STi5518

SINGLE-CHIP SET-TOP BOX DECODER WITH MP3 AND HARD DISK DRIVE SUPPORT

DATA SHEET

- **Integrated 32-bit host CPU up to 81 MHz**
  - 2 Kbytes of Icache, 2 Kbytes of Dcache, and 4 Kbytes of SRAM configurable as Dcache.
- **Audio decoder**
  - 5.1 channel Dolby Digital® /MPEG-2 multi-channel decoding, 3 X 2-channel PCM outputs
  - IEC60958 -IEC61937 digital output
  - SRS®/TruSurround®
  - DTS® digital out and MP3 decoding
  - Alignment beep for satellite dishes.
- **Video decoder**
  - Supports MPEG-2 MP@ML
  - Fully programmable zoom-in and zoom-out
  - NTSC to PAL conversion.
- **DVD and SVCD subpicture decoder**
- **High performance on-screen display**
  - 2 to 8 bits per pixel OSD options
  - Anti-flicker, anti-flutter and anti-aliasing filters.
- **PAL/NTSC/SECAM encoder**
  - RGB, CVBS, Y/C and YUV outputs with 10-bit DACs
  - Macrovision® 7.01/6.1 compatible (optional).
- **Shared SDRAM memory interface**
  - 1 or 2x16-Mbit, or 1x64-Mbit 125 MHz SDRAM.
- **Programmable CPU memory interface for SDRAM, ROM, peripherals...**
- **Front-end interface**
  - DVD, VCD, SVCD and CD-DA compatible
  - Serial, parallel and ATAPI interfaces
  - Hardware sector filtering
  - Integrated CSS decryption and track buffer.
- **Hardware transport-stream demultiplexor**
  - Parallel/serial input
  - DES and DVB descramblers
  - 32 PID support.
- **Integrated peripherals**
  - 2 UARTs, 2 SmartCards, I^2C controller, 3 PWM outputs, 3 capture timers
  - Modem support
  - 44 bits of programmable I/O
  - IR transmitter/receiver.
- **Professional toolset support**
  - ANSI C compiler and libraries.
- **208 pin PQFP package.**

The STi5518 is a highly integrated single-chip decoder, designed for use in feature-rich mass-market set-top boxes. It integrates a high-performance 32-bit CPU, a dedicated block for DVB/DirecTV transport demultiplexing and descrambling, modules for MPEG-2 video and audio decoding with 3D-surround and MP3 support, advanced display and graphics features, a digital video encoder and all of the system peripherals required in a typical low-cost interactive receiver.

To cover the needs of DVD-capable set-top boxes, STi5518 integration options include a CSS decryption block, a Dolby Digital audio decoder and Macrovision copy protection.

An ATAPI interface is built-in, supporting the glueless connection of standard Hard Disk Drives. In this way, the STi5518 is ideal for set-top boxes featuring trick modes such as live TV recording, pausing and time-shifting.

The STi5518 is backward compatible with the popular STi5500 set-top box decoder, allowing easy migration from the previous generation.

The high level of integration in a single PQFP-208 package makes the STi5518 ideally suited for low-cost, high-volume set-top box applications.

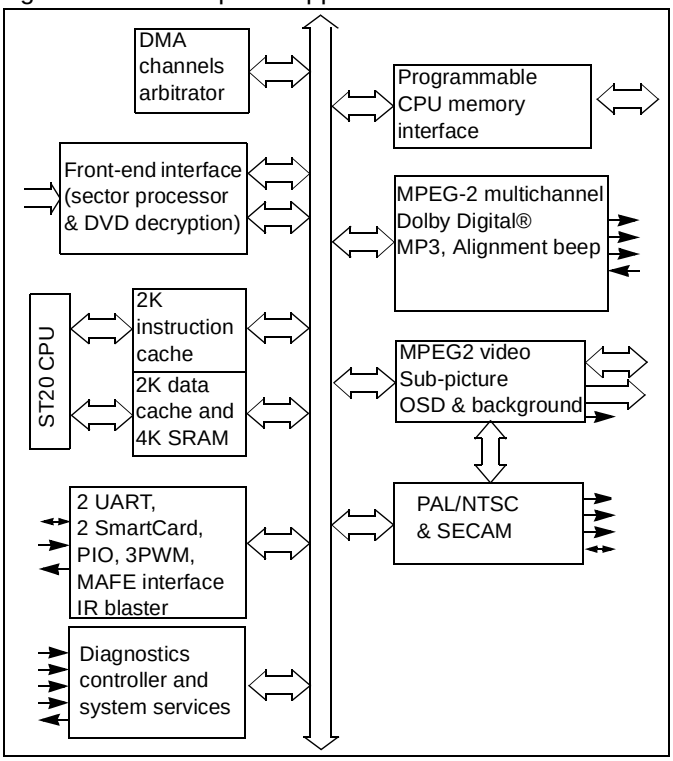


Table of contents

1 Architecture overview

1.1 Introduction

The figure below shows the architecture of the STi5518.

This chapter gives a brief overview of each of the functional blocks of the STi5518.

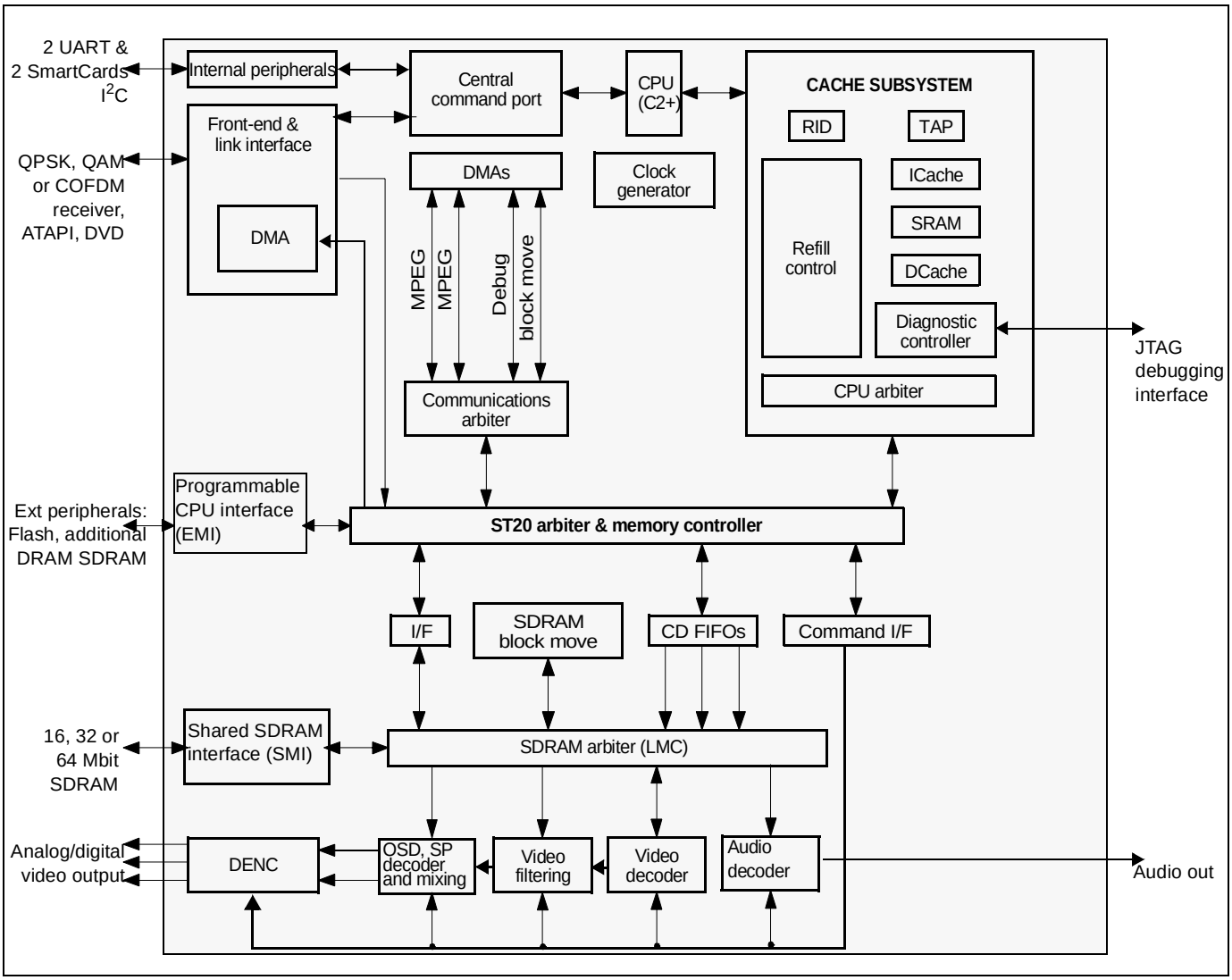


Figure 1 Functional block diagram

1.2 Central processor

The STi5518 Central Processing Unit is a ST20C2+ 32-bit processor core. It contains instruction processing logic, instruction and data pointers, and an operand register. It directly accesses the high-speed on-chip SRAM, which can store data or programs and uses the cache to reduce access time to off-chip program and data memory.

The processor can access memory via the Programmable CPU Interface (often referred to as the EMI) or the Shared Memory Interface (SMI), which is shared with the video, audio, sub-picture and OSD decoders.

1.3 MPEG video decoder

This is a real-time video compression processor supporting the MPEG-1 and MPEG-2 standards at video rates up to 720 x 480 x 60 Hz and 720 x 576 x 50 Hz. Picture format conversion for display is performed by vertical and horizontal filters. User-defined bitmaps can be super-imposed on the display picture by using the on-screen display function.

The display unit is part of the MPEG video decoder, it overlays the four display planes shown in the figure below. The display planes are normally overlaid in the order illustrated, with the background color at the back and the sub-picture at the front (used as a cursor plane). The sub-picture plane can alternatively be positioned between the OSD and MPEG video planes where it can be used as a second on-screen display plane.

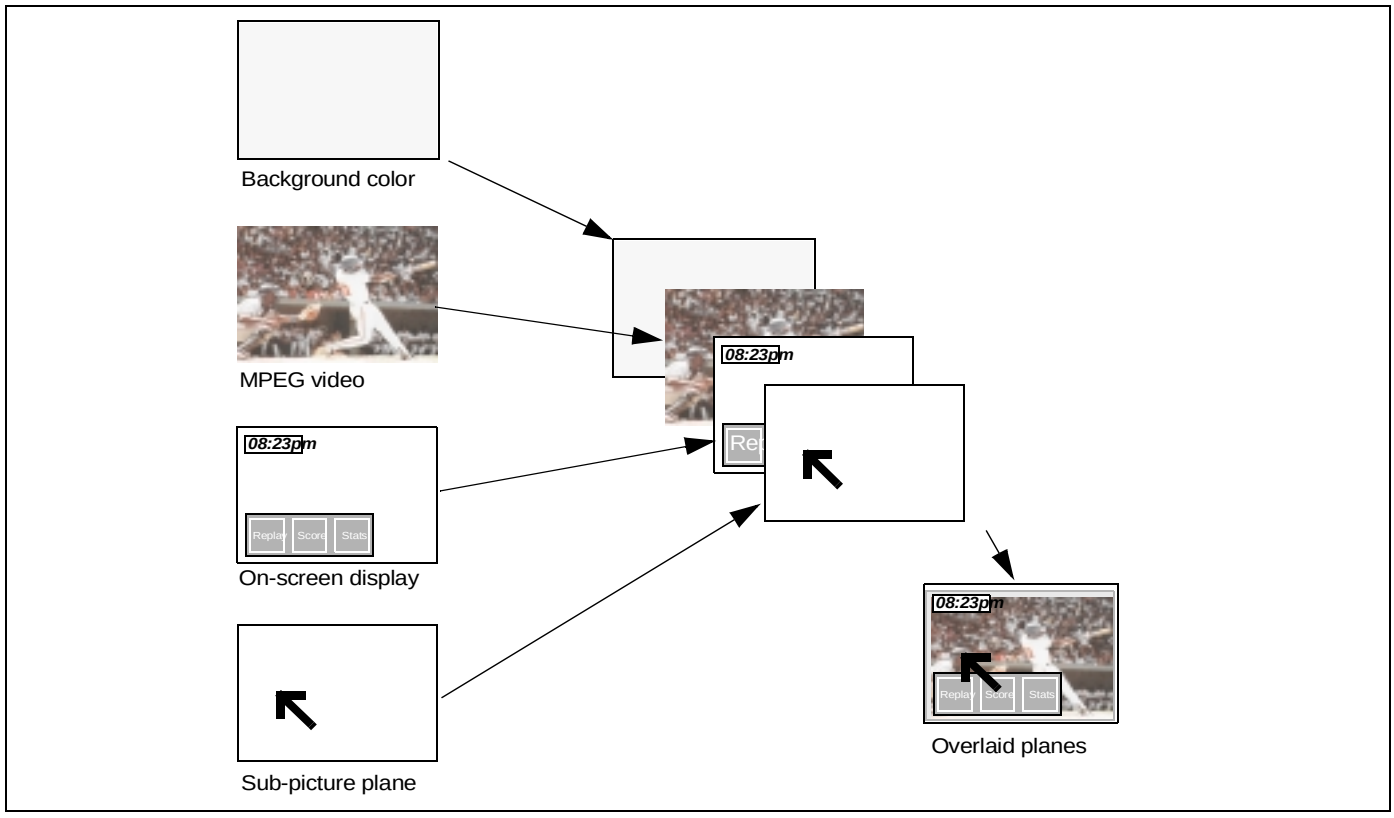


Figure 2 Display planes

1.4 Audio decoder

The audio decoder accepts: Dolby Digital, MPEG-1 layers I, II and III, MPEG-2 layer II 6-channel, PCM, CDDA data formats; MPEG2 PES streams for MPEG-2, MPEG-1, Dolby Digital, MP3, and Linear PCM (LPCM). The audio decoder supports DTS® digital out (DVD DTS and CDDA DTS).

SPDIF input data (IEC-60958 or IEC-61937 standards) is accepted if an external circuitry extracts the PCM clock from the stream.

Skip frame, repeat blocks and soft mute frame features can be used to synchronize audio and video data. PTS audio extraction is also supported.

The device outputs up to 6 channels of PCM data and appropriate clocks for external digital-to-analog converters.

Programmable downmix enables 1,2,3 or 4 channel outputs. Data can be output in either I²S format or Sony format. The decoder can format output data according to IEC-60958 standard (for non compressed data: L/R channels, 16, 18, 20 and 24-bits) or IEC-61937 standard (for compressed data), for F_S = 96 kHz, 48 kHz, 44.1 kHz or 32 kHz.

Sampling frequencies of 96 kHz, 48 kHz, 44.1 kHz, 32 kHz and half sampling frequencies are supported. A downsampling filter (96 kHz/48 kHz) is available.

The decoder supports dual mode for MPEG and Dolby Digital. It includes a Dolby surround compatible downmix and a ProLogic decoder.

A pink noise generator enables the accurate positioning of speakers for optimal surround sound setup.

PCM beep tone is a special mode used for Set Top Box. It generates a triangular signal of variable frequency and amplitude on the left and right channels.

In global mute mode, the decoder decodes the incoming bitstream normally but the PCM and SPDIF outputs are softmuted. This mode is used to prepare a period of decoding mode, to synchronize audio and video data without hearing the audio.

Slow-forward and fast-forward trick modes are available for compressed and non-compressed data.

The control interface of the decoder is activated via memory mapped registers in the ST20 address space.

1.5 IR transmitter/receiver

The STi5518 provides a pulse-position modulated signal for automatic VCR programming by the set-top box. The signal is output to the IR blast pin and an accessory jack pin, simultaneously. The pulse frequency, number of pulses (envelope length) and the total cycle time is controlled by registers.

1.6 Modem analog front-end interface

The Modem Analog Front-end interface is used to transfer transmit and receive DAC and ADC samples between the memory and an external modem analog front-end (MAFE), using a synchronous serial protocol. DMA is used to transfer the sample data between memory buffers and the MAFE interface module, with separate transmit and receive buffers and double buffering of the buffer pointers. FIFOs are used to take into account the access latency to memory, in a worst case system and to allow the use of bursts for memory bandwidth efficiency improvement. The V22 bis standard is supported.

1.7 Memory subsystem

On-chip

The on-chip memory includes 2Kbytes of instruction cache, 2Kbytes of data cache and 4Kbytes of SRAM that can be optionally configured as data cache. The subsystem provides 240M/bytes of internal bandwidth, supporting pipelined 2-cycle internal memory access.

The instruction and data caches are direct-mapped, with a write-back system for the data-cache. The caches support burst accesses to the external memories for refill and write-back. Burst access increases the performance of page-mode DRAM memories.

Off-chip

There are two off-chip memory interfaces:

- The external memory interface (EMI) accessed by the ST20 is used for the transfer of data and programs between the STi5518 and external peripherals, flash and additional SDRAM and DRAM.
- Shared memory interface (SMI) controls the movement of data between the STi5518 and 16, 32 or 64 Mbits of SDRAM. This external SDRAM stores the display data generated by the MPEG decoder and CPU and the C2+ code data.

The EMI uses minimal external support logic to support memory subsystems, and accesses a 32 Mbytes of physical address space (greater if SDRAM or DRAM is used) in four general purpose memory banks of 8 or 16 bits wide, 21 or 22 address lines, and byte select. For applications requiring extra memory, the EMI supports this extra memory with zero external support logic, even for 16-bit SDRAM devices. The EMI can be configured for a wide variety of timing and decode functions by the configuration registers. The timing of each of the four memory banks can be set separately, with different device types being placed in each bank with no need for external hardware.

1.8 Serial communication

Asynchronous serial controllers

The Asynchronous Serial Controller (ASC), also referred to as the UART interface, provides serial communication between the STi5518 and other microcontrollers, microprocessors or external peripherals. The STi5518 has four ASCs, two of which are generally used by the SmartCard controllers.

Eight or nine bit data transfer, parity generation, and the number of stop bits are programmable. Parity, framing, and overrun error detection increase data transfer reliability. Transmission and reception of data can be double-buffered, or 16-deep FIFOs can be used. A mechanism to distinguish the address from the data bytes is included for multiprocessor communication. Testing is supported by a loop-back option. A 16-bit baud-rate generator provides the ASC with a separate serial clock signal.

Two ASCs support full-duplex and 2 half-duplex asynchronous communication, where both the transmitter and the receiver use the same data frame format and the same baud rate. Each ASC can be set to operate in SmartCard mode for use when interfacing to a SmartCard.

Synchronous serial controller

Two Synchronous Serial Controllers (SSC) provide high-speed interfaces to a wide variety of serial memories, remote control receivers and other microcontrollers. The SSCs support all of the features of the Serial Peripheral Interface bus (SPI) and the I^2C bus. The SSCs can be programmed to interface to other serial bus standards. The SSCs share pins with the parallel input/output (PIO) ports, and support half-duplex synchronous communication.

1.9 Front-end interface

The STi5518 can be connected to a front-end through the following interfaces:

- I^2S interface;
- multi-format serial interface;
- multi-format parallel interface;
- ATAPI interface (for Hard Disk Drives and DVD-ROMs)

1.10 On-chip PLL

The on-chip PLL accepts 27 MHz input and generates all the internal high-frequency clocks needed for the CPU, MPEG and audio subsystems.

1.11 Diagnostic controller (DCU)

The ST20 Diagnostic Controller Unit (DCU) is used to boot the CPU and to control and monitor the chip systems via the standard IEEE 1194.1 Test Access Port. The DCU includes on-chip hardware with ICE (In Circuit Emulation) and LSA (Logic State Analyzer) features to facilitate verification and debugging of software running on the on-chip CPU in real time. It is an independent hardware module with a private link from the host to support real-time diagnostics.

1.12 Interrupt subsystem

The interrupt system allows an on-chip module or external interrupt pin to interrupt an active process so that an interrupt handling process can be run. An interrupt can be signalled by one of the following: a signal on an external interrupt pin, a signal from an internal peripheral or subsystem, software asserting an interrupt in the pending register.

Interrupts are implemented by an on-chip interrupt controller and an on-chip interrupt-level controller. The interrupt controller supports eight prioritized interrupts as inputs and manages the pending interrupts. This allows the nesting of pre-emptive interrupts for real-time system design. Each interrupt can be programmed to be at a lower or higher priority than the high priority process queue.

1.13 PAL/NTSC/SECAM encoder

The integrated digital encoder converts a multiplexed 4:2:2 or 4:4:4 YCbCr stream into a standard analog baseband PAL/NTSC or SECAM signal and into RGB, YUV, Yc and CVBS components. The encoder can perform closed-caption, CGMS encoding, and allows Macrovision™ 7.01/6.1 copy protection.

The DENC is able to encode Teletext according to the “CCIR/ITU-R Broadcast Teletext System B” specification, also known as “World System Teletext”.

In DVB applications, Teletext data is embedded within DVB streams as MPEG data packets. It is the responsibility of the software to handle incoming data packets and in particular to store Teletext packets in a buffer, which then passes them to the DENC on request.

1.14 SmartCard interfaces

Two SmartCard interfaces support SmartCards compliant with ISO7816-3. Each interface is has a UART (ASC), a dedicated programmable clock generator, and eight bits of parallel IO port.

1.15 PWM and counter module

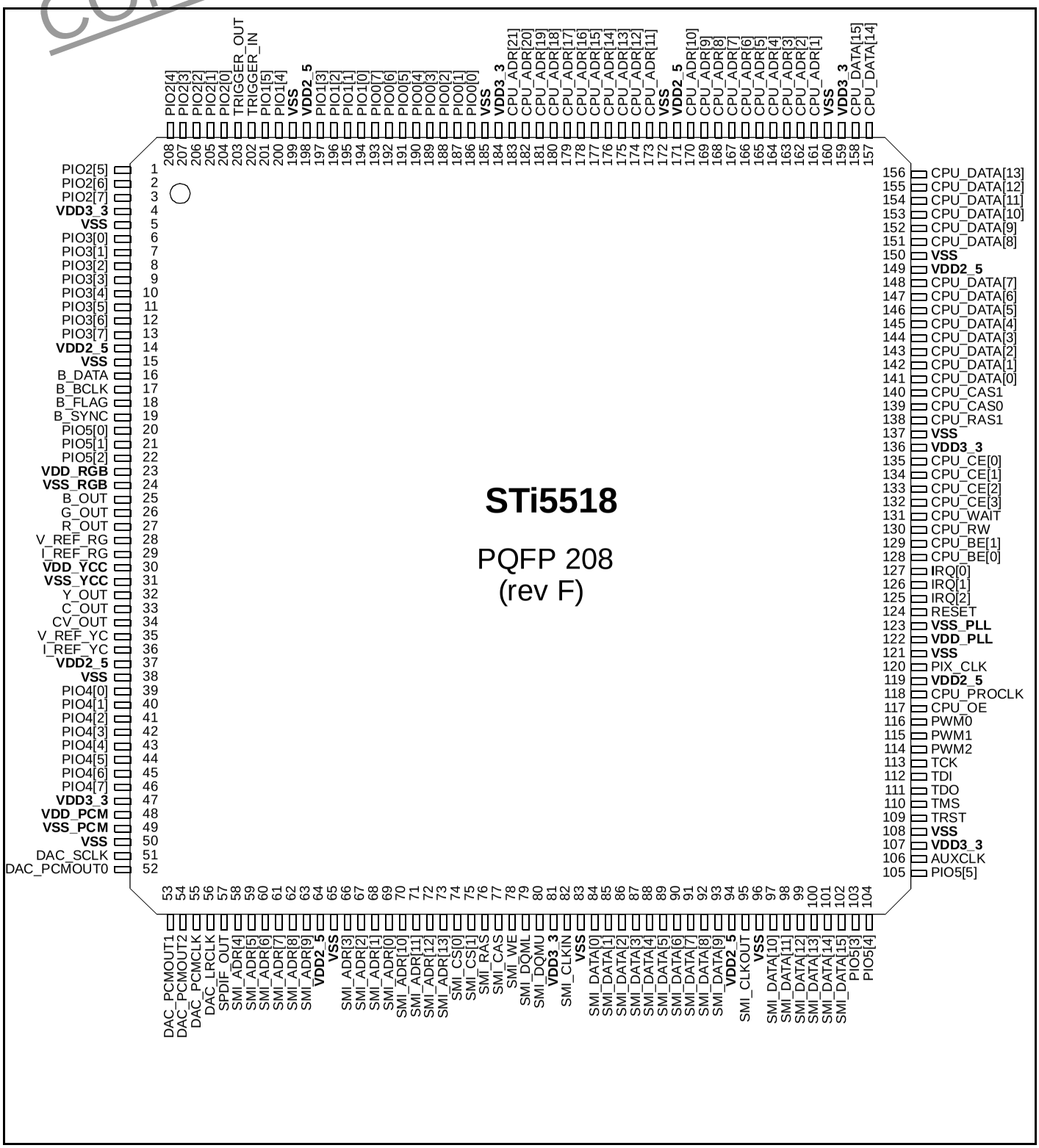
The PWM and counter module provides three PWM encoder outputs, three PWM decoder (capture) inputs and four programmable timers. Each capture input can be programmed to detect rising edge, falling edge, both edges or neither edge (disabled). These facilities are clocked by two independent clocks, one for PWM outputs and one for capture inputs/timers. The PWM counter is 8-bit, with 8-bit registers to set the output-high time. The capture/compare counter and the compare and capture registers are 32-bit. The module generates a single interrupt signal.

1.16 Parallel I/O module

44 bits of parallel I/O are configured in 6 ports, and each bit is programmable as output or input. The output can be configured as a totem-pole or open-drain driver. The input compare logic can generate an interrupt on any change of any input bit. Many parallel IO have alternate functions and can be connected to an internal peripheral signal such as a UART or SSC.

2 Pin data

2.1 Pin out



2.2 Pin list sorted by function

Alternate functions printed in *Italic* show a suggested use of the PIO; alternate functions not printed in *Italic* are multiplexed with a specific hardware.

Pin number	Pin name	Main function	Alternate function		Type
			Input	Output	
Audio DAC					
51	DAC_SCLK	over sampling clock		EXT_AUD_CLK	O
52	DAC_PCMOUT0	PCM output 0		EXT_AUD_DATA	O
53	DAC_PCMOUT1	PCM output 1	EXT_AUD_REQ		I/O
54	DAC_PCMOUT2	PCM output 2			O
55	DAC_PCMCLK	PCM clock			I/O
56	DAC_LRCLK	Left/right clock		EXT_AUD_WCLK	O
57	SPDIF_OUT	SPDIF output			O
48	VDD_PCM	VDD freq synthesizer=2.5V			PWR 2.5V
49	VSS_PCM	VSS freq synthesizer=GND			PWR
Clock & reset					
124	RESET	Chip reset			I
122	VDD_PLL	VDD PLL=2.5V			PWR 2.5V
123	VSS_PLL	GND PLL=GND			PWR
120	PIX _CLK	27 MHz main clock			I
PIOs and communication					
186	PIO0[0]	PIO0[0]	UART0_DATA *(SC0_DATA)*		I/O
187	PIO0[1]	PIO0[1]	TTX_IN_CLOCK	ATAPI_RD	I/O
188	PIO0[2]	PIO0[2]		ATAPI_WR	I/O
189	PIO0[3]	PIO0[3]		SC0_CLOCK	I/O
190	PIO0[4]	PIO0[4]		SC0_RST	I/O
191	PIO0[5]	PIO0[5]		SC0_CMD_VCC	I/O
192	PIO0[6]	PIO0[6]		SC0_DATA_DIR	I/O
193	PIO0[7]	PIO0[7]	SC0_DETECT		I/O
194	PIO1[0]	PIO1[0]	SSC0_DATA (MTSROut/MRSTin)		I/O
195	PIO1[1]	PIO1[1]	SSC0_CLOCK		I/O
196	PIO1[2]	PIO1[2]	SC EXTERNAL CLOCK PARA_DVALID		I/O
197	PIO1[3]	PIO1[3]		UART2_TXD	I/O
200	PIO1[4]	PIO1[4]	UART2_RXD		I/O
201	PIO1[5]	PIO1[5]	PARA_SYNC	UART1_TXD	I/O
202	TRIGGER_IN	Trigger input for DCU			I/O
203	TRIGGER_OUT	Trigger output for DCU			I/O
204	PIO2[0]	PIO2[0]	UART3_DATA *(SC1_DATA)*		I/O
205	PIO2[1]	PIO2[1]	UART1_RXD	MAFEIF_DOUT PARA_REQ	I/O

Table 1 Pins sorted by function

Pin number	Pin name	Main function	Alternate function		Type
			Input	Output	
206	PIO2[2]	PIO2[2]	PARA_STR	MAFEIF_HC1	I/O
207	PIO2[3]	PIO2[3]		SC1_CLOCK	I/O
208	PIO2[4]	PIO2[4]		SC1_RST	I/O
1	PIO2[5]	PIO2[5]		SC1_CMD_VCC	I/O
2	PIO2[6]	PIO2[6]		SC1_DATA_DIR	I/O
3	PIO2[7]	PIO2[7]	SC1_DETECT		I/O
6	PIO3[0]	PIO3[0]	MAFEIF_SCLK PARA_DATA{0]		I/O
7	PIO3[1]	PIO3[1]	MAFEIF_DIN PARA_DATA[1]		I/O
8	PIO3[2]	PIO3[2]	MAFEIF_FSI PARA_DATA[2]		I/O
9	PIO3[3]	PIO3[3]	CAPTURE_IN0 PARA_DATA[3]		I/O
10	PIO3[4]	PIO3[4]	CAPTURE_IN1 PARA_DATA[4]	UART1 RTS (RTS1)	I/O
11	PIO3[5]	PIO3[5]	CAPTURE_IN2 PARA_DATA[5]	UART2 RTS (RTS2)	I/O
12	PIO3[6]	PIO3[6]	PARA_DATA[6] UART1 CTS (CTS1)	COMP_OUT1	I/O
13	PIO3[7]	PIO3[7]	PARA_DATA[7] UART2 CTS (CTS2)	COMP_OUT0	I/O
39-46	PIO4[0:7]	PIO4[0:7]		YC[0:7]	I/O
20	PIO5[0]	PIO5[0]	B_WCLK		I/O
			SSC1_DATA/ NRSS_CLOCK[1]		
21	PIO5[1]	PIO5[1]	B_V4	NRSS_OUT [2 and 1]	I/O
			SSC1_CLOCK		
22	PIO5[2]	PIO5[2]	IRB_IRinput/NRSS_IN[2]		I/O
			SDAV_CLK/ P1394_Clk[3]		
103	PIO5[3][4]	PIO5[3]	IRB_UHFinput[4]		I/O
			SDAV_DATA[3]		
104	PIO5[4]	PIO5[4]		IRB_drivePPMsignal	I/O
			SDAV_DIR / P1394_P_CLK[3]		
105	PIO5[5]	PIO5[5]		IRB_drive0orZ[5] (jack)	I/O
			OSC_IN_CLK[3]		
Auxiliary Clock					
106	Auxiliary Clock				O
EMI Interface					
161-170	CPU_ADR[1:10]	Address[1:10]			O
173-183	CPU_ADR[11:21]	Address[11:21]			O
141-148	CPU_DATA[0:7]	Data[0:7]			I/O

Table 1 Pins sorted by function

Pin number	Pin name	Main function	Alternate function		Type
			Input	Output	
151-158	CPU_DATA[8:15]	Data[8:15]			I/O
138	CPU_RAS1	DRAM RAS		NOT_SDRAM_CS1 CHIPSEL. BANK3	I/O
131	CPU_WAIT	Wait state			I
130	CPU_RW	Read-not-write		NOT_SDRAM_WE	O
128	CPU_BE[0]	Byte 0 enable		DQM[0]	O
129	CPU_BE[1]	Byte 1 enable		DQM[1]	O
139	CPU_CAS0	DRAM CAS 0		SDRAM_CAS/ CPU_ADR[22]	O
140	CPU_CAS1	DRAM		NOT_SDRAM_CS0	O
135	CPU_CE[0]	DRAM RAS 0		SDRAM_RAS NOTCHIPSELBANK0	O
134	CPU_CE[1]	Chip sel. bank 1			O
133	CPU_CE[2]	Chip sel. bank 2			O
132	CPU_CE[3]	Chip sel. bank 3		CS_SUB_BANK3	O
118	CPU_PROCLK	EMI clock			O
117	CPU_OE	Output enable			I/O
Interrupt					
127	IRQ[0]	IRQ[0] *(SERVO_IRQ)*			I
126	IRQ[1]	IRQ[1] *(ATAPI IRQ)*			I
125	IRQ[2]	IRQ[2] *(MD_IRQ)*			I
Timers					
116	PWM0	Pulse Width Modulator 0	HSYNC		I/O
115	PWM1	Pulse Width Modulator 1	BOOT_FROM_ROM[6]		I/O
114	PWM2	Pulse Width Modulator 2	VSYNC		I/O
JTAG					
113	TCK	Test clock			I
112	TDI	Test data in			I
111	TDO	Test data out			O
110	TMS	Test mode select			I
109	TRST[7]	Test reset			I
Front-end					
16	B_DATA	I^2S data	FEC_DATA		I
17	B_BCLK	I^2S bit clock	FEC_B_CLK		I
18	B_FLAG	I^2S error flag dvd	FEC_D_VALID (DVD) FEC_P_CLK (DVB/DSS)		I
19	B_SYNC	I^2S sector/ABS time	FEC_P_START (DVD) FEC_ERROR (DVB/ DSS)		I
Video DAC					
27, 26, 25	R_OUT, G_OUT, B_OUT	R_OUT, G_OUT, B_OUT			O

Table 1 Pins sorted by function

Pin number	Pin name	Main function	Alternate function		Type
			Input	Output	
32, 33, 34	Y_OUT, C_OUT, CV_OUT	Y_OUT, C_OUT, CV_OUT			O
29	I_REF_RGB	RGB DAC reference current			I
28	V_REF_RGB	RGB DAC reference voltage			I
36	I_REF_YCC	YCC DAC reference current			I
35	V_REF_YCC	YCC DAC reference voltage			I
23	VDD_RGB	VDDA_RGB=2.5V			PWR 2.5V
24	VSS_RGB	VSSA_RGB=GND			PWR
30	VDD_YCC	VDDA_YCC=2.5V			PWR 2.5V
31	VSS_YCC	VSSA_YCC=GND			PWR
Shared memory interface					
69-66	SMI_ADR[0:3]	Address bus SDRAM			O
58-63	SMI_ADR[4:9]	Address bus SDRAM			O
70-73	SMI_ADR [10:13]	Address bus SDRAM			O
84-93, 97-102	SMI_DATA[0:15]	Data bus SDRAM			I/O
74, 75	SMI_CS[0,1]	Chip select bank 0,1			O
76	SMI_RAS	RAS SDRAM			O
77	SMI_CAS	CAS SDRAM			O
78	SMI_WE	SDRAM write enable			O
79, 80	SMI_DQML, U	DQ mask en low, up			O
82	SMI_CLKIN	SDRAM clock in			I
95	SMI_CLKOUT	SDRAM clock out			O
Power supply					
4, 47, 81, 107, 136, 159, 184	VDD3_3	3.3 V POWER SUPPLY			PWR
14, 37, 64, 94, 119, 149, 171, 198	VDD2_5	2.5V POWER SUPPLY			PWR
5, 15, 38, 50, 65, 83, 96, 108, 121, 137, 150, 160, 172, 185, 199	VSS	Ground			PWR

Table 1 Pins sorted by function

1. FEI_CFG bits 8 and 9 must be programmed according to the required NRSS configuration.
2. The NRSS_IN and NRSS_OUT pins are swapped around on the STi5518 compared to the STi5508.
3. Register LNK_SDAV_CONF bit 22 (SDE) must be set to 1 to validate the output path.
4. Inverted. ATTENTION! the PIO input is also inverted.
5. The PIO must be configured in open drain.
6. BOOT_FROM_ROM is active during reset.
7. Tie low whenever JTAG is not used.

2.3 Pins sorted by pin number

Pin N°	Pin name	Main function	Alternate function		Dir func.
			Input	Output	
Left Side					
1	PIO2[5]	PIO2[5]		SC1_CMD_VCC	I/O
2	PIO2[6]	PIO2[6]		SC1_DATA_DIR	I/O
3	PIO2[7]	PIO2[7]	SC1_DETECT		I/O
4	VDD3_3	3.3 V power supply			POWER
5	VSS	Ground			POWER
6	PIO3[0]	PIO3[0]	MAFEIF_SCLK PARA_DATA{0]		I/O
7	PIO3[1]	PIO3[1]	MAFEIF_DIN PARA_DATA[1]		I/O
8	PIO3[2]	PIO3[2]	MAFEIF_FSI PARA_DATA[2]		I/O
9	PIO3[3]	PIO3[3]	CAPTURE_IN0 PARA_DATA[3]		I/O
10	PIO3[4]	PIO3[4]	CAPTURE_IN1 PARA_DATA[4]	UART1 RTS (RTS1)	I/O
11	PIO3[5]	PIO3[5]	CAPTURE_IN2 PARA_DATA[5]	UART2 RTS (RTS2)	I/O
12	PIO3[6]	PIO3[6]	PARA_DATA[6] UART1 CTS (CTS1)	COMP_OUT1	I/O
13	PIO3[7]	PIO3[7]	PARA_DATA[7] UART2 CTS (CTS2)	COMP_OUT0	I/O
14	VDD2_5	2.5V power supply			POWER
15	VSS	Ground			POWER
16	B_DATA	I^2S data	FEC_DATA		I
17	B_BCLK	I^2S bit clock	FEC_B_CLK		I
18	B_FLAG	I^2S error flag DVD	FEC_D_VALID (DVD) FEC_P_CLK (DVB/DSS)		I
19	B_SYNC	I^2S sector/ABS time	FEC_P_START (DVD) FEC_ERROR (DVB/DSS)		I
20	PIO5[0]	PIO5[0]	B_WCLK		I/O
			SSC1_DATA/ NRSS_CLOCK[1]		
21	PIO5[1]	PIO5[1]	B_V4	NRSS_OUT[2 and 1]	I/O
			SSC1_CLOCK		
22	PIO5[2]	PIO5[2]	IRB_IRinput/NRSS_IN[2]		I/O
			SDAV_CLK/ P1394_CLK[3]		
23	VDD_RGB	VDDA_RGB=2.5V			POWER
24	VSS_RGB	VSSA_RGB=GND			POWER
25	B_OUT	B output			O

Table 2 Pins sorted by number

Pin N°	Pin name	Main function	Alternate function		Dir func.
			Input	Output	
26	G_OUT	G output			O
27	R_OUT	R output			O
28	V_REF_RGB	RGB DAC reference voltage			I
29	I_REF_RGB	RGB DAC reference current			I
30	VDD_YCC	VDDA_YCC=2.5V			POWER
31	VSS_YCC	VSSA_YCC=GND			POWER
32	Y_OUT	Y output			O
33	C_OUT	C output			O
34	CV_OUT	CV output			O
35	V_REF_YCC	YCC DAC reference voltage			I
36	I_REF_YCC	YCC DAC reference current			I
37	VDD2_5	2.5V power supply			POWER
38	VSS	Ground			POWER
39	PIO4[0]	PIO4[0]		YC[0]	I/O
40	PIO4[1]	PIO4[1]		YC[1]	I/O
41	PIO4[2]	PIO4[2]		YC[2]	I/O
42	PIO4[3]	PIO4[3]		YC[3]	I/O
43	PIO4[4]	PIO4[4]		YC[4]	I/O
44	PIO4[5]	PIO4[5]		YC[5]	I/O
45	PIO4[6]	PIO4[6]		YC[6]	I/O
46	PIO4[7]	PIO4[7]		YC[7]	I/O
47	VDD3_3	3.3 V power supply			POWER
48	VDD_PCM	VDD freq synthesizer=2.5V			POWER
49	VSS_PCM	VSS freq synthesizer=GND			POWER
50	VSS	Ground			POWER
51	DAC_SCLK	Sampling clock		EXT_AUD_CLK	O
52	DAC_PCMOUT0	PCM output 0		EXT_AUD_DATA	O
Bottom side					
53	DAC_PCMOUT1	PCM output 1	EXT_AUD_REQ		I/O
54	DAC_PCMOUT2	PCM output 2			O
55	DAC_PCMCLK	PCM clock			I/O
56	DAC_LRCLK	Left/right clock		EXT_AUD_WCLK	O
57	SPDIF_OUT	SPDIF output			O
58	SMI_ADR[4]	Address bus SDRAM			O
59	SMI_ADR[5]	Adress bus SDRAM			O
60	SMI_ADR[6]	Adress bus SDRAM			O
61	SMI_ADR[7]	Adress bus SDRAM			O
62	SMI_ADR[8]	Adress bus SDRAM			O
63	SMI_ADR[9]	Adress bus SDRAM			O
64	VDD2_5	2.5V power supply			POWER

Table 2 Pins sorted by number

Pin N°	Pin name	Main function	Alternate function		Dir func.
			Input	Output	
65	VSS	Ground			POWER
66	SMI_ADR[3]	Adress bus SDRAM			O
67	SMI_ADR[2]	Adress bus SDRAM			O
68	SMI_ADR[1]	Adress bus SDRAM			O
69	SMI_ADR[0]	Adress bus SDRAM			O
70	SMI_ADR[10]	Adress bus SDRAM			O
71	SMI_ADR[11]	Adress bus SDRAM			O
72	SMI_ADR[12]	Adress bus SDRAM			O
73	SMI_ADR[13]	Adress bus SDRAM			O
74	SMI_CS[0]	Chip select bank 0			O
75	SMI_CS[1]	Chip select bank 1			O
76	SMI_RAS	RAS SDRAM			O
77	SMI_CAS	CAS SDRAM			O
78	SMI_WE	SDRAM write enable			O
79	SMI_DQML	DQ mask en low			O
80	SMI_DQMU	DQ mask en up			O
81	VDD3_3	3.3 V power supply			POWER
82	SMI_CLKIN	SDRAM clock in			I
83	VSS	Ground			POWER
84	SMI_DATA[0]	Data bus SDRAM			I/O
85	SMI_DATA[1]	Data bus SDRAM			I/O
86	SMI_DATA[2]	Data bus SDRAM			I/O
87	SMI_DATA[3]	Data bus SDRAM			I/O
88	SMI_DATA[4]	Data bus SDRAM			I/O
89	SMI_DATA[5]	Data bus SDRAM			I/O
90	SMI_DATA[6]	Data bus SDRAM			I/O
91	SMI_DATA[7]	Data bus SDRAM			I/O
92	SMI_DATA[8]	Data bus SDRAM			I/O
93	SMI_DATA[9]	Data bus SDRAM			I/O
94	VDD2_5	2.5V power supply			POWER
95	SMI_CLKOUT	SDRAM clock out			O
96	VSS	Ground			POWER
97	SMI_DATA[10]	Data bus SDRAM			I/O
98	SMI_DATA[11]	Data bus SDRAM			I/O
99	SMI_DATA[12]	Data bus SDRAM			I/O
100	SMI_DATA[13]	Data bus SDRAM			I/O
101	SMI_DATA[14]	Data bus SDRAM			I/O
102	SMI_DATA[15]	Data bus SDRAM			I/O
103	PIO5[3]	PIO5[3]	IRB_UHFinput[4]		I/O
			SDAV_DATA[3]		

Table 2 Pins sorted by number

Pin N°	Pin name	Main function	Alternate function		Dir func.
			Input	Output	
104	PIO5[4]	PIO5[4]		IRB_drivePPM signal	I/O
			Sdav_dir / P1394_P_CLK[3]		
Right side					
105	PIO5[5]	PIO5[5]		IRB_drive0orZ[5] (jack)	I/O
			OSC_IN_CLK[3]		
106	Auxiliary Clock				O
107	VDD3_3	3.3 V power supply			POWER
108	VSS	Ground			POWER
109	TRST[6]	Test reset			I
110	TMS	Test mode select			I
111	TDO	Test data out			O
112	TDI	Test data in			I
113	TCK	Test clock			I
114	PWM2	Pulse Width Modulator 2	VSYNC		I/O
115	PWM1	Pulse Width Modulator 1	BOOT_FROM_ROM[7]		I/O
116	PWM0	Pulse Width Modulator 0	HSYNC		I/O
117	CPU_OE	Output enable			I/O
118	CPU_PROCLK	EMI clock			O
119	VDD2_5	2.5V power supply			POWER
120	PIX _CLK	27 MHz main clock			I
121	VSS	Ground			POWER
122	VDD_PLL	VDD PLL=2.5V			POWER
123	VSS_PLL	GND PLL=GND			POWER
124	RESET	Chip reset			I
125	IRQ[2]	IRQ[2] *(MD_IRQ)*			I
126	IRQ[1]	IRQ[1] *(ATAPI IRQ)*			I
127	IRQ[0]	IRQ[0] *(SERVO_IRQ)*			I
128	CPU_BE[0]	Byte 0 enable		DQM[0]	O
129	CPU_BE[1]	Byte 1 enable		DQM[1]	O
130	CPU_RW	Read-not-write		NOT_SDRAM_WE	O
131	CPU_WAIT	Wait state			I
132	CPU_CE[3]	Chip select bank 3		CS_SUB_BANK3	O
133	CPU_CE[2]	Chip select bank 2			O
134	CPU_CE[1]	Chip select bank 1			O
135	CPU_CE[0]	DRAM RAS 0		SDRAM_RAS	O
136	VDD3_3	3.3 V power supply			POWER
137	VSS	Ground			POWER
138	CPU_RAS1	DRAM RAS 1		NOT_SDRAM_CS1 CHIPSEL. BANK3	I/O

Table 2 Pins sorted by number

Pin N°	Pin name	Main function	Alternate function		Dir func.
			Input	Output	
139	CPU_CAS0	DRAM CAS 0		SDRAM_CAS CPU_ADR[22]	O
140	CPU_CAS1	DRAM CAS 1		NOT_SDRAM_CS0	O
141	CPU_DATA[0]	Data[0]			I/O
142	CPU_DATA[1]	Data[1]			I/O
143	CPU_DATA[2]	Data[2]			I/O
144	CPU_DATA[3]	Data[3]			I/O
145	CPU_DATA[4]	Data[4]			I/O
146	CPU_DATA[5]	Data[5]			I/O
147	CPU_DATA[6]	Data[6]			I/O
148	CPU_DATA[7]	Data[7]			I/O
149	VDD2_5	2.5V power supply			POWER
150	VSS	Ground			POWER
151	CPU_DATA[8]	Data[8]			I/O
152	CPU_DATA[9]	Data[9]			I/O
153	CPU_DATA[10]	Data[10]			I/O
154	CPU_DATA[11]	Data[11]			I/O
155	CPU_DATA[12]	Data[12]			I/O
156	CPU_DATA[13]	Data[13]			I/O
Top side					
157	CPU_DATA[14]	Data[14]			I/O
158	CPU_DATA[15]	Data[15]			I/O
159	VDD3_3	3.3 V power supply			POWER
160	VSS	Ground			POWER
161	CPU_ADR[1]	Address[1]			O
162	CPU_ADR[2]	Address[2]			O
163	CPU_ADR[3]	Address[3]			O
164	CPU_ADR[4]	Address[4]			O
165	CPU_ADR[5]	Address[5]			O
166	CPU_ADR[6]	Address[6]			O
167	CPU_ADR[7]	Address[7]			O
168	CPU_ADR[8]	Address[8]			O
169	CPU_ADR[9]	Address[9]			O
170	CPU_ADR[10]	Address[10]			O
171	VDD2_5	2.5V power supply			POWER
172	VSS	Ground			POWER
173	CPU_ADR[11]	Address[11]			O
174	CPU_ADR[12]	Address[12]			O
175	CPU_ADR[13]	Address[13]			O
176	CPU_ADR[14]	Address[14]			O

Table 2 Pins sorted by number

Pin N°	Pin name	Main function	Alternate function		Dir func.
			Input	Output	
177	CPU_ADR[15]	Address[15]			O
178	CPU_ADR[16]	Address[16]			O
179	CPU_ADR[17]	Address[17]			O
180	CPU_ADR[18]	Address[18]			O
181	CPU_ADR[19]	Address[19]			O
182	CPU_ADR[20]	Address[20]			O
183	CPU_ADR[21]	Address[21]			O
184	VDD3_3	3.3 V power supply			POWER
185	VSS	Ground			POWER
186	PIO0[0]	PIO0[0]	UART0_DATA *(SC0_DATA)*		I/O
187	PIO0[1]	PIO0[1]	TTX_IN_CLOCK	ATAPI_RD	I/O
188	PIO0[2]	PIO0[2]		ATAPI_WR	I/O
189	PIO0[3]	PIO0[3]		SC0_CLOCK	I/O
190	PIO0[4]	PIO0[4]		SC0_RST	I/O
191	PIO0[5]	PIO0[5]		SC0_CMD_VCC	I/O
192	PIO0[6]	PIO0[6]		SC0_DATA_DIR	I/O
193	PIO0[7]	PIO0[7]	SC0_DETECT		I/O
194	PIO1[0]	PIO1[0]	SSC0_DATA (MTSROut/MRSTin)		I/O
195	PIO1[1]	PIO1[1]	SSC0_CLOCK		I/O
196	PIO1[2]	PIO1[2]	SC EXTERNAL CLOCK PARA_DVALID		I/O
197	PIO1[3]	PIO1[3]		UART2_TXD	I/O
198	VDD2_5	2.5V power supply			POWER
199	VSS	Ground			POWER
200	PIO1[4]	PIO1[4]	UART2_RXD		I/O
201	PIO1[5]	PIO1[5]	PARA_SYNC	UART1_TXD	I/O
202	TRIGGER_IN	Trigger input for DCU			I/O
203	TRIGGER_OUT	Trigger output for DCU			I/O
204	PIO2[0]	PIO2[0]	UART3_DATA *(SC1_DATA)*		I/O
205	PIO2[1]	PIO2[1]	UART1_RXD	MAFEIF_DOUT PARA_REQ	I/O
206	PIO2[2]	PIO2[2]	PARA_STR	MAFEIF_HC1	I/O
207	PIO2[3]	PIO2[3]		SC1_CLOCK	I/O
208	PIO2[4]	PIO2[4]		SC1_RST	I/O

Table 2 Pins sorted by number

1. FEI_CFG bits 8 and 9 must be programmed according to the required NRSS configuration.
2. The NRSS_IN and NRSS_OUT pins are swapped around on the STi5518 compared to the STi5508.
3. Register LNK_SDAV_CONF bit 22 (SDE) must be set to 1 to validate the output path.
4. Inverted. ATTENTION! the PIO input is also inverted.
5. The PIO must be configured in open drain.

6. Tie low whenever JTAG is not used
7. BOOT_FROM_ROM is active during reset.

3 Central processing unit

The STi5518 Central Processing Unit is a ST20C2+ 32-bit processor core. It contains instruction processing logic, instruction and data pointers, and an operand register. It directly accesses the high-speed on-chip SRAM, which can store data or programs, and uses the cache to reduce access time to off-chip program and data memory.

The CPU can access memory via the general purpose external memory interface (EMI) or the local memory interface (LMI), which is shared with the MPEG decoder.

The processor performs the following manipultations:

- fast integer-multiply - 4 cycle multiply;
- fast bit-shift - single cycle barrel shifter;
- byte and part-word handling;
- scheduling and interrupt support;
- 64-bit integer arithmetic support.

The scheduler provides a single level of pre-emption. In addition, multi-level pre-emption is provided by the interrupt subsystem. Additionally, there is a per-priority trap handler to improve the support for arithmetic errors and illegal instructions.

3.1 Registers

The CPU contains six registers which are used in the execution of a sequential integer process. The six registers are:

- Workspace pointer (Wptr) which points to an area of store where local data is kept.
- Instruction pointer (Iptr) which points to the next instruction to be executed.
- Status register (Status).
- Areg, Breg and Creg registers which form an evaluation stack.

The Areg, Breg and Creg registers are the sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes Breg into Creg, and Areg into Breg, before loading Areg. Storing a value from Areg, pops Breg into Areg and Creg into Breg. Creg is left undefined.

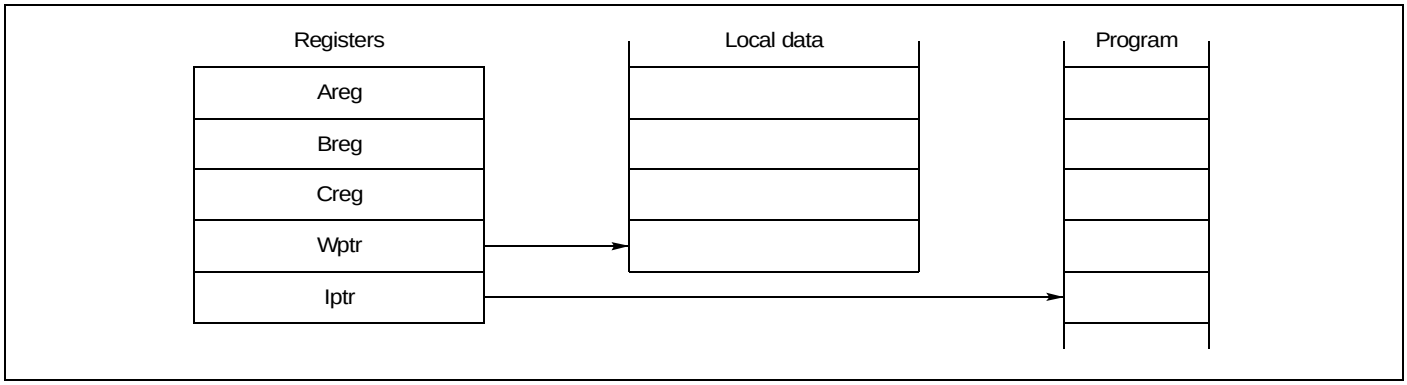


Figure 3 Registers used in sequential integer processes

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to explicitly specify the location of their operands. No hardware mechanism is

provided to detect that more than three values have been loaded onto the stack; it is easy for the compiler to ensure that this never happens.

Note that a location in memory can be accessed relative to the workspace pointer, enabling the workspace to be of any size.

The use of shadow registers provides fast, simple and clean context switching.

3.2 Processes and concurrency

This section describes the default behavior of the CPU and it should be noted that the user can alter this behavior, for example by disabling timeslicing or installing a user scheduler.

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. The CPU can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each.

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel, although kernels can still be written if desired.

At any time, a process may be

active
- being executed
- interrupted by a higher priority process
- on a list waiting to be executed

inactive
- waiting to input
- waiting to output
- waiting until a specified time

The scheduler operates in such a way that inactive processes do not consume any processor time. Each active high priority process executes until it becomes inactive. The scheduler allocates a portion of the processor's time to each active low priority process in turn (see section Section 3.3). Active processes waiting to be executed are held in two linked lists of process work spaces, one of high priority processes and one of low priority processes. Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the linked process list shown below, process *S* is executing and *P*, *Q* and *R* are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones behave in a similar manner.

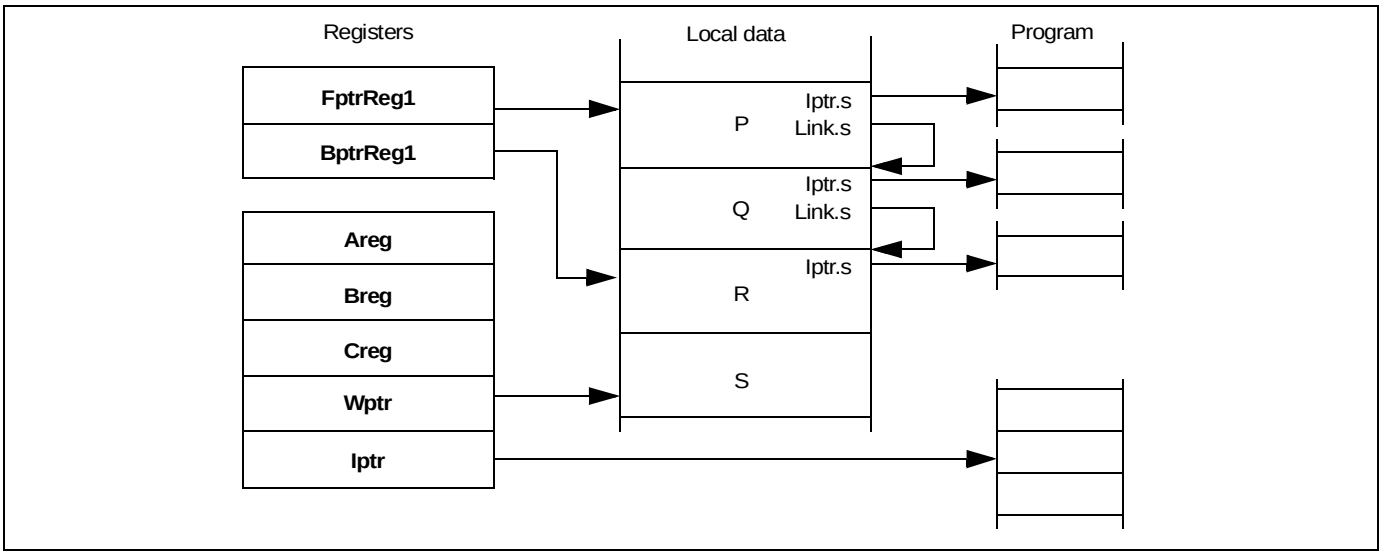


Figure 4 Linked process list

Function	High priority	Low priority
Pointer to front of active process list	FptrReg0	FptrReg1
Pointer to back of active process list	BptrReg0	BptrReg1

Table 3 Priority queue control registers

Each process runs until it has completed its action or is descheduled. In order for several processes to operate in parallel, a low priority process is only permitted to execute for a maximum of two timeslice periods. After this, the machine deschedules the current process at the next timeslicing point, adds it to the end of the low priority scheduling list and instead executes the next active process. The timeslice period is 1ms.

There are only certain instructions at which a process may be descheduled. These are known as descheduling points. A process may only be timesliced at certain descheduling points. These are known as timeslicing points and are defined in such a way that the operand stack is always empty. This removes the need for saving the operand stack when timeslicing. As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list.

The processor core provides a number of special instructions to support the process model, including *startp* (start process) and *endp* (end process). When a main process executes a parallel construct, *startp* is used to create the necessary additional concurrent processes. A *startp* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *endp* instruction. This uses a data structure that includes a counter of the parallel construct components which have still to terminate. The counter is initialized to the number of components before the processes are started. Each component ends with an *endp* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

3.3 Priority

The following section describes ‘default’ behavior of the CPU and it should be noted that the user can alter this behavior, for example, by disabling timeslicing and priority interrupts.

The processor can execute processes at one of two priority levels, one level for urgent (high priority) processes, one for less urgent (low priority) processes. A high priority process will always execute in preference to a low priority process if both are able to do so.

High priority processes are expected to execute for a short time. If one or more high priority processes are active, then the first on the queue is selected and executes until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is active, but one or more processes at low priority are active, then one is selected. Low priority processes are periodically timesliced to provide an even distribution of processor time between tasks which use a lot of computation.

If there are *n* low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is the order of 2*n* timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolizes the time of the CPU; i.e. it has frequent timeslicing points.

The specific condition for a high priority process to start execution is that the CPU is idle or running at low priority and the high priority queue is non-empty.

If a high priority process becomes able to run while a low priority process is executing, the low priority process is temporarily stopped and the high priority process is executed. The state of the low priority process is saved into ‘shadow’ registers and the high priority process is executed. When no further high priority processes are able to run, the state of the interrupted low priority process is re-loaded from the shadow registers and the interrupted low priority process continues executing. Instructions are provided on the processor core to allow a high priority process to store the shadow registers to memory and to load them from memory. Instructions are also provided to allow a process to exchange an alternative process queue for either priority process queue. These instructions allow extensions to be made to the scheduler for custom run-time kernels.

A low priority process may be interrupted after it has completed execution of any instruction. In addition, to minimize the time taken for an interrupting high priority process to start executing, the potentially time consuming instructions are interruptible. Also some instructions may be aborted, and are restarted when the process next becomes active (refer to Chapter 4: *Instruction set* on page 36).

3.4 Process communications

Communication between processes takes place over channels, and is implemented in hardware. Communication is point-to-point, synchronized and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same CPU is implemented by a single word in memory; a channel between processes executing on different processors is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being *in* (input message) and *out* (output message).

The *in* and *out* instructions use the address of the channel to determine whether the channel is internal or external. This means that the same instruction sequence can be used for both hard and soft channels, allowing a process to be written and compiled without knowledge of where its channels are implemented.

Communication takes place when both the inputting and outputting processes are ready. Consequently, the process which first becomes ready must wait until the second one is also ready. The inputting and outputting processes only become active when the communication has completed.

A process performs an input or output by loading the evaluation stack with, a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *in* or *out* instruction.

3.5 Timers

There are two 32-bit hardware timer clocks which ‘tick’ periodically. These are independent of any on-chip peripheral real time clock. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented approximately every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and runs 64 times slower, giving 15625 ticks per second. It has a full period of approximately 76 hours.

Actual timer speeds are derived from the processor speed CPU_PROCLK and are given in the *Clocks* chapter. The periods may be calculated as follows:

High_priority_clock_period = 1µs × *Nominal_speed* / CPU_PROCLK_*speed*

Low_priority_clock_period = High_priority_clock_period x 64

Register	Function
ClockReg0	Current value of high priority (level 0) process clock.
ClockReg1	Current value of low priority (level 1) process clock.
TnextReg0	Indicates time of earliest event on high priority (level 0) timer queue.
TnextReg1	Indicates time of earliest event on low priority (level 1) timer queue.
TptrReg0	High priority timer queue.
TptrReg1	Low priority timer queue.

Table 4 Timer registers

The current value of the processor clock can be read by executing a *ldtimer* (load timer) instruction. A process can arrange to perform a *tin* (timer input), in which case it will become ready to execute after a specified time has been reached. The *tin* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process becomes active. In addition, the *ldclock* (load clock), *stclock* (store clock) instructions allow total control over the clock value and the *clockenb* (clock enable), *clockdis* (clock disable) instructions allow each clock to be individually stopped and re-started.

Figure 5 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

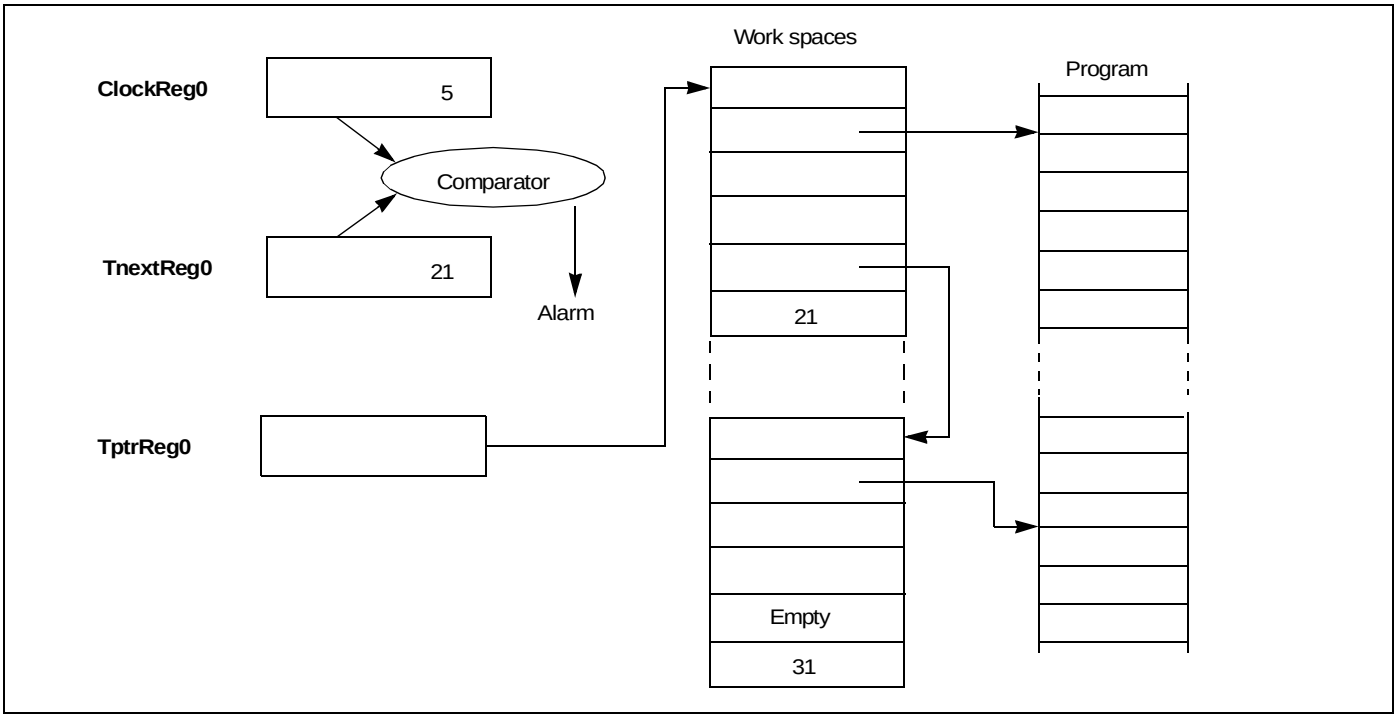


Figure 5 Timer registers

3.6 Traps and exceptions

A software error, such as arithmetic overflow or array bounds violation, can cause an error flag to be set in the CPU. The flag is directly connected to the **ErrorOut** pin. Both the flag and the pin can be ignored, or the CPU stopped. Stopping the CPU on an error means that the error cannot cause further corruption. As well as containing the error in this way it is possible to determine the state of the CPU and its memory at the time the error occurred. This is

particularly useful for postmortem debugging where the debugger can be used to examine the state and history of the processor leading up to and causing the error condition.

In addition, if a trap handler process is installed, a variety of traps/exceptions can be trapped and handled by software. A user supplied trap handler routine can be provided for each high/low process priority level. The handler is started when a trap occurs and is given the reason for the trap. The trap handler is not re-entrant and must not cause a trap itself within the same group. All traps can be individually masked.

3.6.1 Trap groups

The trap mechanism is arranged on a per priority basis. For each priority there is a handler for each group of traps, as shown in Figure 6 .

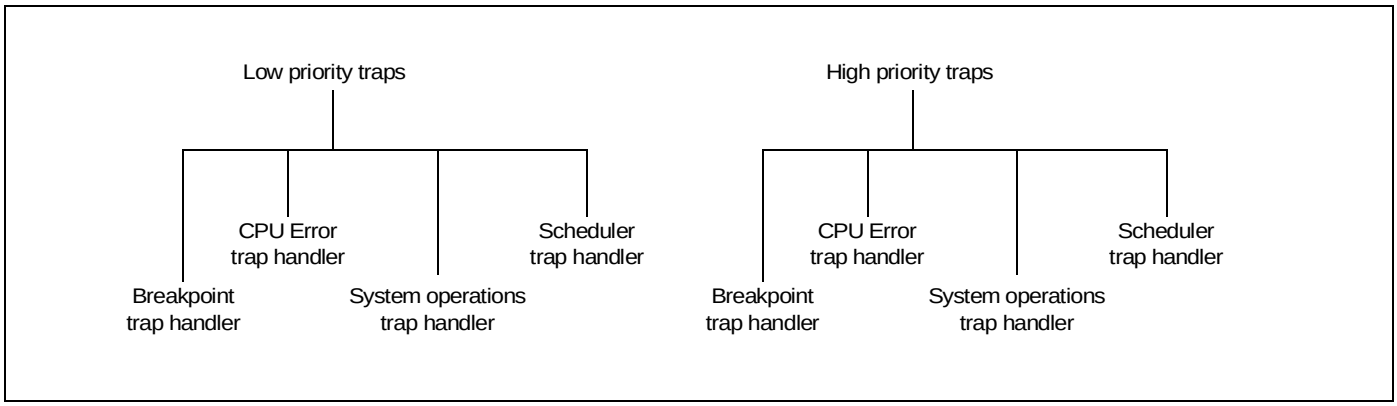


Figure 6 Trap arrangement

There are four groups of traps, as detailed below.

Breakpoint trap: The breakpoint instruction (*j0*) calls the breakpoint routine via the trap mechanism.

Errors: The traps in this group are *IntegerError* and *Overflow*. *Overflow* represents arithmetic overflow, such as arithmetic results which do not fit in the result word. *IntegerError* represents errors caused when data is erroneous, for example when a range checking instruction finds that data is out of range.

System operations: This group consists of the *LoadTrap*, *StoreTrap* and *IllegalOpcode* traps. The *IllegalOpcode* trap is signalled when an attempt is made to execute an illegal instruction. The *LoadTrap* and *StoreTrap* traps allow a kernel to intercept attempts by a monitored process to change or examine trap handlers or trapped process information. It enables a user program to signal to a kernel that it wishes to install a new trap handler.

Scheduler: The scheduler trap group consists of the *ExternalChannel, InternalChannel, Timer, TimeSlice, Run, Signal, ProcessInterrupt* and *QueueEmpty* traps. The *ProcessInterrupt* trap signals that the machine has performed a priority interrupt from low to high. The *QueueEmpty* trap indicates that there is no further executable work to perform. The other traps in this group indicate that the hardware scheduler wants to schedule a process on a process queue, with the different traps enabling the different sources of this to be monitored.

The scheduler traps enable a software scheduler kernel to use the hardware scheduler to implement a multi-priority software scheduler.

Note that scheduler traps are different from other traps as they are caused by the micro-scheduler rather than by an executing process.

Trap groups encoding is shown in below. These codes are used to identify trap groups to various instructions.

Trap group	Code
Breakpoint	0
CPU errors	1
System operations	2
Scheduler	3

Table 5 Trap group codes

In addition to the trap groups mentioned above, the **CauseError** flag in the **Status** register is used to signal when a trap condition has been activated by the *causeerror* instruction. It can be used to indicate when trap conditions have occurred due to the user setting them, rather than by the system.

3.6.2 Events that can cause traps

Table 6 summarizes the events that can cause traps and gives the encoding of bits in the trap **Status** and **Enable** words.

Trap cause	Status/Enable codes	Trap group	Comments
Breakpoint	0	0	When a process executes the breakpoint instruction (*j0*) then it traps to its trap handler.
IntegerError	1	1	Integer error other than integer overflow - e.g. explicitly checked or explicitly set error.
Overflow	2	1	Integer overflow or integer division by zero.
IllegalOpcode	3	2	Attempt to execute an illegal instruction. This is signalled when *opr* is executed with an invalid operand.
LoadTrap	4	2	When the trap descriptor is read with the *ldtraph* instruction or when the trapped process status is read with the *ldtrapped* instruction.
StoreTrap	5	2	When the trap descriptor is written with the *sttraph* instruction or when the trapped process status is written with the *sttrapped* instruction.
InternalChannel	6	3	Scheduler trap from internal channel.
ExternalChannel	7	3	Scheduler trap from external channel.
Timer	8	3	Scheduler trap from timer alarm.
Timeslice	9	3	Scheduler trap from timeslice.
Run	10	3	Scheduler trap from *runp* (run process) or *startp* (start process).
Signal	11	3	Scheduler trap from *signal*.
ProcessInterrupt	12	3	Start executing a process at a new priority level.
QueueEmpty	13	3	Caused by no process active at a priority level.
CauseError	15 (Status only)	Any, encoded 0-3	Signals that the *causeerror* instruction set the trap flag.

Table 6 Trap causes and status/enable codes

3.6.3 Trap handlers

For each trap handler there is a trap handler structure and a trapped process structure. Both the trap handler structure and the trapped process structure are in memory and can be accessed via instructions, see section Section .

The trap handler structure specifies what should happen when a trap condition is present, see .

The trapped process structure saves some of the state of the process that was running when the trap was taken.

In addition, for each priority, there is an **Enables** register and a **Status** register. The **Enables** register contains flags to enable each cause of trap. The **Status** register contains flags to indicate which trap conditions have been detected. The **Enables** and **Status** register bit encodings are given in Table 6 .

	Comments	Location
IPTR	**Iptr** of trap handler process.	Base + 3
WPTR	**Wptr** of trap handler process. A null **Wptr** indicates that a trap handler has not been installed.	Base + 2
Status	Contains the **Status** register that the trap handler starts with.	Base + 1
Enables	A word which encodes the trap enable and global interrupt masks, which will be ANDed with the existing masks to allow the trap handler to disable various events while it runs.	Base + 0

Table 7 Trap handler structure

	Comments	Location
Iptr	Points to the instruction after the one that caused the trap condition.	Base + 3
Wptr	**Wptr** of the process that was running when the trap was taken.	Base + 2
Status	The relevant trap bit is set, see for trap codes.	Base + 1
Enables	Interrupt enables.	Base + 0

Table 8 Trapped process structure

A trap will be taken at an interruptible point if a trap is set and the corresponding trap enable bit is set in the **Enables** register. If the trap is not enabled then nothing is done with the trap condition. If the trap is enabled then the corresponding bit is set in the **Status** register to indicate the trap condition has occurred.

When a process takes a trap the processor saves the existing **Iptr**, **Wptr**, **Status** and **Enables** in the trapped process structure. It then loads **Iptr**, **Wptr** and **Status** from the equivalent trap handler structure and ANDs the value in **Enables** with the value in the structure. This allows the user to disable various events while in the handler, in particular a trap handler must disable all the traps of its trap group to avoid the possibility of a handler trapping to itself.

The trap handler then executes. The values in the trapped process structure can be examined using the *ldtrapped* instruction (see section Section). When the trap handler has completed its operation it returns to the trapped process via the *tret* (trap return) instruction. This reloads the values saved in the trapped process structure and clears the trap flag in **Status**.

Note that when a trap handler is started, **Areg**, **Breg** and **Creg** are not saved. The trap handler must save the **Areg**, **Breg**, **Creg** registers using *stl* (store local).

Trap instructions

Trap handlers and trapped processes can be set up and examined via the *ldtraph*, *sttraph*, *ldtrapped* and *sttrapped* instructions. Table 9 describes the instructions that may be used when dealing with traps.

Instruction	Meaning	Use
ldtraph	load trap handler	Load the trap handler from memory to the trap handler descriptor.
sttraph	store trap handler	Store an existing trap handler descriptor to memory.
ldtrapped	load trapped	Load replacement trapped process status from memory.
sttrapped	store trapped	Store trapped process status to memory.
trapenb	trap enable	Enable traps.
trapdis	trap disable	Disable traps.
tret	trap return	Used to return from a trap handler.
causeerror	cause error	Program can simulate the occurrence of an error.

Table 9 Instructions which may be used when dealing with traps

The first four instructions transfer data to/from the trap handler structures or trapped process structures from/to an area in memory. In these instructions **Areg** contains the trap group code and **Breg** points to the 4 word area of memory used as the source or destination of the transfer. In addition **Creg** contains the priority of the handler to be installed/examined in the case of *ldtraph* or *sttraph. ldtrapped* and *sttrapped* apply only to the current priority.

If the *LoadTrap* trap is enabled then *ldtraph* and *ldtrapped* do not perform the transfer but set the **LoadTrap** trap flag. If the *StoreTrap* trap is enabled then *sttraph* and *sttrapped* do not perform the transfer but set the **StoreTrap** trap flag.

The trap enable masks are encoded by an array of bits (see Table 6) which are set to indicate which traps are enabled. This array of bits is stored in the lower half-word of the **Enables** register. There is an **Enables** register for each priority. Traps are enabled or disabled by loading a mask into **Areg** with bits set to indicate which traps are to be affected and the priority to affect in **Breg**. Executing *trapenb* ORs the mask supplied in **Areg** with the trap enables mask in the **Enables** register for the priority in **Breg**. Executing *trapdis* negates the mask supplied in **Areg** and ANDs it with the trap enables mask in the **Enables** register for the priority in **Breg**. Both instructions return the previous value of the trap enables mask in **Areg**.

3.6.4 Restrictions on trap handlers

There are various restrictions that must be placed on trap handlers to ensure that they work correctly.

- Trap handlers must not deschedule or timeslice. Trap handlers alter the **Enables** masks, therefore they must not allow other processes to execute until they have completed.
- Trap handlers must have their **Enable** masks set to mask all traps in their trap group to avoid the possibility of a trap handler trapping to itself.
- Trap handlers must terminate via the *tret* (trap return) instruction. The only exception to this is that a scheduler kernel may use *restart* to return to a previously shadowed process.

4 Instruction set

This chapter provides information on the ST20-C2+ instruction set. It contains tables listing all the instructions, and where applicable provides details of the number of processor cycles taken by an instruction.

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits (MSB) of the byte are a function code and the four least significant bits (LSB) are a data value, as shown below.

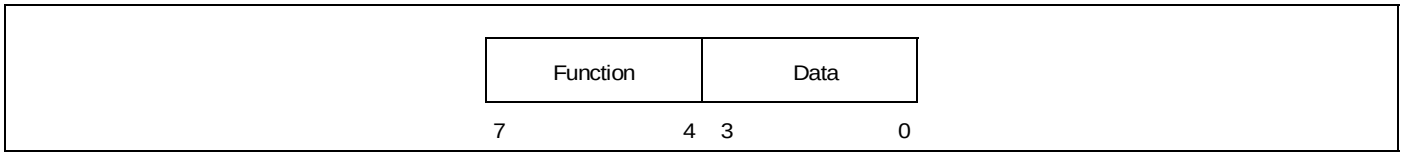


Figure 7 Instruction format

For further information on the instruction set refer to the *ST20C2/C4 Instruction Set Manual* (document number 72-TRN-273).

4.1 Instruction cycles

Timing information is available for some instructions. However, it should be noted that many instructions have ranges of timings which are data dependent.

Where included, timing information is based on the number of clock cycles assuming any memory accesses are to 2 cycle internal memory and no other subsystem is using memory. Actual time will be dependent on the speed of external memory and memory bus availability.

Note that the actual time can be increased by:

- The instruction requiring a value on the register stack from the final memory read in the previous instruction – the current instruction will stall until the value becomes available.
- The first memory operation in the current instruction can be delayed while a preceding memory operation completes - any two memory operations can be in progress at any time, any further operation will stall until the first completes.
- Memory operations in current instructions can be delayed by access by instruction fetch or subsystems to the memory interface.
- There can be a delay between instructions while the instruction fetch unit fetches and partially decodes the next instruction – this will be the case whenever an instruction causes the instruction flow to jump.

Note that the instruction timings given refer to ‘standard’ behavior and may be different if, for example, traps are set by the instruction.

4.2 Instruction characteristics

Table 12 on page 38 gives the basic function code of each of the primary instructions. Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*. Examples of *pfix* and *nfix* coding are given in Table 10 .

Mnemonic		**Function code**	**Memory code**
ldc	#3	#4	#43
ldc	#35		
is coded as			
pfix	#3	#2	#23
ldc	#5	#4	#45
ldc	#987		
is coded as			
pfix	#9	#2	#29
pfix	#8	#2	#28
ldc	#7	#4	#47
ldc	-31 (*ldc* #FFFFFFE1)		
is coded as			
nfix	#1	#6	#61
ldc	#1	#4	#41

Table 10 Prefix coding

Any instruction which is not in the instruction set tables is an invalid instruction and is flagged illegal, returning an error code to the trap handler, if loaded and enabled.

The **Notes** column of the tables indicates the features of an instruction as described in Table 11 .

Ident	**Feature**
E	Instruction can set an *IntegerError* trap
L	Instruction can cause a *LoadTrap* trap
S	Instruction can cause a *StoreTrap* trap
O	Instruction can cause an *Overflow* trap
I	Interruptible instruction
A	Instruction can be aborted and later restarted.
D	Instruction can deschedule
T	Instruction can timeslice

Table 11 Instruction features

4.3 Instruction-set tables

Function code	Memory code	Mnemonic	Processor cycles	Name	Notes
0	0X	j	5	jump	D, T
1	1X	ldlp	1	load local pointer	
2	2X	pfix	0 to 1	prefix	
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	ldnlp	1	load non-local pointer	
6	6X	nfix	0 to 1	negative prefix	
7	7X	ldl	1	load local	
8	8X	adc	1	add constant	O
9	9X	call	8	call	
A	AX	cj	1 or 5	conditional jump	
B	BX	ajw	2	adjust workspace	
C	CX	eqc	1	equals constant	
D	DX	stl	1	store local	
E	EX	stnl	2	store non-local	
F	FX	opr	0	operate	

Table 12 Primary functions

Memory code	Mnemonic	Processor cycles	Name	Notes
22FA	testpranal	2	test processor analyzing	
23FE	saveh	3	save high priority queue registers	
23FD	savel	3	save low priority queue registers	
21F8	sthf	1	store high priority front pointer	
25F0	sthb	1	store high priority back pointer	
21FC	stlf	1	store low priority front pointer	
21F7	stlb	1	store low priority back pointer	
25F4	sttimer	2	store timer	
2127FC	lddevid	1	load device identity	
27FE	ldmemstartval	1	load value of **MemStart** address	

Table 13 Processor initialization operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
24F6	and	1	and	
24FB	or	1	or	
23F3	xor	1	exclusive or	
23F2	not	1	bitwise not	
24F1	shl	1	shift left	
24F0	shr	1	shift right	
F5	add	1	add	A, O
FC	sub	1	subtract	A, O
25F3	mul	4	multiply	A, O
27F2	fmul	6	fractional multiply	A, O
22FC	div	5 to 37	divide	A, O
21FF	rem	5 to 40	remainder	A, O
F9	gt	1	greater than	A
25FF	gtu	1	greater than unsigned	A
F4	diff	1	difference	
25F2	sum	1	sum	
F8	prod	4	product	A
26F8	satadd	2	saturating add	A
26F9	satsub	2	saturating subtract	A
26FA	satmul	5	saturating multiply	A

Table 14 Arithmetic/logical operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
21F6	ladd	2	long add	A, O
23F8	lsub	2	long subtract	A, O
23F7	lsum	2	long sum	
24FF	ldiff	2	long diff	
23F1	lmul	5 to 6	long multiply	A
21FA	ldiv	5 to 39	long divide	A, O
23F6	lshl	2	long shift left	A
23F5	lshr	2	long shift right	A
21F9	norm	2 to 5	normalize	A
26F4	slmul	5	signed long multiply	A, O
26F5	sulmul	5	signed times unsigned long multiply	A, O

Table 15 Long arithmetic operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
F0	rev	1	reverse	
23FA	xword	4	extend to word	A
25F6	cword	3	check word	A, E
21FD	xdble	2	extend to double	
24FC	csngl	3	check single	A, E
24F2	mint	1	minimum integer	
25FA	dup	1	duplicate top of stack	
27F9	pop	1	pop processor stack	
68FD	reboot	1	reboot	

Table 16 General operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
F2	bsub	1	byte subscript	
FA	wsub	1	word subscript	
28F1	wsubdb	1	form double word subscript	
23F4	bcnt	1	byte count	
23FF	wcnt	1	word count	
F1	lb	1	load byte	
23FB	sb	2	store byte	
24FA	move		move message	I

Table 17 Indexing/array operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
22F2	ldtimer	1	load timer	
22FB	tin		timer input	I
24FE	talt	3	timer alt start	
25F1	taltwt		timer alt wait	D, I
24F7	enbt	2 to 8	enable timer	
22FE	dist		disable timer	I

Table 18 Timer handling operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
F7	in		input message	D
FB	out		output message	D
FF	outword		output word	D
FE	outbyte		output byte	D
24F3	alt	2	alt start	
24F4	altwt	4 to 7	alt wait	D
24F5	altend	9	alt end	
24F9	enbs	1 to 2	enable skip	
23F0	diss	1	disable skip	
21F2	resetch	3	reset channel	
24F8	enbc	2 to 5	enable channel	
22FF	disc	2 to 7	disable channel	

Table 19 Input and output operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
22F0	ret	3	return	
21FB	ldpi	1	load pointer to instruction	
23FC	gajw	3	general adjust workspace	
F6	gcall	6	general call	
22F1	lend	5 to 8	loop end	T

Table 20 Control operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
FD	startp	5	start process	
F3	endp	4 to 6	end process	D
23F9	runp	3	run process	
21F5	stopp	2	stop process	
21FE	ldpri	1	load current priority	

Table 21 Scheduling operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
21F3	csub0	2	check subscript from 0	A, E
24FD	ccnt1	3	check count from 1	A, E
22F9	testerr	2	test error false and clear	
21F0	seterr	2	set error	
25F5	stoperr	2 to 3	stop on error (no error)	D
25F7	clrhalterr	1	clear halt-on-error	
25F8	sethalterr	1	set halt-on-error	
25F9	testhalterr	2	test halt-on-error	

Table 22 Error handling operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
25FB	move2dinit	3	initialize data for 2D block move	
25FC	move2dall		2D block copy	I
25FD	move2dnonzero		2D block copy non-zero bytes	I
25FE	move2dzero		2D block copy zero bytes	I

Table 23 2D block move operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
27F4	crcword	36	calculate crc on word	A
27F5	crcbyte	12	calculate crc on byte	A
27F6	bitcnt	3	count bits set in word	A
27F7	bitrevword	2	reverse bits in word	
27F8	bitrevnbits	2	reverse bottom n bits in word	A

Table 24 CRC and bit operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
27F3	cflerr	3	check floating point error	E
29FC	fptesterr	1	load value true (FPU not present)	
26F3	unpacksn	10	unpack single length floating point number	A
26FD	roundsn	7	round single length floating point number	A
26FC	postnormsn	9	post-normalize correction of single length floating point number	A
27F1	ldinf	1	load single length infinity	

Table 25 Floating point support operation codes

Memory code	Mnemonic	Processor cycles	Name	Notes
2CF7	cir	3	check in range	A, E
2CFC	ciru	3	check in range unsigned	A, E
2BFA	cb	3	check byte	A, E
2BFB	cbu	2	check byte unsigned	A, E
2FFA	cs	3	check sixteen	A, E
2FFB	csu	2	check sixteen unsigned	A, E
2FF8	xsword	3	sign extend sixteen to word	A
2BF8	xbword	3	sign extend byte to word	A

Table 26 Range checking and conversion instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
2CF1	ssub	1	sixteen subscript	
2CFA	ls	1	load sixteen	
2CF8	ss	2	store sixteen	
2BF9	lbx	1	load byte and sign extend	
2FF9	lsx	1	load sixteen and sign extend	

Table 27 ndexing/array instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
2FF0	devlb	3	device load byte	A
2FF2	devls	3	device load sixteen	A
2FF4	devlw	3	device load word	A
62F4	devmove		device move	I
2FF1	devsb	3	device store byte	A
2FF3	devss	3	device store sixteen	A
2FF5	devsw	3	device store word	A

Table 28 Device access instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
60F5	wait	5 to 11	wait	D
60F4	signal	7 to 12	signal	

Table 29 Semaphore instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
60F0	swapqueue	4	swap scheduler queue	
60F1	swaptimer	5	swap timer queue	
60F2	insertqueue	3 to 4	insert at front of scheduler queue	
60F3	timeslice	3 to 4	timeslice	
60FC	ldshadow	6 to 31	load shadow registers	A
60FD	stshadow	6 to 17	store shadow registers	A
62FE	restart	20	restart	
62FF	causeerror	7 to 8	cause error	
61FF	iret	3 to 11	interrupt return	
2BF0	settimeslice	2	set timeslicing status	
2CF4	intdis	2	interrupt disable	
2CF5	intenb	2	interrupt enable	
2CFD	gintdis	5	global interrupt disable	
2CFE	gintenb	5	global interrupt enable	

Table 30 Scheduling support instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
26FE	ldtraph	12	load trap handler	L
2CF6	ldtrapped	12	load trapped process status	L
2CFB	sttrapped	12	store trapped process status	S
26FF	sttraph	12	store trap handler	S
60F7	trapenb	4	trap enable	
60F6	trapdis	4	trap disable	
60FB	tret	8 to 10	trap return	

Table 31 Trap handler instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
68FC	ldprodid	1	load product identity	
63F0	nop	1	no operation	

Table 32 Processor initialization and no operation instructions

Memory code	Mnemonic	Processor cycles	Name	Notes
64FF	clockenb	2	clock enable	
64FE	clockdis	2	clock disable	
64FD	ldclock	2	load clock	
64FC	stclock	2	store clock	

Table 33 Clock instructions

5 Interrupt system

5.1 Introduction

The interrupt system allows an on-chip module or external interrupt pin to interrupt an active process so that an interrupt handling process can be run. Interrupts are signalled by one of the following:

- a signal on an external interrupt pin;
- a signal from an internal peripheral or subsystem;
- software asserting an interrupt in the pending register.

Interrupts are implemented by an on-chip **interrupt controller** and an on-chip **interrupt level controller**. The interrupt level controller multiplexes the 31 incoming interrupt sources onto the eight programmable interrupt level inputs of the interrupt controller. This multiplexing is controlled by software. This is illustrated in the figure below.

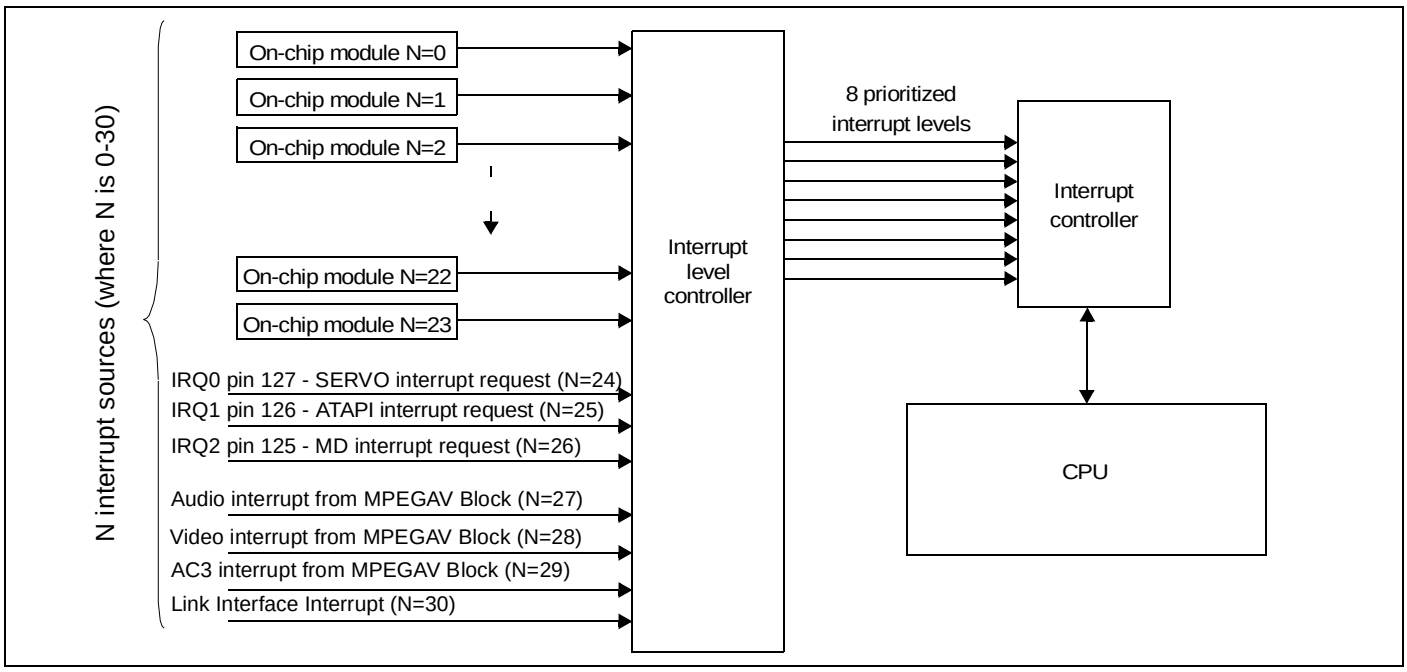


Figure 8 STi5518 Interrupt system

5.2 Interrupt controller

The interrupt controller supports eight prioritized interrupts as inputs, and manages the pending interrupts. This allows nested pre-emptive interrupts for real-time system design. Interrupt level 7 has the highest priority and interrupt level 0 has the lowest priority.

All interrupts are at a higher priority than the low-priority process queue. Each interrupt can be programmed to be at a lower or higher priority than the high-priority process queue by writing to the priority bit in the INC_HandlerWptr registers. Interrupts which are specified as higher priority must be contiguous from the highest numbered interrupt downwards. For example, if 4 interrupts are programmed as high-priority and 4 as low-priority, then the higher priority interrupts must be set to Interrupt7:4 and the lower priority interrupts to Interrupt3:0.

Each of the eight interrupt levels of the interrupt controller can be programmed with a interrupt trigger mode, using the INC_TriggerMode register. The trigger mode can be set to be high or low level, or rising edge, falling edge or any edge sensitive. Note that all on-chip module interrupt sources produce active-high level interrupt signals. Therefore the

interrupt level that these interrupt sources are multiplexed onto (by the interrupt level controller) must be programmed with a high-level trigger mode.

Furthermore, each of the eight interrupt levels can be programmed to be enabled or disabled by the INC_MASK register. The default state of INC_MASK is that all interrupt levels are disabled. A corresponding level bit is set in the INC_PENDING register if the interrupt signal from the interrupt level controller matches the level trigger condition. If this is the highest priority bit set in the INC_PENDING register, the CPU will then execute the interrupt handler associated with that level by the INC_HANDLERWPTR register and the INC_PENDING bit will then be reset. If the level bit set in the INC_PENDING register is not the highest priority bit set, then the bit remains set until it is the highest priority level bit, then the CPU executes the associated interrupt handler for that level. Note the CPU will only execute the interrupt handler and then clear the INC_PENDING register bit if it is enabled in the INC_MASK register. Software can write to the INC_PENDING register to generate a software interrupt on any of the eight interrupt-levels.

Programming of the INC_MASK, INC_PENDING and INC_TRIGGERMODE registers is supported via the operating system run time library functions of STLite (a.k.a OS20).

The interrupt controller also contains an INC_EXEC register used by the interrupt controller logic to keep a record of which interrupt-level handler is currently executing on the CPU (or was previously executing before being pre-empted by a high priority process, for low priority interrupts) and which levels have been pre-empted by higher priority interrupt levels. This register can be read by user software, if required, but the register must never be written to as its behavior is undefined.

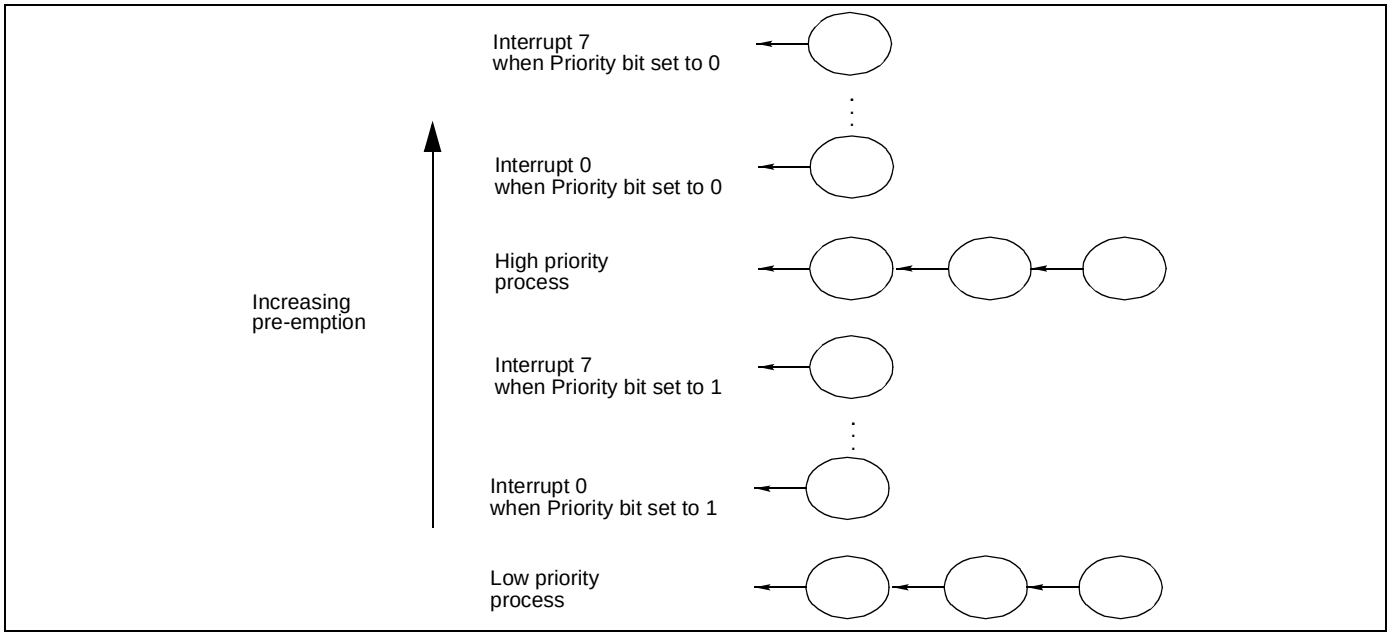


Figure 9 Interrupt priority

5.3 Interrupt vector table

The interrupt controller contains a table of pointers to interrupt handlers. There are 8 interrupt handlers, each controlled by a work-space register INC_HandlerWptr 0-7. The table of pointer values contains a work-space pointer for each interrupt level.

The INC_HandlerWptr registers access the code, data and interrupt-save area of the interrupt handler. The position of the INC_HandlerWptr register in the interrupt table sets the priority of the interrupt.

The operating system run time library (STLite a.k.a. OS20) supports the setting and programming of the vector table.

5.4 Interrupt handlers

At any interruptible point in its execution, the CPU can receive an interrupt request from the interrupt controller. The CPU immediately acknowledges the request.

In response to receiving an interrupt, the CPU performs a procedure call to the process in the vector table. The state of the interrupted process is stored in the work space of the interrupt handler as shown in Figure 10 . Each interrupt level has its own work space.

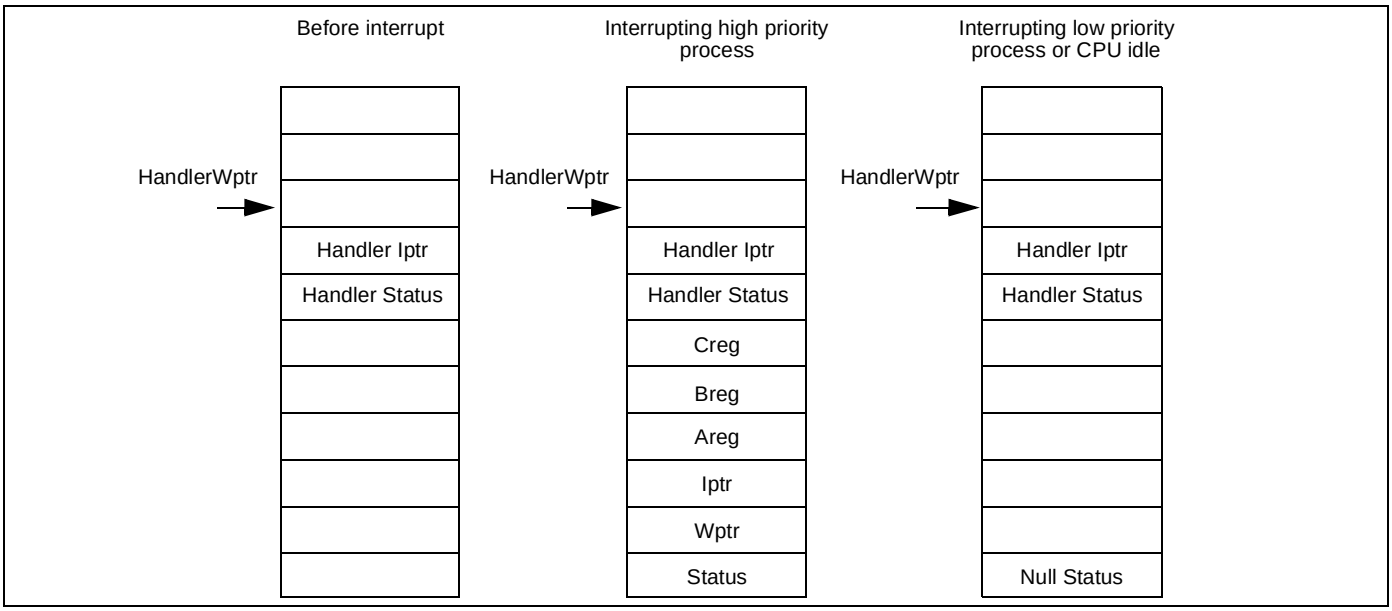


Figure 10 State of interrupted process

The interrupt routine is initialized with space below **HandlerWptr**. The **Iptr** and **Status** word for the routine are stored there permanently. This should be programmed before the **HandlerWptr** is written into the vector table.

The behavior of the interrupt differs depending on the priority of the CPU when the interrupt occurs. If an interrupt occurs when the CPU is running at high priority, and the interrupt is set at a higher priority than the high priority process queue, the CPU saves the current process state (Areg, Breg, Creg, Wptr, Iptr and Status) into the workspace of the interrupt handler. The value **HandlerWptr**, which is stored in the interrupt controller, points to the top of this work space. The values of **Iptr** and **Status** to be used by the interrupt handler are loaded from this work space and starts executing the handler. The value of **Wptr** is then set to the bottom of this save area.

If an interrupt occurs when the CPU is running at high priority, and the interrupt is set at a lower priority than the high priority process queue, no action is taken and the interrupt waits in a queue until the high priority process queue is empty (see Pre-emption and interrupt priority on page 48).

Interrupts always take priority over low priority processes. If an interrupt occurs when the CPU is idle or running at low priority, the **Status** is saved. This indicates that no valid process is running (*Null Status*). The interrupted processes (low priority process) state is stored in shadow registers. This state can be accessed via the *ldshadow* (load shadow registers) and *stshadow* (store shadow registers) instructions. The interrupt handler is then run at high priority.

When the interrupt routine has completed it must adjust **Wptr** to the value at the start of the handler code and then execute the *iret* (interrupt return) instruction. This restores the interrupted state from the interrupt handler structure and signals to the interrupt controller that the interrupt has completed. The processor will then continue from where it was before being interrupted.

5.5 Interrupt latency

The interrupt latency depends on the type of data being accessed, and the position in memory of the interrupt handler and the interrupted process. This allows a trade-off of between fast internal SRAM memory and interrupt latency.

5.6 Pre-emption and interrupt priority

Each interrupt channel has an implied priority fixed by its place in the interrupt vector table. All interrupts cause scheduled processes of any priority to be suspended and the interrupt handler started. Once an interrupt has been sent from the controller to the CPU the controller keeps a record of the current executing interrupt priority in the INC_EXEC register. This is only cleared when the interrupt handler executes a return from interrupt (*iret*) instruction. Interrupts of a lower priority arriving are blocked by the interrupt controller until the interrupt priority is low enough for the routine to execute. An interrupt of a higher priority than the currently executing handler is passed to the CPU and causes the current handler to be suspended until the higher priority interrupt is serviced. In this way, interrupts can be nested and a higher priority interrupt always pre-empts a lower priority one.

Note: deep nesting and the placing of frequent interrupts at high priority can result in systems where low priority interrupts are never serviced or CPU time is consumed in nesting interrupt priorities instead of executing the interrupt handlers.

5.7 Restrictions on interrupt handlers

For optimum interrupt handling, the following restrictions are placed on interrupt handlers:

- Interrupt handlers must not deschedule.
- Interrupt handlers must not execute communication instructions. However they may communicate with other processes through shared variables using the semaphore *signal* to synchronize.
- Interrupt handlers must not perform CPU 2D block move instructions.
- Interrupt handlers must not cause program traps. However they may be trapped by a scheduler trap.

5.8 Interrupt level controller

The interrupt level controller multiplexes 31 incoming interrupt source signals onto the eight interrupt level inputs of the interrupt controller. In this way, it gives programmable control of the priority of the interrupt sources and extends the number of possible interrupts to 31.

The incoming interrupt signals can be generated by on-chip subsystems or received from external pins. Table 34 on page 50 assigns each of the interrupt sources to a number N from 0-30. Software assigns a signal ***n*** to one of the 8 interrupt levels by writing the priority of the required input in the register INC_IntnPriority. Each of the 31 interrupt sources in the **interrupt level controller** can be selectively enabled or disabled at source, by writing to the INC_SRC_MASK register. This is in addition to the individual masking of the 8 levels in the interrupt controller. This means that the user can disable just the interrupt source from generating an interrupt, without disabling all other interrupt sources mapped onto that interrupt level. This would be the case if the INC_MASK register in the interrupt controller was used. Each interrupt source can be used to trigger an interrupt and can be programmed to trigger on rising or falling edges, or on the high or low logic level of the incoming interrupt source signal. This is controlled by writing to the INC_SRC_TRIGGERMODE registers.

The STi5518 enhanced feature **interrupt level controller** is software backward compatible with the **interrupt level controller** in the STi5500, STi5505 and STi5508. Backward compatibility can be maintained by setting the interrupt trigger on the interrupt levels in the **interrupt controller,** as was done before. The default state of the **interrupt level controller** trigger mode registers is high, therefore, these new registers do not need to be programmed. In this case the **interrupt level controller** will effectively pass the interrupt source signal through, unmodified, to the **interrupt controller**. The default state of all of the INC_SRC_MASK register bits is 1, meaning that all of the interrupt sources are enabled. This is again to maintain software backwards compatibility.

If the non-default trigger modes in the **interrupt level controller** are to be used, the corresponding trigger mode for the interrupt level in the **interrupt controller** that the source(s) are mapped to, must be programmed to high level. Otherwise, the trigger mode in the **interrupt controller** and the **interrupt level controller** may conflict. The INC_InputInterrupt register has the same function as before in STi5500, STi5505, STi5508 and can be used to indicate the current logic state of the all the interrupt sources. Note that this register is just a buffered version of the interrupt source signals before the trigger mode detection stage and does not latch the signal, as does the INC_SRC_STATUS register, for interrupt sources defined with an edge sensitive trigger mode. The INC_SRC_STATUS register is more useful, because of this feature, as it can be read by the interrupt handler software routine to determine which interrupt sources have triggered.

For example, if the interrupt source is external and provides a pulse, the interrupt level controller would have the interrupt source trigger mode set to be rising edge. On a rising edge the corresponding bit in the INC_SRC_STATUS register would be set high and would remain set until explicitly cleared by the interrupt handler routine writing to the corresponding bit in the INC_SRC_CLEAR register. However if the pulse was short, by the time the interrupt handler was executed and it read the INC_InputInterrupt register, the pulse may have returned to a logic low and the bit would be read as zero. Thus the cause of interrupt could not be determined if more than one interrupt source had been multiplexed onto the interrupt level.

So now, using the INC_SRC_STATUS register, it is possible to multiplex interrupt sources of different types, including edge sensitive, onto the same interrupt level in the interrupt controller.

The STi5518 interrupt level controller also has two new registers mapped into its register address space, that have no connection with normal interrupt operation. These registers are for controlling waking up the CPU by an external interrupt pin, when it has been put into low power mode, by the low power controller module. The register INC_SELNOTINV controls whether the three external interrupt pins are active high or low, to wake up the CPU from low power mode. Note that the setting of this register has no effect on the triggering of the external interrupt pins in the interrupt level controller. The register INC_EN_INT is a mask register to enable or disable the external interrupt pins from waking up the CPU from low power mode. Again, this has no effect on the masking of these interrupts in the interrupt level controller.

5.9 Interrupt assignments

All interrupts are active high. Interrupts from the internal peripherals and external pins are assigned as in the table below.

INT N	Peripheral	Description of the functions
0	PIO 0	Compare function
1	PIO 1	Compare function
2	PIO 2	Compare function
3	PIO 3	Compare function
4	PIO 4	Compare function
5	SSC0	SSC0TIR, SSC0RIR, SSC0EIR I^2C MASTER
6	SSC1	SSC1TIR, SSC1RIR, SSC1EIR I^2C MASTER
7	UART 3	ASC3TIR, ASC3TBIR, ASC3RIR, ASC3EIR
8	UART 2	ASC2TIR, ASC2TBIR, ASC2RIR, ASC2EIR
9	UART 1	ASC1TIR, ASC1TBIR, ASC1RIR, ASC1EIR
10	UART 0	ASC0TIR, ASC0TBIR, ASC0RIR, ASC0EIR
11	PWM and Capture	PWMFunctions, (Capture0Int, Capture1Int TBD)
12	MPEG3DMA 0	MPEG3DMA0 Interrupt
13	MPEG3DMA 1	MPEG3DMA1 Interrupt
14	MPEG3DMA 2	MPEG3DMA 2 inside the Link Interface interrupt
15	BLOCK MOVE	Blockmove Interrupt
16	MODEM DMA	MAFE modem Interface Interrupt
17	PIO5	Compare Function
18	IR Blaster	Tx, Rx interrupts
19	TeleText	Ttxt DMA interrupt
20-23	Reserved	Tied low internally
24	IRQ0 pin 127	SERVO interrupt request
25	IRQ1 pin 126	ATAPI interrupt request
26	IRQ2 pin 125	MD interrupt request
27	MPEGAV block	Audio interrupt from MPEGAV block
28	MPEGAV block	Video interrupt from MPEGAV block
29	MPEGAV block	Sector processor interrupt or HDD link interrupt
30	Link interface interrupt	Link interface interrupt

Table 34 STi5518 Interrupt assignments

6 Memory map

6.1 Overview

The STi5518 has a 32-bit signed 2s-complement address space. A byte of memory is addressed by a 30-bit word address plus a 2-bit byte-selector identifier in the word. A word of memory is addressed by a 30-bit word address with the byte-selector set to zero.

Memory is divided into areas with different purposes. Some areas are dedicated to a specific purpose, either because they contain memory-mapped devices or because they are reserved by the system. The figure below shows the memory map.

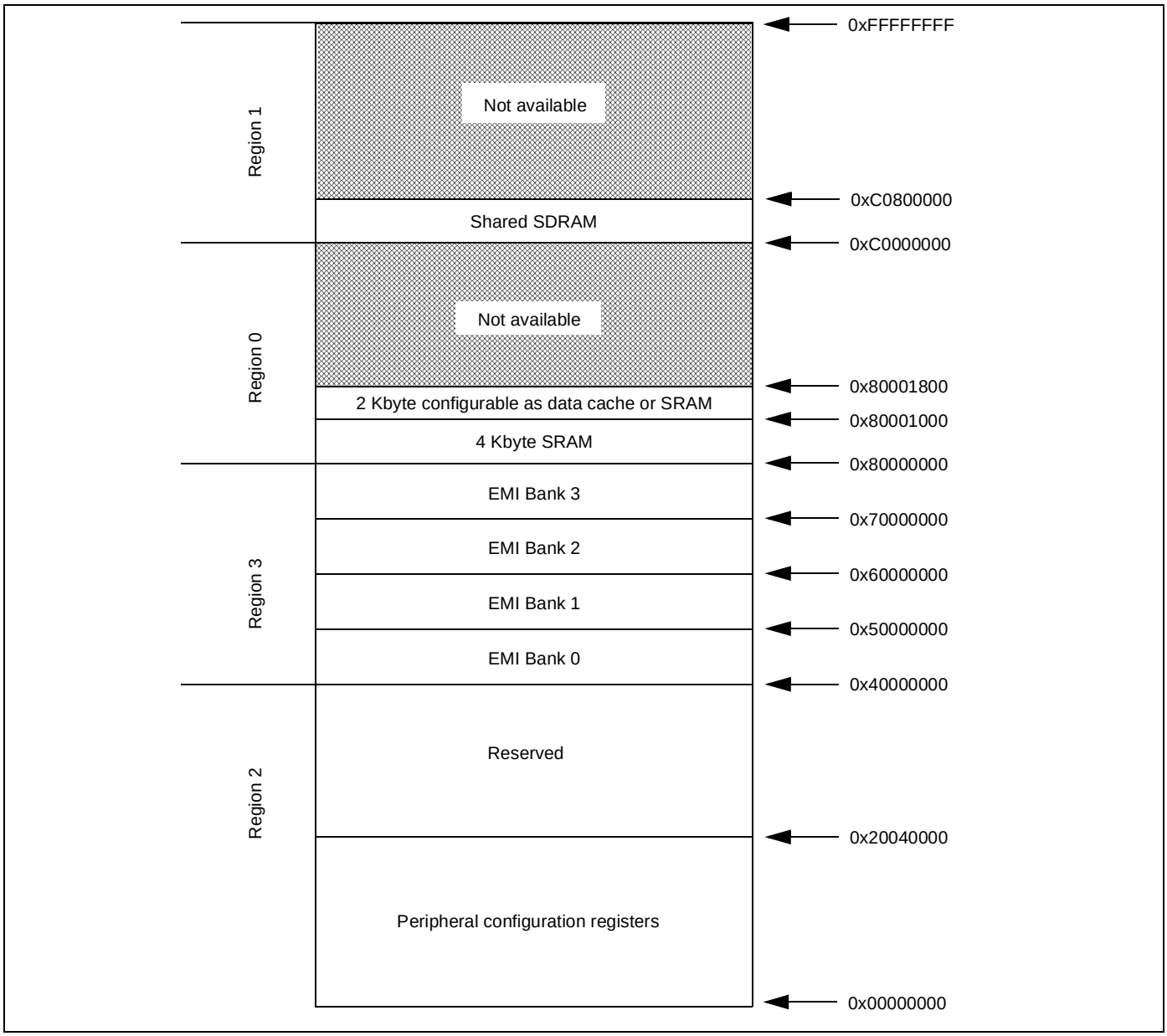


Figure 11 Memory map

Memory is normally accessed by the **load**, **store**, **block move** and **channel** instructions. These will use data cache if it is enabled, and do not guarantee the order of accesses to different addresses.

6.2 Mapping

The address space is divided into the following regions:

- Region 0: DCache or SRAM, the bottom 4 Kbytes (or 6 Kbytes if the data cache is not used) is occupied by on-chip SRAM.
- Region 1: Shared SDRAM, the 8 Mbyte area from 0xC0000000 to 0xC07FFFFF is for SDRAM and is shared with the MPEG decoders;
- Region 2: Peripheral configuration registers, the area from 0x00000000 to 0x3FFFFFFF is dedicated to memory-mapped or command-mapped on-chip peripherals;
- Region 3: EMI banks0 to 3, 0x40000000 to 0x7FFFFFFF is for external memory and peripherals, accessed through the EMI

Designation	Start	End	Description
MPEG	#00000000	#000001FF	MPEG Video
	#00000200	#000002FF	MPEG Audio
	#00000400	#000005FF	Sub-Picture decoder
	#00000600	#000006FF	DENC decoder
	#00000700	#000007FF	Reserved
	#00000800	#000009FF	MPEG Video fifos accesses
	#00000A00	#00000BFF	MPEG Audio fifos accesses
	#00000C00	#00000DFF	Sub-Picture decoder accesses
	#00000E00	#00000FFF	MPEG control registers
Configuration	#00002000	#00002FFF	EMI configuration
	#00003000	#00003FFF	DCU
	#00004000	#00004FFF	Cache configuration
	#00005000	#1FFFFFFF	Reserved

Table 35 STi5518 memory map

Designation	Start	End	Description
Peripherals	#20000000	#20000FFF	Int. controller
	#20003000	#20003FFF	UART0 (ASC0) Smartcard 0
	#20004000	#20004FFF	UART1 (ASC1)
	#20005000	#20005FFF	UART2 (ASC2)
	#20006000	#20006FFF	UART3 (ASC3) Smartcard 1
	#20007000	#20007FFF	SCCG0 (Smcard 0 clkgen)
	#20008000	#20008FFF	SCCG1 (Smcard1 clkgen)
	#20009000	#20009FFF	SSC0
	#2000A000	#2000A0FF	SSC1
	#2000A100	#2000A1FF	PIO5
	#2000A200	#2000A2FF	IR Blaster
	#2000A300	#2000A3FF	TTXT
	#2000A400	#2000AFFF	Reserved
	#2000B000	#2000BFFF	PWM
	#2000C000	#2000CFFF	PIO 0
	#2000D000	#2000DFFF	PIO 1
	#2000E000	#2000EFFF	PIO 2
	#2000F000	#2000FFFF	PIO 3
	#20010000	#20010FFF	PIO 4
	#20011000	#20011FFF	ILC
	#20024000	#20024FFF	MPEGDMA 0
	#20025000	#20025FFF	MPEGDMA 1
	#20026000	#20026FFF	Block Move DMA
	#20027000	#20027FFF	MODEM DMA
	#20030000	#20037FFF	Reserved
	#20038000	#2003FFFF	Link extra (Link Interface)

Table 35 STi5518 memory map

Designation	Start	End	Description
Region 3 EMI banks0 to 3	0x40000000	0x4FFFFFFF	EMI bank 0. SDRAM/DRAM supported
	0x50000000	0x5FFFFFFF	EMI bank 1. SDRAM/DRAM supported
	0x60000000	0x6FFFFFFF	EMI bank 2. SDRAM/DRAM not supported
	0x70000000	0x7FFFFFFF	EMI bank 3, normally used for boot ROM. DRAM not supported
	0x7FFFFFFE		Boot entry point
Region 0 DCache or SRAM	0x80000000	0x80000003	Reserved
	0x80000004	0x8000000F	Reserved
	0x80000010	0x80000013	Reserved
	0x80000014	0x8000003F	Reserved
	0x80000040	0x8000004F	High priority Breakpoint trap handler
	0x80000050	0x8000005F	High priority Breakpoint trapped process
	0x80000060	0x8000006F	High priority Error trap handler
	0x80000070	0x8000007F	High priority Error trapped process
	0x80000080	0x8000008F	High priority SystemOperations trap handler
	0x80000090	0x8000009F	High priority SystemOperations trapped process
	0x800000A0	0x800000AF	High priority Scheduler trap handler
	0x800000B0	0x800000BF	High priority Scheduler trapped process
	0x800000C0	0x800000CF	Low priority Breakpoint trap handler
	0x800000D0	0x800000DF	Low priority Breakpoint trapped process
	0x800000E0	0x800000EF	Low priority Error trap handler
	0x800000F0	0x800000FF	Low priority Error trapped process
	0x80000100	0x8000010F	Low priority SystemOperations trap handler
	0x80000110	0x8000011F	Low priority SystemOperations trapped process
	0x80000120	0x8000012F	Low priority Scheduler trap handler
	0x80000130	0x8000013F	Low priority Scheduler trapped process
	0x80000140	0x80000FFF	Internal SRAM: < 4 Kbytes user code, data and stack
	0x80001000	0x800017FF	Internal SRAM if data cache is not enabled.User-code, data and stack
	0x80001800	0xBFFFFFFF	Reserved
Region 1 Shared SDRAM	0xC0000000	0xC07FFFFF	SDRAM. Video memory, user code, data, stack
	0xC0400000	0xFFFFFFFF	Reserved

Table 35 STi5518 memory map

6.3 System memory use

The following sections of address space are reserved for system use:

- The locations below the address **MemStart** at the bottom of memory are dedicated to processor use. The address of **MemStart** is returned by the *ldmemstartval* instruction.
- When booting from ROM, the system boots from the predefined location **BootEntry** (0x7FFFFFFE) at the top of memory.

Areas of memory reserved for processor use should not be accessed directly. Special instructions are provided for manipulating these areas.

The special address **MemStart** marks the base of user memory space.

Peek and *poke* use the two words above **MemStart**, i.e. memory locations 0x80000140 to 0x80000147. The use of peek and poke is described in System services on page 82.

Subsystem channels memory

Each channel based DMA subsystem is allocated a word of storage below **MemStart**. This is used by the processor to store information about the state of that channel. This information should not normally be examined directly, although debugging kernels may need to do so.

Interrupting DMA subsystems do not have a channel word allocated and rely on interrupts to perform synchronization with the processes running on the processor.

Memory for trap handlers

The area of memory reserved for trap handlers is broken down hierarchically as follows:

- each high/low process priority has a set of trap handlers;
- each set of trap handlers has a handler for each of the four trap groups;
- each trap group handler has a trap handler structure and a trapped process structure;
- each of the structures contains four words.

The contents of these addresses can be accessed via *ldtraph*, *sttraph*, *ldtrapped* and *sttrapped* instructions.

Boot ROM

When the processor boots from ROM, it jumps to a boot program held in ROM with an entry point 2 bytes from the top of memory at 0x7FFFFFFE. These 2 bytes are used to encode a negative jump of up to 256 bytes down in the ROM program. For large ROM programs it may then be necessary to encode a longer negative jump to reach the start of the routine.

7 Memory

7.1 External memory

Programmable CPU interface memory

The programmable CPU interface memory (commonly referred to as the EMI) decodes region 3 of the address space into four banks, into which different external memories and peripherals can be mapped. Further details of the EMI can be found in Programmable CPU memory interface on page 63. Two of the banks support DRAM and one bank is normally used for boot ROM.

- Locations 0x40000000 to 0x5FFFFFFF (banks 0 and 1) are generally used for SDRAM/DRAM, but may be used for any external memory or peripherals.
- The locations 0x60000000 to 0x6FFFFFFF (bank 2) may be used for any external memory or peripherals except SDRAM/DRAM.
- The locations 0x70000000 to 0x7FFFFFFF (bank 3) may be used for any external memory or peripherals except SDRAM/DRAM, but are generally used for boot ROM. When booting from ROM, the system boots from the predefined location BootEntry (0x7FFFFFFE) at the top of memory space.

Accessing some areas of memory causes special access characteristics (strobes etc.) to be generated depending on the way the EMI is programmed.

The EMI provides address decoding, address and data buses, timing strobes, enabling signals and refresh where appropriate.

Shared SDRAM memory

The shared SDRAM memory occupies the first 16, 32 or 64 Mbits of region 1, and is shared with the MPEG decoders. OSD bitmaps, for example, are stored in this memory.

For details of the shared SDRAM memory interface configuration and set-up, refer to the Register Manual.

7.2 On-chip SRAM memory

This internal memory module, known as on-chip memory, contains 6 Kbytes of SRAM, which is mapped into the lowest 6 Kbytes of memory space from MinInt (0x80000000) extending upwards, as shown in Figure 11 on page 51.

Part of the lowest 4 Kbytes of memory is committed to system use; see System memory use on page 55 for details. The remainder of the lowest 4 Kbytes of memory is uncommitted and can be used to store on-chip data, stack or code for time-critical routines.

The upper 2 Kbytes of the on-chip memory is also uncommitted SRAM, and is contiguous with the lower 4 Kbytes. However, it can be configured to be the data cache, as described below, in which case it is not available as SRAM.

Locations between 0x80001800 (or 0x80001000 if data cache is used) and 0xBFFFFFFF should not be addressed.

7.3 Caching

Cache can be used to reduce the average access delay imposed on the CPU when it accesses a memory location to read or write. Some locations should not be cached, for example those to which other modules have direct memory access (DMA) and the CPU also requires access to those locations. This is because DMA operations always bypass the data cache and so cache incoherence problems will occur. It is therefore recommended to configure such memory locations as being non-cacheable.

The STi5518 cache subsystem provides:

- 2 Kbytes of direct-mapped write-back data cache
- 2 Kbytes of direct-mapped read-only instruction cache

The cache configuration is held in memory-mapped registers. The registers must be accessed using the device access instructions.

Device access instructions can also be used to force access to external memory as they bypass the cache. Device writes do not change the value in the cache. These instructions can be used to solve any cache coherency issues, for example where data is being DMA copied from a cached location. However care must be taken in using device access instructions to access memory (rather than device registers) as cache coherency problems may occur rather than being solved, if the CPU also performs normal (therefore cached) memory accesses to these locations.

Registers are provided to configure areas of memory to be cacheable or non-cacheable for data access, as described in Cacheable and non-cacheable memory locations on page 60.

Note that the correct cache initialization sequences, described in Cache initialization on page 58, must be used before the caches are enabled.

7.3.1 Outline of operation

The cache is four 32-bit words (16 bytes) wide and 128 lines (2 KBytes, 512 words) high. It is direct-mapped (sometimes called *one way set associative*). This is shown in Figure 12 below.

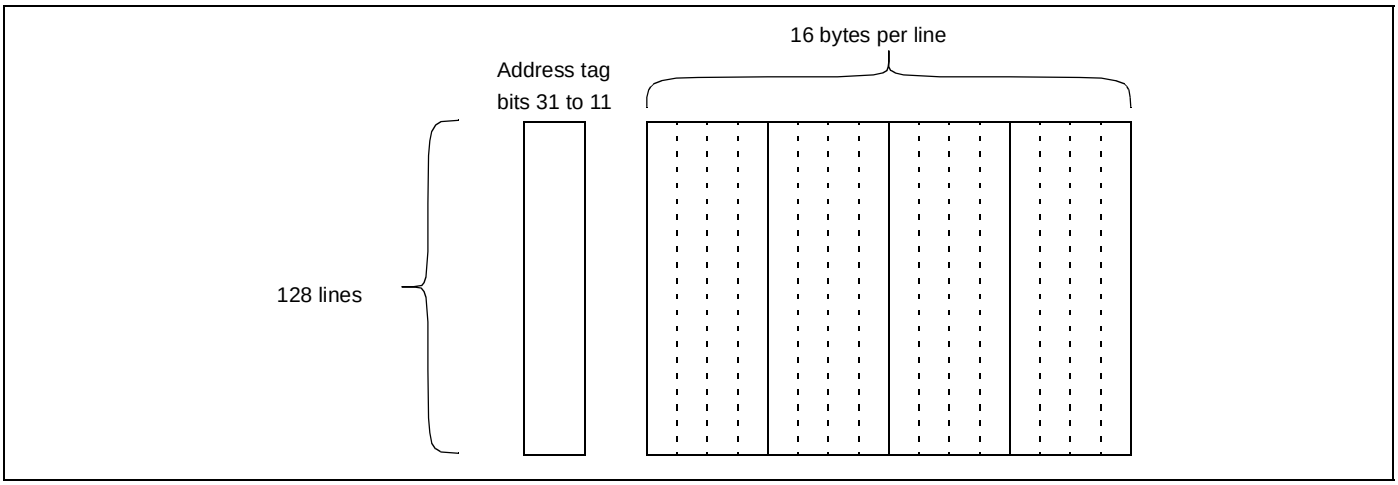


Figure 12 Kbyte data or instruction cache2

Each line of the cache can only store data from specific four-word sections of memory at 2 Kbyte intervals, with the bottom line of the cache coinciding with the 4 words just above each 2 Kbyte boundary. Thus the line number of the cache pinpoints the four-word section of memory within a 2 Kbyte block, i.e. bits 4 to 10 of the address. The 21 most significant bits of the address selects the 2 Kbyte block. These 21 bits are stored in 128 tag registers, with one tag

register corresponding to each cache lines. The significance of the parts of the address when using the cache are shown in the figure below.

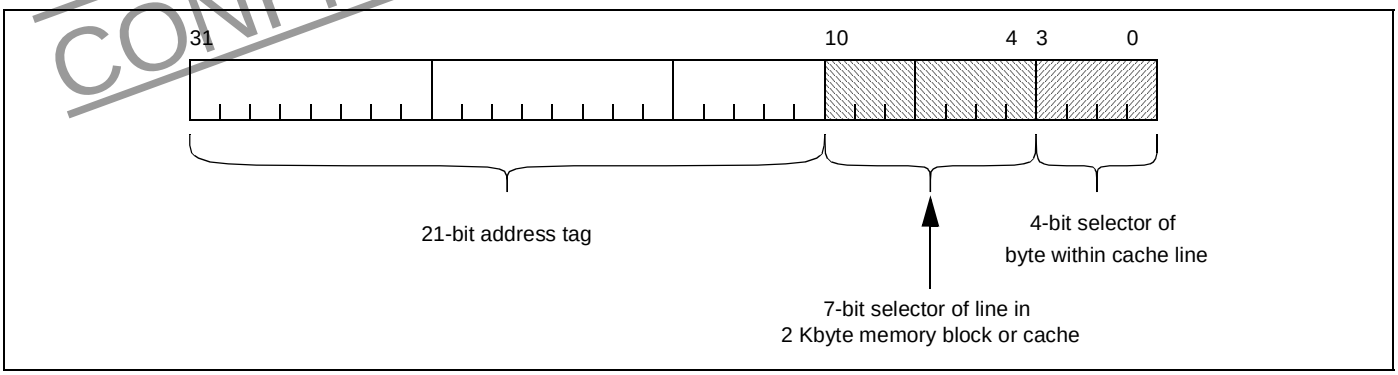


Figure 13 Address fields when using cache

If a request is made to access a cacheable memory location, and a copy of that location is held in cache, then the access is said to have made a cache hit. A hit is identified by comparing the address bits 11 to 31 with the address tag for the cache line given by the address bits 4 to 10. If the cache is hit, then the access is completed by the cache subsystem. If the cache is missed, the appropriate cache line is written back to memory, and if necessary the new location in memory is read into that cache line. All cache reads and writes to memory are complete lines because of the efficiency of accessing the memory in burst mode.

7.3.2 Cache initialization

Before the caches are enabled, they must be correctly initialized. To do this the cache must first be invalidated before it is accessed. To ensure this occurs, the invalidate bit of each cache must be set with the cache disabled and then the enable bit set to enable the cache.

This sequence has the effect of forcing a cache to be invalid, which initializes the cache state before any other accesses are considered by the cache.

7.3.3 Cache subsystem control

The cache subsystem registers control cache functions such as flushing and invalidation, and are used to mark sections of memory space as cacheable or not cacheable. Registers should be accessed using the device access instructions.

The CAC_CACHECONTROLLOCK must be 0 before some registers can be changed. After changing registers, the CAC_CACHECONTROLLOCK should be set to 1. Once this lock is set it cannot be cleared except by a reset. It is not recommended to change the cache configuration other than at reset.

7.3.4 Data cache

It is possible to select either data cache or an extra 2 Kbyte of on-chip SRAM. This is done by writing to the CAC_DCACHENOTSRAM register. The default is to enable the extra on-chip SRAM.

Set the CAC_DCACHENOTSRAM bit to 1 to select data cache mode and enable the cache. Do not access locations 0x80001000 to 0x800017FF when using the data cache. The cache invalidate bit should be set before enabling the cache. Invalidating a cache marks every line as not containing valid data. This is done by setting the CAC_INVALIDATEDCACHE register to 1. This register is automatically reset to 0 on completion of the task, but it cannot be directly read by the CPU as the register is write only. Therefore bit 2 is provided in the CAC_CACHESTATUS register to indicate when the invalidate operation has completed.

Note that for the data cache, setting the CAC_INVALIDATEDCACHE register before the cache is enabled does not actually cause the invalidation sequence to begin. This only occurs when the cache is enabled when writing to the CAC_DCACHENOTSRAM register. So the initialize sequence of operations should be:

- write 1 to the CAC_INVALIDATEDCACHE register;
- write 1 to the CAC_DCACHENOTSRAM register as an “atomic” operation.
  Where, “atomic” means than no other CPU software processes / tasks, trap handlers or interrupt handlers can execute in the time between the write to invalidate register and the write to cache enable register. Note: this will normally be the case as the cache initialization should be performed during the initialization stage of the application code, before any operating system (STLite/OS20) or interrupt handlers have been installed.
- wait for the invalidate sequence (128 CPU clock cycles) to finish.
  This can done by a software delay loop and/ or by polling bit 2 of the cache status register.

Note it is recommended to ensure that the CPU does not attempt to execute any code or access any data in cacheable memory locations before the invalidate sequence has finished. Therefore if the cache initialization is done by the CPU (rather than by poking the registers via the TAP and DCU, before booting the application code from the software toolset) the CPU software (code and data) that waits for the invalidate sequence to finish should reside in non-cacheable memory locations, so that the cache is not accessed while it is still invalidating. If this is not ensured it is possible for data corruption to occur.

It is not recommended to change selection from data cache to SRAM during operation. However, if it is necessary to do so, it is essential to flush the cache to maintain memory integrity before making the change.

Flushing the cache means forcing a write-back to memory of every dirty line in the cache. A dirty line is a line of cache that has been written to since it was loaded or last written back. Only the data cache can be flushed; the instruction cache never needs flushing since it is read only.

To flush the data cache, set the CAC_FLUSHINGDCACHE register to 1. It is automatically reset to 0 on completion of the task, but cannot be read directly by the CPU as the register is write only. Therefore to check that the flush operation has finished it is necessary to poll bit 4 of the cache status register.

Ensure that the cache-flush software function is “atomic” and operates as follows:

- writes to the flush register;
- waits for the flush to finish;
- writes to the CAC_DCACHENOTSRAM register as an “atomic” operation.

Where, “atomic” means than no other software processes / tasks, trap handlers or interrupt handlers can execute in the time between writing to the flush register and the DCache being changed back to SRAM mode.

Note: the CPU code function and associated data used to perform this sequence must be placed in non-cacheable memory locations, so the cache is not accessed while it is still flushing.

7.3.5 Instruction cache

To use the instruction cache it must be first be invalidated by writing 1 to the CAC_INVALIDATEICACHE register. Invalidating a cache marks every line as not containing valid data. This is done by setting the CAC_INVALIDATEICACHE register to 1. This register is automatically reset to 0 on completion of the task (128 CPU cycles). However this register can not be read by the CPU, as it is write only, therefore bit 3 (Invalidating ICache) in the CAC_CACHESTATUS register can be read instead to check that the invalidate operation has completed. The instruction cache must be allowed to complete the invalidate operation before the cache is enabled. Note: unlike the data cache the instruction cache actually starts the invalidation sequence when the CAC_INVALIDATEICACHE is written to and so can be allowed to complete before enabling the cache by writing to the CAC_ENABLEICACHE register.

The instruction cache can then be enabled by writing 1 to the CAC_ENABLEICACHE register; the default condition is disabled, no instruction cache.

The CAC_CACHESTATUS register is read only, and shows the current state of the caches.

The cache configuration can be locked by writing a 1 to the CAC_CACHECONTROLLOCK register bit. Reset of this flag is only performed by a hardware reset. This bit should be set to 1 after all the cache configuration registers have been written.

7.3.6 Cacheable and non-cacheable memory locations

Note: The cacheability control registers for both region1 and region 3 should be programmed before the caches are enabled. In this way the cacheable and non-cacheable memory areas are already defined when the caches are enabled, see Cache initialization on page 58. Do not dynamically change these registers after cache initialization.

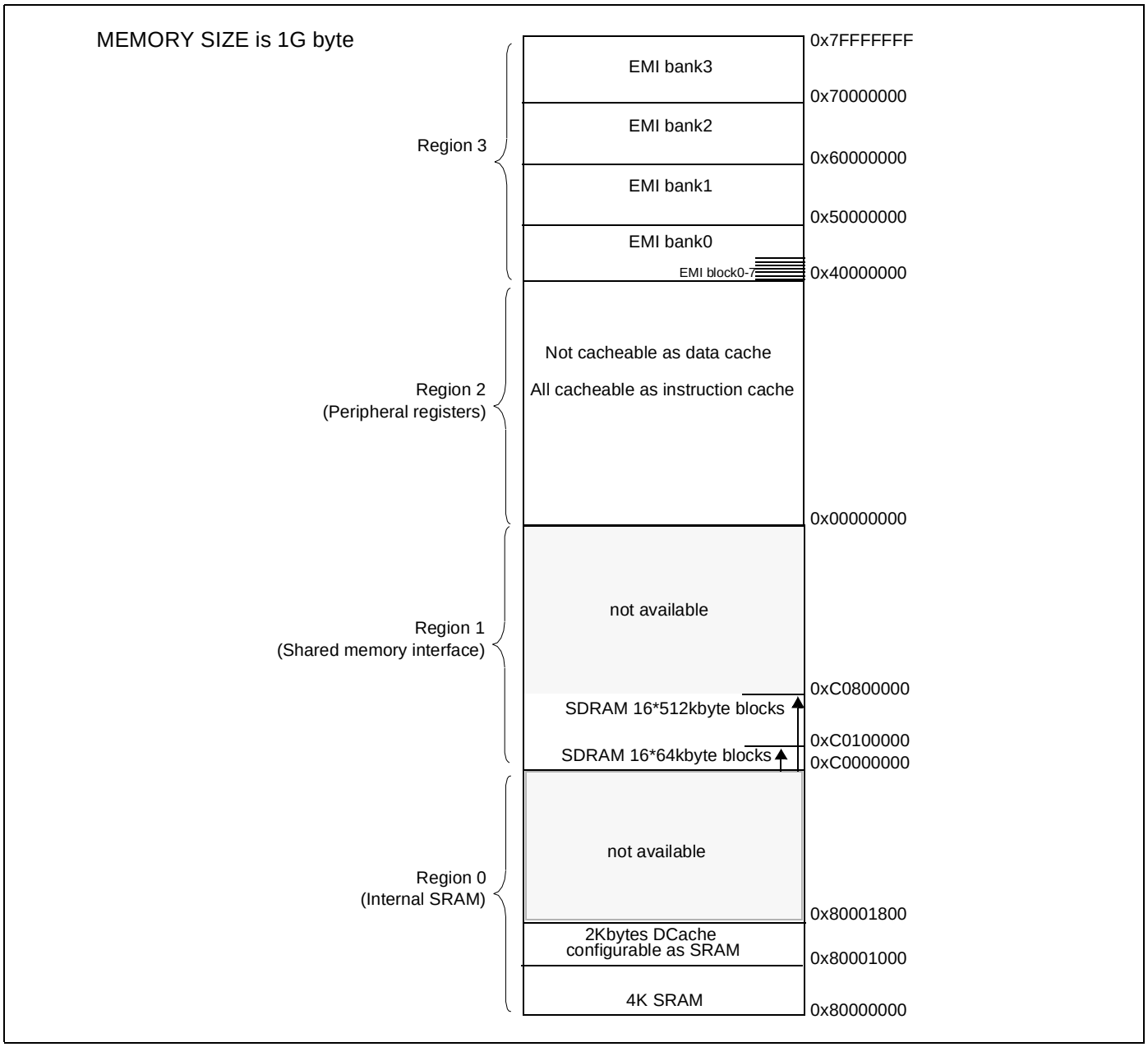


Figure 14 Memory cacheability map

Region 0

REGION 0 is used only to access internal SRAM and is, therefore, never cacheable for either ICACHE or DCACHE.

There are always 4K of SRAM. Because the DCACHE can be configured to be SRAM, there can be another 2K SRAM (for a total amount of 6K SRAM). This is controlled by the CAC_DCACHENOTSRAM register, which must be configured to'0' for SRAM, or to'1' for DCACHE.

Region 1

This region is mostly non-cacheable for data access. It can be programmed to be cacheable (default is non-cacheable) in either 16 blocks of 64K bytes or 16 blocks of 512K bytes. These blocks are contiguous and are placed starting from the address 0xC0000000. Each one of these blocks may be selected individually as cacheable or non-cacheable for DCACHE memory accesses. This is controlled by two CAC registers (CAC_CACHECONTROL0 and 1), where each bit controls the cacheability characteristics of a particular 64K/512K block. To select between 64K byte or 512K byte block size, bit 8 of CAC_CACHECONTROL0 register is used. After reset, all blocks are marked non-cacheable and the size of each block is set to 64K byte. The registers must then be programmed to mark certain blocks cacheable.

The remainder of REGION1 is non-cacheable for DCACHE.

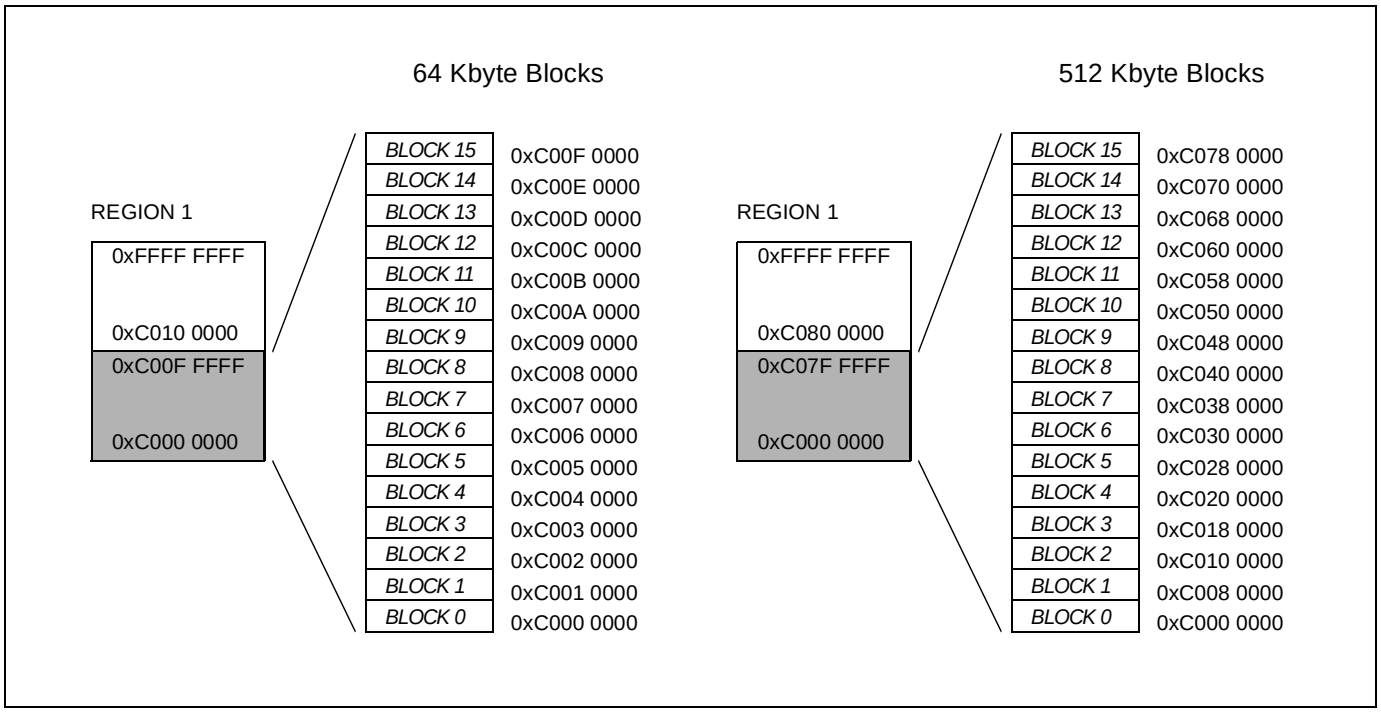


Figure 15 Region1 cacheable area with block sizes of 64K byte and 512Kbyte.

Region 2

Always non-cacheable for DCACHE, but is all cacheable for ICACHE.

Region 3

This region is split into 4 EMI banks, 0-3. For ICACHE they are all cacheable. For DCACHE they can be set as cacheable or non-cacheable by the 4-bit register CAC_CACHECONTROL3, the default is non-cacheable. EMI bank 0 is split into two parts; the upper part is cacheable in the same way as EMI banks 1-3 and the lower part, containing 8 x 64Kbyte blocks, can be set non-cacheable. These blocks, located between 0x40000000 and 0x4007FFFF, can be set cacheable by the 8-bit register CAC_CACHECONTROL2.

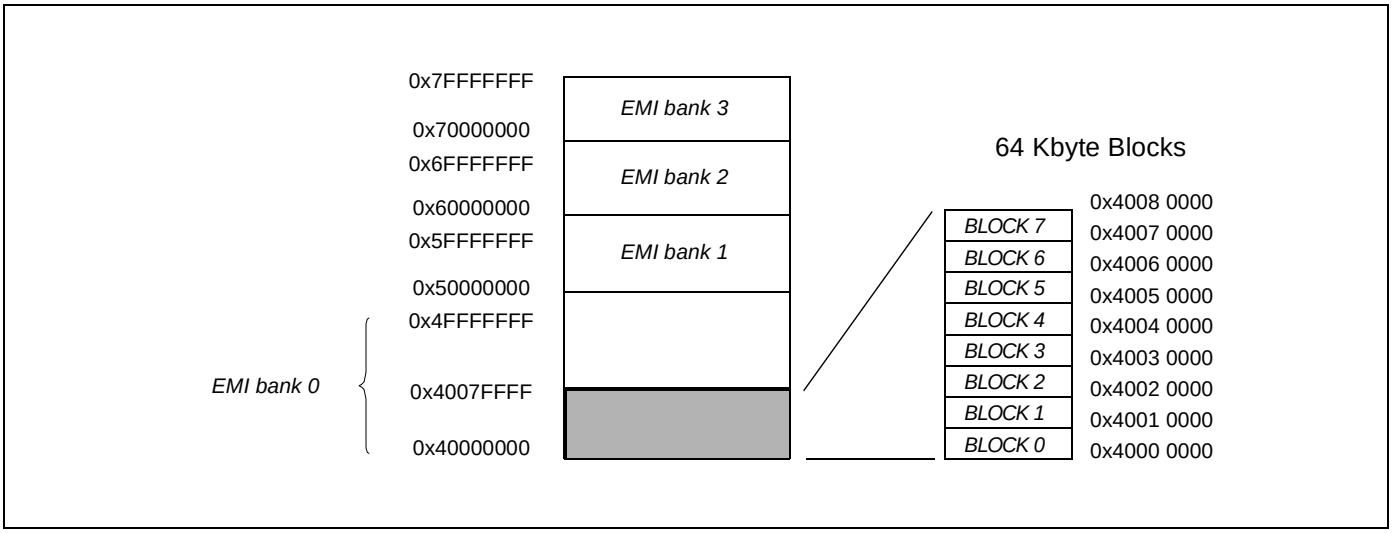


Figure 16 Region 3

8 Programmable CPU memory interface

The Programmable CPU Interface (EMI) controls the movement of data between the STi5518 and off-chip memory, except for the shared SDRAM which is connected to a dedicated interface (SMI). The EMI uses minimal external support logic to support memory subsystems.

The EMI accesses a 32 Mbytes physical address space (greater if SDRAM or DRAM is used) in four general purpose memory banks of 8 or 16 bits wide, 21 or 22 address lines, and byte select.

For DVD applications requiring extra memory, the EMI supports this extra memory with zero external support logic.

The interface can be configured for a wide variety of timing and decode functions through configuration registers.

The EMI maps external memory into the top quarter of the address space and is partitioned into four banks with each bank occupying one sixteenth of the total address space.

Systems can use several memory types such as SDRAM, DRAM, SRAM, Flash EPROM, and other peripherals. The figure below illustrates the Programmable CPU Interface memory allocation.

Warning! *It is not possible to have SDRAM and DRAM on the Programmable CPU Interface at the same time.*

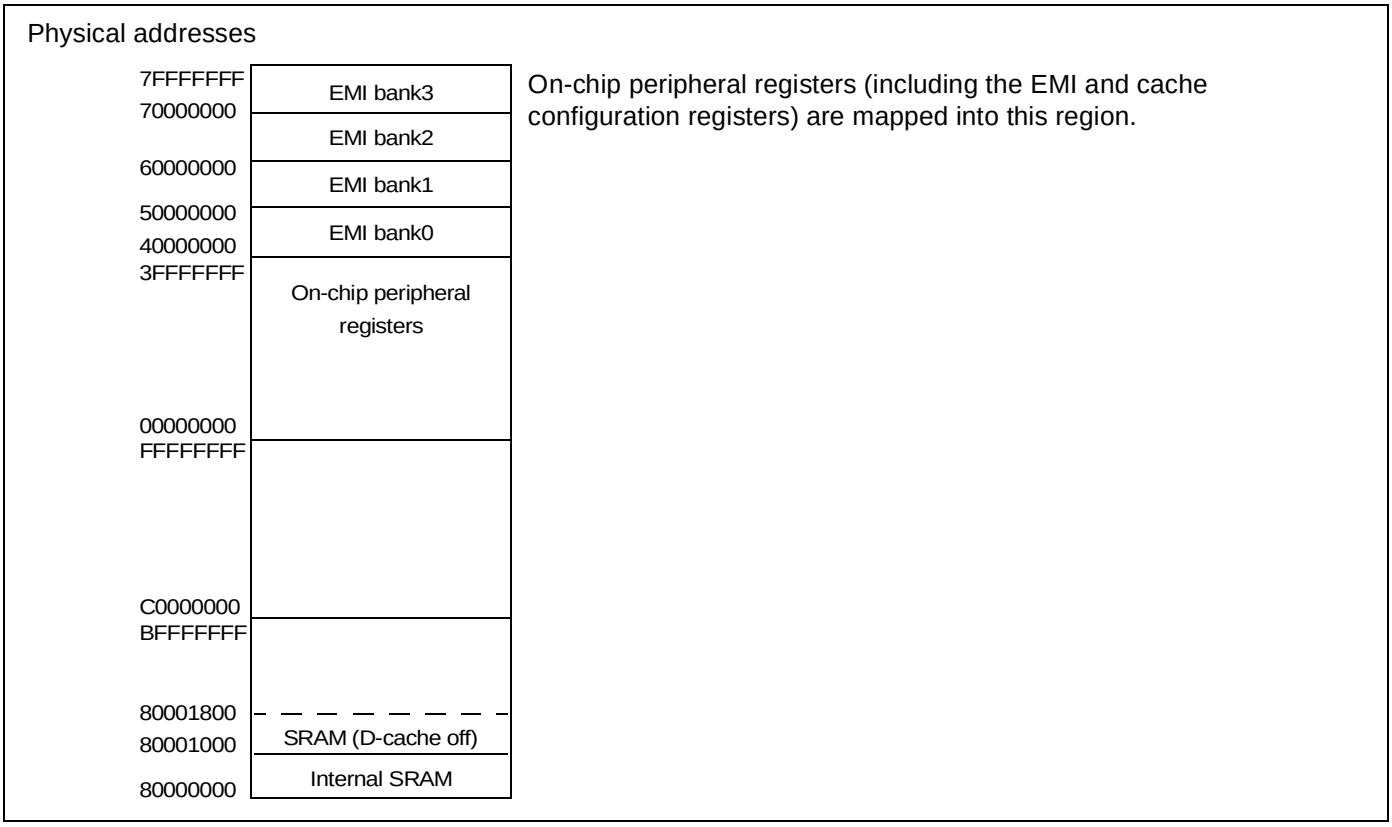


Figure 17 Memory allocation

The timing of each of the four memory banks can be selected separately, with different device types being placed in each bank with no external hardware support. Banks can be configured to contain 8-bit wide or 16-bit wide devices.

The EMI supports three memory types:

- SDRAM with a multiplexed row and column address;
- DRAM with a multiplexed row and column address used to support fast page mode;
- SRAM or peripherals, which is used to support SRAMs, peripherals, EPROM or Flash ROMs.

EMI banks 0 and 1 support either memory type, while banks 2 and 3 only support SRAM or peripheral memory types.

Words of 1 byte and 2 bytes can be addressed.

The behavior of some strobes depends on whether the bank being accessed has been configured as SDRAM, DRAM or SRAM / peripheral

8.1 Pin functions

Note A cycle is defined as one processor clock cycle, and a phase as a half of one processor clock cycle.

The table below describes the functions of the Programmable CPU Interface pins. Note that a signal name prefixed by **not** indicates that the pin is active-low. The pins are listed in alphabetical order.

Pin	Function
BOOT_FROM_ ROM	When the BOOT_FROM_ROM pin is held low, the STi5518 boots from the DCU. When the BOOT_FROM_ROM pin is held high, the STi5518 boots from ROM. Boot code is run from an external ROM in bank 3 (at the top of memory). The BOOT_FROM_ROM pin is also used to encode the size of bank3 (which is 16-bit), and this value overrides the PortSize value in the bank 3 configuration registers. If the STi5518 is being booted by the DCU, then the bootstrap must execute from internal memory until the EMI has been configured.
CPU_ADDR[1:21] and [22]	The address bus operates in both multiplexed and non-multiplexed modes. When a bank is configured to contain SDRAM, DRAM, or another multiplexed memory, the device type is set to SDRAM or DRAM, and the internally generated 32-bit address is multiplexed as row and column addresses through the external address bus. An extra address bit CPU_ADDR[22] is used when no DRAM or SDRAM support is required on the Programmable CPU Interface. This pin allows direct addressing of up to 8Mbytes in peripheral SRAM modes. It is also used for the notCPU_CAS[0] signal.
CPU_DATA[0:15]	The data bus transfers 16 or 8-bit data items depending on the bus width configuration. The least significant bit of the data bus is always CPU_DATA[0]. The most significant bit varies with bus width. It will be CPU_DATA15 for 16-bit data items, and CPU_DATA7 for 8-bit data items.
CPU_PROCLK	This is a reference signal for external bus cycles, which oscillates at the processor clock frequency.
CPU_RW	This signal indicates whether the current cycle is a read or a write cycle. During writes, the signal is asserted low at the beginning of the access (i.e. at the start of RASTime for DRAM banks and at the start of CSTime for SRAM / peripheral banks) and de-asserted high at the end of the access (end of CASTime / CSTime). At all other times this signal is held high.
CPU_WAIT	Wait states can be generated by taking CPU_WAIT high. CPU_WAIT is only sampled during SRAM or peripheral accesses. CPU_WAIT retains the state of any strobe during the cycle after the one in which it was asserted, until it is de-asserted. When CPU_WAIT is de-asserted the access continues as programmed by the configuration interface. The CPU_WAIT signal can be treated as synchronous or asynchronous to the CPU_PROCLK clock, depending on the state of a bit 5 in the EMI_ConfigPadLogic register. This pin can not be disabled by software. If it is not used, this pin should be pulled down with a resistor.

Table 36 Programmable CPU interface pin descriptions

<table>
<tr><th>Pin</th><th>Function</th></tr>
<tr><td>notCPU_BE[0:1]</td><td>The EMI uses word addressing, two byte-enable strobes are provided, and the use of the byte enable pins depends on the bus width.

16-bit wide memory is defined as an array of 2-byte words: 31 address-bits in the ST20 memory space select a 2-byte word and “notCPU_BE[0:1]” selects one byte within the word.

8-bit wide memory is defined as an array of 1-byte words with 32 address-bits selecting a word. For 8-bit wide memory, the lower order address bit (A0) is multiplexed onto the unused byte-enable pin notCPU_BE[1] to give a 32-bit address bus. This address bit can be made available to the address bus by configuring the bank width as below.
<table>
<tr><th>Pin</th><th>16-bit external port size</th><th>8-bit external port size</th></tr>
<tr><td>notCPU_BE[1]</td><td>enables CPU_Data[8:15]</td><td>notCPU_ADDR[0]</td></tr>
<tr><td>notCPU_BE[0]</td><td>enables CPU_Data[0:7]</td><td>enables CPU_Data[8:15]</td></tr>
</table>
For banks configured as SDRAM, notCPU_BE pins provide DQ mask signals.

For banks configured as DRAM, notCPU_BE strobes are valid from the start of CASTime to one phase before the end of CASTime.

For banks configured as SRAM, notCPU_BE pins used as data-enable strobes have the same timing and may be configured to be active on read cycles, write cycles, or both read and write cycles.</td></tr>
<tr><td>notCPU_CAS[0:1]</td><td>The two notCPU_CAS[0:1] strobes have different meanings depending on the contents of banks 0 & 1 - DRAM (bank and byte mode), SDRAM and SRAM. The three configurations are described below.</td></tr>
</table>

Table 36 Programmable CPU interface pin descriptions

<table>
<tr><th>Pin</th><th>Function</th></tr>
<tr><td>notCPU_CAS[0:1]
DRAM
configuration</td><td>
The CAS strobes can be programmed on a per-bank basis in one of two modes.

Bank mode in which only one CAS strobe is used for the entire bank and sub-banks (if any).
Byte mode in which each CAS strobe is used as a byte decoded CAS strobe and can be used across both banks (and any sub-banks).

Byte mode supports 16-bit wide DRAMs or DRAM modules that provide multiple CAS strobes, one for each byte, and a single write signal for byte write operations.

The alternative type DRAMs that have multiple write signals, one for each byte, and a single CAS to allow byte write operations or banks that are constructed from 1, 4, or 8-bit wide DRAMs can be interfaced using bank mode.

Note Bank or byte mode can be selected independently for banks 0 and 1.

CAS strobes in bank mode (DRAM)

If banks 0 and1 are set to DRAM device type with bank mode selected then notCPU_CAS[0] is the sole CAS strobe for bank 0 and notCPU_CAS[1] is the sole CAS strobe for bank1. Unused CAS strobes remain inactive during an access.

CAS strobes in byte mode (DRAM)

For banks containing DRAM, which require byte decoded CAS strobes, one programmable CAS strobe is allocated to each byte. Each of the CAS strobes in this mode will have the timing programmed into the CAS timing configuration registers, of the bank being accessed, if they are active during that cycle. Byte mode CAS strobes are active during an access if the byte corresponding to the strobe is being accessed. During refresh cycles, all CAS strobes will go low at the start of the cycle and remain low until the end of the cycle.

The table below shows how the CAS strobes are used in byte mode. Note that the strobes are common to both banks and any sub-banks. Only the CAS strobes that enable bytes which are being accessed will be active during an access cycle.

The table below summarizes the Byte mode notCPU_CAS[0:1] strobe pins
<table>
<tr><td>CAS strobe</td><td>Bank 0, 16-bits wide</td><td>Bank 1, 16-bits wide</td></tr>
<tr><td>notCPU_CAS[1]</td><td>enables CPU_DATA[8:15]</td><td>enables CPU_DATA[8:15]</td></tr>
<tr><td>notCPU_CAS[0]</td><td>enables CPU_DATA[0:7]</td><td>enables CPU_DATA[0:7]</td></tr>
</table>
Mixing bank and byte mode (DRAM)

For full flexibility, any permutation of bankwidth CAS mode (byte / bank) is supported for both banks 0 and 1. The following table gives a full listing of the active strobes for all permutations
<table>
<tr><td>No of DRAMs</td><td>Bank Configuration</td><td>notCPU_CAS[0]</td><td>notCPU_CAS[1]</td></tr>
<tr><td rowspan="2">One DRAM in bank0 or 1</td><td>bank mode</td><td>active</td><td>unused</td></tr>
<tr><td>byte mode</td><td>active CPU_DATA[0:7]</td><td>active CPU_DATA[8:15]</td></tr>
<tr><td rowspan="4">Two DRAMs in bank 0 and1</td><td>DRAM in bank0 bank mode</td><td>active</td><td>unused</td></tr>
<tr><td>DRAM in bank0 byte mode</td><td>active CPU_DATA[0:7]</td><td>active CPU_DATA[8:15]</td></tr>
<tr><td>DRAM in bank1 bank mode</td><td>unused</td><td>unused</td></tr>
<tr><td>DRAM in bank1byte mode</td><td>active CPU_DATA[0:7]</td><td>active CPU_DATA[8:15]</td></tr>
</table>
</td></tr>
</table>

Table 36 Programmable CPU interface pin descriptions

<table>
<tr><th>Pin</th><th>Function</th></tr>
<tr><td>notCPU_CAS[0:1] SDRAM configuration</td><td>
<table>
<tr><th>No of DRAMs</th><th>notCPU_CAS[0]</th><th>notCPU_CAS[1]</th></tr>
<tr><td>One SDRAM in bank0 or
One SDRAM in bank1</td><td>active (CAS strobe for SDRAM)</td><td>active chip select for SDRAM</td></tr>
<tr><td>One SDRAM in bank0 or
One SDRAM in bank1</td><td>active (CAS strobe for both SDRAMs)</td><td>active chip select for SDRAM in bank 0</td></tr>
</table>
</td></tr>
<tr><td>notCPU_CAS[0:1] SRAM configuration</td><td>For banks which do not contain SDRAM or DRAM the notCPU_CAS[1] pin is inactive. If there is no SDRAM or DRAM at all on the programmable CPU interface, notCPU_CAS[0] is CPU_ADDR[22] to allow up to 8MBytes SRAM addressing.</td></tr>
<tr><td>notCPU_CE[0:3]</td><td>These four signals are used for programmable strobe (for example, chip select when the corresponding bank is configured as SRAM/peripheral

When SRAM/DRAM is used on the corresponding CPU interface, notCPU_CE[0:3] is used as the RAS signal.</td></tr>
<tr><td>notCPU_OE</td><td>The behavior of the notCPU_OE signal depends on the type of memory being accessed. If the access is to a bank configured as DRAM then the notCPU_OE strobe is active only during a read access when it is asserted low CASe1Time after the start of CASTime, and de-asserted high at the end of CASTime. For accesses to configured as SRAM / peripheral the notCPU_OE strobe is programmable and will behave according to the values in the EMIConfigData registers for that bank.</td></tr>
<tr><td>notCPU_RAS[0:1]</td><td>These two signals control the RAS strobe for SDRAM and DRAM. The two signals do not necessarily correspond to individual banks. Bank0 (only) may be sub-decoded.

For DRAM, notCPU_RAS[0:1] strobes are used as the RAS strobes for bank0, bank1, or sub-banks.

For SDRAM, one RAS strobe is used for all devices in the bank, and sub-decoding is carried out using the notCPU_RAS[1] and notCPU_CAS[1]pins.

If bank0 is not programmed as SDRAM or DRAM, the notCPU_RAS[0] strobe is used as chip-select (notCPU_CE[0]) for the bank.
<table>
<tr><th>Signal</th><th>Type</th><th>Definition</th></tr>
<tr><td>notCPU_RAS[0]</td><td>out</td><td>RAS strobe for SDRAM/DRAM in Bank 0, or
RAS strobe for lowest DRAM sub-bank in Bank0, or
chip select for Bank0</td></tr>
<tr><td>notCPU_RAS[1]</td><td>out</td><td>RAS strobe for SDRAM/DRAM in Bank1, or
RAS strobe for highest DRAM sub-bank in Bank0, or
SDRAM chip select signal for highest sub-bank of Bank0</td></tr>
</table>
The table below summarizes the notCPU_RAS[0:1] strobes for banks 0 and 1.
<table>
<tr><th>Bank Configuration</th><th>notCPU_RAS[0]</th><th>notCPU_RAS[1]</th></tr>
<tr><td>Bank 0: DRAM with no sub-decoding</td><td>Bank 0 RAS strobe</td><td>Unused for bank0</td></tr>
<tr><td>Bank 0: DRAM with two sub-banks</td><td>Bank 0 sub-bank0 RAS strobe</td><td>Bank 0 sub-bank1 RAS strobe</td></tr>
<tr><td>Bank 0: SDRAM with no sub-decoding</td><td>Bank 0 RAS strobe</td><td>Unused for bank0</td></tr>
<tr><td>Bank 0: SDRAM with two sub-banks</td><td>Bank 0 sub-bank0 RAS strobe</td><td>Bank 0 sub-bank1 with chip select</td></tr>
<tr><td>Bank 0: SRAM / peripherals</td><td>Bank 0 chip select strobe</td><td>Bank3 sub-decoding</td></tr>
<tr><td>Bank 1DRAM with no sub-decoding</td><td>Bank 1RAS strobe</td><td>Unused for bank1</td></tr>
<tr><td>Bank 1DRAM with two sub-banks</td><td>not possible</td><td>not possible</td></tr>
<tr><td>Bank 1: SDRAM with no sub-decoding</td><td>Bank 1 RAS strobe</td><td>Unused for bank1</td></tr>
<tr><td>Bank 1: SDRAM with two sub-banks</td><td>not possible</td><td>not possible</td></tr>
<tr><td>Bank 1contains SRAM / peripheral</td><td>Unused for bank1</td><td>Bank3 sub-decoding</td></tr>
</table>
</td></tr>
</table>

Table 36 Programmable CPU interface pin descriptions

8.2 Configuration list

The following tables illustrate the different configurations supported by SDRAM, DRAM and peripheral memories, the different strobes used in each case. Note that it is not possible to have SDRAM and DRAM on the EMI at the same time.

EMI bank configuration				Strobes
bank0	**bank1**	**bank2**	**bank3**	
Peripheral	Peripheral	Peripheral	Peripheral	notCPU_CE[0] for bank0, notCPU_CE[1]for bank1, notCPU_CE[2]for bank2 and notCPU_CE[3]for bank3. notCPU_RAS1 available to subdecode bank3 notCPU_CAS0 is CPU_ADD[22] to allow up to 8 Mbytes of SRAM addressing notCPU_CAS1 not used notCPU_OE shared by each bank notCPU_BE[1:0] shared by each bank CPU_R/W shared by each bank CPU_WAIT shared by each bank
DRAM	Peripheral	Peripheral	Peripheral	If DRAM in bank 0: notCPU_CE[1] for bank1
or				If DRAM in bank1: notCPU_CE[1] for bank0
Peripheral	DRAM	Peripheral	Peripheral	notCPU_CE[2]for bank2 notCPU_CE[3]for bank3 notCPU_CE[0] (RAS0) for DRAM despite of DRAM position notCPU_CAS0 (CAS0) used by DRAM notCPU_CAS1 (CAS1) used by DRAM only if multi-byte mode enabled If DRAM in (and only in) bank0, then subdecoding up to 2 DRAM sub-banks using CPU_RAS1 If DRAM in bank0 and no DRAM subdecoding OR DRAM in bank1, CPU_RAS1 available to subdecode bank3 notCPU_OE shared by each bank notCPU_BE[1:0] shared by each bank. CPU_R/W shared by each bank. CPU_WAIT shared by each bank

Table 37 List of strobes used for all EMI configurations

EMI bank configuration				Strobes
bank0	bank1	bank2	bank3	
DRAM	DRAM	Peripheral	Peripheral	notCPU_CE[0] (RAS0) for DRAM in bank0, notCPU_RAS1 for DRAM in bank1, notCPU_CE[2] for bank2, notCPU_CE[3] for bank 3, notCPU_CE[1] not used. No subdecoding is allowed for either DRAM. No subdecoding of bank3 Multi-byte mode can be allowed for both DRAMS at the same time: If multi-byte mode is not enabled: notCPU_CAS0 for DRAM in bank0 notCPU_CAS1 for DRAM in bank1 If multi-byte mode is enabled: notCPU_CAS0 used as shared by both DRAMs notCPU_CAS1 used as shared by both DRAMs notCPU_OE is shared by each bank notCPU_BE[1:0] is shared by each bank. CPU_R/W shared by each bank. CPU_WAIT shared by each bank
SDRAM	Peripheral	Peripheral	Peripheral	If SDRAM in bank 0: notCPU_CE[1] for bank1, If SDRAM in bank1: notCPU_CE[1] for bank0 In both above cases (only 1 SDRAM): notCPU_CAS1 used as chip select (sdram_cs(0)) for SDRAM notCPU_CE[0] used as RAS0 strobe for SDRAM notCPU_CAS0 used as CAS0 strobe for SDRAM notCPU_RAS1 used to subdecode bank3 If two SDRAM in bank0 (SDRAM subdecoding in bank0): notCPU_CAS1 used to map sdram_cs(0) for SDRAM 0 notCPU_RAS1 used to map sdram_cs(1) for SDRAM 1 (NOTE: In this case not possible subdecode bank3) notCPU_CE[0] used as shared RAS0 strobe by both SDRAMs notCPU_CAS0 used as shared CAS0 strobe by both SDRAMs In any case: notCPU_CE[2] for bank2 and notCPU_CE[3] for bank3 notCPU_OE shared by each bank notCPU_BE[1:0] (with DQM functionality for SDRAM) shared by all banks CPU_R/W shared by each bank. CPU_WAIT shared by each bank
or				
Peripheral	SDRAM	Peripheral	Peripheral	

Table 37 List of strobes used for all EMI configurations

EMI bank configuration				Strobes
bank0	bank1	bank2	bank3	
SDRAM	SDRAM	Peripheral	Peripheral	notCPU_CAS1 used as chip select (sdram_cs(0)) for SDRAM in bank0
				notCPU_RAS1 used as chip select (sdram_cs(2)) for SDRAM in bank1
				notCPU_CE[2] for periph in bank2
				notCPU_CE[3] for periph in bank3
				notCPU_CE[0] used as shared RAS strobe for both SDRAMs
				notCPU_CE[1] not used
				notCPU_CAS0 used as shared CAS strobe by both SDRAM
				No subdecoding is allowed for both SDRAMs.
				No subdecoding of bank3.
				notCPU_OE shared by each bank
				notCPU_BE[1:0] (with DQM functionality for SDRAM) shared by each bank.
				CPU_R/W shared by each bank.
				CPU_WAIT shared by each bank

Table 37 List of strobes used for all EMI configurations

8.3 External bus cycles

The external memory interface supports dynamic memory and other devices such as static memory and IO devices. This flexibility is provided by allowing the required wave-forms to be programmed via configuration registers (see Section 8.4: *EMI configuration* on page 80).

Memory is byte addressed, with words aligned on four-byte boundaries and half-words on two-byte boundaries.

During read cycles, byte-level addressing is performed internally by the STi5518. The EMI can read bytes, half-words or words. It always reads the size of the bank.

During write cycles, the STi5518 uses the notCPU_BE[0:1] strobes to perform addressing of bytes. If a particular byte is not to be written then the corresponding data outputs are tri-stated. Writes can be less than the size of the bank.

The internally generated address is indicated on pins notCPUAddr[1:21, 22]. The least significant bit of the data bus is always CPU_DATA[0]. The most significant bit is adjusted dynamically to suit the required external bus size.

The following sections describe the access cycles for the three device types, DRAM, SDRAM and SRAM/peripherals.

8.3.1 DRAM

DRAM access cycles are supported in Banks 0 and 1 only when these are set to device type DRAM.

The EMI can support either one DRAM bank set in one of the 4 possible banks or two DRAM banks set in bank0 and bank1.

A DRAM memory access cycle consists of a number of defined periods or times, as shown in the figure below. All of the named times shown in this diagram together with other parameters such as RAS address shift and page size are programmable to suit a wide variety of DRAM types.

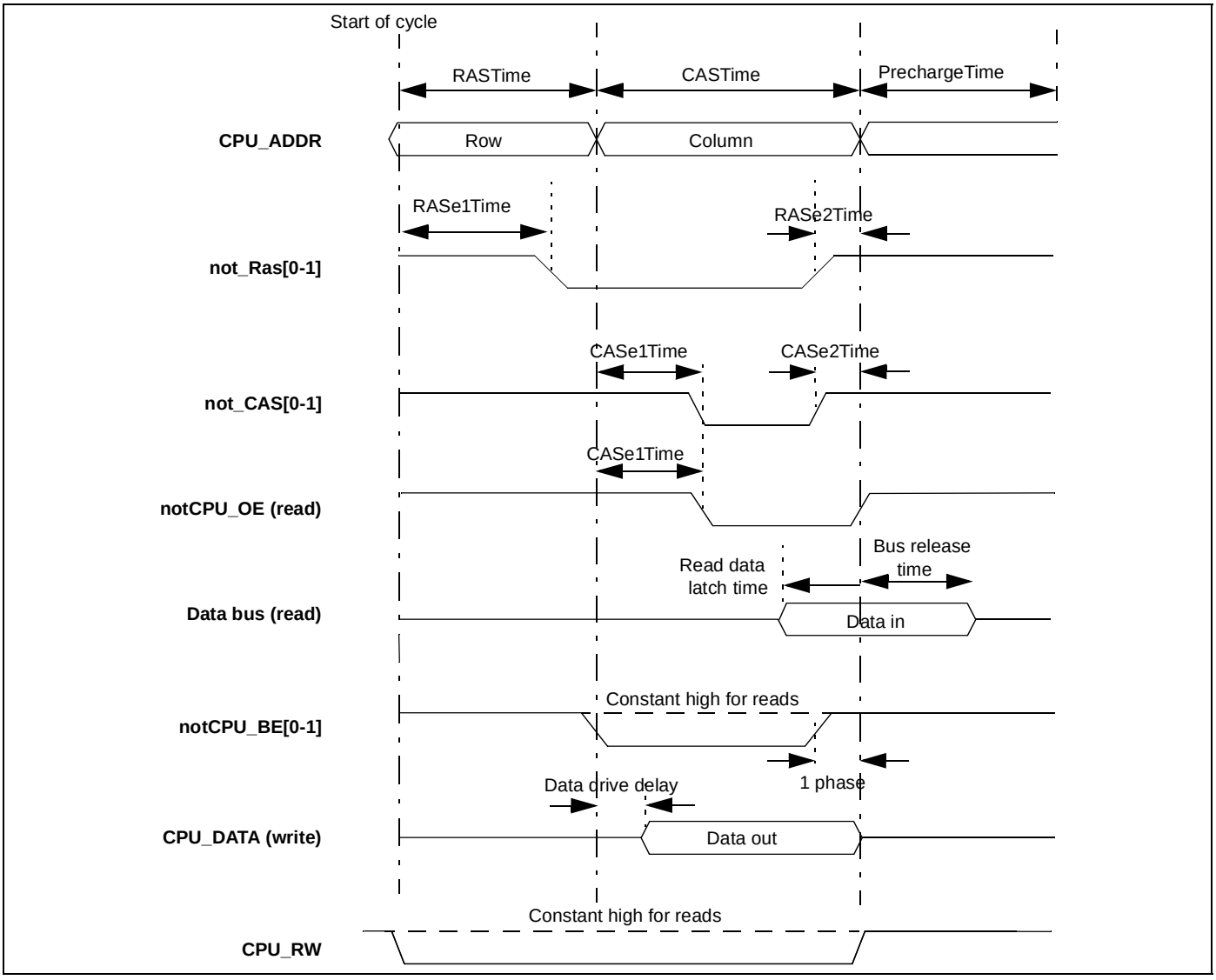


Figure 18 DRAM memory cycle

Signals RASTime and CASTime are consecutive. The CASTime can be followed by concurrent Precharge and BusRelease times.

Thus for DRAM, these times are used for RAS address latching, CAS address latching, RAS precharge and output driver tristate times respectively. For consecutive access to the same bank of DRAM, RASTime will only occur when there is a page miss. The next access will not commence until the PrechargeTime for a previous access to the same bank has completed. During the RASTime, a transition can only be programmed on the RAS strobes.

During the CASTime the CAS strobes and either the byte-enable or notCPU_OE strobes are active. The address is output on the address bus without being RAS shifted. Write data is valid during CASTime. Read data is latched into the interface at the point defined by the LatchPoint bit in the EMIConfigData3 register for the bank being accessed.

The PrechargeTime and BusReleaseTime commence concurrently at the end of the CASTime. A PrechargeTime will occur, and the active notCPU_RAS strobe will be taken high if:

- the next access is to the same bank but to a different row address;
- the next access is to a different bank.

The BusReleaseTime runs concurrently with the PrechargeTime and will occur if:

- the current cycle is a read and the next cycle is a write;
- the current cycle is a read and the next cycle is a read from a different bank.

The BusReleaseTime is provided to allow an accessed device to float to a high impedance state.

Page mode

DRAM pages are delineated using the RASBits configuration parameter. These bits are used as an address mask for comparison with the previous dram address. If an access is requested by an internal subsystem of the STi5518 to a DRAM bank while a DRAM access is in progress, the new address is compared to the current access address. If the row addresses are the same, the access may proceed as a page mode access. There is no specific configuration bit to select pagemode DRAM. If all the RASBits are set to 0, then no pagehits will be caused and normal DRAM RAS/CAS cycles will always be produced.

A page mode access does not include the RASTime period. The notCPU_RAS strobe is not taken high before commencing the page mode access. If the current access is a read and the page mode access is due to be a write, a BusReleaseTime is inserted as shown in figure below. The notCPU_RAS strobe is held low during this period.

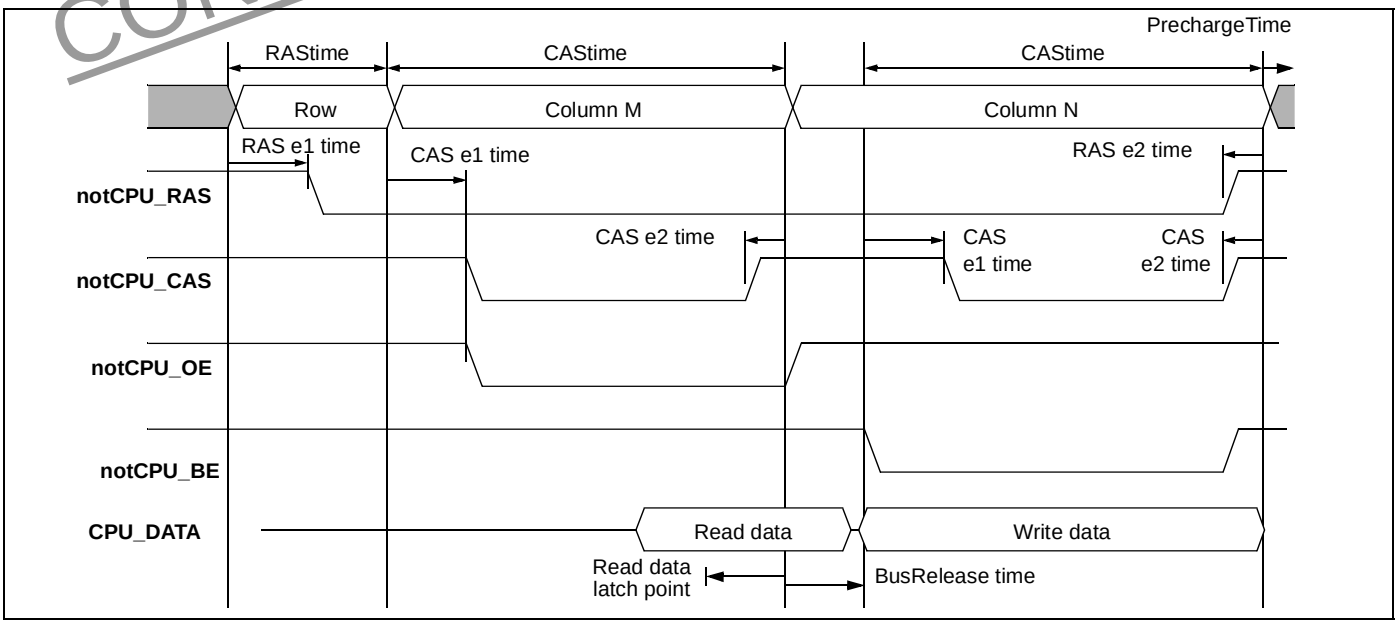


Figure 19 Read followed by page mode write

When setting the RASBits, care must be taken to consider the port size. Also, if the bank has been sub-decoded, the sub-bank selection address bits must be included in the comparison, so the RASBits corresponding to these addresses must be set.

For example, if the DRAM bank is composed of 2 x 256k * 16 devices, the sub-bank selection address bit is A19, so the RASBits corresponding to address bits A_{19}-A_{10} must be set.

When page mode is active, the RASe2time must be programmed to zero.

SubBank	SubBankSize	PortSize	SubBank selection address	RAS strobe selection
2	256K 1M 4M 16M	8 bit	Address<18> Address<20> Address<22> Address<24>	0 = notCPU_RAS[0] 1 = notCPU_RAS[1]
	256K 1M 4M 16M	16 bit	Address<19> Address<21> Address<23> Address<25>	
	256K 1M 4M 16M	32 bit	Address<20> Address<22> Address<24> Address<26>	

Table 38 Address decoding

Refresh

DRAM banks are periodically refreshed at intervals specified by the Refresh Interval configuration parameter.

The notCPU_CAS strobe(s) is taken low at the beginning of the refresh time. The position of the RAS falling edge (RASedge) is programmable and the minimum width of the CAS pulse is the sum of the RASTime and CASTime values specified for random access. If there is more than one bank of DRAM the refresh configuration will then be taken from the lowest numbered bank configured as DRAM.

All sub banks are refreshed in the same access and a cycle is inserted between each bank and/or sub-bank in order to spread current peaks. If no DRAM has been programmed for a bank then no transitions occur on the relevant RAS or CAS strobes and all unused RAS and CAS strobes (i.e. strobes not used due to the choice of bank/byte mode, sub-banks and bankwidth) will remain inactive during a refresh cycle.

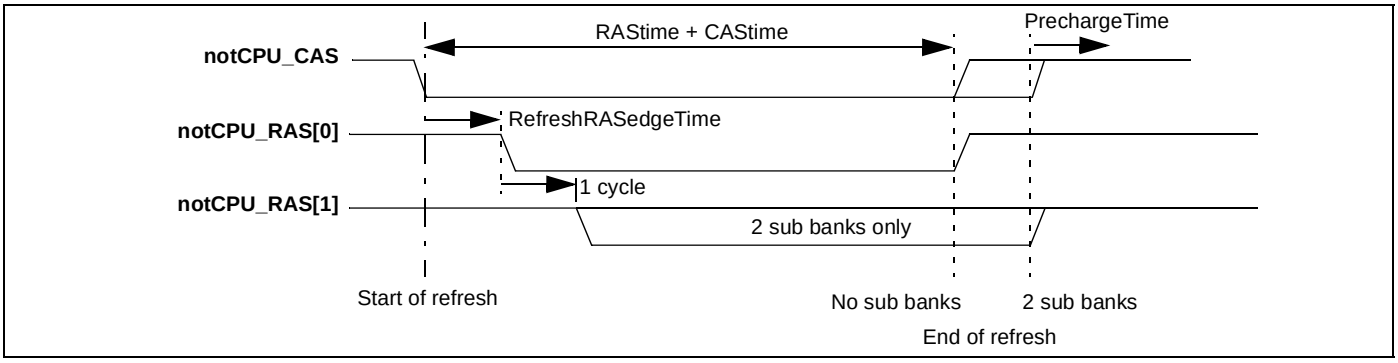


Figure 20 Generic refresh access for one DRAM bank

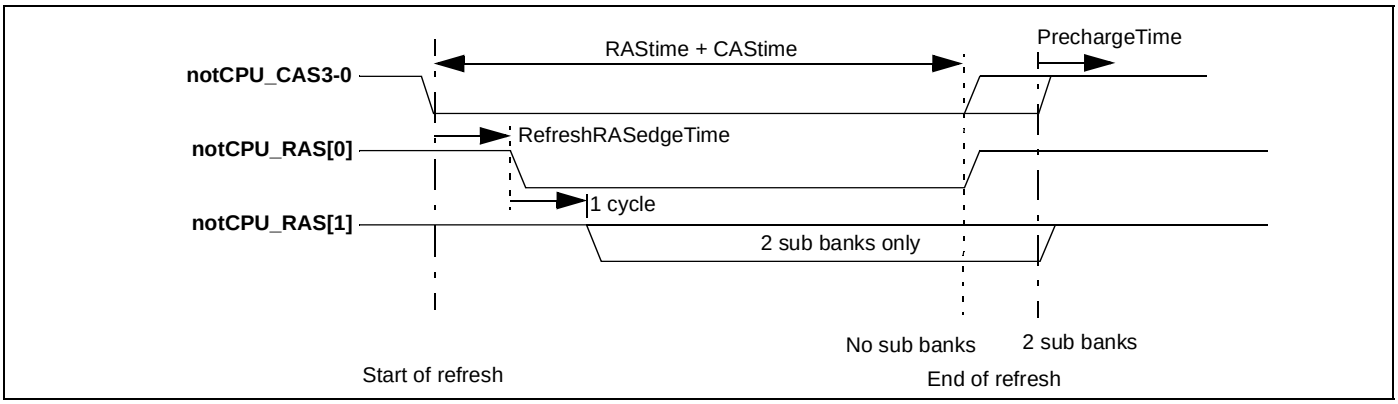


Figure 21 Generic refresh access for two DRAM banks

Name	Programmable value
PrechargeTime	1 - 8 cycles
RefreshInterval	(1 - 16) * 128 cycles
RefreshRASedgeTime	1 or 2 cycles after start of refresh

Table 39 Refresh parameters

The EMI ensures that notCPU_CAS and notCPU_RAS are both high for the required time before every refresh cycle by inserting a PrechargeTime in the last bank being accessed and ensuring all PrechargeTimes are complete.

Note, no refreshes will take place until after a DRAMinitialize command in the ConfigCommand register is performed.

8.3.2 SDRAM

Note These diagrams show the waveforms at device's pads, NOT the outputs from the generic EMI block.

Typically the EMI pad logic will re-time the EMI's outputs to the next cycle. All signals on this interface are synchronous to the system clock, which is fed to the SDRAMs on the CPU_PROCLK pad.

8.3.2.1 Typical access

The following diagrams shows typical read and write accesses to an SDRAM. This example shows a bank activation, due to a page miss, then two write accesses in the same bank are performed in page mode.

A precharge is then done in anticipation of another bank activation command. If as in this example only one SDRAM word is to be written then the notCPU_BE signal is used as a data mask, so that only the correct word is updated.

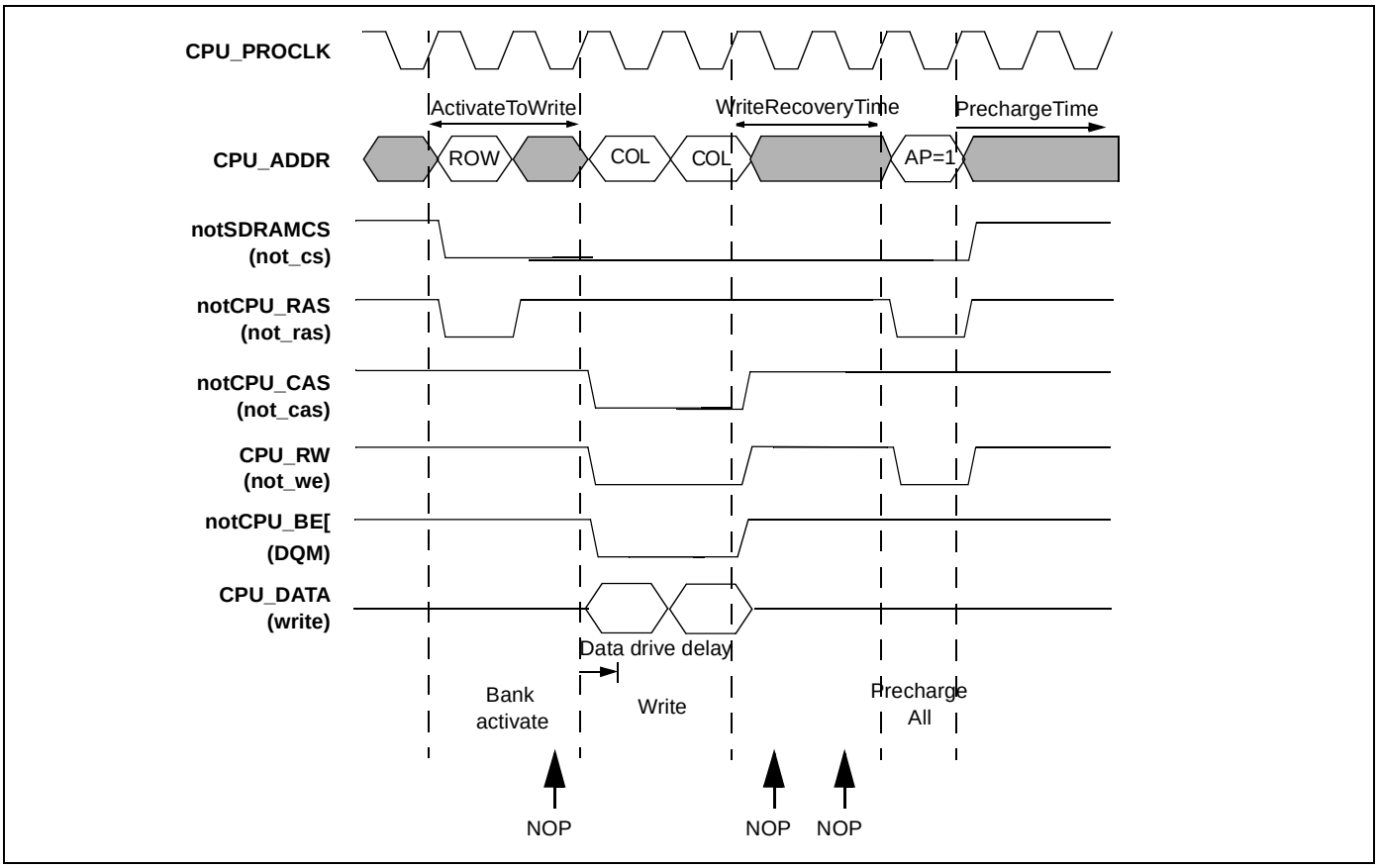


Figure 22 SDRAM write accesses

The following figure shows a bank activation, due to a page miss, then two read accesses in the same bank are performed in page mode. A precharge is then done in anticipation of another bank activation command. If as in this example only one SDRAM word is to be read then the notCPU_BE signal is used as a data output enable.

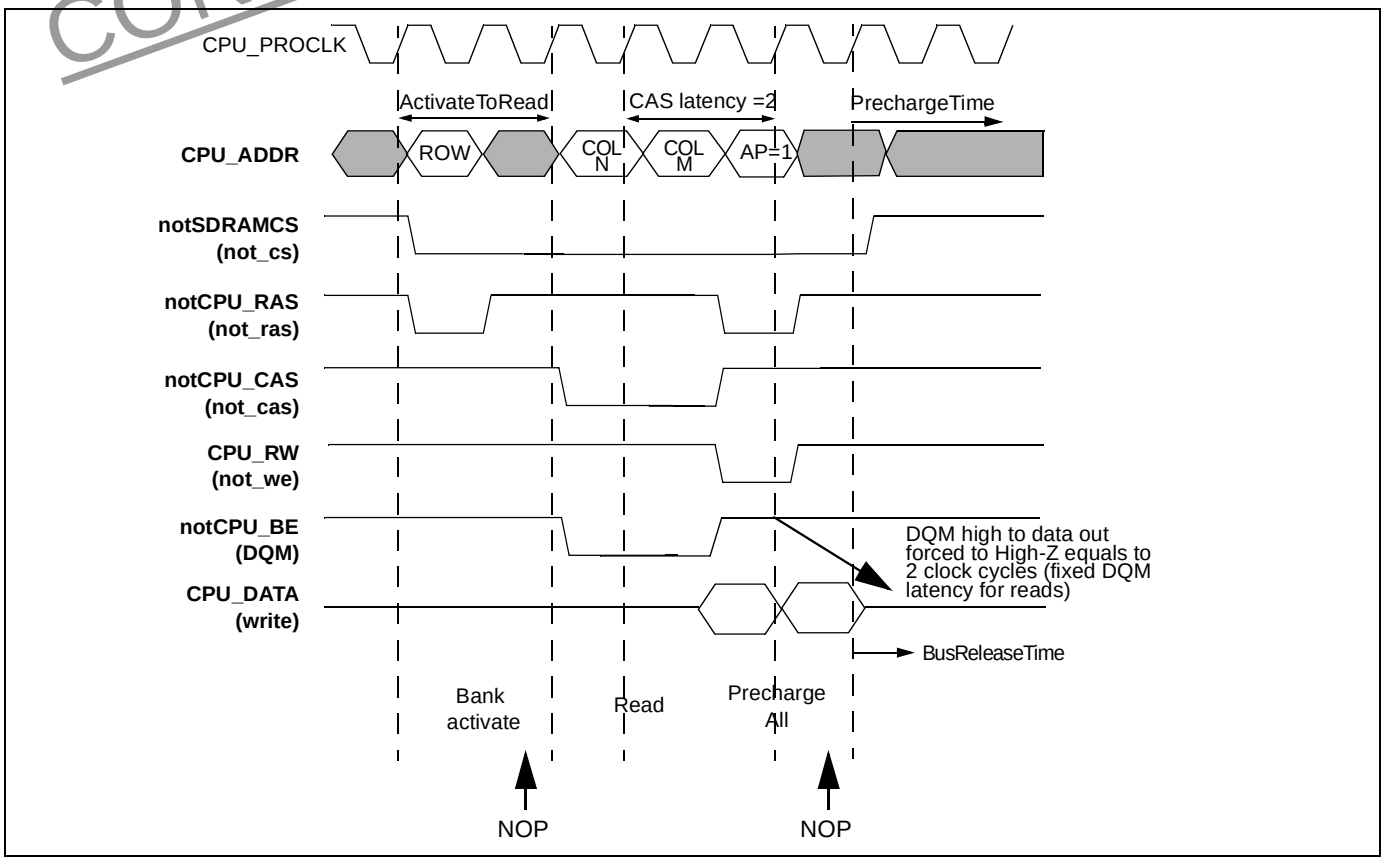


Figure 23 SDRAM read accesses with CAS latency = 2 cycles

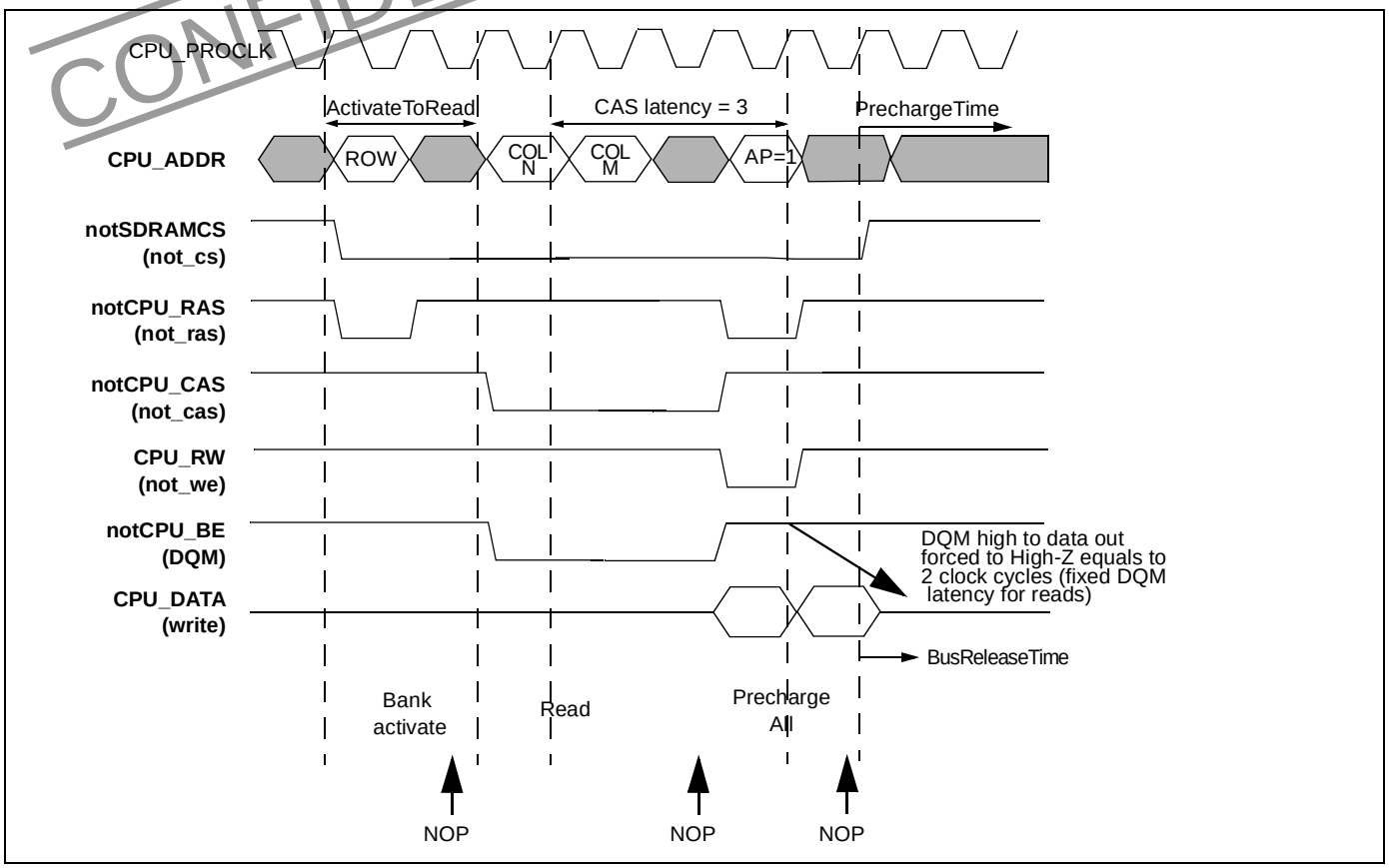


Figure 24 SDRAM read accesses with CAS latency = 3 cycles

8.3.3 SRAM or peripheral access cycles

A generic peripheral (e.g. SRAM, EPROM, FLASH, etc.) type of access is provided which is suitable for direct interfacing to a wide variety of SRAM, ROM, Flash and other peripheral devices. No internal sub-decoding is provided with banks in this configuration. All of the named times shown in Section 8.3.4 together with other parameters such as bank size and bank size dependent shifts are programmable to suit a wide variety of device types.

For details of the configuration of the EMI see Section 8.4: *EMI configuration* on page 80.

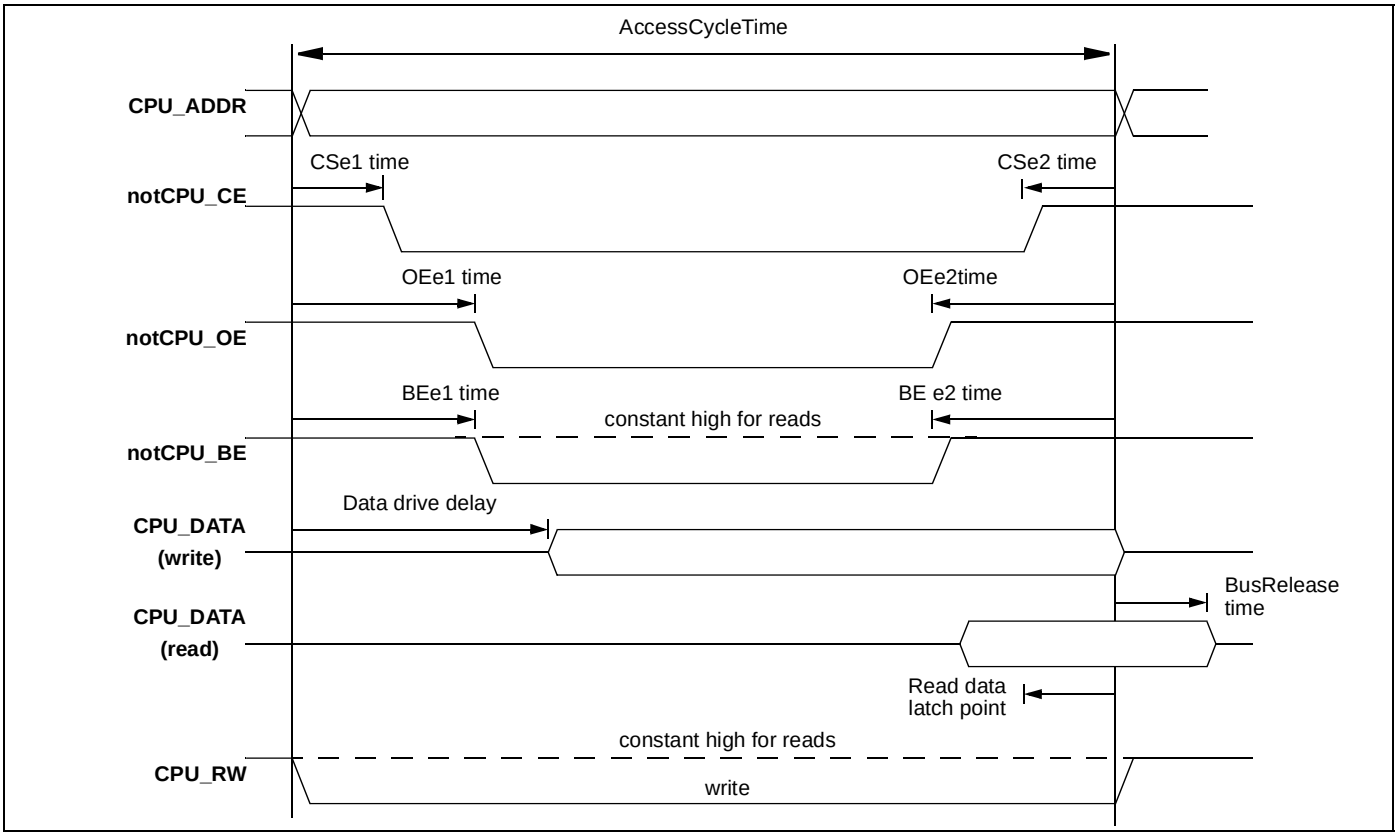


Figure 25 Generic peripheral access

The distance between signal time pairs OEe1/OEe2 and BEe1/BEe2 is primarily set by the registers EMI_ConfigData1Bank and EMI_ConfigData2Bank. If this distance is set to maximum by these registers, then an additional delay of up to 3 wait cycles can be inserted for peripheral accesses to the upper portion of EMIbank3, using the EMI_Configpadlogic register bit WaitCycles. These bits control a wait period that is inserted both at the start of the EMI access cycle and at the end (triggered by the $\overline{CS}$ low-to-high transition). For a 60 MHz clock, this gives a maximum additional delay of 133ns.

8.3.4 Wait

CPU_WAIT is provided so that external cycles can be extended to enable variable access times, for example, shared memory access. CPU_WAIT is only effective during accesses to SRAM / peripheral banks and is ignored during accesses to DRAM banks. The STi5518 can accept either synchronous or asynchronous CPU_WAIT signals. If CPU_WAIT is synchronous, then wait states can be inserted at precise times during the access. An asynchronous CPU_WAIT does not require any external synchronization but cannot accurately insert wait states during an access. The following description and diagrams assume that a synchronous CPU_WAIT is being used. The CPU_WAIT signal can be enabled on a per-bank basis. Note that the selection of the asynchronous or synchronous CPU_WAIT signal is the same for all banks. CPU_WAIT freezes the state of the strobes for the duration of the cycles in which it was sampled high. Any strobe transitions occurring on the sampling edge or the falling edge immediately after this will not be inhibited, however, transitions on the rising and falling edges of the following cycle will not occur. The following two figures illustrate the extension of the external memory cycle and the delaying of strobe transitions.

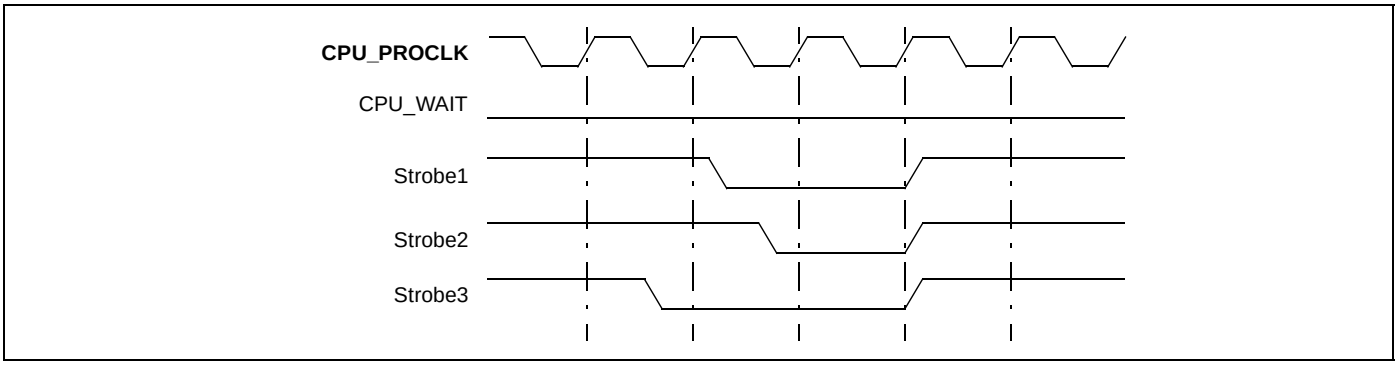


Figure 26 Strobe activity without CPU_WAIT

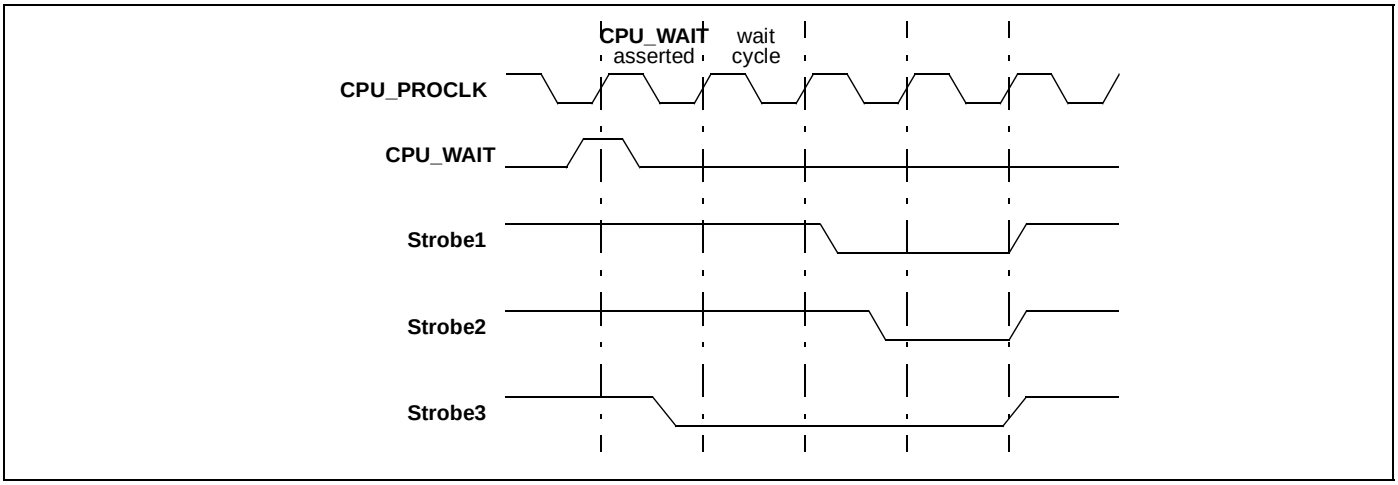


Figure 27 Strobe activity with CPU_WAIT

The asynchronous CPU_WAIT uses an extra clock edge to synchronize the signal before it is sampled in the EMI. Apart from this extra cycle of latency, the response to the two types of CPU_WAIT is the same. Configuration of the CPU_WAIT pin to a synchronous or asynchronous wait signal is performed by bit 5 of the EMI_ConfigPadLogic register. Setting this bit high selects a synchronous wait signal, setting it low selects an asynchronous wait signal. Note the asynchronous CPU_WAIT does not need to meet setup and hold times to the CPU_PROCLK signal rising edges.

8.3.5 Bank-width based address shifting

Address shifting can be enabled on a per bank basis to allow population options on boards which vary bank widths to be handled more easily. The shifts can only be enabled in banks programmed to the SRAM or peripheral device type, the shift amount being dependent on the width of the bank. Shifting is enabled or disabled by 4 bits, one for each bank, of the EMI padlogic register ConfigPadLogic0-3 where bit 0 refers to bank 0 and so on. The table below shows the addresses presented on the CPU_ADDR[1:21] pins for different configurations. Note that the addresses presented on the notCPU_BE[0:1] signals for 16 or 8 bit banks are not affected by the shifting.

Bank device type	configpadlogic0-3	current_portsize<1:0>	CPU_ADDR[1:21]
DRAM	-	--	Address2-23 during CAS time
SRAM or Peripheral	0 = shift disabled	01 (32 bit)	Address2-23
		10 (16 bit)	Address1-22
	1 = shift enabled	11 (8 bit)	Address0-21

Table 40 SRAM address shifting

8.4 EMI configuration

The EMI configuration is held in memory-mapped registers. The function of the registers is to eliminate external decode and timing logic. Each EMI bank has several parameters which can be configured. The parameters define the structure of the external address space and how it is allocated to the four banks and the timing of the strobe edges for the four banks.

Each EMI bank has 64 bits of configuration data which is held in four 16-bit configuration registers In addition there is an EMIConfigLock register for each bank, an EMIConfigStatus register, the EMIDRAMInitialize register and an EMIConfigPadlogic register. For safe configuration, each of the four banks should be configured after reset and then have their configuration locked by writing to the EMIConfigLock register before any access to an external bank is made.

8.5 Default configuration

The default configuration is loaded into all four banks on reset. It should allow the EMI to read data from a slow ROM memory. The following parameters are set.

Parameter	Default value
DataDriveDelay	101 (5 phases)
BusReleaseTime	10 (2 cycles)
CSactive	01 (active during read only)
OEactive	01 (active during read only)
BEactive	00 (inactive)
Portsize	Value of the signal portsize_init
DeviceType	000 (peripheral)
AccessTimeRead	1000 (8+2=10 cycles)
CSe1TimeRead	00 (0 phases)
CSe2TimeRead	00 (0 phases)
OEe1TimeRead	00 (0 phases)
OEe2TimeRead	00 (0 phases)
LatchPoint	00 (end of access cycle)

Table 41 Default configuration

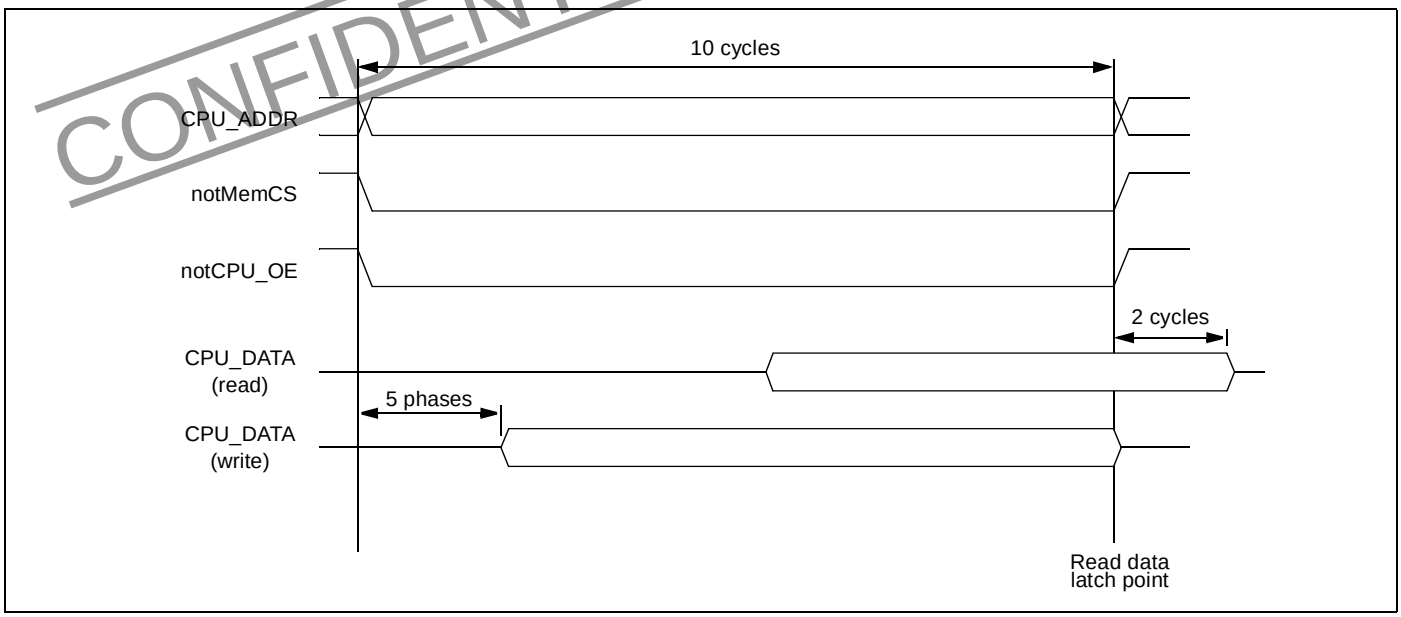


Figure 28 Default configuration for SDRAMModeReg0/1 registers

SDRAMModeReg0/1			
	latency mode	**burst type**	**burst length**
15:7	**6:4**	**3**	**2:0**
000000000	010	0	010

Table 42 Default configuration for SDRAMModeReg0/1 registers

9 System services

The system services module includes all of the necessary logic to initialize the device. Device initialization and debugging can also be done with the diagnostic controller unit (DCU); see the Diagnostic controller on page 83.

9.1 Power-on hard reset

The RESET pin provides a power-on or “hard” reset function. It must be asserted (low) before the clocks and power supply are stable. When the RESET pin is asserted (regardless of any other inputs), all modules are asynchronously forced into their power-on reset state.

The RESET pin should only be de-asserted (high) after both of the following events have taken place:

- the clocks and power are stable, to guarantee well-defined behavior;
- the **notTRST** TAP reset pin has been asserted.

When the RESET pin is de-asserted, the CPU enters its boot sequence. The sequence starts only after the rising edge of the RESET pin is internally synchronized and the clocks are stable.

Bootstrap code can either be in off-chip ROM or can be received through the DCU.

9.2 Bootstrap

The STi5518 can be bootstrapped from the diagnostic controller (DCU), or from ROM.

Booting from the DCU

The STi5518 can be booted from the DCU at any time by setting up the test access port (TAP) to do so. The procedure is explained in the *Diagnostic controller* chapter.

If the device is not set up to boot from DCU then the STi5518 will boot from ROM as soon as it comes out of reset.

Booting from ROM

When not booting from DCU, the **BOOT_FROM_ROM** pin sets the STi5518 to boot from ROM as it comes out of reset.

If the BOOT_FROM_ROM pin is high, boot code is run from a slow external ROM placed in bank 3 at the top of memory. The ROM width is 16-bit wide. When booting from ROM, the value in the configuration registers for the PortSize for bank 3 is disregarded.

When booting from ROM, the STi5518 starts to execute code from the top two bytes in external memory, at address #7FFFFFFE, which should contain a backward jump to a bootstrap program in ROM.

10 Diagnostic controller

The ST20 Diagnostic Controller Unit (DCU) is used to boot the CPU and to control and monitor all of the systems on the chip, via the standard IEEE 1194.1 Test Access Port. The DCU includes on-chip hardware with ICE (In Circuit Emulation) and LSA (Logic State Analyzer) features to facilitate verification and debugging of software running on the on-chip CPU in real time. It is an independent hardware module with a private link from the host to support real-time diagnostics.

10.1 Diagnostic hardware

The on-chip diagnostic controller assists in debugging, while reducing or eliminating the intrusion into the target code space, the CPU utilization, and impact on the application. As shown in Figure 29 , the DCU and TAP provide a means of connecting a diagnostic host to a target board with a suitable JTAG port connector and interface.

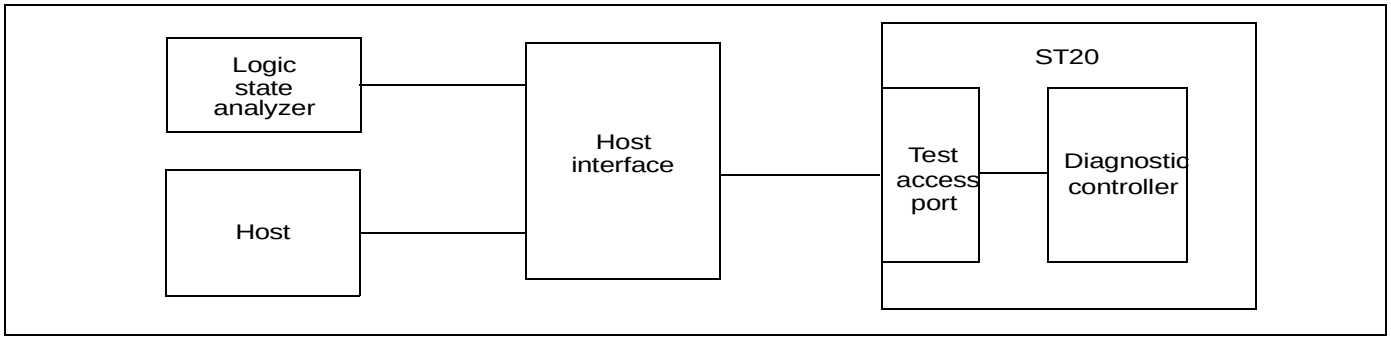


Figure 29 Debugging hardware

The diagnostic controller provides the following facilities for debugging from a host:

- control of target CPU and subsystems including CPU boot;
- hardware breakpoint, watchpoint, datawatch and single instruction step;
- complex trigger sequencing and choice of subsequent actions;
- non-intrusive jump trace and instruction pointer profiling;
- access to the memory of the target while the device is powered up, regardless of the state of the CPU;
- full debugging of ROM code.

When running multi-tasking code on the target, one or more processes can be single-stepped or stopped while others continue running in real time. In this case, the running threads can be interrupted by incoming hardware interrupts, with a low latency.

The host can communicate with the DCU via a private link, using the 5 standard test pins.

Target software also has access to the diagnostic facilities and access through the DCU to the host memory.

A logic state analyzer can be connected to the TRIGGER_IN and TRIGGER_OUT pins. The response to TRIGGER_IN and the events that cause a TRIGGER_OUT signal can be controlled by the host or by target software.

The diagnostic controller provides debugging facilities with much less impact on the software and target performance. In particular it gives:

- non-intrusive attachment to the host system;
- no intrusion into the performance of the CPU or any subsystems;

- no intrusion into the code space, so the application builder does not need to add a debugging kernel;
- no intrusion into any on-chip functional modules, including any communications facilities;
- no functional external connection pins are used.

The connections between the diagnostic controller and other on-chip modules and external hardware may vary between ST20 variants.

10.2 Access features

Access to target memory and peripheral registers from host

Full read and write access to the entire on-chip and external memory space and the register space is available via the TAP. This is independent of the state of the CPU.

Access from target CPU process

The CPU itself can program its own diagnostic controller. Further access may be explicitly prevented by the lock mechanism so that the application being debugged cannot interfere with the breakpoint and watchpoint settings. When the breakpoint or watchpoint match occurs, then the diagnostic controller may release the lock according to settings in the control register.

Access to host memory from target

If the target CPU accesses any address in the top half of the DCU memory space, then these accesses are mapped on to host memory via the TAP as target initiated peek and poke messages. Peek accesses and poke accesses are specifically enabled by separate property bits.

10.3 Software debugging features

Control of the target CPU including boot

Various state information about the target CPU may be monitored and the CPU may be controlled from the diagnostic controller via the TAP. The control of the CPU extends to stalling, forcing a trap and booting.

Non-intrusive IPTR profiling

A copy of the **IPTR** is visible as a read-only register in the diagnostic controller. This register may be read at any time. Reading this register is not intrusive on the CPU or its memory space.

Events

Support is provided by the diagnostic controller to trigger actions when certain predefined events occur.

Event	Action
Breakpoint	The function of the breakpoint is to break before the instruction is executed, but only if it really was going to be executed. A 32-bit comparator is used to compare the breakpoint register against the instruction pointer of the next instruction to be executed. The matched instruction is not executed and the CPU state, including all CPU registers, is defined as at the start of the instruction. The previous instruction is run to completion.
Breakpoint range	The function of a breakpoint range is equivalent to any single breakpoint but where the breakpoint address can be anywhere within a range of addresses bounded by lower and upper register values.

Table 43 Software debugging events

Event	Action
Watchpoint	The function of a watchpoint is to trigger after a memory access is made to an address within the range specified by a pair of 32-bit registers. The CPU pipeline architecture allows for the CPU to continue execution of instructions without necessarily waiting for a write access to complete. So, by the time a watchpoint violation has been detected, the CPU may have executed a number of instructions after the instruction which caused the violation. If the subsequent action is to stall the CPU or to take a hardware trap, then the last instruction executed before the stall or trap may not be the instruction which caused the violation.
Datawatch	The function of a datawatch is to trigger after a data value specified in one 32-bit register is written to a memory word address specified in another 32-bit register. The subsequent action is equivalent to a watchpoint.

Table 43 Software debugging events

Following a watchpoint match, or any other condition detectable by the diagnostic controller, the subsequent action may be programmed to be one of the following:

- stall the CPU, i.e. inhibit further instructions from being executed by the CPU;
- wait until the end of the current instruction, then signal a hardware trap;
- signal an immediate hardware trap;
- continue without intrusion.

In addition, the diagnostic controller may take any combination of the following actions:

- signal on TRIGGER_OUT to a logic state analyzer;
- send a *triggered* message via the TAP to the host;
- unlock access by the target CPU.

Hardware single instruction step

The function of single stepping one CPU instruction is performed by using a breakpoint range over the code to be single stepped. The DCU includes a mechanism to prevent the breakpoint trap handler single-stepping itself. By selecting an inverse range, the effect of single stepping one high level instruction can be achieved.

Jump trace

Jump tracing monitors code jumps, where a jump is any change in execution flow from the stream of consecutive instructions stored in memory. A jump may be caused by a program instruction, an interrupt or a trap.

When the jump occurs, a 32-bit DCU register is loaded with the origin of the jump. This value points to the instruction which would have been executed next if the jump had not occurred. The CPU may not have completed the instruction prior to the change in flow. The diagnostic controller can be set to trace the origin of each jump, the destination, or both.

The DCU copies the details of each jump to a rolling trace buffer in memory. The trace buffer may be located in host memory, but using target memory will have less impact on performance. The tracing facility has two modes:

- Low intrusion. In this mode the DCU uses dead memory cycles to write the trace into the buffer. This means that the CPU is not delayed, but some trace information may be lost.
- Complete trace. In this mode, the CPU is stalled on every jump to ensure the data can be written to the buffer. This means that no trace information is lost, but the CPU performance is affected.

Logic state analyzer (LSA) support

Two signals, TRIGGER_IN and TRIGGER_OUT, are provided to support diagnostics with an external LSA. The action by the DCU on receiving a TRIGGER_IN signal is programmable. The selection of internal events which trigger a TRIGGER_OUT signal is also programmable.

Trigger combinations and sequences

Complex trigger conditions can be programmed. For example:

- the fifth time that breakpoint 3 is encountered;
- enable a watchpoint when a breakpoint occurs.

There is no software intrusion imposed by this mechanism.

10.4 Controlling the diagnostic controller

This section gives a summary of host communications with the diagnostic controller.

The diagnostic controller has direct access to:

- the instruction pointer,
- a selection of CPU state control signals,
- the memory bus,
- memory-mapped peripheral configuration registers.

This access does not depend on the state of the CPU. Access to non-memory-mapped peripheral configuration registers is via the CPU, and for this the CPU must be active and running the appropriate handler.

The host can give two commands to the diagnostic controller: *peek* and *poke*. *Peek* reads memory locations or configuration registers, and *poke* writes to memory locations or configuration registers. The diagnostic controller responds to a *peek* command with a *peeked* message, giving the contents of the peeked addresses.

The diagnostic controller has registers, which are accessed from the host using *peek* and *poke* commands. The registers are used to control breakpoints, watchpoints, datawatch, tracing and other facilities.

The target CPU can also access these registers using the normal load and store instructions, so the target software running on the CPU can program its own diagnostic controller. A lock is provided to prevent CPU access, which can be released by the diagnostic controller when a breakpoint or watchpoint match occurs.

In addition, the target CPU can *peek* and *poke* the host via the diagnostic controller by reading or writing addresses in the top half of the memory space of the diagnostic controller. This facility can be disabled.

Various different types of CPU events can be selected as *trigger events*. When an trigger event occurs, the diagnostic controller can send a *triggered* message.

The four types of message are summarized in the table below. The messages are distinguished by the two least significant bits of the message header byte.

Message type	Direction	Bit 1	Bit 0	Meaning
poke	Command.	0	0	Write to one or more addresses.
peek	Command.	0	1	Read from one or more addresses.
peeked	Opposite to *peek* command.	1	0	The result of a *peek* command.
triggered	DCU to host.	1	1	A trigger event has occurred.

Table 44 Diagnostic controller message types

Messages may be initiated from either the host or the target. Target initiated messages, which constitute asynchronous or unsolicited messages, can be enabled by a property bit.

Messages are composed of a header byte followed by zero or more data bytes, depending on the type of message. The formats for the four message types are shown in Figure 30 .

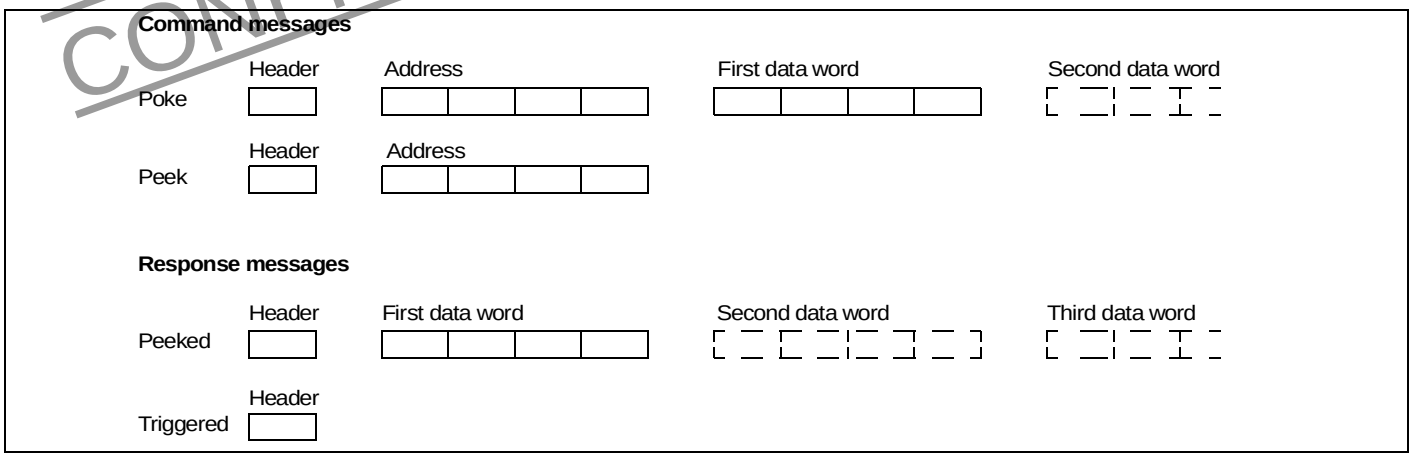


Figure 30 Message formats

10.5 Peeking and poking the host from the target

The target CPU can *peek* and *poke* the host via the diagnostic controller. This is done by reading or writing a single word to a block of addresses within the DCU register block. The DCU will then send a *peek* or *poke* message to the host. After a host *peek*, the target CPU will wait until the host responds with a *peeked* message, which the DCU returns to the CPU as memory read data.

Peeking and poking the host from the target can be enabled or disabled. After reset, these bits are cleared, so peek and poke from the target are disabled.

11 Test access port

The STi5518 Test Access Port (TAP) conforms to IEEE standard 1149.1.

The TAP has pins as listed in Table 45 TDO can be over-driven to the power rails, and TCK can be stopped in either logic state. None of the TAP pins has an internal pull-up.

Pin	In/Out	Function
TDI	in	Test data input
TDO	out	Test data output
TMS	in	Test mode select
TCK	in	Test clock
notTRST	in	Test logic reset

Table 45 STi5518 TAP pins

The instruction register is 5 bits long with no parity. The pattern “00001” is loaded into the register during the *Capture-IR* state.

There are four defined public instructions, outlined inthe table below. All other instruction codes are reserved.

Instruction code[1]					Instruction	Selected register
0	0	0	0	0	EXTEST	Boundary-scan
0	0	0	1	0	IDCODE	Identification
0	0	0	1	1	SAMPLE/PRELOAD	Boundary-Scan
1	1	1	1	1	BYPASS	Bypass

Table 46 Instruction codes

1. MSB ... LSB; LSB closest to TDO.

There are three test data registers; Bypass, Boundary-Scan and Identification. These registers operate according to 1149.1. The operation of the Boundary-Scan register is defined in the BSDL description.

Identification code.

bit 31																												bit 0*
Mask rev				b	ST20 Family			Variant								STMicroelectronics Manufacturers id												c
1				D				4				0				5				0				4				1			
0	0	0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1

Table 47 Identification code

Cut identification

The cut identification register (address 0x00000180) contains the product cut number coded as cut x.y, where decimal x occupies the 4 MSBs as binary coded decimal (BCD) and decimal y occupies the 4 LSBs as BCD.

12 Data flow

This chapter describes the data flow through the STi5518, from the incoming transport stream to the outgoing analog video and PCM audio. It shows how the picture and sound modules are used together. The individual modules are described in the appropriate chapters.

12.1 On-chip modules

The STi5518 reads in an MPEG2 transport stream, demultiplexes it, decodes the audio and video elementary streams and creates a video picture and audio PCM. Demultiplexing extracts the video and audio MPEG streams plus other PES data such as DVB subtitles. Hardware modules are provided on-chip for decoding the MPEG video and audio. The data before decoding is called compressed data (CD), and digital video data after decoding is called pixel data.

The on-chip modules which process the compressed data streams from the incoming transport stream to the decoders are shown below:

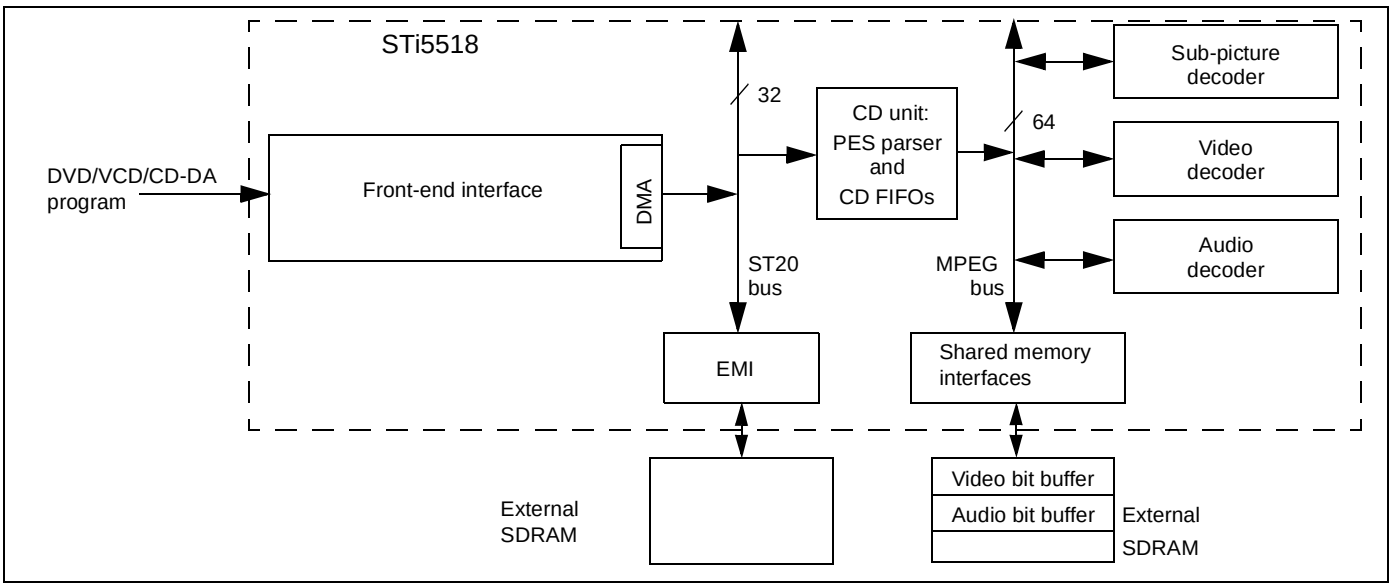


Figure 31 Compressed data modules

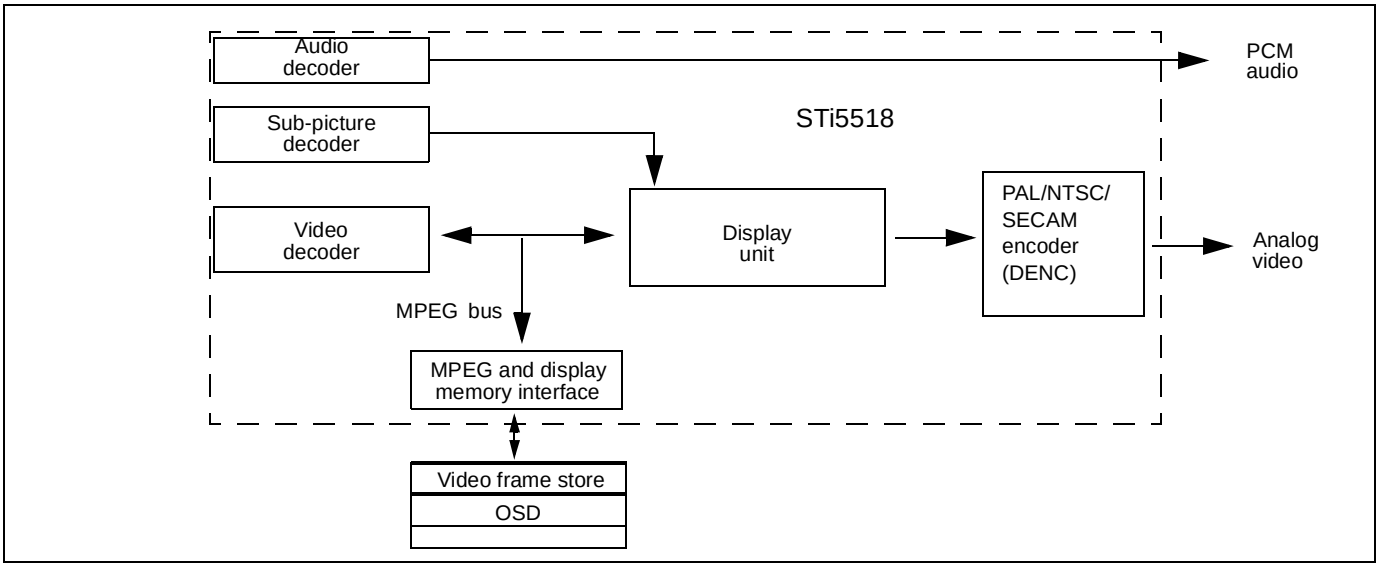


Figure 32 Decoded data modules

12.2 Video data flow

The data flow for MPEG-2 video streams is illustrated below.

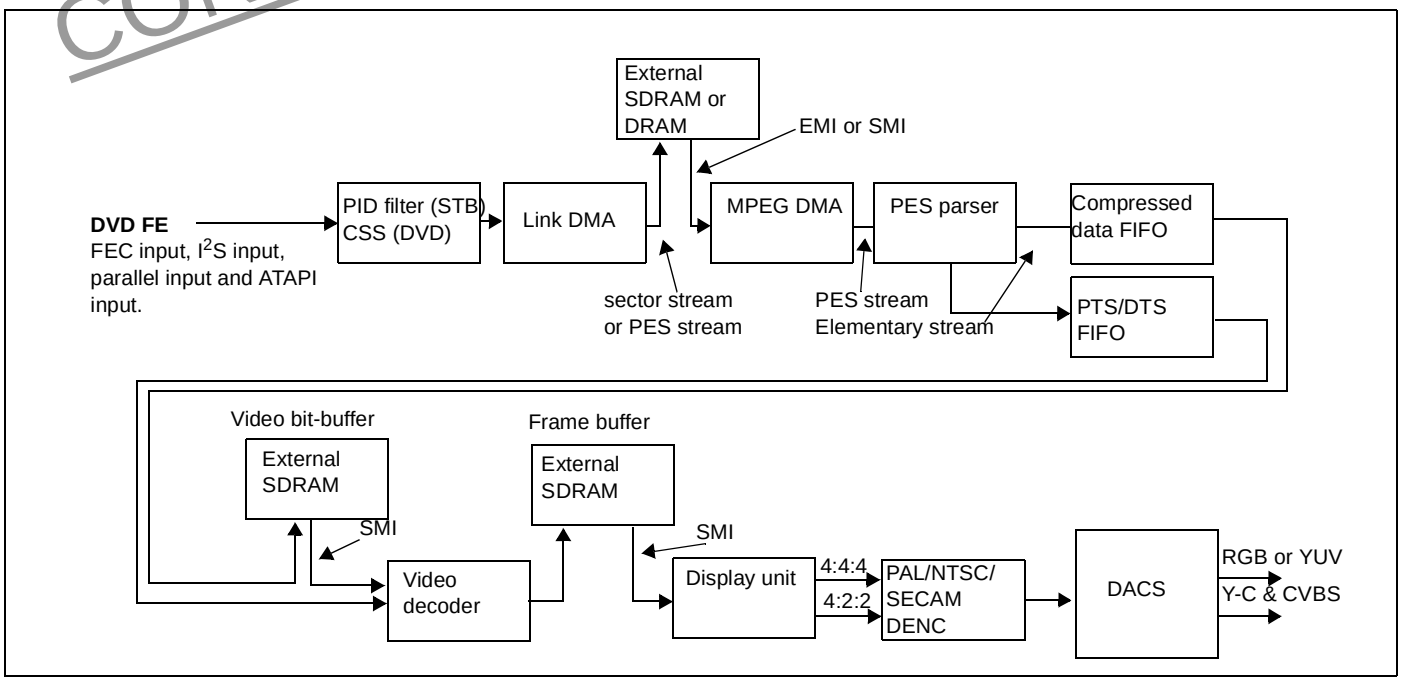


Figure 33 Video data flow

DVD, VCD, and CDDA data enters the STi5518 through the I^2S, parallel, FEC or ATAPI input. These data formats are described in Front-end interface on page 92. DVD data are descrambled by the CSS decryption module.

DVD, VCD, and CDDA data are transfered by DMA (see Link on page 105) to the track buffer which can be on the programmable CPU interface or the shared memory interface. The data in the track buffer is demultiplexed by software into seperate componatnts (video, audio, subpicture/OGT, navigation).

Video data enters the CD unit through the PES parser, which passes the data to the video CD FIFO. The video CD FIFO holds 128 bytes and writes 512-bit bursts into the video bit buffer in the SDRAM on the shared meory interface.

The video decoder (described in MPEG video decoder on page 133) reads 1024-bit bursts from the video bit-buffer. It decodes the compressed bit stream and produces a pixel stream. I-frames and P-frames and B-frames are written into video frame stores. Different programmable pointers allow enhanced trick modes.

The display unit (described in Display planes on page 156) converts the blocks of pixels into rows, and performs filtering (zoom-in, zoom_out...) and pan/scan. It then mixes the video with the other display planes and sends two pixel streams to the on-chip PAL/NTSC/SECAM encoder (DENC). One pixel stream is in 4:4:4 format and is generally used for TV display, while the other is in 4:2:2 format, and is generally used for output to a VCR.

The DENC converts the pixel streams into analog signals for output from the device. The 4:4:4 pixel stream is converted into YUV and RGB signals, and the 4:2:2 pixel stream is converted into CVBS and YC signals.

12.3 Audio data flow

The data flow for audio streams is illustrated in the figure below.

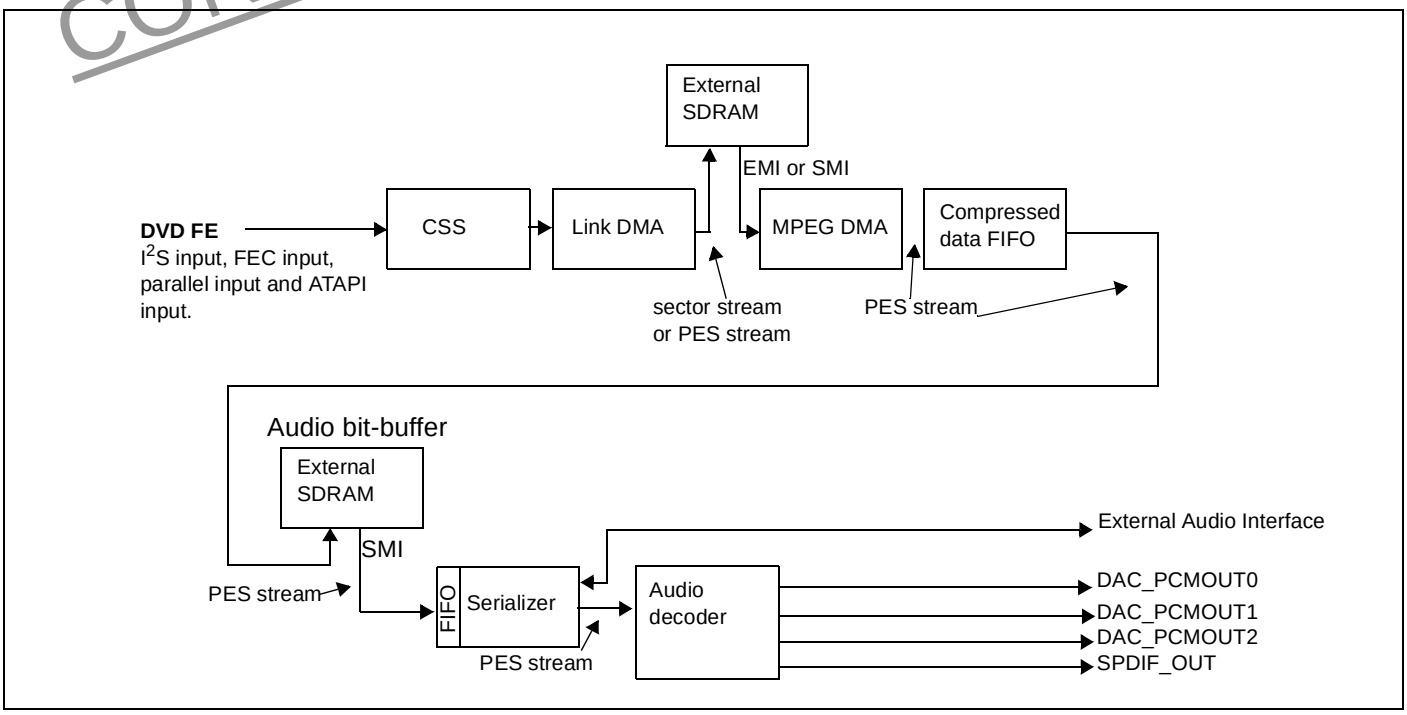


Figure 34 Audio data flow

Up to the track buffer, the audio data follows the same path as the video data flow described above.

Audio data are sent to the audio CD FIFO and do not pass through the PES parser in the CD unit.

The audio CD FIFO writes 512-bit bursts into the audio bit buffer in external SDRAM on the shared memory interface. The CD unit is described in MPEG video decoder on page 133.

The audio decoder (described in Audio decoder on page 219) unit includes its own FIFO and PES parser. It transfers data from the audio bit-buffer into its FIFO. The data may be either passed to the audio DSP or to an external audio interface for transfer to an external audio decoder (see External audio decoder interface on page 241).

13 Front-end interface

13.1 Introduction

The figure below illustrates the front-end interface (FEI) architecture. For STB applications, data enters through the FEC interface.

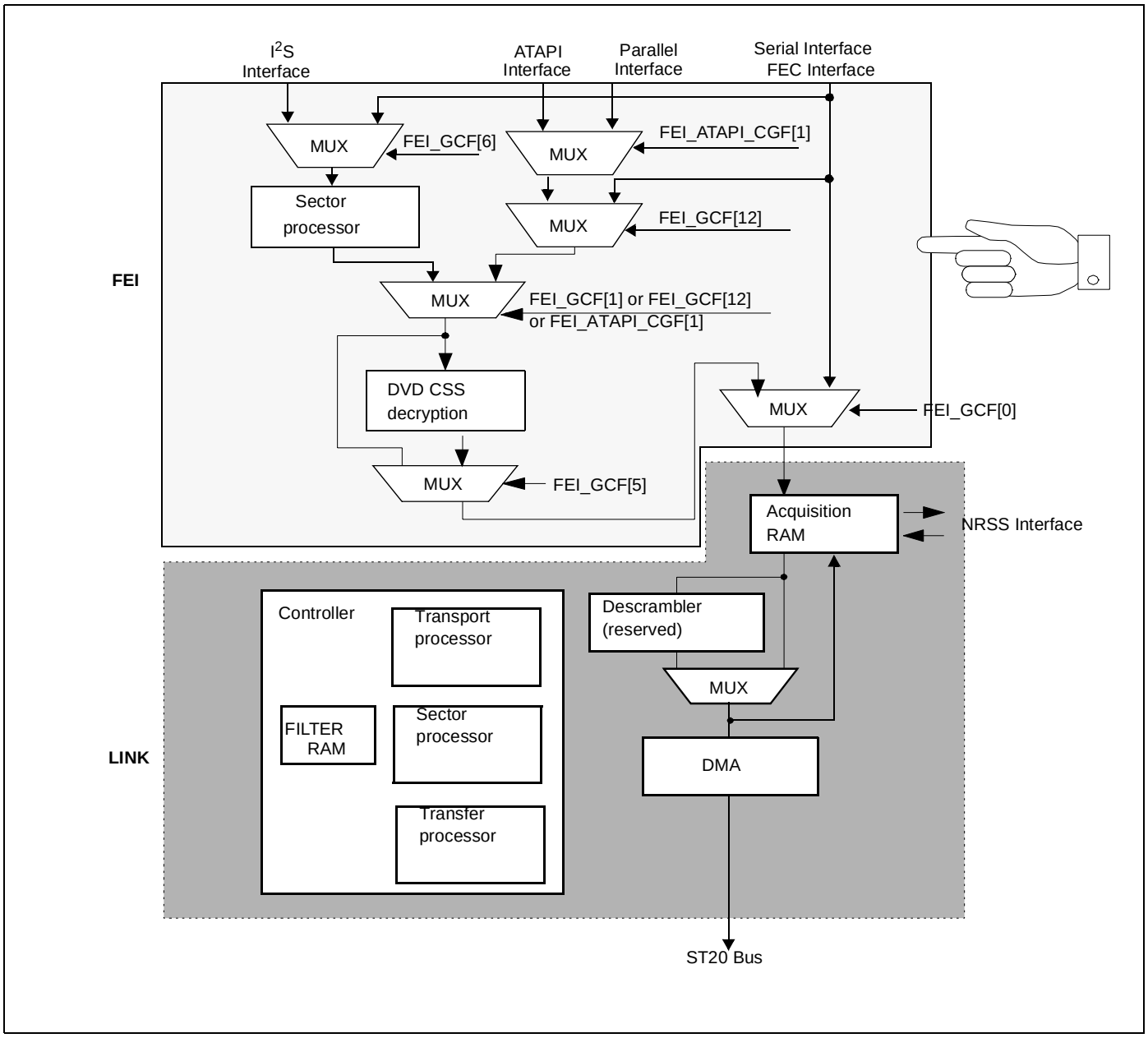


Figure 35 Link architecture

13.2 Serial interface

The DVD front-end serial interface supports many different formats and can be used for DVD, VCD and CD-DA applications. For DVD applications, sectors of 2048, 2064, 2066... bytes are accepted. For VCD and CD-DA applications, subcodes are multiplexed with data, and the CPU software demultiplexes the data. In serial mode, the STi5518 signals are renamed as in the table below.

Signal name	Signal name in serial mode	Pin number	Type
B_DATA	FEC_DATA	16	I
B_BCLK	FEC_B_CLK	17	I
B_FLAG	FEC_D_VALID (DVD) FEC_P_CLOCK (DVB/DSS)	18	I
B_SYNC	FEC_P_START (DVD) FEC_ERROR (DVB/DSS)	19	I

Table 48 Serial mode signal names

Signal FEC_DATA corresponds to the serial stream data and signal FEC_B_CLK corresponds to the serial stream clock. FEC_B_CLK can be up to 59.5Mbits, or 60Mbits if the clock is synchronized with the ST20 clock.

The FEC_D_VALID signal is active during a burst transmission (data are supposed to be transferred in a burst of at least 8 bits long). There can be gap of one or more clock cycles between bytes, but there can be no gaps within the 8 bits of a byte. When the FEC_P_START signal is active during the first bit of the first byte of a packet, the serial front-end interface detects the beginning of a packet. The rising edge is detected on the FEC_P_START pin and the internal FIFO counter is reset.

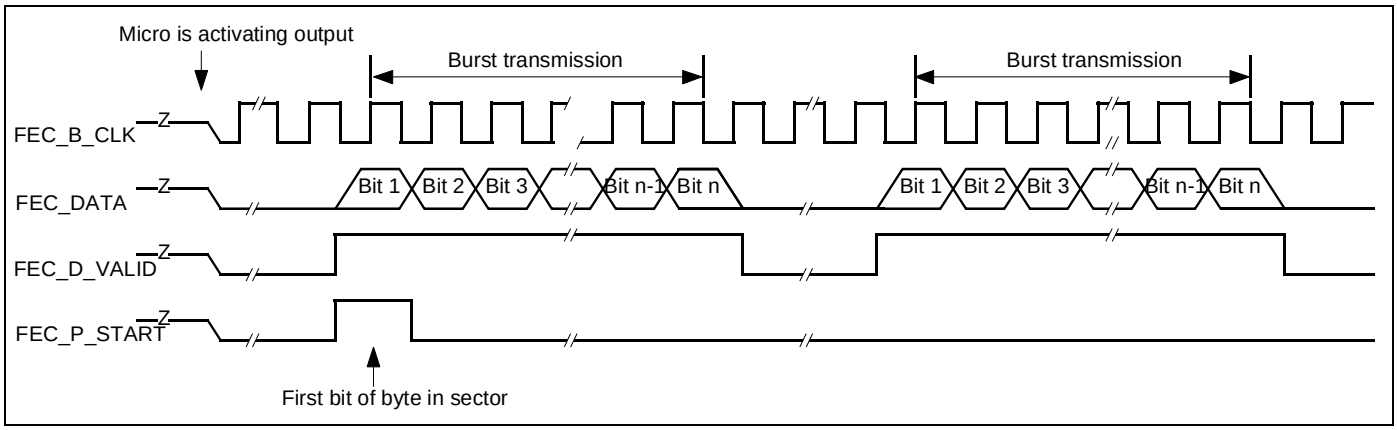


Figure 36 DVD serial interface

When accessing the link through the FEI, the minimum delay between 2 consectutive sectors is ten serial interface clock cycles.

13.3 DVB-CI mode (optional)

This option is intended for set-top box applications using the digital video broadcast common interface (DVB-CI). All DVD modes are removed on versions with this option.

The DVB-CI mode uses the three FEC interface control pins (17, 18 and 19) , and the PIO3 pins for the transport data:

Pin number	Pin name	Function	Pin type
6-13	PIO3[0:7]	MDI[0:7]	Input
17	B_BCLK	MICLK	Input
18	B_FLAG	MISTRT	Input
19	B_SYNC	MIVAL	Input

Table 49 Pin functions for the DVB-CI mode

- MICLK is the data clock, as defined in the DVB-CI specifications (EN 50221).
- The data MDI[7:0] are clocked out of the STV700/1 on the falling edge of MICLK. The STi5518 samples this data on the rising edge of MICLK in order to comply with the setup and hold times specified in the DVB-CI specifications.
- MISTRT is valid only during the first byte of the input transport packet, that is, when MDI[7:0]=0x47.
- MIVAL indicates valid data bytes on MDI[7:0]. MIVAL may be either at logic 1 for the duration of the transport packet (burst valid data), or go to logic 0 at any time to indicate data bytes which must be ignored.

To select the front-end mode, you have to program bit 0 of the FEI configuration register, that is FEI_GCF[0]. All the other bits of FEI_GCF are related to DVD modes and have no effect on this mode:

FEI_GCF[0] = 0 DVB-CI compliant mode

FEI_GCF[0] = 1 FEC mode.

The diagram below shows how the different devices are connected together. You can insert either one DVB-CI module using STV701, or two DVB-CI modules using STV700. These ICs are inserted between front-end and back-end products, such as the STV399 and the STi5518.

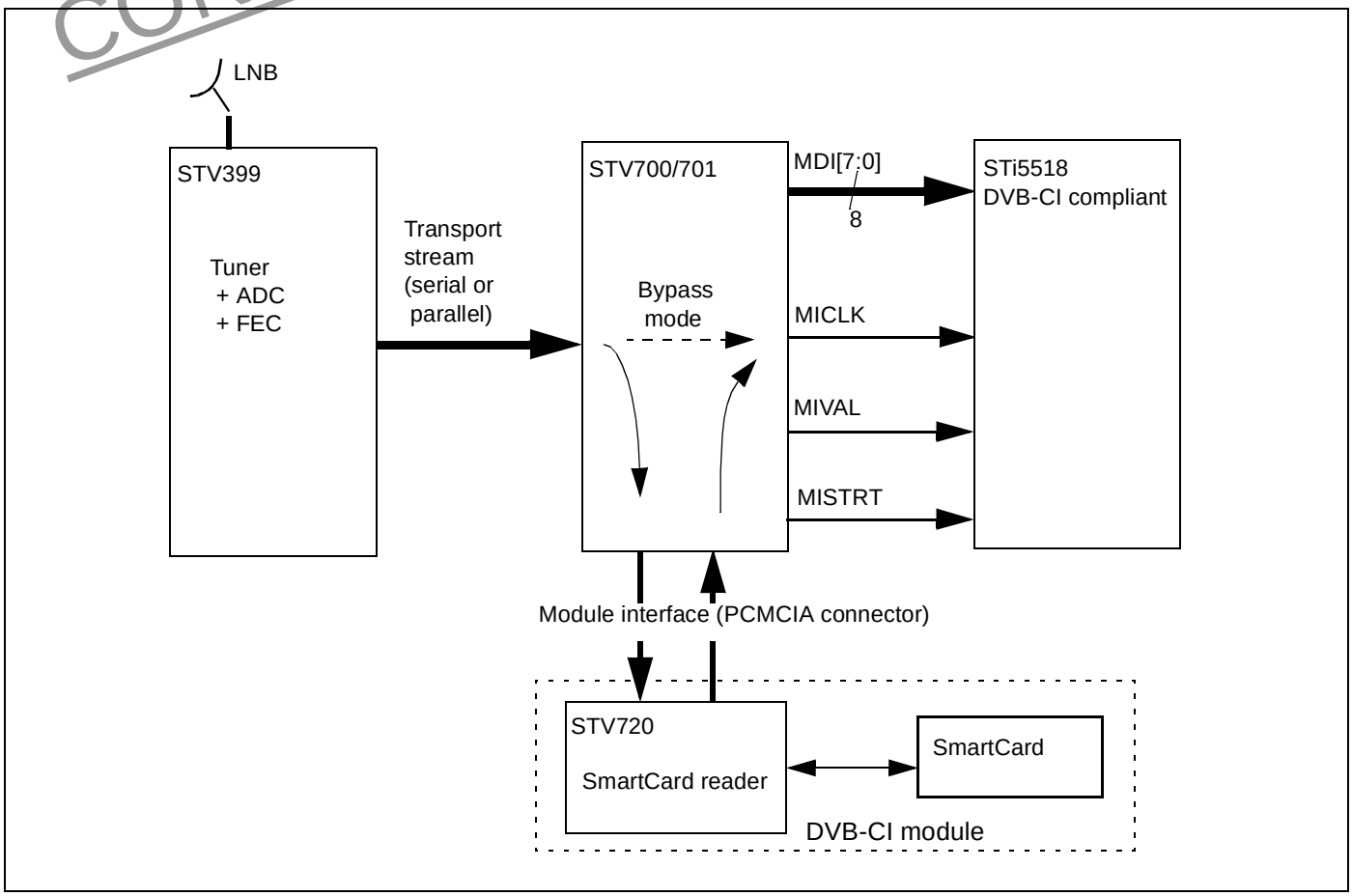


Figure 37 : Satellite set-top box transport flow diagram using the DVB-CI

13.4 Parallel interface

The front-end parallel interface is a generic interface that can be used to input data directly to the decryption cell.This interface has the pins described in the table below:

Pins	Name	Type	Function
13 - 6 (PIO3[7:0])	PARA_DATA[7:0]	I	Parallel input data bus
206 (PIO2[2])	PARA_STR	I	Parallel input strobe
205 (PIO2[1])	PARA_REQ	O	Parallel output request
201 (PIO1[5])	PARA_SYNC	I	Parallel synchronization
196 (PIO1[2])	PARA_DVALID	I	Parallel input data valid

Table 50 Parallel interface pins

In order to ensure data continuity in the serializer (60Mbit/s) at the input to the DMA, there is an elastic buffer behind the asynchronous parallel interface. The handshake over the interface is controlled by the output signal PARA_REQ, which is, in effect, the "empty signal" for the 2-byte buffer. It goes active when the buffer requires more data. Figure 38, below, illustrates typical waveforms. The operation of the interface is as follows:

When PARA_REQ is active then the data on the interface must be written to the buffer. PARA_REQ then goes inactive since the buffer is no longer empty. However, the buffer is not yet full since it is 2 bytes long; hence, a second byte can be written if desired. When the buffer needs more data PARA_REQ will again go active.

In terms of FEI clock cycles (of 60 MHz), PARA_REQ goes active after 8 cycles if 1 byte is written, and after 16 cycles if 2 bytes are written. When the last byte of a sector is written PARA_REQ will stay inactive for an additional 10 cycles (that is, 10+8 or 10+16 cycles). Furthermore, following an encrypted sector, PARA_REQ stays inactive for an additional 10 cycles (that is, 10+8 or 10+16 cycles) after the first byte of the new sector is written.

The DVD data entering the parallel interface are 2048-byte sectors (if the DVD header has been stripped by the front-end) or 2064-byte sectors (the sector length is defined in FEI_SLG register).

The data latching signal is selected by register FEI_GCF[13] as the rising edge (for non-inverted polarities) of either PARA_STR, Figure 38, or (PARA_DVALID & PARA_STR), Figure 39. The polarities of the signals PARA_STR and PARA_REQ are programmable by register FEI_GCF[3,4]. They must be set up before the parallel interface is selected (FEI_GCF[1]). With inverted polarities, the clocking edge of PARA_STR is the falling edge.

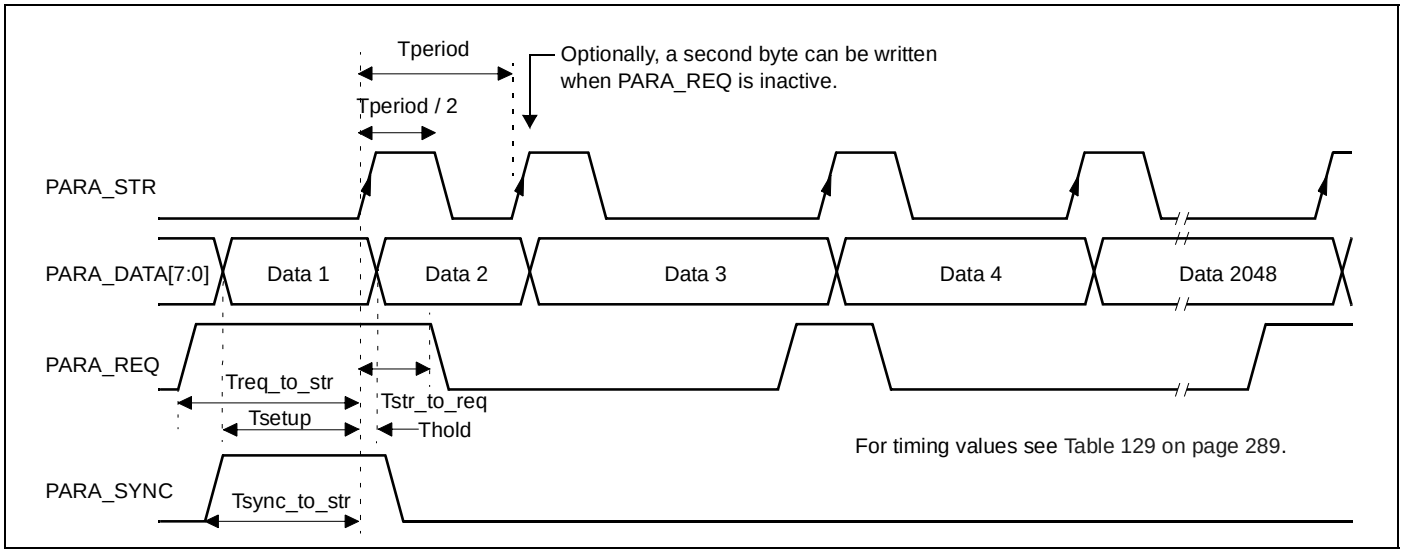


Figure 38 Generic parallel interface waveforms (PARA_DVALID not used)

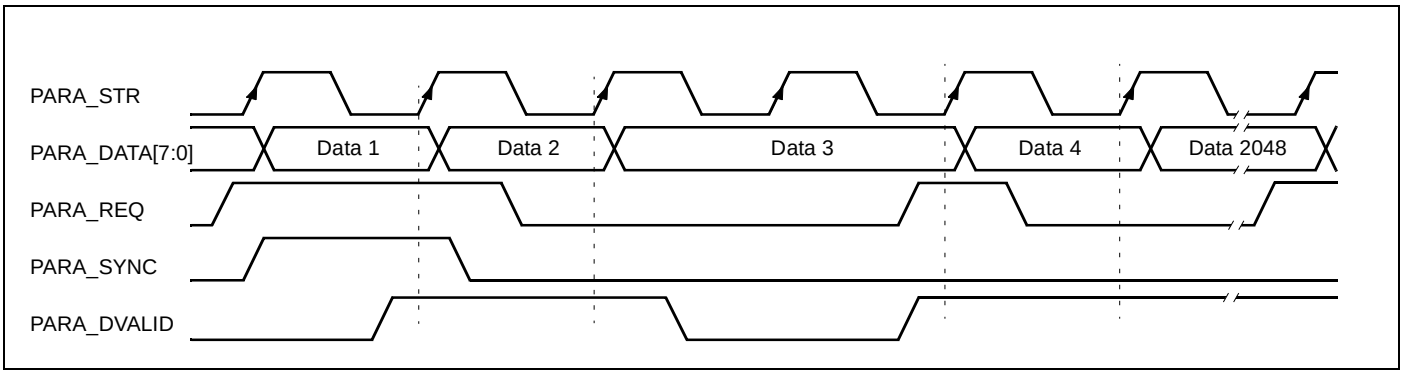


Figure 39 Generic parallel interface waveforms (PARA_DVALID activated)

The parallel interface is automatically configured when FEI_GCF[1] is set. However, since all the parallel interface pins have alternate functions, they must be separately set to the correct I/O type. That is, each one must be set to 'input' except PARA_REQ which must be set to 'output'. The I/O type is set by registers PIO_PnC2:0.

The "sector start" signal can be either an external asynchronous input on pin PARA_SYNC or an internally generated signal using register FEI_SLG. The setup is via register FEI_GCF[10,11].

The PARA_SYNC signal may be used for example to process DVD shortened sectors (that is, sectors that have less than 2048 data bytes due to, for example, front-end disturbances which cause loss of sync). Shortened sectors are completed to 2048 bytes by adding extra bytes after the next sync, that is, at the beginning of the following sector. This following sector is then ignored.

The maximum width of PARA_SYNC is unlimited but it must be asserted 20ns before the first byte of the sector is written. The minimum width of PARA_SYNC is 20ns. The PARA_SYNC signal must not be active if PARA_REQ is inactive.

13.5 ATAPI interface

Introduction

The ATA interface (commonly known as ATAPI) is mainly used for mass storage peripherals in desktop computing. However, some consumer cost-effective DVD drives use the ATAPI interface. The STi5518 interfaces gluelessly to these drives.

ATAPI drives are memory mapped devices. A set of registers and an interrupt, control the drive. Data can be transferred in the following two ways:

- programmed input/output (PIO), a memory mapped data register. This is also used to describe one form of data transfers;
- DMA transfer.

The ATAPI interface is accessed through bank 1 of the CPU programmable interface.

Connecting to the ATAPI drive

The ATAPI drive uses the STi5518 programmable CPU interface for register IO, and block move DMA to move the data from the ATAPI to the decryption unit. Only the PIO modes up to Mode 2 (Read and Write) are supported by the decryption unit. The databus width is 16-bits, and the data is connected to the STi5518 front-end interface internally. The ST20 programmable CPU interface (otherwise called EMI) must be programmed according to the DVD drive speed. The figure below shows how the ATAPI interface should be connected to the STi5518.

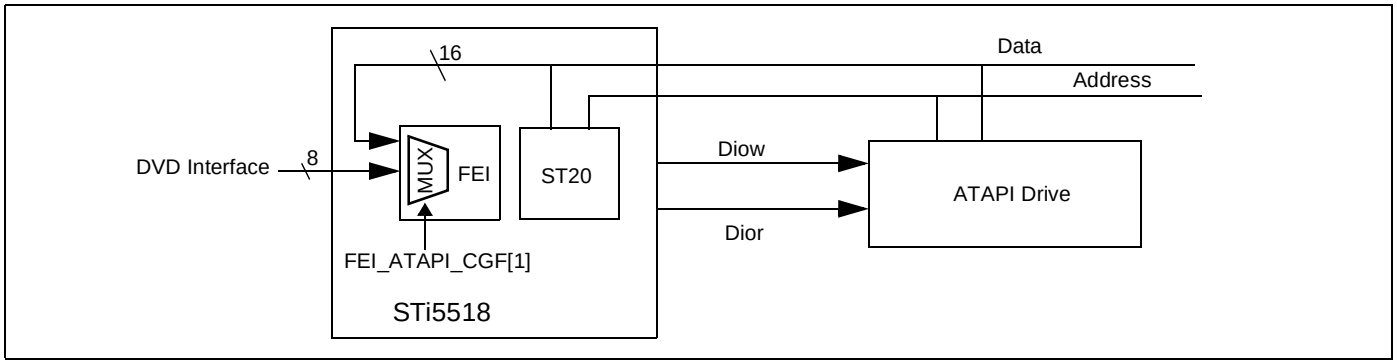


Figure 40 Connection of an ATAPI drive to the STi5518

Operating the ATAPI interface

The following signals are used to operate the ATAPI interface in PIO mode:

- DIOW (Device register write), write strobe signal ATAPI_WR
- DIOR (Device register read), read strobe signal ATAPI_RD
- ATAPI enable (interface enable), notCE[1]

- IRQ ATAPI, IRQ1
- 3 address pins, CPU_ADR[16:18]
- 16 data pins, CPU_DATA[0:15]
- 2 chip selects, CPU_ADR[19:20]

The following figure shows the block diagram of the ATAPI interface

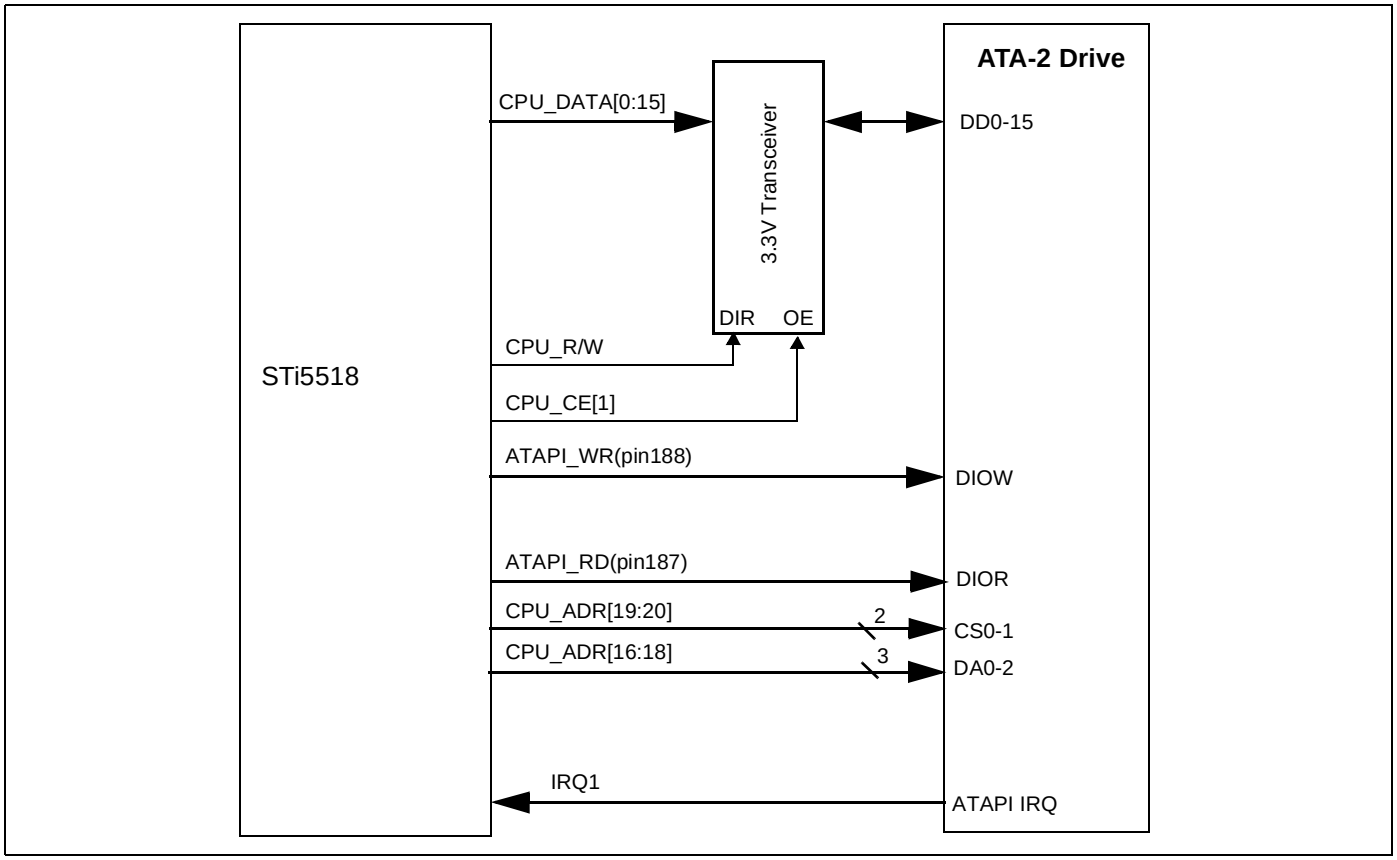


Figure 41 ATAPI interface block diagram

Authentication

The ATAPI drive requires authentication as a legal DVD drive. The authentication process carried out by both hardware and software. The hardware part in the CSS decryption block is accessed via register reads and writes from the ST20.

PIO data transfer

The EMI is programmed by the ST20 to the PIO transfer speed that is supported by the drive. The ATAPI drives (DVD drive and HDD) support different transfer speeds. Speeds up to and including Mode 4 are supported by the STi5518 if the CSS decryption unit is not used.

13.6 I^2S interface

Introduction

The STi5518 supports DVD drives with an I^2S interface. A hardware sector processor is associated to the interface. Formats accepted on this input are given in the sector processor (SP) and are described below.

Sector processor

The sector processor (SP) only accepts data from the serial interface. The activities of the sector processor are:

- Sector capturing (DVD, VCD, CDDA)
- Subcode capturing (VCD, CDDA)
- Sector filtering (DVD)
- Sector header bytes strip off (DVD, VCD)
- Flywheel subcode error correction
- Serial to parallel conversion for data and subcodes
- Selectable error strategy for received sectors
- Indication of navigation pack reception (DVD)

The main function of the sector processor is to allow the user to filter and capture groups of contiguous DVD sectors by programming the first and last sector numbers of a group.

This mechanism is used to ensure only the DVD required sectors reach the track buffer. In normal, the use of the sector processor implies having the track buffer in STi5518 memory space (memory savings).

The sector header information is also stripped and stored. It can be read by the microprocessor via a number of user registers (useful information for decryption).

The resulting sector payload is then sent to the DMA engine via the decryption engine.

The sector processor also indicates via an interrupt when a navigation pack is received in the sector stream.

Using these capture and filter functions, the overspeed processing function in DVD can be realized.

In CD-DA mode the sector processor uses the subcode information it receives to “sectorize” the data thus allowing overspeed processing to be handled for these backward compatible modes.

Since there is no general configuration register the user must configure each of the three modes DVD, VCD, CD-DA separately.

In DVD mode the sector processor expects a sectorized serial bit stream according to the DVD format (Figure 42 and Figure 43). The B_SYNC signal indicates the start of each sector.

Erroneous sectors are flagged by the B_FLAG signal at the last byte of the sector (CRC check is done by the front end).

Whenever a captured sector is flagged erroneous, an interrupt is given to the ST20.

Depending on the error strategy (SE_EMR_ERRO Mode Register) the following will happen:

1 The last stored sector is removed and the sector processor waits for a new arrival of this erroneous sector (the ST-20 is expected to issue a seek command).

2 The last stored sector is not removed and capturing continues with the next sector.

Start/stop capturing is checked on a 3 bytes sector address in the sector address. The 2048 bytes of user data are stored after decryption in the track buffer through DMA. When a navigation pack enters the sector processor, it is signaled to the ST-20 by means of an interrupt (NPRENAV).

In VCD mode the sector processor receives a sectorized serial bit-stream according to the CDROM-XA format. In this mode the B_SYNC signal is not valid as a sector sync. Sector synchronization is obtained by detecting the 12 bytes CDROM sector sync (00 FF FF FF FF FF FF FF FF FF FF 00) in the I^2S data. Before interpreting, the sector data is

descrambled. Erroneous bytes are flagged by the B_FLAG signal. Whenever a captured sector contains erroneous bytes, an interrupt (RDERR) is given to the ST-20 and the behavior will be as described above in DVD mode.

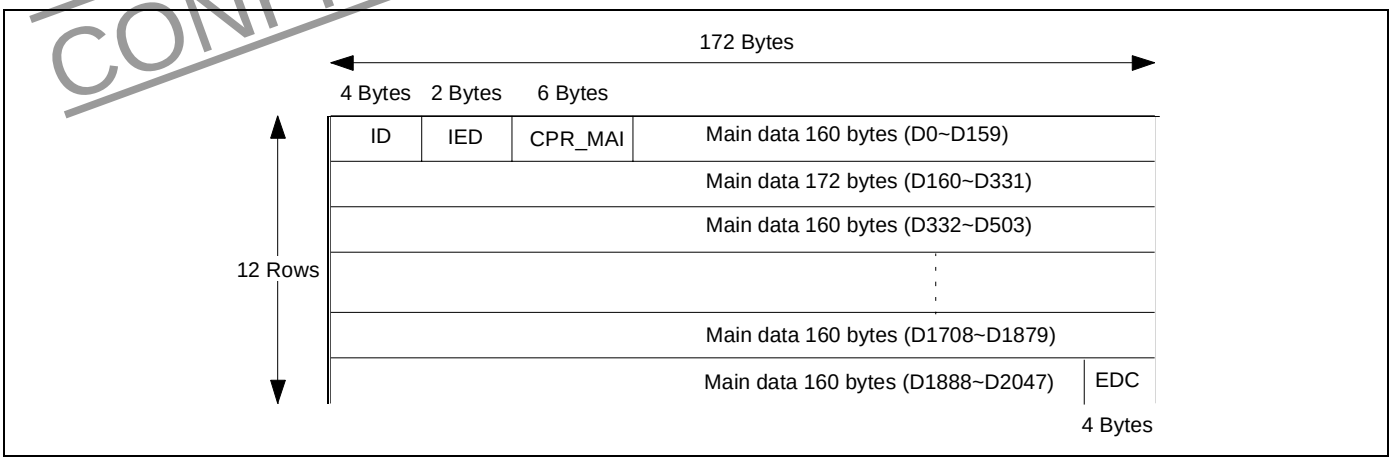


Figure 42 DVD data sector structure

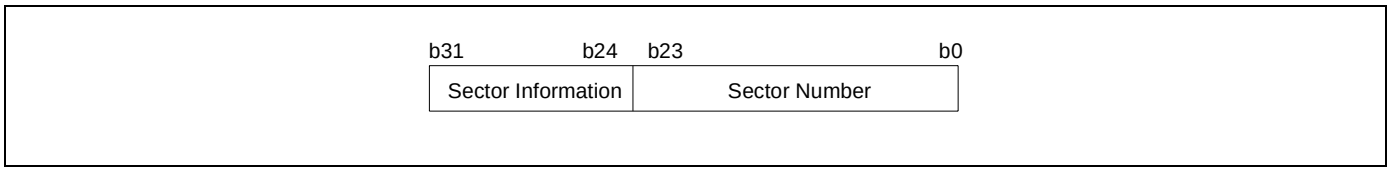


Figure 43 DVD data sector identifier

After descrambling, start/stop is checked on a 3 byte sector address in MSF (Minutes Seconds Frames) format in the Header (see Figure 46).

From the real-time sectors, only the sectors containing video and audio (indicated by bits A set or V set of the sub mode byte in the sub header) are stored into the track buffer (see Figure 47).

Also interrupts are given if certain types of sectors enter the sector processor (Sub header-sub mode byte, bits EOF, EOR or T set).

Since the sub header is present twice in a sector, some error strategy is implemented. If one of the 2 is erroneous, the error flag is suppressed and the right one is taken. If both are erroneous a RDERR interrupt will be given.

Set the SE_MOD register to capture static (file system) sectors (mode 2 form 1). The complete sector, excluding sector sync is stored into the track buffer. Thus enabling the ST-20 to carry out error correction if needed. Error correction is not done by the sector processor. In the case of real-time data (mode 2 form 2), only sectors which contain audio and video data (indicated by the submode bytes) are sent to the DMA engine (see Figure 47).

In CDDA mode the sector processor receives a serial bitstream of audio samples.

Start/stop and overspeed control is done by means of the absolute time coded within Q-channel of subcode.

The subcode has a fixed relation to the serial bitstream of audio samples enabling the sector processor to split up the CD-DA data stream in “sectors” of 2352 bytes.The sector processor uses a flywheel for absolute time, which is set according incoming subcode, but increments if the subcode is erroneous (CRC check).

This allows to start/stop capturing, even if the subcode of the start/stop sector is erroneous.The subcode is available on pin B_V4.

As with VCD the subcode information is stored into the sub code buffer.

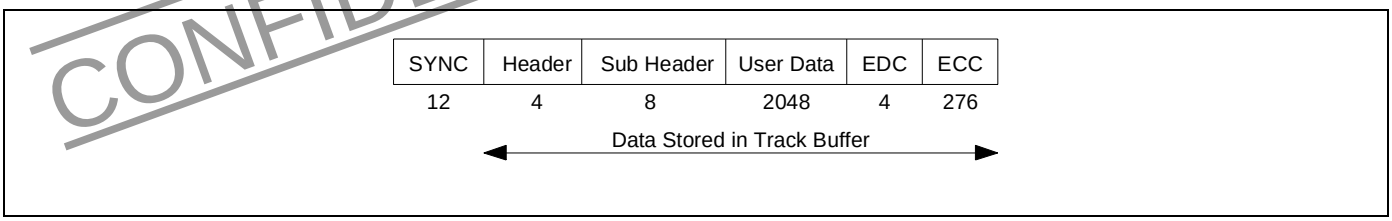


Figure 44 VCD sector format (mode 2 form 1)

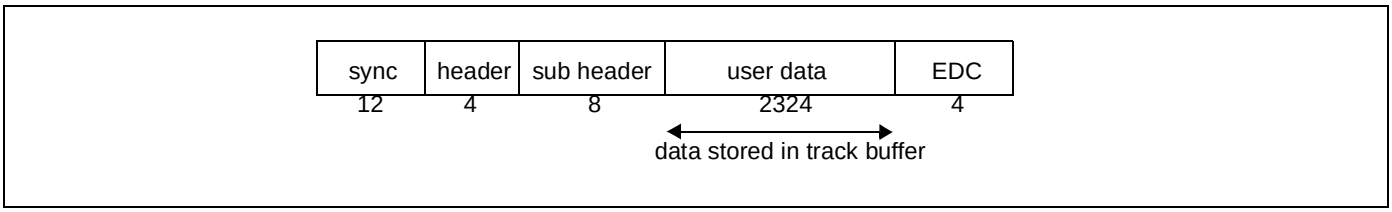


Figure 45 VCD sector format (mode 2 form 2)

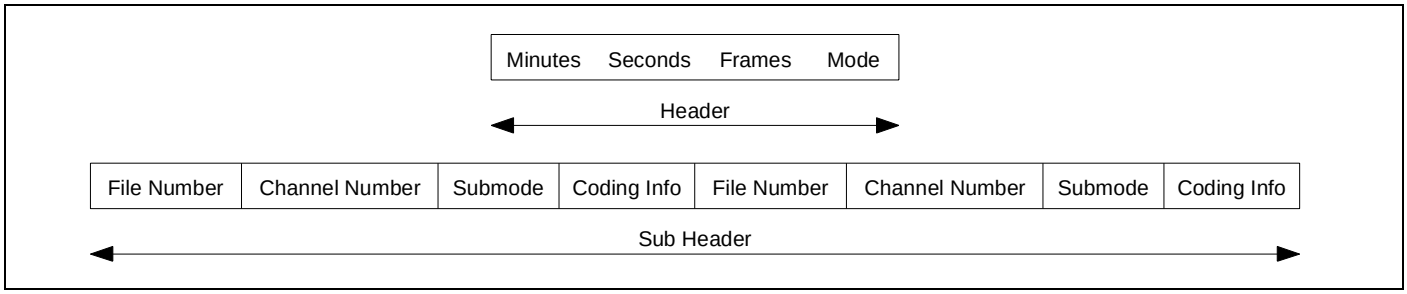


Figure 46 VCD header and sub header format

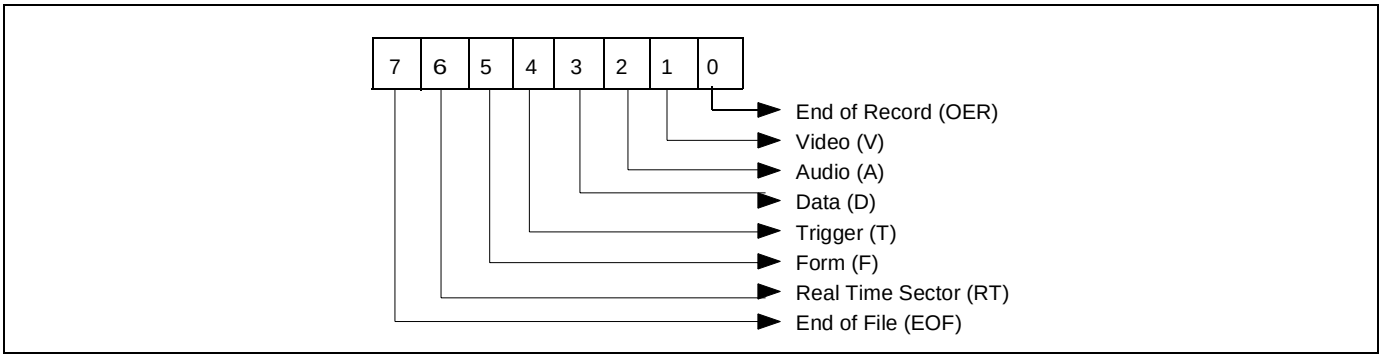


Figure 47 VCD submode byte format

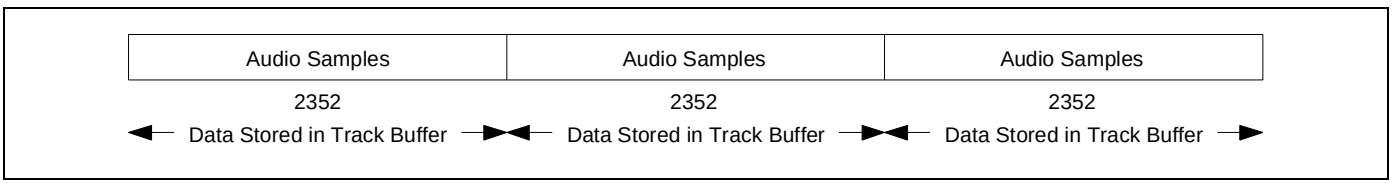


Figure 48 CD-DA sector format

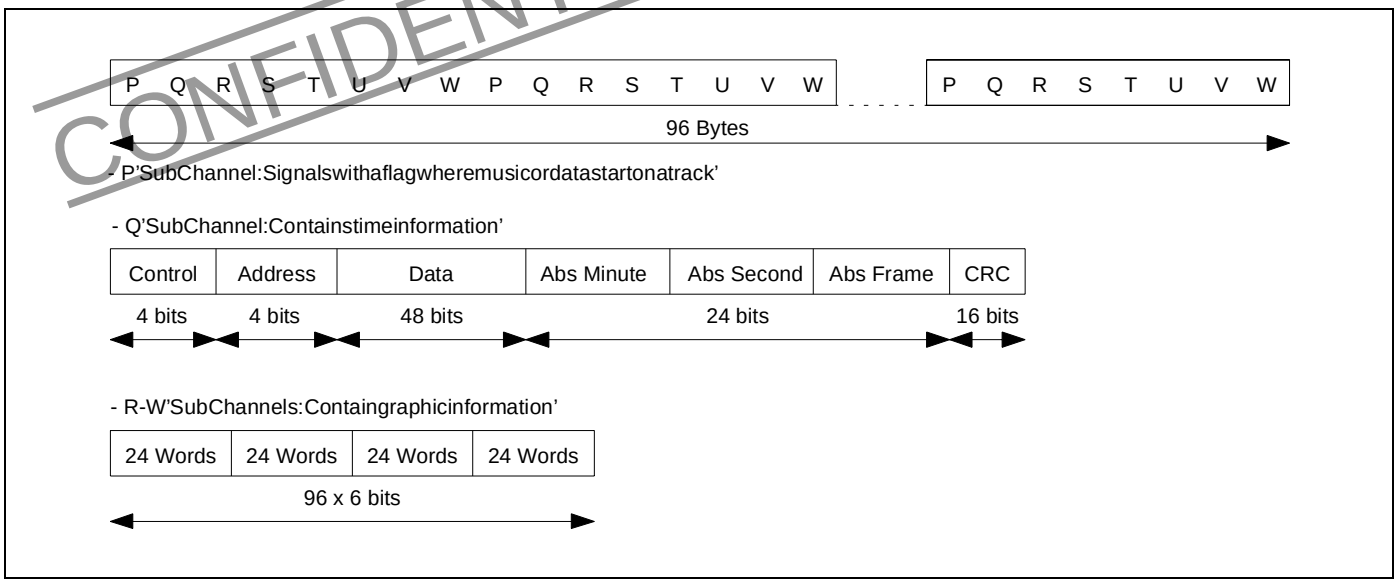


Figure 49 CD-DA subcode format

V4 interface

The V4 interface is used in CD-DA and VCD modes to transfer subcode information. The format on B_V4 pin is similar to the RS232.

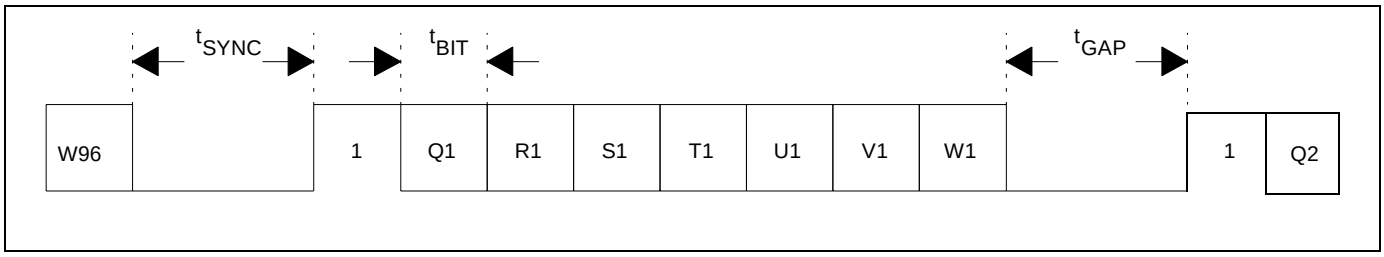


Figure 50 Subcode format and timing at b_v4 pin

Signals

The table below details I^2S interface pins.

Pin N°	Name	Type	Function
16	B_DATA	I	I^2S Data
17	B_BCLK	I	I^2S Bit Clock
18	B_FLAG	I	Error Flag
19	B_SYNC	I	Sector Sync/Abs Time Sync
20	B_WCLK	I	I^2S Word Clock

Table 51 I^2S interface pins

The inter-IC sound (I^2S) bus was initially a serial link for digital audio. It has been extended to VCD and DVD.

The following four figures illustrate the different I2S modes supported.

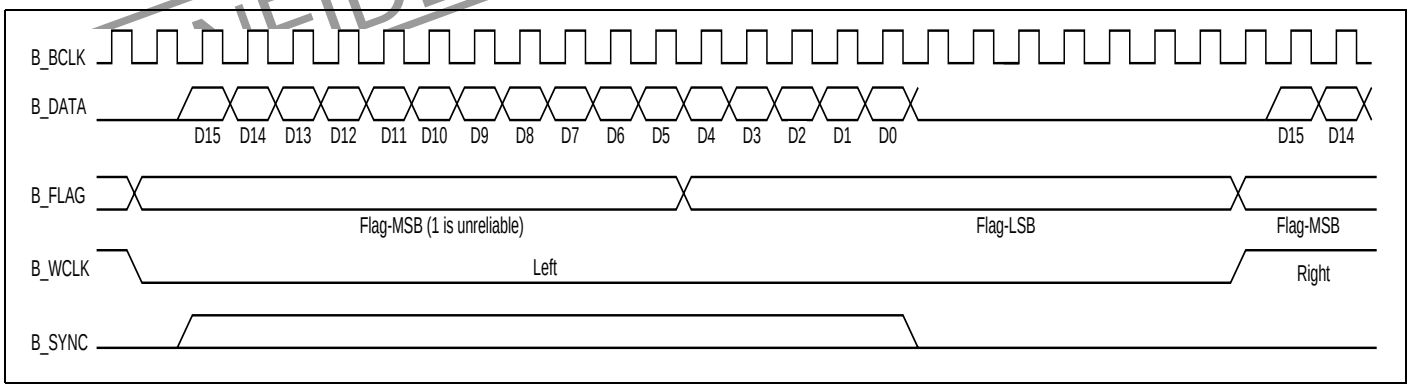


Figure 51 I²S bus data format (16-bit word length)

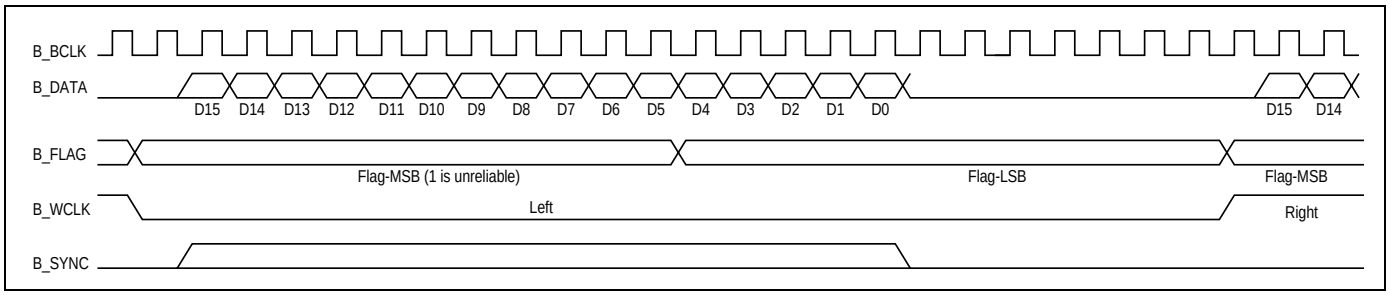


Figure 52 I²S bus data format (24-bit word length)

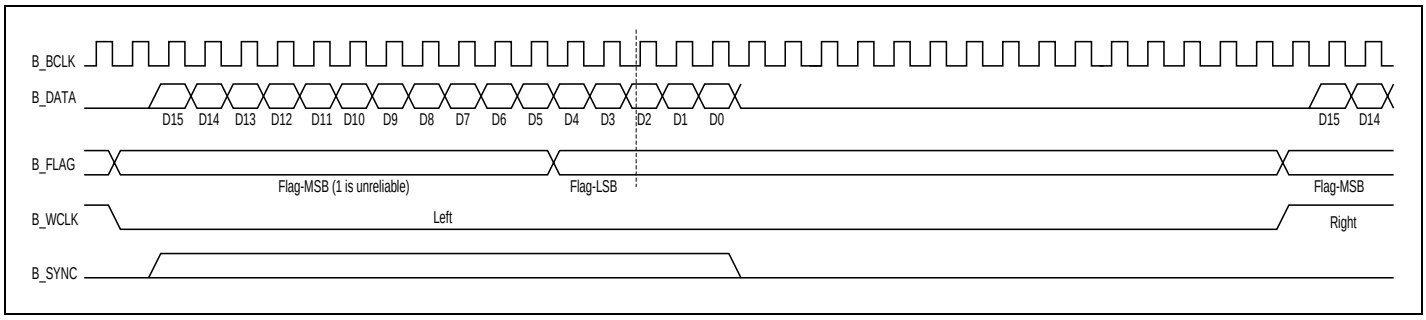


Figure 53 I²S bus data format (32-bit word length)

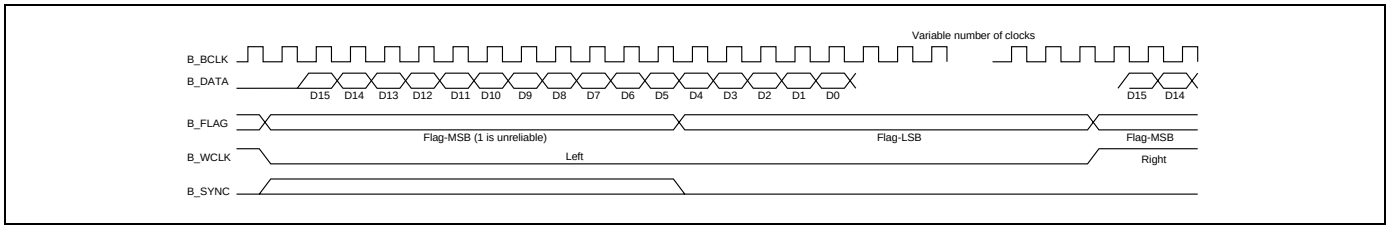


Figure 54 I²S bus data format (variable word length)

13.7 Decryption cell

The decryption cell provides all the necessary logic to decrypt DVD data as well as perform the transformations required for authentication. All necessary keys are internally coded and unreadable externally.

Data decryption is performed by the cell after the ST20 has read, from the sector header information registers (in the sector processor) the title keys and written them into the decryption cell via the key load register. After a hard reset, the decryption is in reset mode (FEI_GCF[7] is set).

The decryption cell is able to handle automatically encrypted or non-encrypted sectors.

Before initialization, the decryption needs to be bypassed (FEI_GCF[5]).

14 Link

14.1 Introduction

The link interface accepts a constrained DVB or DSS transport stream input and extracts a Packet Elementary Streams (PES) for decode and play. Section streams are extracted from the bitstream and stored in buffers for use by the decoder control unit.

A high-speed SDAV (Simplified Digital Audio Video) interface transfers transport packets between the STi5518 and external units for recording or playback. This interface supports an external P1394 link layer controller.

The link includes a National Renewable Security System (NRSS) interface for external descrambling. The figure below illustrates the link architecture.

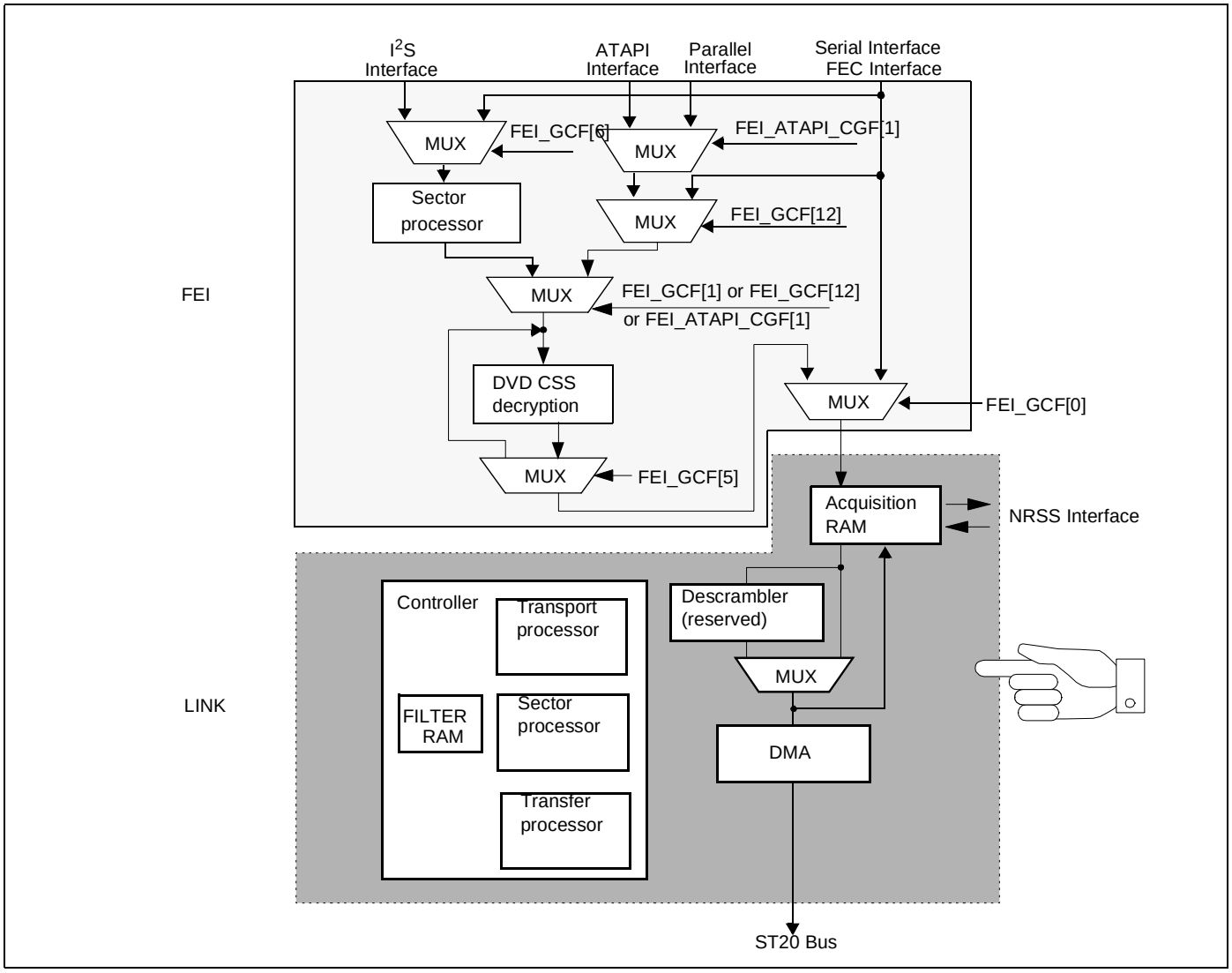


Figure 55 Link architecture

14.2 MPEG-2 & DSS systems layers

Two layers are used to describe link interface processing:

- Transport Packets (TP) layer,
- Packetized elementary stream (PES) or sections (for program specific information) layers.

The link interface performs a complete processing at the TP layer and possibly at PES or section layers.

Function	DVB layer	DSS layer
Acquisition	TP	TP
Descrambling	TP or PES	TP
H/W Filtering (PSI for DVB, CA for DSS)	SECTION	TP

Table 52

14.3 Overview

The link connects the front-end interface to the MPEG decoders and the ST20. It is composed of the following units, and the figure below shows the block diagram:

- Acquisition RAM (AR) + NRSS interface
- Descramblers (DESCR)
- SDAV/P1394 interface
- Filter RAM
- Processor units, including a transport processor, section processor and transfer processor
- Adaptation field filtering
- Clock recovery
- DMA engine

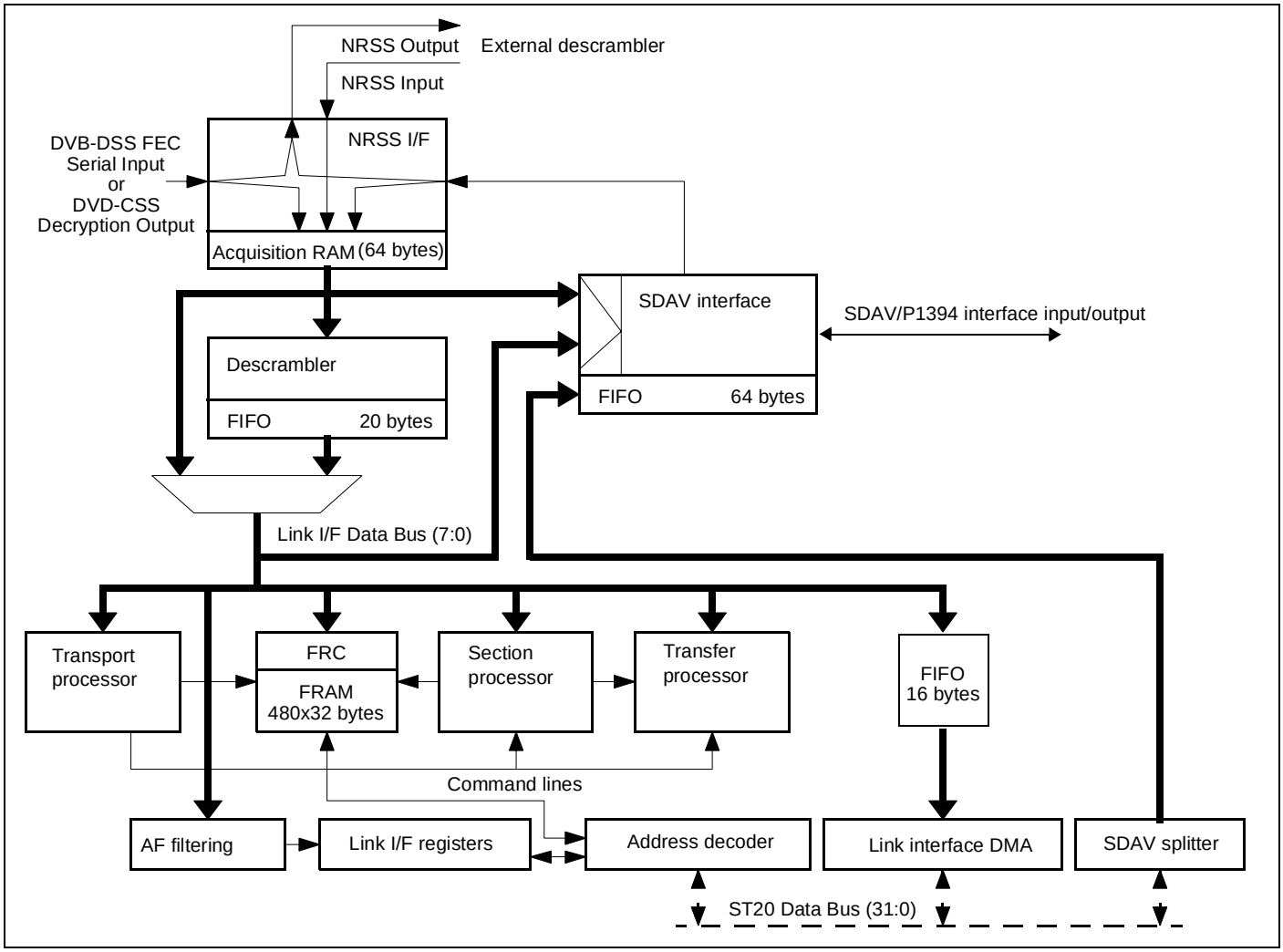


Figure 56 Link block diagram

Input interface (acquisition RAM + NRSS)

Signals at the front-end interface and link interface are asynchronous. Data received from the front-end interface and provided by the serial-parallel converter, are buffered in the Acquisition RAM (AR), which is a FIFO memory.

The packet processing must start *before* the watchdog signal is activated (at least a few bytes before the AR is full).

Descrambling

DVB and DES descramblers are both implemented. For DVB, TP and PES level descrambling are supported. For DSS the descrambling is only done at TP level only.

Up to 8 different key-sets can be used to descramble up to 32 streams. The descrambling keys are located in the FRAM and are automatically loaded after PID filtering. If the payload contained in an acquired TP is scrambled, the descrambler is set-up to handle descrambling and to return descrambled bytes. If the payload is not scrambled, the payload bytes are sent directly.

Note The descrambler is not used for DVD applications.

SDAV/P1394 interface

The high-speed SDAV/P1394 digital interface transfers either scrambled or non-scrambled transport packets between the STi5518 and an external unit, for recording or playback.

The simplified digital A/V bus is a point-to-point connection. It only allows one source on any bus segment at a time.

The IEEE1934 standard provides a single I/O interface with a simple connector that can handle numerous devices through a single port.

This block sends a single stream out to a high-speed serial digital bus, or plays back a stream from that bus. SDAV and IEEE1394 bus formats are supported.

PID filtering

This block contains a filter to receive the TP of one program. It extracts the transport packets of up to 32 streams from the incoming bitstream.

Note PID filtering is not used for DVD applications.

Section filtering

A second filter function is applied to all section-type data. The section header can be compared to up-to 32 targets for each stream. The maximum length of the targets is 16 bytes for DSS and 14 bytes for DVB.

Each bit of each target can be masked individually. For one target byte, two bytes of RAM are required. The total number of target bytes is defined by the size of the filter RAM array.

Note Section filtering is not used for DVD applications.

Adaptation field filtering

Adaption field filtering extracts PCR information, or discards any undesired data contained in the extracted TP.

Note Adaption field filtering is not used for DVD applications.

Processor units and DMA

The transport processor handles the TP-layer relevant bytes, the section processor handles the section-layer relevant bytes and the transfer processor counts the transferred bytes or discards unwanted data.

The DMA engine handles the transfer or relevant bytes to the appropriate ST20 memory buffer.

14.4 Detailed description

14.4.1 Input interface

The link interface receives the TP through the input interface section; it is a fully asynchronous FIFO buffer (64bytes) that decouples write and read clocks.

Data are latched on the falling edge of FEC_B_CLK.

The FEC_P_CLK is active-high during the significant bits of the packet (188 x 8 for DVB, 130x8 for DSS). On this pin, the rising edge is detected and the internal FIFO counter is reset.

The FEC_ERROR signal should be active high for an entire packet if there is an error somewhere in the packet. These packets will not be written into the AR. This signal should only transition at the rising edge of FEC_P_CLK.

The maximum input data rate (FEC_B_CLK) is 59.5Mbit/s or 7.43Mbytes/s (7/8 of 68Mbit/s). The internal link interface block data rate is 60Mbits.T

Clocks	Description	Value
SYS_CLK	descrambler clock	60 MHz
FEC_B_CLK	bit clock signal	up to 59.5 MHz

Table 53 Link interface data rates

14.4.2 NRSS interface

The figure below shows the NRSS interface block diagram. The incoming signal comes from the SDAV/P interface for DVB applications, or from the DVD-FEI for DVD applications. This signal is transferred through the NRSS interface to an external descrambler, after descrambling the signal is transferred back over the NRSS interface to the NRSS input. The descrambled signal is then moved into the acquisition RAM.

In DVB applications, the signal can bypass the NRSS interface and pass from the SDAV interface to the acquisition RAM. The data can pass through without going out to the NRSS.

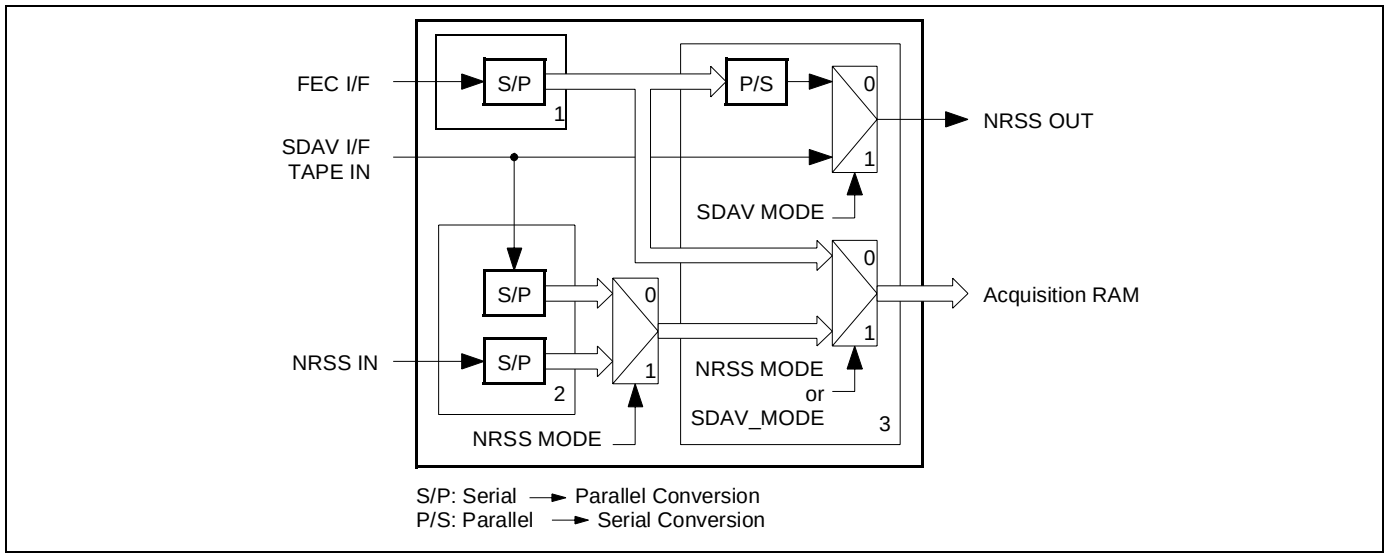


Figure 57 NRSS interface block diagram

The following paths are used:

FEC	->	ARAM		
FEC	->	NRSS	->	ARAM
SDAV	→	ARAM		
SDAV	→	NRSS	→	ARAM

Two MUX controls select whether the input is comes from the FEC interface or a reserved input, and whether the NRSS is used or not. The other FEC signals pass through this block to ensure that proper timing is maintained. Serial/parallel conversion is performed after NRSS. Register LNK_MODE sets MSB or LSB first in the serial-parallel converter.

When the input is FEC, the NRSS_CLK coming from the link interface is discontinuous. Consequently, the data has to be maintained at the end of each byte.

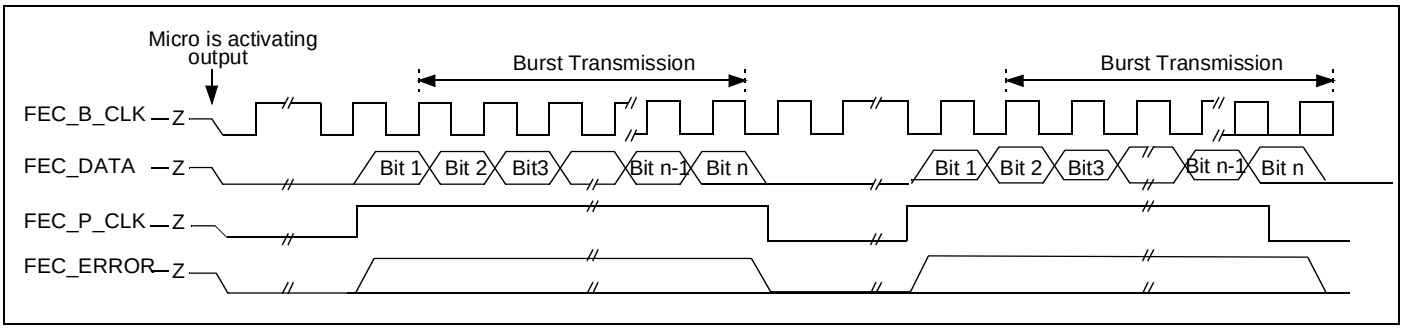


Figure 58 Serial input I/F from channel IC or link IC

The NRSS block inputs are shifted from a serial bit-stream into a serial-to-parallel converter shift-register, using the incoming clock, FEC_B_CLK.

The parallel byte is then loaded into a register and a single bit is generated that toggles on each new byte. That signal is then sampled with the SYS_CLK. This asynchronous sampling takes a couple of clock cycles.

The byte is then loaded into the output shift register for the NRSS interface and is shifted out using the SYS_CLK. If the SYS_CLK is faster than the incoming clock then there will be SYS_CLK cycles where there is no data available to shift out.

When there is no available data, the NRSS_CLK output is forced to remain low for that clock cycle. This mechanism can be broken by making the FEC clock (FEC_B_CLK) faster than SYS_CLK.

Normally the acquisition RAM has all four of the FEC input signals (FEC_B_CLK, FEC_DATA, FEC_P_CLOCK, and FEC_ERROR). Since the NRSS interface has only clock and data, there is no indication of the beginning of a packet. Within the NRSS interface the Link Interface looks for a synchronization byte (0x47) coming from the NRSS card to indicate the beginning of a packet and uses that to generate a packet clock.

Acquisition RAM size

The internal FIFO counter is reset by the rising edge of the packet clock signal in DVB/DSS mode, or by the rising edge of sector start in DVD mode.

When the following packet arrives, the last bytes of the packet in process have to be read. To ensure this, the upper limit of this FIFO is programmable by software so that the last byte of a packet is written as high as possible in the FIFO, as shown in the figure below.

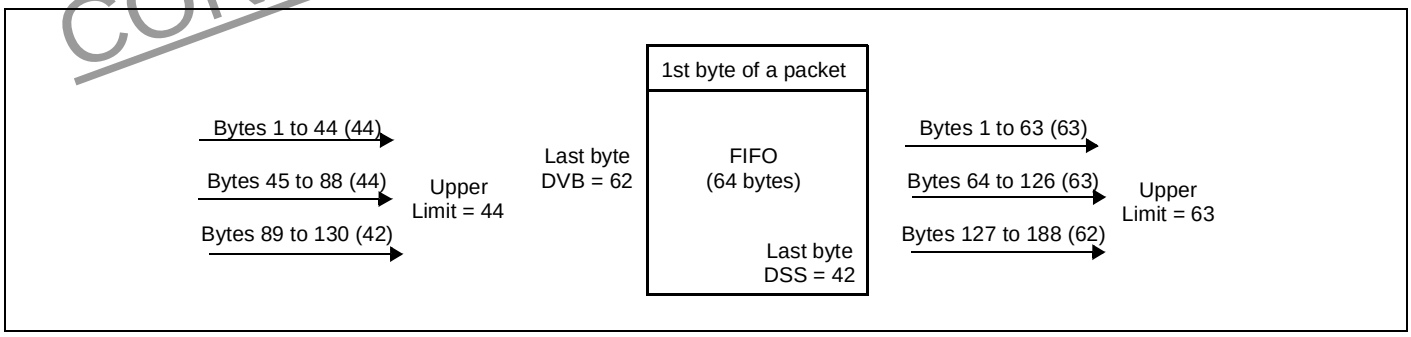


Figure 59 Acquisition RAM

The optimal acquisition RAM size for each mode is as follows:

- DSS mode = 44 (2*44 + 42 = 130)
- DVB mode = 63 (2*63 + 62 = 188)
- DVD mode if sector size is 2066 = 53 (38*53 + 52 = 2066)
- CD mode if sector size is 2048 = 58 (43*58 + 54 = 2548)

14.4.3 Descrambler

The figure below shows the TP and PES header format. The link interface contains two descramblers conforming to the DVB/DES descrambler specifications. Byte # 1 is the 1st byte of the TP - bit(7) = MSB Descrambler DVB.

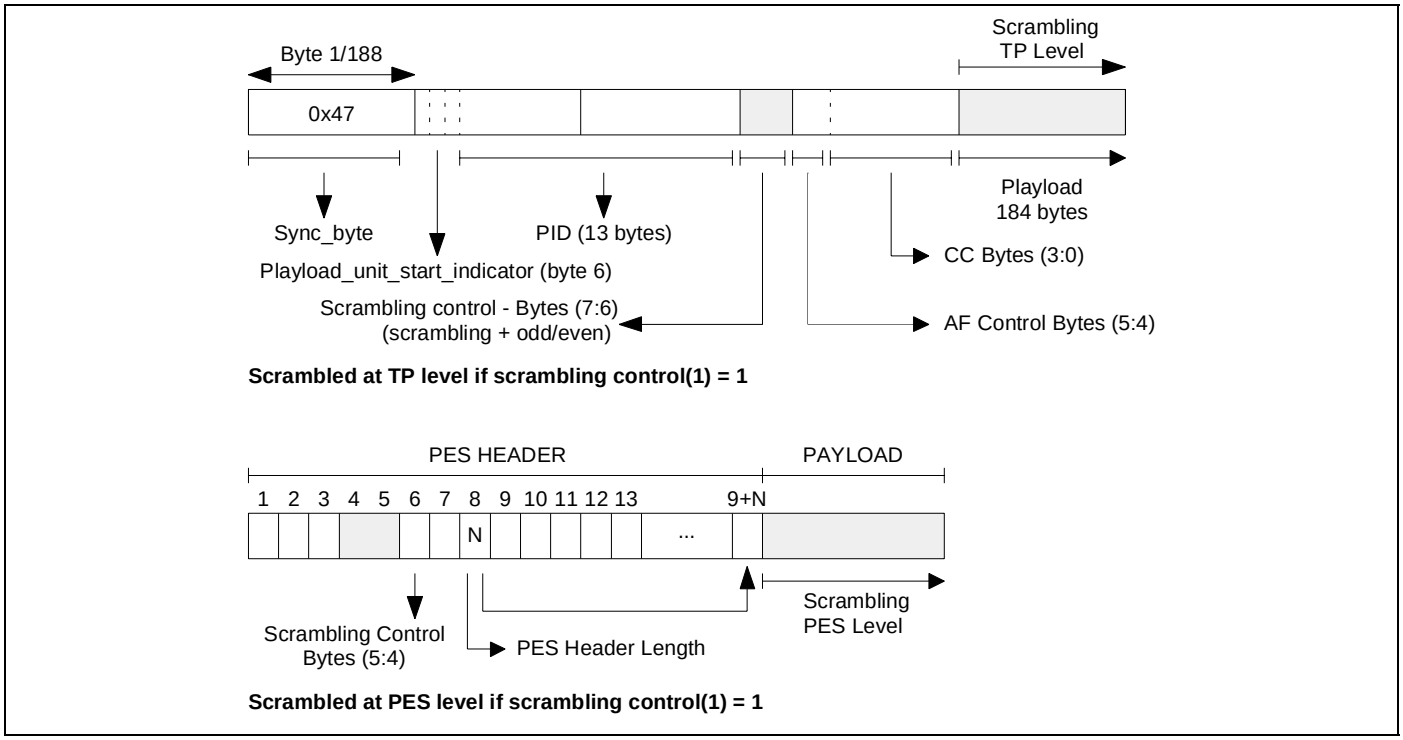


Figure 60 TP & PES Headers

	Scrambling Control Bits	MSB	LSB (if MSB = 1)
DVB TP Level	Byte # 4 Bits(7:6)	1: scrambled 0: non-scrambled	1: odd key 0: even key
DVB PES Level	Byte # 11 Bits(5:4)	1: scrambled 0: non-scrambled	1: odd key 0: even key
DSS	Byte # 1 Bits(5:4)	1: non-scrambled 0: scrambled	1: odd key 0: even key

Table 54

The scrambling algorithm operates on the payload of a TP in the case of TS-level scrambling.

A structuring of PES packets is used to implement PES-level scrambling with the same scrambling algorithm.

The scrambling of MPEG-2 sections is at TP level.

PES-level scrambling

The PES data format is shown in the figure below. The following recommendations must be followed for PES-level scrambling.

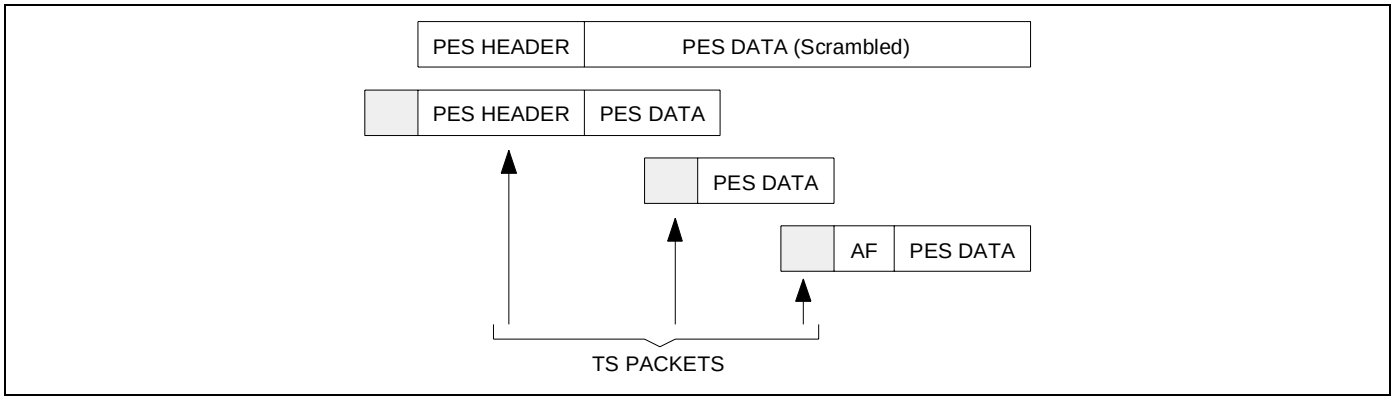


Figure 61 PES data format

1 The PES packet header must not be scrambled.

2 The header of a scrambled packet must not span multiple TP.

3 The TP containing parts of a scrambled PES packet should not contain an Adaptation Field (with the exception of the TP containing the end of the PES packet).

4 The TP carrying the start of a scrambled PES packet must be filled by the PES header and the first part of the PES payload.

In this way, the first part of the PES packet payload is scrambled exactly as a TP with a similar payload. The remaining part of the PES packet payload is split in super-blocks of 184 bytes. Each block is scrambled exactly as a TP payload of 184 bytes.

5 The end of the PES packet payload is aligned with the end of the TP by inserting an Adaptation Field of suitable size (as required in ISO/IEC 13818-1).

If the length of a packet is not a multiple of 184 bytes, the last part of the PES packet payload (from 1 to 183) is scrambled exactly as a TP with a similar payload.

14.4.4 SDAV/P1394 interface

The SDAV/P1394 interface carries a single stream out to a high-speed serial digital bus, or plays back a stream from that bus. SDAV and IEEE1394 bus formats are supported, this section describes each format.

SDAV bus format

This format uses a 49.152 MHz bit-rate in a non-return-to-zero (NRZ) encoding method. The following signals are used:

Signal direction	Pin name	Pin number	Pin direction	SDAV direction	SDAV description
In	SDAV_DATA	103	I/O	I	STROBE_RX (49.1 MHz) (NRZ decoding) (only header + playload)
	SDAV_CLK	22	I/O	I	DATA_RX (NRZ decoding)
	SDAV_DIR	104	I/O	O	DIRECTION (tape in)
Out	SDAV_DATA	103	I/O	O	DATA_TX (NRZ encoding)
	SDAV_CLK	22	I/O	O	STROBE_TX (49.1 MHz) (NRZ encoding) (only header + payload)
	SDAV_DIR	104	I/O	O	DIRECTION (tape out)

Table 55 SDAV bus format on the SDAV/P1394 interface

The signals STROBE_TX and DATA_TX are reversed for transmit and receive, i.e. the signal STROBE_TX acts as the data signal in receive mode and the strobe in transmit mode, the DATA_TX signal acts as the strobe signal in receive mode and data in transmit mode. The signal DIRECTION controls the mode; when DIRECTION is high, the mode is transmit and the other two signals are outputs.

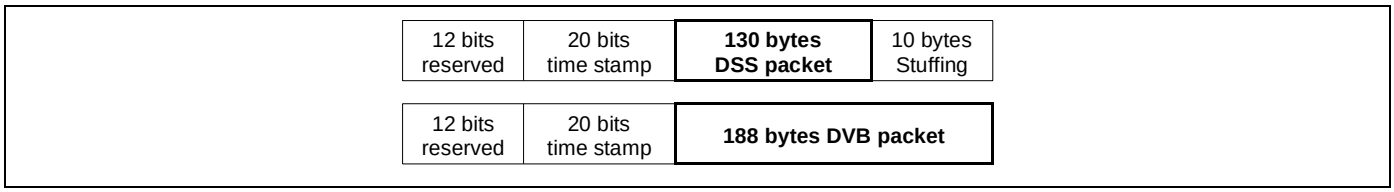


Figure 62 Format for DSS and DVB in SDAV Mode

The time stamp, which is used to maintain proper packet placement, is the value of the LSB's of a continuously running 27 MHz clock. This time stamp is added for SDAV bus and optionally for 1394 bus in tape-out mode, but it is simply discarded when received from these busses in tape-in mode.

For SDAV, during packet transmission, there is only a single mode transmitting on the bus so the media can operate in a half-duplex mode using two signals: DATA_TX/RX and STROBE_TX/RX. NRZ.Data is transmitted on DATA_TX/RX and is a accompanied by the STROBE_TX/RX signal, which changes state whenever two consecutive NRZ bits are the same. This ensures that a transition occurs on either DATA_TX/RX or STROBE_TX/RX for each bit. A clock with transition on each bit-period is derived from the exclusive-or of DATA_TX/RX with STROBE_TX/RX as show below.

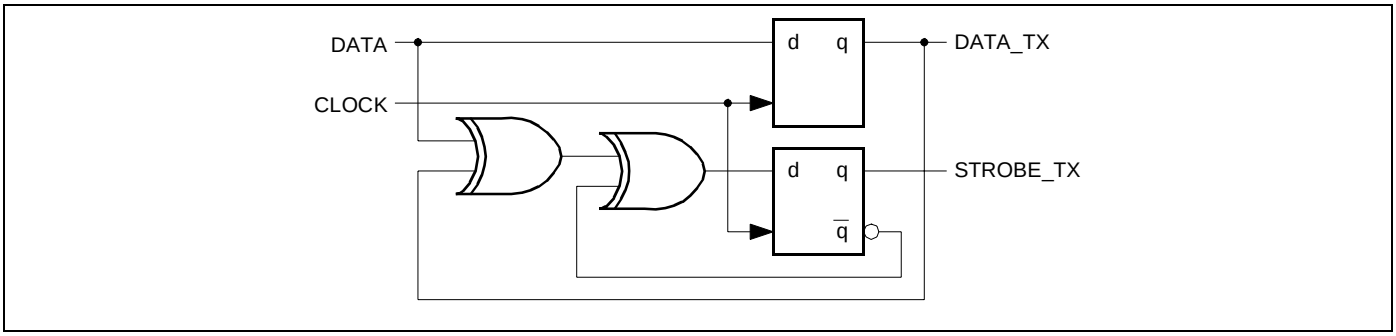


Figure 63 DATA_STROBE NRZ Encoding (inside STi5518)

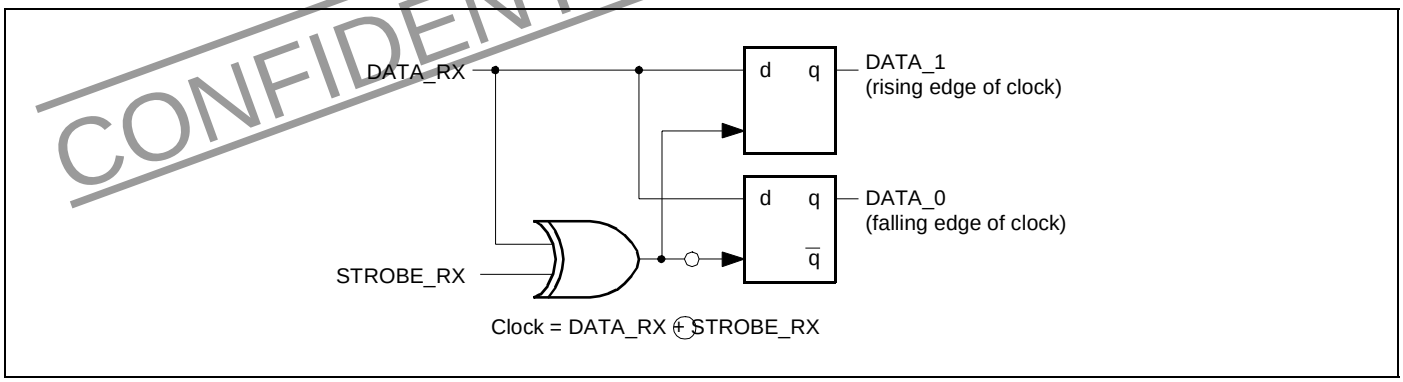


Figure 64 DATA_STROBE NRZ Decoding (inside STi5518)

Packets are synchronized by introducing a clock “gap” of 16 clock (49.152 MHz) cycles (= clock stopped in Figure 63 and Figure 64). So if such a gap is detected, then the packet is finished. The following edge of the clock indicates the beginning of a new packet.

P1394 bus format

This mode uses the STi5518 in conjunction with an external 1394 link layer circuit to interface to the physical bus. The following signals are used.

Signal direction	Pin name	Pin number	Pin direction	SDAV direction	SDAV description
In	SDAV_DATA	103	I/O	I	DATA_IN (no NRZ decoding)
	SDAV_CLK	22	I/O	I	CLOCK_IN continuous (up to 60 MHz)
	SDAV_DIR	104	I/O	I	DATA_VALID_IN (packet clock)
Out	SDAV_DATA	103	I/O	O	DATA_OUT (no NRZ encoding)
	SDAV_CLK	22	I/O	O	CLOCK continuous (60 MHz)
	SDAV_DIR	104	I/O	O	DATA_VALID_OUT (packet clock)

Table 56 SDAV bus format on the SDAV/P1394 interface

All three signals are outputs for tape-out mode, and inputs for tape-in mode. The clock is continuous and the DATA_VALID signal is active for the entire packet, without gaps between the bytes.

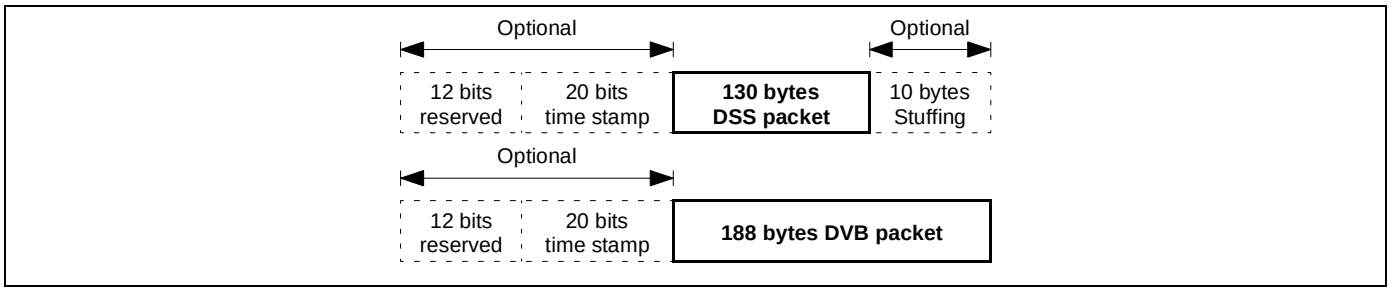


Figure 65 Format for DSS and DVB in 1394 Mode

The DATA_VALID signal defines the size of the packet. The rising edge of DATA_VALID determines the start of the packet. As for SDAV mode, a clock “gap” of 16 clock cycles (up to 60 MHz) is needed before the next rising edge of DATA_VALID.

If the rising edge is asserted before this period has passed, the behavior is undefined.

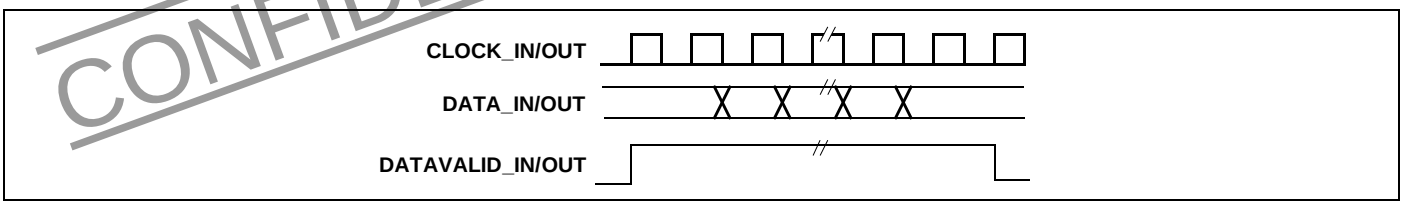


Figure 66 Timings in P1394 Mode

Bus	Mode	Incomplete DMA	Header_ enable	Stuffing_ enable	Header bytes	TP bytes	Padding bytes	Stuffing bytes	Total
SDAV	DVB	0	X	X	4	188			192
SDAV	DVB	1	X	X	4	X	188-X		192
SDAV	DSS	0	X	X	4	130		10	144
SDAV	DSS	1	X	X	4	Y	130-Y	10	144
1394	DVB	0	0	X		188			188
1394	DVB	0	1	X	4	188			192
1394	DVB	1	0	X		X	188-X		188
1394	DVB	1	1	X	4	X	188-X		192
1394	DSS	0	0	0		130			130
1394	DSS	0	0	1		130		10	140
1394	DSS	0	1	0	4	130			134
1394	DSS	0	1	1	4	130		10	144
1394	DSS	1	0	0		Y	130-Y		130
1394	DSS	1	0	1		Y	130-Y	10	140
1394	DSS	1	1	0	4	Y	130-Y		134
1394	DSS	1	1	1	4	Y	130-Y	10	144

Table 57

HEADER if SDAV or (1394 and header_enable) and not(PCM or DVD).

PADDING if incomplete DMA and not(PCM or DVD).

STUFFING if DSS and (SDAV or (1394 and header_enable).

Data path

The maximum input rate is about 7.5Mbytes/s (7/8 * 68Mbits/s). The data are sent to the SDAV interface either scrambled or not. Note that the descrambler works up to 60Mbits/s. Null packets are not transmitted to the digital bus.

Packets concerning other programs, and any useless information are discarded. In such cases, packets may be generated by the ST20 and transmitted by DMA transfer for example.

Those packets must be isochronous with the packets extracted from the original multiplex. Some tables(PAT, PMT) may be modified by S/W to create new program guides for use in playback mode

Tape-in

In tape-in mode the SDAV interface serves as a data source similar to, and instead of, the FEC input. The data is received from the interface and sent to the NRSS block and into the acquisition RAM. The data can be sent directly or re-synchronized to SYS_CLK.

Any header or stuffing, if enabled, are stripped from the packet and discarded with the exception of the 12 reserved bits in the header. These reserved bits are latched in the register LNK_EXTRA_BITS. If these bits differ from the contents of the register prior to the latching, an interrupt is generated.

The SDAV_overflow interrupt is generated to indicate to the ST20 that some extra_bits are available. Both fields (SDAV_overflow and SDAV_underflow) are set in the LNK_STAT_FIFO.

This IRQ can be masked by register bit LNK_EXTRA_BITS[0], and is generated only when the incoming extra_bits changes value.

	SDAV Input	SDAV Output
SDAV_OVERFLOW	EXTRA_BITS_IRQ	SDAV_OVERFLOW
SDAV_UNDERFLOW	when both = 1	SDAV_UNDERFLOW

Table 58

Tape-out

In Tape-out mode, the SDAV block receives data from the acquisition RAM, or from the descrambler. The Link I/F system clock (SYS_CLK) is used as CLK_IN (60 MHz).

The output clock is the interface clock (49.1 for SDAV, up to 60 MHz for 1394). The input data stream speed varies with the speed of the incoming FEC data, or it will be at whatever speed the DMA engine can provide.

The data are latched (at the CLK_IN frequency) into a single port RAM to guarantee the output of one complete packet at the corresponding clock frequency. This means that the SDAV block will receive about one byte every 8 clock cycles (CLK_IN).

As soon as there is enough information in the RAM (not to run out of data before the end of the packet), the SDAV I/F generates the header information and then converts the data to the SDAV bus serial format.

CPU generated packets

The CPU can create custom program guide packets for insertion into the bitstream and DMA the packets to the SDAV block.

The packets are inserted into the out-going stream where possible. The DMA is set-up to send 32-bit word data to the SDAV block. Before enabling the DMA, the ST20 must set the value of the LNK_SDAV_DMA_EN register bits FIRST_BYTE_POSIT and LAST_BYTE_POSIT.

- FIRST_BYTE_POSIT should be loaded with the 2 LSB of the address of the first byte to be transferred.
- LAST_BYTE_POSIT should be loaded with the 2 LSB of the address of the last byte to be transferred.

If the data that is being sent starts at address 0x40001000 and ends at address 400010FF, the value of FIRST_BYTE_POSIT is 00 and the value of LAST_BYTE_POSIT is 0x11. If the data to be sent starts on an odd boundary such as 0x40001001, the DMA should start with the address 0x40001000 but the FIRST_BYTE_POSIT should be loaded with 01. Similarly, if the data to be sent ends on an odd boundary such as 0x400010FE, the DMA should transfer the entire 32 bit word starting at address 0x400010FC but the LAST_BYTE_POSIT should be loaded with 10.

14.4.5 FRAM

Introduction

The FRAM is a dual port 480 x 32 bit RAM that holds all of the link interface information. This includes the filter information, the stream configurations, descrambler keys and the IRQ words. One port is used by the micro to initialize the RAM. The other port is used by the processor, the filter and the descrambler.

After the PID is filtered, the stream number is used to generate the address for the stream initialization. This configuration determines how stream is processed (section, error code, filtering...). For power reduction, the FRAM is only enabled when an access from one of the processors occurs. This happens in case of PID filtering, AF filtering, section filtering, the loading of the stream configuration, the saving of the stream configuration at the end of the packet (section length if section is over 2 TP, CC...) and the IRQ status word read and write operations.

Filtering

Filtering is done by pre-calculating the result. The byte to be filtered generates the address in the FRAM. A “1” at an address means a match. For a 8 bit value, this would give 256 bits in the RAM.

To reduce the RAM size, filtering is done in four steps per incoming byte. In each step, 2 bits of the incoming byte plus a 2-bit pointer, generate the address in the RAM. This gives a 16-bit RAM for one filter byte. Each bit of the byte to be filtered can be masked individually as described below. Because the RAM is 32 bits wide, 32 targets can be filtered in parallel. However, it is also possible to filter on less then 32 targets. In this case the MUX at the output of the FRAM allows selection of only a part of the data.

With each new byte to be filtered, the address of this MUX changes (by adding the filter number) and selects a different part of the output data. Therefore, the lower bit of the filter match register holds the result of the filter process. For example, if only one target is filtered, the LSB of the register holds the result.

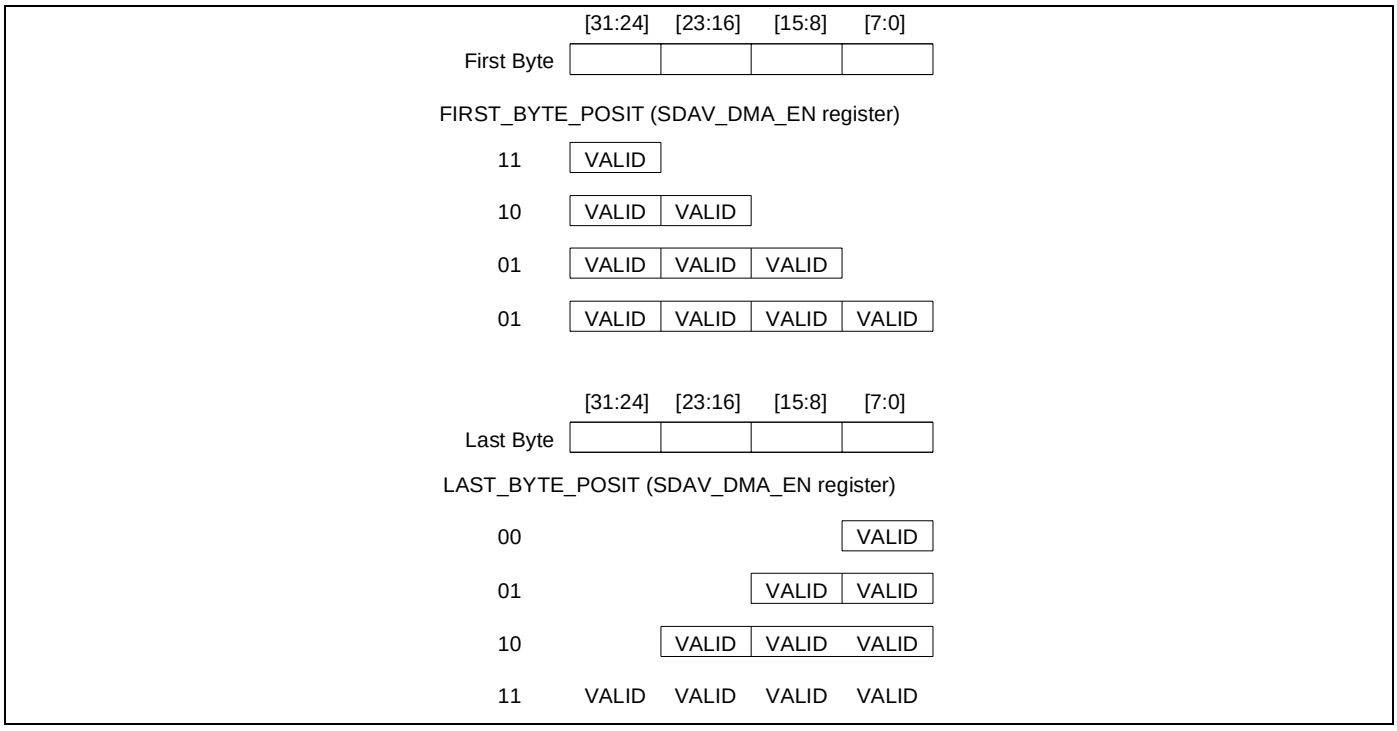


Figure 67 First and last bytes for DMA transfer to the SDAV interface

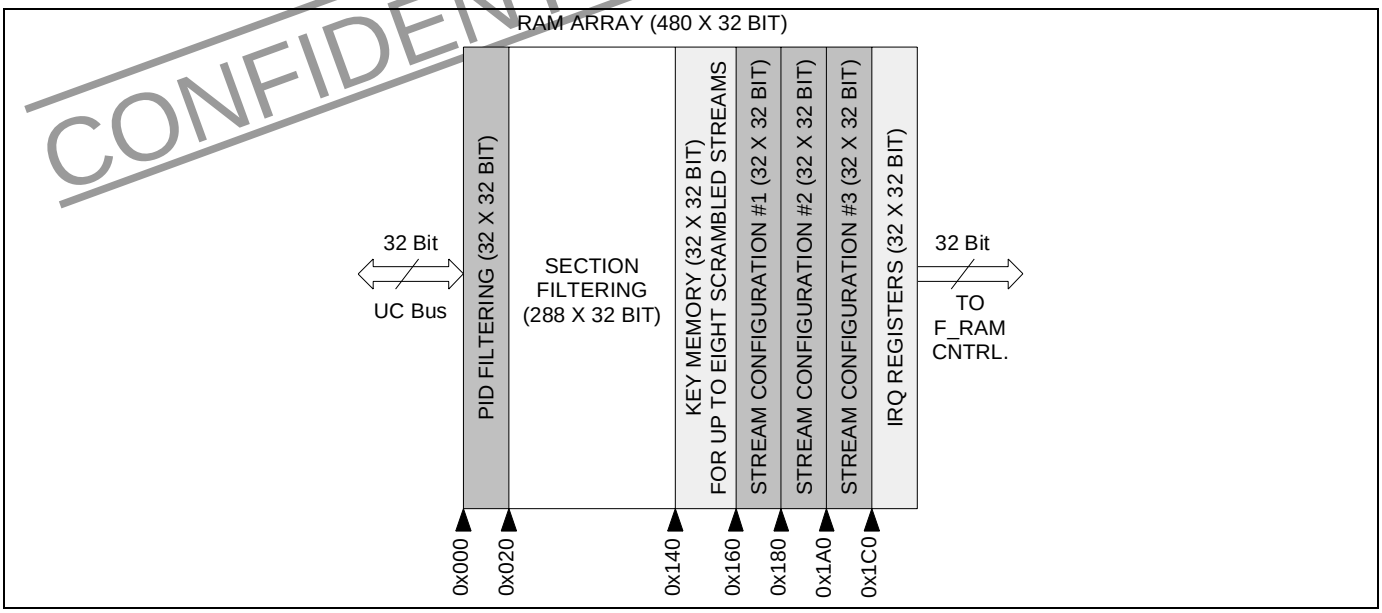


Figure 68 FRAM organization

Filter Mask	Value in FRAM	Filter Mask	Value in FRAM
01101100 11111111	0100 0010 0001 1000	...	
01101100 11111110	0100 0010 0001 1100	01101100 00000010	1111 1111 1111 1100
01101100 11111101	0100 0010 0001 1010	01101100 00000001	1111 1111 1111 1010
...		01101100 00000000	1111 1111 1111 1111 All input bytes are valid

Table 59 Error filtering examples

Error procedures

The following error mechanisms are applied.

Error	Mode	Description
FEC	DVB DSS	This signal is delivered by the Link_IC and signals a packet error. In this case the transport packet is not processed, it is not written into the AR.
SYNC_BYTE	DVB	If the sync_byte in the packet header is not correct (!= 0x47), the TP is rejected.
TRANSPORT_ERROR _INDICATOR	DVB	This bit belongs to the TP header; if it is set the TP is rejected. This is done with the filter process.
CC	DVB, DSS	If the received CC does not match the expected one, different mechanisms are applied according to the stream type.

Table 60 Error mechanisms

The CC error code insertion can be switched on or off for each stream individually, by the stream configuration.

Event	Action
Suspend = ’0’	No CC Error Generated

Table 61 CC error code

CC Error and ERROR_PATT = '1' and not (only AF) and PES	Error code is Generated: B4 00 00 01 B4
CC Error for a Section Stream	If section is active, the stream is disabled and IRQ is generated (see below)

Table 61 CC error code

The following table summarizes the CC processing for DVB.

AF	Payload	Event	Link I/F processing
0	0	CC is Incremented or not	- Skipped TP: Dummy Xfer
1	0	CC n ÷ CC n-1- CC n 1 CC n-1	- End of CC processing - CCstored is Modified by the Link I/F (1)
0	1	CC is Incremented CC n ÷ CC n-1- CC is not Incremented	- End of CC processing - Duplicated TP: Dummy Xfer - Section + Suspend = 1: sTream Disabled (2) - Section + Suspend = 0: Nothing - PES + Suspend = 1: Insert Error Code - PES + Suspend = 0: Nothing
1	1	Same processing as previous combination (2)	

Table 62 CC processing for DVB

1 This will create a CC error for the next packet on this PID.

2 If discontinuity_indicator is set (in the AF) and if an error code was inserted, this code must be removed.

Not equal filtering

The not equal filtering mode can be used only in DVB, not in DSS.

This mode filters on a not-equal condition (defined in register STREAM_CONF_1). The byte to which this is applied is programmable. This filtering can also be done in parallel, on up-to 8 different targets. When using this function, one additional byte has to be filtered after the 'not-equal' byte.

After the section length, register LNK_STREAM_CONF_1 bit SEC_F_INV is loaded. It cannot be loaded with "111". The function is activated by register LNK_STREAM_CONF_1 bit SEC_F_INV.

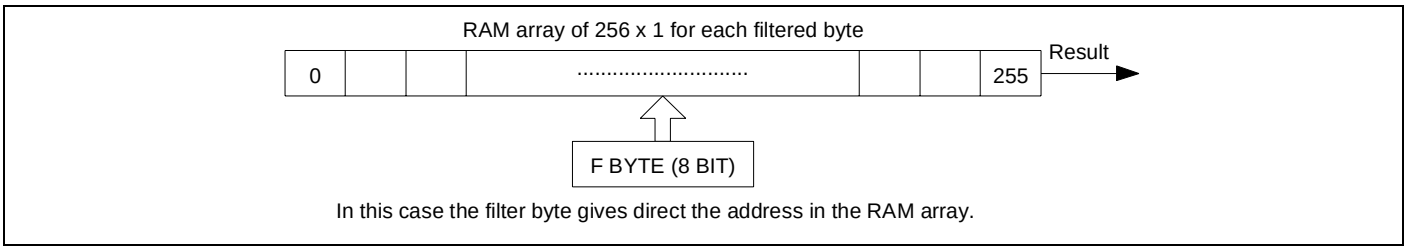


Figure 69 Basic principle of filter mechanism

One byte of each target can be checked to be different from a specific value. At the beginning, all sections can be received. This is done by masking the specified byte (the FRAM is initialized with 00h in not_equal mode). After each section has arrived, the filter for the specified byte can be written into the FRAM (see Table 63).

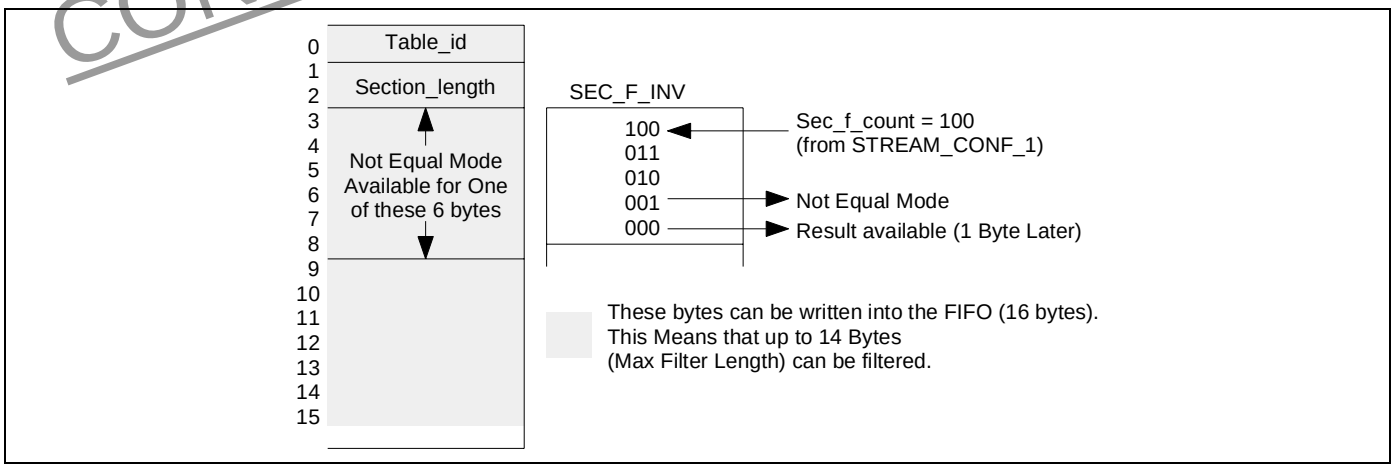


Figure 70 Section filtering example

Filter Mask	Value in FRAM	Equal mode	Not-equal mode
01101100 11111111	0100 0010 0001 1000	01101100 valid	All Valid Except 01101100
01101100 01101100	0100 0010 0001 1100	01101100 and01101101 valid	All valid except 01101100 and 01101101
01101100 10011111	1100 1010 0001 1010	01101100 and 00101100 and 01001100 and 00001100 valid	All valid except 01101100 and 00101100 and 01001100 and 00001100 valid
...	...	...	...
01101100 00000000	1111 1111 1111 1111	All bytes valid	No byte valid
01101100	0000 0000 0000 0000At least 1 nibble = 0000	No byte valid	All bytes valid

Table 63 Equal and not-equal filtering

Note 0: bit is masked, 1: bit is not masked

14.4.6 DMA

MPEG audio, video and system data can be transferred to any location in the ST20 address space (internal SRAM, external SDRAM or DRAM, MPEG decoders) via a DMA controller.

The DMA transfers the data from the link interface to a destination address which can be set individually for each stream.

DMA configuration

The figure below shows the DMA address configuration, the following table gives the DMA configuration registers.

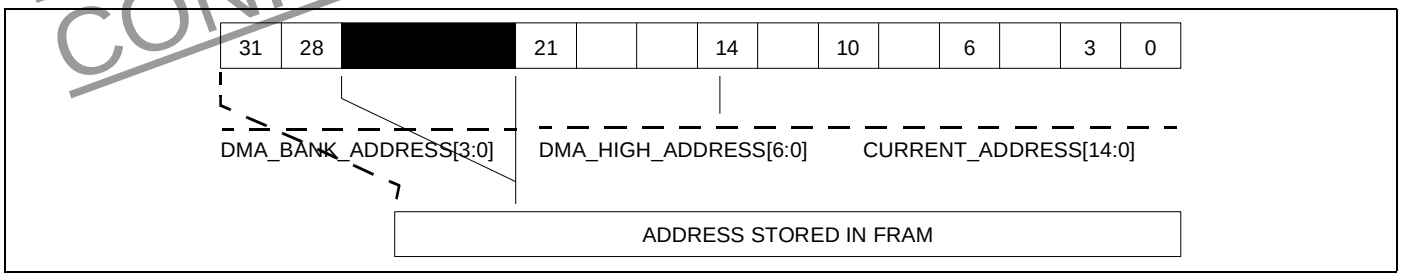


Table 64 DMA address

Register	Bits	Circular buffer
LNK_STREAM_CONF_2		
Buffer_size	26:24	Circular Buffer Size
Stop_address	23:14	Dma_stop_address(14:5) to (7:0)
Start_address	13:11	Dma_start_address(14:12) to (7:5)
Dma_bank_addr	10:7	Dma_address(31:28)
Dma_high_addr	6:0	Dma_address(21:15)
LNK_STREAM_CONF_3		
Dma_low_addr	26:12	Dma_address(14:0)

Table 65 DMA configuration registers

DMA description

At the beginning of the packet, the DMA is initialized by registers LNK_STREAM_CONF_2 and LNK_STREAM_CONF_3. There are two basic DMA modes: incremental (circular and linear) and non incremental.

At the end of the packet, the current DMA address can be stored back to the FRAM. This is reloaded by the link interface when the next packet on this stream arrives. Address write-back is not performed in DVD mode.

- If register LNK_STREAM_CONF_2 bit INCREMENT is set to ’1’, and register LNK_MODE bit DFB is set to ’0’, the last transfer will be a burst transfer, where unused bytes are filled with dummy data.
- If register LNK_STREAM_CONF_2 bit INCREMENT is set to ’0’, or register LNK_MODE bit DFB is set to ’1’, the exact number of bytes is transferred (MPEG decoders).

The address counter keeps the value of the last valid byte that is transferred. Two buffers are used in the DMA (two times 4x32 bits). While one buffer is transferred, the second one is filled. This allows data to be read from the link interface even if the DMA has to wait for the ST20 bus.

The address pointer, which is normally increment every time something is transferred, must be limited in order not to destroy information from other buffers or program code. Therefore, circular buffers are implemented.

Circular buffer (incremental mode)

To stop the transfer when the circular buffer is full, the CPU writes the STOP_ADDRESS in the link interface. In this case, when the transfer pointer (START_ADDRESS) reaches this value, the DMA is aborted. This prevents overwriting of data which has not been processed.

The CPU updates the STOP_ADDRESS each time it has finished processing data. A START_ADDRESS(2:0) and a STOP_ADDRESS(9:0) are specified. The meaning of these bits depends on the BUFFER_SIZE(2:0). The following figure illustrates how the buffer-size is defined.

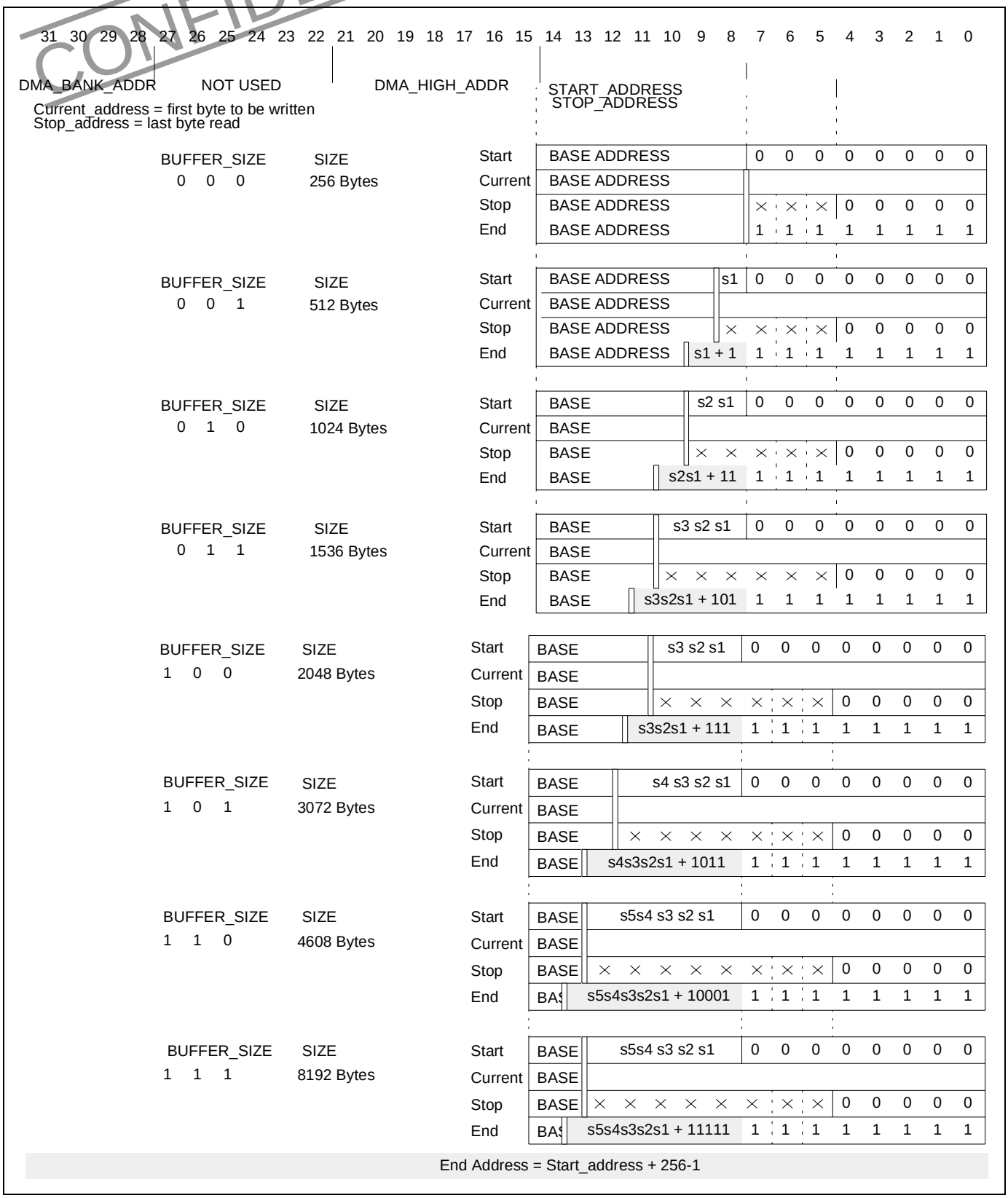


Figure 71 Buffer_size definition

When the transfer address reaches the top of the circular buffer, it is reset to the bottom of the circular buffer and the transfer continues. The figure below illustrates the circular buffer operation.

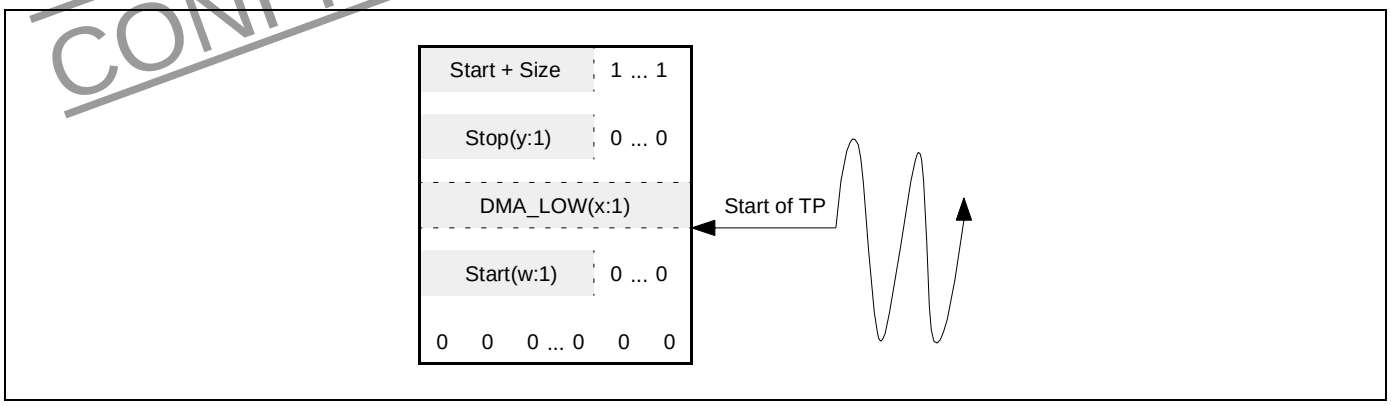


Figure 72 Circular buffer diagram

If the transfer is aborted (if STOP_ADDRESS is reached), the DMA engine generates a DMA overflow and the rest of the packet is discarded. The DMA buffer is flushed to its destination, STREAM_CONF_3 is saved back to the FRAM, and the stream is automatically disabled. The DMA_overflow-bit is set, along with the STREAM_NUMBER in the status word. The status word is written into the LNK_STAT_FIFO register.

DVD buffer (linear mode)

In DVD mode, the DMA can be incremental or non-incremental. If it is incremental, the circular buffer is not used.

- If register bit LNK_STREAM_CONF_2.INCREMENT=1, the start address is set by register bit LNK_STREAM_CONF_3.DMA_LOW_ADDR, and is incremented after each access, independent of the start and stop addresses.
- If register LNK_STREAM_CONF_2.INCREMENT=0, all of the data is written to the address specified by register bits LNK_STREAM_CONF_2.DMA_HIGH_ADDR and LNK_STREAM_CONF_3.DMA_LOW_ADDR.

Non-incremental buffer

This mode is used when addressing the CD FIFOs. The current_address(1:0) is incremented by ’1’ after each access.

<table>
<tr><th></th><th>Circular buffer</th><th>DVD buffer</th><th>Non-incremental buffer</th></tr>
<tr><td>BUFFER_SIZE</td><td rowspan="3"></td><td></td><td rowspan="3">not used</td></tr>
<tr><td>STOP_ADDRESS</td><td rowspan="2">not used</td></tr>
<tr><td>START_ADDRESS</td></tr>
<tr><td>DMA_BANK_ADDRESS</td><td colspan="3" rowspan="3">Used to initialize the DMA</td></tr>
<tr><td>DMA_HIGH_ADDRESS</td></tr>
<tr><td>DMA_LOW_ADDRESS</td></tr>
</table>

Table 66 Non-incremental buffer

14.4.7 Clock recovery

This block assists the control unit in the clock recovery and clock synchronization processes. It uses local counters clocked with the video clock (27 MHz).

The counter values are latched in four registers that can be read by the control unit: PCR_EXT, PCR and V_PTS.

LNK_PCR_EXT(8:0) and LNK_PCR(31:0) are updated with the local time counters when a new packet occurs. If an AF with PCR is found (if PCR flag is present in LNK_AF byte #0), a sample of the 27 Mhz clock is latched and the first 8

bytes of the incoming Adaptation Field are stored in LNK_AF[1:0]. It allows the controller unit to synchronize the decoder clock reference (27 MHz). A new PCR can't be latched if the current PCR (AF[1]) has not been read by the micro. The signal PCR_LATCH_EN is set to indicate that a new PCR can be latched.

LNK_V_PTS (or LNK_A_PTS) is updated with the local time counter(31:0) when a rising edge is detected on V_PTS _LATCH (or A_PTS_LATCH from the audio decoder or external AC3).

The V_PTS_LATCH and A_PTS_LATCH are reset by the clock recovery entity after the register has been read by the control unit.

A counter (19:0) is also required to generate the time stamp used for the SDAV header. The time stamp value is updated at the beginning of each packet, as illustrated below.

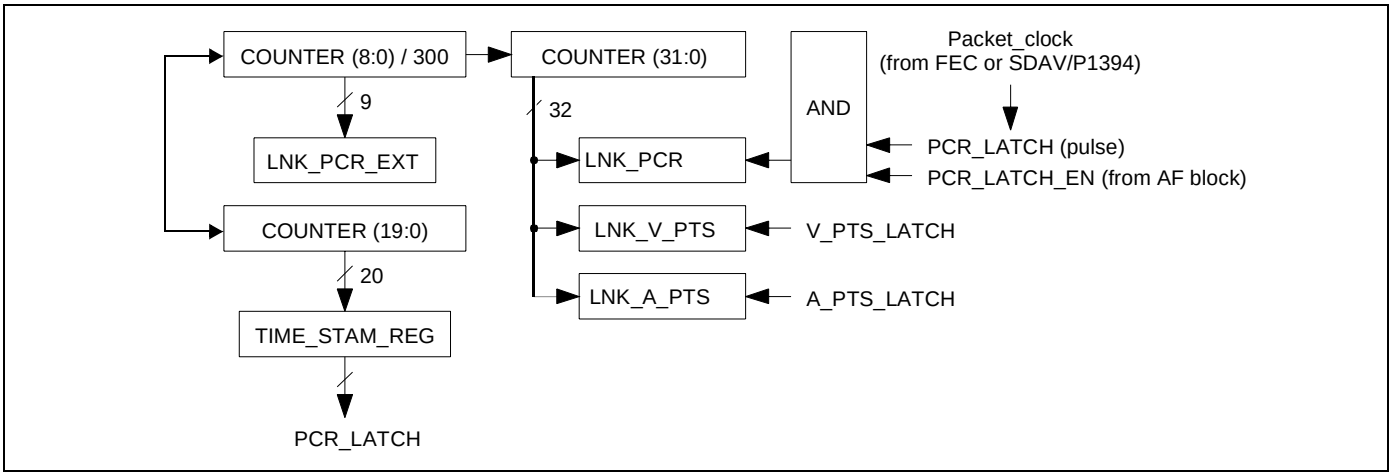


Figure 73 Clock recovery

14.4.8 Interrupts

There are nine interrupt sources for each stream. Each interrupt source has a corresponding status bit in the register LNK_STAT_FIFO. If the interrupt is enabled (mask=1) and the corresponding status bit is set, an interrupt event is generated.

Successive interrupts generate successive status words (LNK_STAT_FIFO) that are stored in chronological order in a 32-word FIFO. This is useful if multiple interrupts occur within a packet (multiple sections per packet). The FIFO is implemented as a circular buffer in the FRAM.

In the status-word, the stream-number field (LNK_STAT_FIFO.SN) and interrupt status bits are always applicable. The target-match (LNK_STAT_FIFO.TM) and other-match (LNK_STAT_FIFO.OM) are only valid on end-of-filter (LNK_STAT_FIFO.EOF) and incomplete-filter interrupts (LNK_STAT_FIFO.IF). In addition, the filter_offset field is only valid on incomplete-filter interrupts.

The LNK_STAT register indicates whether interrupts are pending (fifo-not-empty, LNK_STAT_FIFO.FNE). Each time a status-word is written to (resp. read from) the FIFO, a write (resp. read) pointer is incremented (LNK_STAT). The FIFO is read by the micro through the LNK_STAT_FIFO register. When a word is read, it is removed from the FIFO.

The ST20 checks the LNK_STAT (for FIFO emptiness or overflow) prior to reading the LNK_STAT_FIFO. The ST20 does not read the FIFO if it is empty. When at least one status word is present in the LNK_STAT_FIFO, the interrupt line to the ST20 is set active and kept active as long as the FIFO is not empty. The interrupt line becomes inactive when the last word is read from the FIFO. If the FIFO is full and new interrupt events occur, no further status-word is written to the FIFO and the FIFO overflow bit (LNK_STAT.FO) is set. This bit is reset when the ST20 reads a word out of the FIFO.

The table below describes the interrupt sources.

Interrupt source	Description
AR overflow Status bit LNK_STAT_FIFO.AO	Not maskable, no stream disabling **IRQ generation** An AR overflow occurs if at least one of the following condition is true: • a new packet has started to be stored in the AR, and its processing has not started as “AR_timeout” bytes have been stored; • the write pointer of the AR reaches the read pointer. This can happen if the link interface hangs, or if the ST20 bus traffic prevents data to be output fast enough. **IRQ processing** The stream is not disabled when such an AR overflow occurs, the current packet processing is aborted. All packet data which have not already been passed to the DMA write-buffer are discarded. The DMA write-buffer is flushed to its destination. An internal reset signal is generated to put the whole link interface into a state where it expects a new packet to arrive: • if a start-of-packet is present in the AR, data input is not stopped and overwrites the discarded data; • if a start-of-packet is already present in the AR, its processing starts immediately. The stream_conf_3 of the aborted packet is not saved back to FRAM. Therefore, appropriate error processing will be performed due to CC discontinuity on the next packet of the same PID. The ar_overflow bit is set along with the stream_number in the status-word, and the status-word is written into the link_stat_fifo.
DMA overflow Status bit LNK_STAT_FIFO.DO	Not maskable, stream disabling possible. **IRQ generation** A DMA_overflow interrupt occurs when the write-pointer of the DMA transfer reaches the stop value stored in stream_conf_2. **IRQ processing** The stream is disabled. The rest of the packet is discarded. The DMA buffer is flushed to its destination. stream_conf_3 is saved back to FRAM. The DMA_overflow bit is set along with the stream_number in the status-word, and the status-word is written into the link_stat_fifo.

Table 67 Interrupt sources

Interrupt source	Description
Bad section Status bit LNK_STAT_FIFO.BS	Not maskable, stream disabling possible. **IRQ generation** When saving a section to memory, the link interface counts the section length and detects the end of the section. The following conditions will generate a bad section interrupt: • The PUS of the current packet is set and the pointer_field at the beginning of the packet does not correspond to the number of remaining bytes in the currently transferred section. • The CC of the current packet has the wrong value on a section packet when a section is being processed. • The PUS is not set and the bytes following the current section are not stuffing bytes (FF). **IRQ processing** The stream is disabled.In this case, the bad_sec bit is set along with the stream_number in the status-word, and the status-word is written into the link_stat_fifo.(Note that the CC is checked on PES streams if the error_patt bit of the stream_conf_1 is set, but no IRQ is generated there.)
End-of-section filtering Status bit LNK_STAT_FIFO.EOF	Maskable, no stream disabling. **IRQ generation** DSS/DVB mode: An EOF interrupt is generated if the eof_irq bit is set for the current stream and a filtering operation has been completed with a match. DVD mode: An EOF interrupt is generated if the eof_irq bit is set for the stream 0 and the DMA engine has been initialized with the parameters stored in FRAM. **IRQ processing** Normal processing continues. DSS/DVB mode: The eof_flag is set along with the stream_number in the status word. The values of target_match and other_match are stored in the status word which is pushed into the link_stat_fifo. This information can be useful for the application software. DVD mode: The EOF_flag is set along with the STREAM_NUMBER in the status word.
End-of-section transfer Status bit LNK_STAT_FIFO.EOS	Maskable, no stream disabling. **IRQ generation** DSS/DVB mode: An eos interrupt is generated if the eos_irq bit is set for the current stream number and a section has been completely transferred by the DMA controller to memory. DVD mode: An eos interrupt is generated if the eos_irq bit is set for the stream 0 and the DMA engine has finished the transfer of a packet. **IRQ processing** Normal processing continues. DSS/DVB mode: The eos_flag is set along with the stream_number in the status word. The status word is written into the link_stat_fifo. DVD mode: The EOS_flag is set along with the STREAM_NUMBER in the status word.

Table 67 Interrupt sources

Interrupt source	Description
Incomplete filtering Status bit LNK_STAT_FIFO.IF	Maskable, no stream disabling. **IRQ generation** An incomplete filtering interrupt is generated if the incomplete_irq bit is set, a section filtering operation is not complete at the end of a packet and at least one temporary match is pending. The filtering is considered as successful and the section transfer starts. **IRQ processing** Normal processing continues. The values of target_match and other_match are stored in the status word and the incomplete_filter bit is set along with the stream_number and filter_offset in the status word and pushed into the link_stat_fifo. The application software can use this information to complete the section filtering.
AF Status bit LNK_STAT_FIFO.AF	Maskable, no stream disabling. **IRQ generation** An LNK_AF interrupt is generated if the af_irq bit is set for the current stream number and an AF with at least one flag set is present in the packet and the previous one has been read by the ST20. The first 8 bytes of the AF are stored into the AF[1..0]. If the AF is less than 8 bytes, some payload bytes are written in the AF buffer. If AF length is 0, there is no match and no storage. **IRQ processing** Normal processing continues. Once the Adaptation Field information has been written into the AF registers, the PCR counter value is latched and no new AF information can overwrite it until the CPU has read the register AF[1].If another AF is received before the previous one has been read by the CPU, it is discarded.The af_flag is set in the status word along with the stream_number, and an interrupt is generated.
SDAV underflow Status bit LNK_STAT_FIFO.SU	Not maskable, no stream disabling. **IRQ generation** A SDAV underflow is generated if a SDAV packet has started to be output to the SDAV port and no more data is present in the SDAV block to be sent. **IRQ processing** The stream is not disabled. The output packet is corrupted and further data belonging to that packet is discarded. Since the SDAV operates independently from the rest of the Link I/F, the stream_number present in the status word may not be relevant.When such a SDAV underflow error occurs, the sdav_underflow bit is set along with the stream_number in the status word and the status word is written into the link_stat_fifo.

Table 67 Interrupt sources

Interrupt source	Description
SDAV overflow Status bit LNK_STAT_FIFO.SO	Not maskable, no stream disabling. **IRQ generation** A SDAV underflow is generated if the SDAV buffer is full and new data is presented at the input of the buffer to be stored. **IRQ processing** The stream is not disabled. Meanwhile, a new packet has possibly started being processed by the Link I/F. Note that the SDAV block is supposed to accept data as delivered by the rest of the Link I/F and can in no way suspend Link I/F operation. When this occurs, the output of the current packet is aborted and all remaining data belonging to that packet is discarded. The read and write pointers are reset. If a new packet_start is already present, data input is not stopped and overwrites the discarded data. The read pointer points now to the first byte of this new packet. Since the SDAV operates independently from the rest of the Link I/F, the stream_number present in the status word may not be relevant. When such a SDAV overflow error occurs, the sdav_overflow bit is set along with the stream_number in the status word and the status word is written into the link_stat_fifo.

Table 67 Interrupt sources

14.5 DVD/link data analyzer

The DVD data analyzer can only be used for the sector data structures described below.

The DVD data analyzer facilitates trick modes and uses packet identification and video start-code detection functions. The DVD data analyzer is enabled by the start-code detector register bit LNK_MODE.E_SCD. The start-code detection circuits in the link are enabled when E_SCD is set and DVDmode is selected bit (LNK_MODE.DVD_M).

Packet identification and video start-code detection are two of the DVD data-analyzer functions. A byte counter tracks the start-code locations and is reset by the Sector_Start signal. The start-code type and location are written to the FRAM. The packet-type and the number of start-codes inside the sector are set by the LNK_STAT_FIFO register.

Processing a new sector

When both the **DATA_VALID** and **Sector_Start** signals go high, the first bit of a 2066 byte sector is transmitted over the serial interface from the front-end. Each sector is treated independently of the adjacent sectors. Any potential start-codes that are cross-sectored (straddled between two sectors) are flagged on the first sector. *The ST20 checks for a start-code at the beginning of the next appropriate sector when the start-code continuation flag has been set from the first sector.*

- If LNK_MODE.E_SCD=0, all sectors are processed as in the STi5518.
- If LNK_MODE.E_SCD=1, all sectors are processed as in the STi5518, but the link hardware non-destructively evaluates the data as it passes through the link.

Sector data structure

Each sector consists of 2066 bytes transmitted over the serial interface in the following order:

- Bytes 1:4 (4 bytes) for the sector ID (Identification Information) (1 byte Sector Information, 3 bytes, byte 2 to 4, Sector Number);
- bytes 5:6 (2 bytes) for the IEC (ID Error Correction Code);
- bytes 7:12 (6 bytes) of Copyright Management Information (CPR_MAI);

- bytes 13:2060 (2048 bytes) of the pack data referenced as D1:D2048;
- bytes 2061:2064 (4 bytes) for the EDC (Error Detection Code);
- bytes 2065:2066 (2 bytes) for the Read Solomon error codes sent by the channel IC.

The Sector number or sector ID is saved for later processing. “Pack data” is the area containing the packet identification and the MPEG data with the start codes. All other parts are passed without processing.

4 bytes (1:4)	2 bytes (5:6)	6 bytes (7:12)	2048 bytes (13:1060)	4 bytes (2061:2064)	2 bytes (2065:2066)
ID	IEC	CPR_MAI	pack data (D1:D2048)	EDC	RSEC

Table 68 Sector data structure

Identifying a packet

1 Determine if bytes 13:16 (4 bytes) or D1:D4 contain the **pack_start_code (0x000001BA)**.
   - If the pack-start code is found, continue.
   - If the pack-start code is not found, write a ‘100’ in register bit LNK_STAT_FIFO.PT.

2 Determine the type of packet contained in the sector by parsing the packet header.

3 Extract the first 4 bytes of the packet header, D15:D18, and set the packet-type bits of the LINK_STAT_FIFO register, according to the table below.

4 Follow the actions identified for the packet-type.

Packet type	Start code	Action
Video packet	0x000001E0	Set the flag identifying the packet as a video packet in the Link_stat_fifo register and continue processing the data per the section on video packet processing.
Navigation packet	0x000001BB	Set the flag identifying the packet as a navigation packet in the Link_stat_fifo register. No further action is required on this sector.
Audio packet or sub-picture packet (private_stream_1)	0x000001BD	Skip the next 4 bytes D19:D22 for the PES structure information. Save the next byte, D23, and use its value in the next step. Skip the number of bytes defined in D23. The next byte is the sub_stream_id byte. For example, if D23 = 2 then skip D24:D25. D26 would be the sub_stream_id byte. If D23 = 0 then D24 would be the sub_stream_id byte. • If the sub_stream_id byte contains **10100xxxb**, set the flag identifying the packet as a **LPCM audio packet** in the Link_stat_fifo register. No further action is required on this sector. • If the sub_stream_id byte contains **10000xxxb**, set the flag identifying the packet as an **AC3 audio packet** in the Link_stat_fifo register. No further action is required on this sector. • If the sub_stream_id byte contains **001xxxxxb**, set the flag identifying the packet as an **subpicture packet** in the Link_stat_fifo register. No further action is required on this sector. • If the sub_stream_id byte contains any other value, set the flag identifying the packet as unknown in the Link_stat_fifo register. No further action is required on this sector.

Table 69 Packet types and start codes

Packet type	Start code	Action
MPEG audio packet	0x000001CX	Set the flag identifying the packet as an MPEG audio packet in the Link_stat_fifo register. No further action is required on this sector.
MPEG-2 audio packet containing an extension audio stream	0x000001DX	Set the flag identifying the packet as an MPEG audio packet in the Link_stat_fifo register. No further action is required on this sector.
Unknown packet type	Any other code	No further action is required on this sector.

Table 69 Packet types and start codes

Processing a video packet

1 Find the beginning of the video MPEG bit stream payload within the sector and locate a video packet (0x000001E0).

2 Skip bytes D19:D22 for the PES structure information.

3 Save byte D23 for use in the next step.

4 Skip the number of bytes defined in D23. The next byte is the first byte of the MPEG bit stream payload of this sector. The byte range is from D24 to D44 (bytes 37-57 from start of sector). If D23 = 2, skip bytes D24 and D25, use byte D26 as the first MPEG byte. If D23 = 0, then D24 is the first MPEG byte.

5 Provide a start-code detector for the incoming video bitstream.

For start-codes located inside a sector (byte aligned) use the following table:

Start code		Start code type
0x00000100	SC = ‘001’	picture_start_code (32 bits)
0x000001B3	SC = ‘010’	sequence_header_code (32 bits)
0x000001B4	SC = ‘011’	sequence_error_code (32 bits)
0x000001B5	SC = ‘100’	extension_start_code (32 bits) * Report it only If the next four bits are one of the following: 0001 Sequence Extension ID 0111 Picture Display Extension ID 1000 Picture Coding Extension ID * if not, do not report it in FRAM and continue to the next start code
0x000001B7	SC = ‘101’	sequence_end_code (32 bits)
0x000001B8	SC = ‘110’	group_start_code (32 bits)
SC = ‘000’ and SC = ‘111’		not used

Table 70 Identity for start-codes located inside a sector

For cross-sector start-codes, process the last 4 bytes (D2045:D2048) of the sector as follows:

Byte		Action
D2045:D2048	0x000001B5	Set flag (bit 7) in link_stat_fifo to indicate continuation of a SC into the next sector and write SC into FRAM using ‘001’ as number of bytes carrying to the next sector.
D2046:D2048	0x000001	Set flag (bit 7) in Link_stat_fifo register to indicate continuation of a SC into the next sector and write SC into FRAM using ‘010’ as number of bytes carrying to the next sector.

Table 71 Identity for cross-sector start-codes

D2047:D2048	0x0000	Set flag (bit 7) in Link_stat_fifo register to indicate continuation of a SC into the next sector and write SC into FRAM using ‘011’ as number of bytes carrying to the next sector.
D2048	0x00	Set flag (bit 7) in Link_stat_fifo register to indicate continuation of a SC into the next sector and write SC into FRAM using ‘100’ as number of bytes carrying to the next sector.

Table 71 Identity for cross-sector start-codes

Reporting start codes

The start-code information is reported in both FRAM and LNK_STAT_FIFO register. A portion of FRAM (from 0x00 to 0x7F) containing 128 32-bit words is used for reporting SCs in this mode. The FRAM is partitioned into 4 sections of 32 words, each is used for one sector of SC information. The FRAM has data written to it only when the start-codes are found within the sector being processed, but the LNK_STAT_FIFO register is updated after every sector is transferred.

FRAM address

The FRAM address is defined as follows:

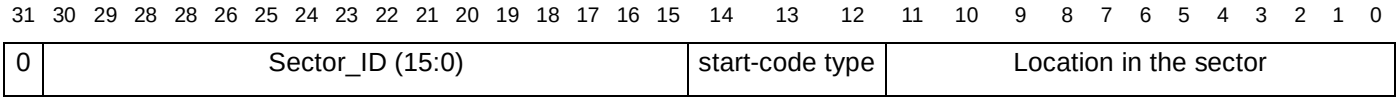
Four sections of the FRAM can be addressed using the two MSBs of the 7-bit address. The two MSBs are incremented in value for the next sector, only when the present sector has data to be written into the FRAM. If no data is written into the FRAM for the present sector, the two MSBs remain the same for the present and the next sector. In this way, the ST20 can keep track of data during jumps and skipping of non-video sectors. These two bits are reset by setting the Start_code_enable signal to inactive, and then resetting it to active.

The 5 LSBs of the FRAM address start at ‘00000’ at each sector start. Each time a start code is written into the FRAM, the 5 LSBs are incremented. A maximum of 31 words (‘11111’) are written into the FRAM for any sector. All writing of data into the FRAM stops after ‘11111’ for any sector, to prevent overwriting of earlier start codes found in this sector. Therefore, the FRAM can report maximum of 31 start-codes in one sector. If a sector has more than 31 start-codes, the ST20 must take action. The two MSBs used to define the section of FRAM containing the start-code is communicated in bits 6:5 of the LNK_STAT_FIFO register. The number of start-codes found in the sector and reported in the FRAM, are communicated in bits 4:0 of the LNK_STAT_FIFO register.

FRAM data

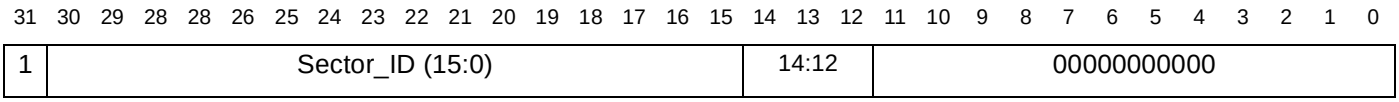
Start-code reports must contain the following information in a 32 bit word:

For start-codes within a sector:



Bitfield	Description
11:0	Location of the last byte of the start-code in that sector. For the extension_start_code 0x000001B5, report the location of the next byte
14:12	Start-code type
30:15	Sector_ID LSBs
31	Indicates possibility of cross-sector start code case. If this bit is set, the software must check the beginning of the next continuous video sector to see if there is a start code

For cross-sector start-code, one additional FRAM write is performed in the following format:



Bitfield	Description
11:0	00000000000
14:12	Number of bytes carrying to the next sector of a possible start code (max value is ‘100”).
30:15	Sector_ID LSBs
31	Indicates possibility of cross-sector start code case. If this bit is set, the software must check the beginning of the next continuous video sector to see if there is a start-code

If start-codes of more that 32 bits are found, the bits above bit 31 are not processed and the information is lost.

14.6 Hard disk drive buffer control

A HDD_BC (HDD buffer controller), positioned between the link interface and the ST20 interconnect, traps the addresses associated with the video pid and adds an offset in order to establish a much bigger buffer within the external memory.

In normal operation, the base address of the circular buffer in external memory is set by DMA_LOW_ADDRESS programmed in LNK_STREAM_CONF3, and DMA_HIGH_ADDRESS and DMA_BANK_ADDRESS programmed in LNK_STREAM_CONF2 register. The circular buffer size can be programmed to a maximum of 8K by LNK_STREAMCONF2.

In HDD mode, the buffer size is programmable but its position in the external memory must be on a boundary that is an integer multiple of its size. To use the HDD_BC, the link circular buffer must be set to its maximum 8KByte size. In HDD mode the base address of the circular buffer is set in exactly the same way as in normal mode until the buffer reaches 8K, then the DMA_HIGH_ADDRESS is incremented by the HDD_BC and programmed in the HDD register LNK_HDD_ADDRHIGH. The size of the circular buffer is now controlled by the HDD_BC through register LNK_HDD_BUFSIZE, and the stop mechanism in the link interface must be disabled by programming the stop pointer location outside the 8K circular buffer (by register LNK_STREAM_CONF_2 bits STOP_ADDR). This causes an HDD link interrupt (interrupt assignment number N=29). All of the HDD dedicated registers are described in the HDD Buffer Control section in the Link chapter of the STi5518 Register Manual.

15 MPEG video decoder

The MPEG video decoder decompresses a MPEG 2 bit-stream and constructs a picture. The display functions are described in Sub-picture decoder on page 150 and the MPEG video decoder registers are described in the STi5518 *Register Manual*.

15.1 Decoder operation

The video decoder is a picture decoder; it decodes one picture and then stops until instructed to decode the next picture in the video bit-stream.

Normally, the decoding of a new picture starts in response to the start-of-display of a new picture. The registers whose contents can change from picture to picture are double-banked and are updated automatically when decoding starts. The bit-stream is read from the bit-buffer into the variable-length code decoder (VLD), and picture can be built. Any required predictors are fetched from the appropriate area of the external memory, and the reconstructed picture is written back into the area of this memory assigned to the decoded picture.

While a picture is being decoded, the start-code detector locates the start of the next picture header. The CPU then uses this to set-up the double-banked registers to decode the next picture.

All of these tasks can be synchronized using interrupts generated on start-code hits and vertical-sync signals.

Start code search

The video decoder is able to decode in its entirety a video bit-stream from the slice layer downwards. The higher layers (i.e. picture and upwards) are decoded by the driver in order to extract the information needed for decoding and set up the appropriate video decoder registers and quantization tables. Since the header information is byte-aligned and requires minimal interpretation, this task represents only a small load on the CPU.

The start code detector parses the bit-stream stored in the bit buffer and locates start codes corresponding to picture layer and above. When one of these start codes has been found, the start code detector stops and raises an interrupt.

The CPU is then able to read the header data following the start code. The start code detector starts automatically whenever the decoding of a new picture starts and on user command. In normal operation, start code parsing is performed one picture in advance of decoding.

Bandwidth reduction mode

Bandwidth reduction mode is the only decoding mode that is supported by the device. In this mode, where I, P and B-frames are decoded into and displayed from frame buffers in external memory, the decoder uses three frame buffers in external memory. This mode opimizes memory bandwidth use.

15.2 Reset

Hard reset is a global signal and is described in the *System Services* chapter. The following types of soft reset can be used for the video decoder:

- Total soft reset is generated by setting and resetting bit VID_CTL.SRS and register AUD_RES. They must be set for a duration of at least 1µs.
- Audio, video and sub-picture subsystems may be individually *soft reset* by setting and resetting VID_SRA and AUD_RES, VID_SRV and SPD_SPR respectively.
- Pipeline reset is generated by setting and resetting bit VID_CTL.PRS. It must be set for a duration of at least 40ns.

After a soft reset, all processes concerning decoding and bit buffer control are reset. Any data remaining in the bit buffer, the compressed data FIFO and the start code detector FIFO are lost.

The interrupt unit is reset. All registers maintain their contents and the display process is not disturbed. A soft reset would normally be used when the decoding of the current bit-stream must be terminated and it is required to restart on a new sequence.

After a hard or a soft reset or a video soft reset, the first task performed by the pipeline when it has been enabled will always be a search for the beginning of a new sequence. The bit buffer data is flushed until the first picture start code following a sequence start code is detected by the pipeline, at which time it stops. At this point normal picture decoding behavior is resumed. After a hard or a soft reset, the first search performed by the start code detector in response to the first DSYNC will always be a search for a sequence start code, after which it stops. After this, the start code detector operates normally.

A pipeline reset terminates the decoding of the current picture. The remaining bits of the picture are flushed from the bit buffer until the next picture start code is detected by the pipeline. At this point normal behavior is resumed, i.e. the pipeline waits for the next picture decoding instruction. No other part of the circuit is affected by a pipeline reset. A pipeline reset would normally be used as part of a manual error recovery procedure. A pipeline reset has no effect if the decoding pipeline is in its idle state.

15.3 Bit buffer and start-code detection (video)

15.3.1 Bit buffer

The transfer of compressed data is carried out using the link DMA engines. Compressed data can be taken from any memory space visible to the CPU and transferred to the relevant elementary stream decoder.

15.3.2 Start code detection

The start code detector operates in parallel with the decoding pipeline. The purpose of this unit is to allow external access to the header data which follows start codes in the input bit-stream. Compressed data is read twice from the bit buffer- once into the pipeline, and once into the start code detector through the 128-byte header FIFO. The transfer of data into the header FIFO does not affect the bit buffer level; only the data transfer into the pipeline can reduce the bit buffer level.

Start code detection is initiated in two ways:

- Automatically whenever the internal event DSYNC occurs. DSYNC is derived from VSYNC as described in Decoding task on page 144. A DSYNC is generated every time the pipeline starts a new picture decoding task.
- By software writing to the VID_HDS register with bit VID_HDS.HDS set.

When start code detection has been started, data is read continuously from the bit buffer into the header FIFO and parsed by the start code detector, which receives the FIFO output data. When a start code is detected, the data scanning stops and the status bit VID_STA.SCH becomes 1. When a start code has been detected, it can be identified by reading the VID_HDF register. The start code detector detects all start codes other than the codes from 0x00000102 through to 0x000001AF. The first slice start code 0x00000101 can be optionally detected to help driver development.

The register VID_HDF should always be read twice to return a 16-bit value. The most significant byte is read first. After detection of a start code, VID_HDF will return one of the 16-bit values shown below:

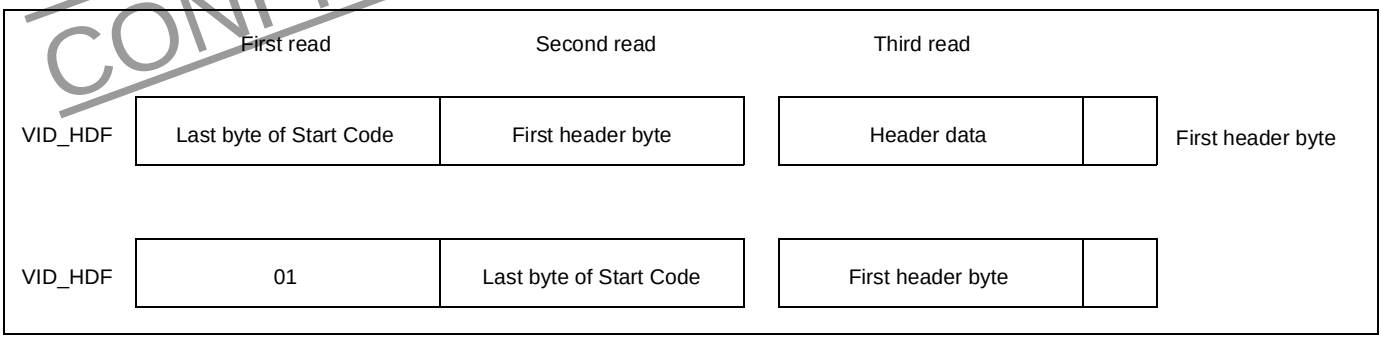


Figure 74 States of VID_HDF after detection of a start code

The first step is to examine the first byte read from VID_HDF. If this contains 0x01, then the start code can be identified by a second read at the same address. If the first byte is not 0x01 then it must be the last byte of the start code and the second byte is the first byte of the header data. In both cases subsequent reads from VID_HDF will give access to the header data which follows the start code.

Scanning for start codes will recommence on the next DSYNC or a write to VID_HDS.HDS. *Whenever a start code has been detected, the* VID_HDF *register must be read in order for the start code detector to restart correctly.* The number of reads before a manual or automatic (DSYNC) restart must always be even.

The first start code search after a hard or soft reset will be a search for a sequence header start code; all other start codes will be ignored. When this start code has been read, all subsequent searches will look for any start codes other than slice start codes.

The two status bits VID_STA.HFE (header FIFO empty) and VID_STA.HFF (header FIFO full) indicate the state of the header FIFO. Reading from HDF must never be performed if VID_STA.HFE is 1. VID_STA.HFF is set whenever the header FIFO contains at least 66 bytes.

The start code detector can also be programmed to stop on the first slice of the picture. This allows the use of the start code search even after reception of the picture start code. All header data that is not used by the application can then be skipped without risk, in order to jump to the next picture start code.

This mode is enabled by setting bit VID_HDS.SOS. To differentiate between first slice start code (00 00 01 01) and other start codes, it is possible to detect at which position (MSB or LSB) the Last Byte of Start code is positioned in the VID_HDF register. Register bit VID_HDS.SCM when set indicates that the Last Byte of Start code is held by the MSB of VID_HDF; it is zero otherwise.

15.3.3 Handling time-stamps

The video decoder accept MPEG-1 system streams and MPEG-2 packet streams.

For video/audio elementary streams, time-stamps contained in the video packet headers are associated with the picture decode time. This is important because the number of pictures which may be stored in the bit-buffer at any instant is unknown, and therefore there is a variable delay between the input of a picture into the bit buffer and its entry into the decoding pipeline.

There is a 24-bit counter at the input and at the output of the CD FIFO - bit buffer - header FIFO chain, as shown in Figure 75. Each time a byte is written into the CD FIFO the counter “CDcount” is incremented. Each time a 16-bit word

is read from the header FIFO, the counter “SCDcount” is incremented. Both of the counters are reset by a hard or soft reset. Both are modulo 2^{24}, that is, the state following FFFFFF is 000000.

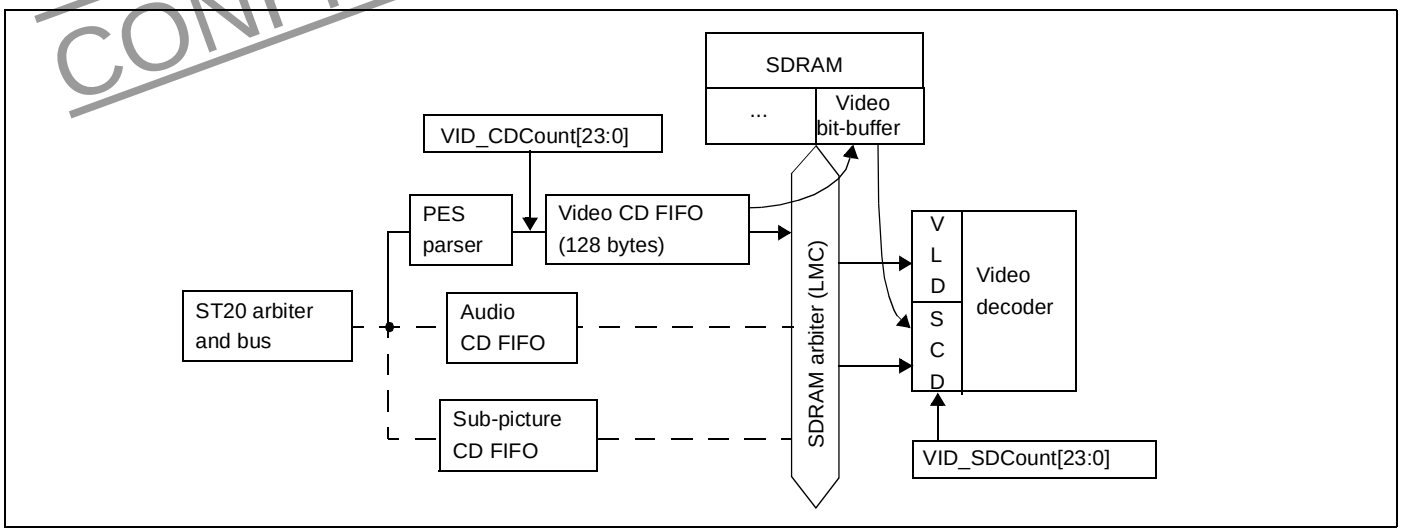


Figure 75 Handling time-stamps with VID_CDCount and VID_SDCount

When the first byte of video data from a new packet containing a time-stamp is written into the CD FIFO, VID_CDcount[23:0] is read.

This value and the time-stamp is recorded in a FIFO. When a picture start code is detected by the start code detector, VID_SCDcount is read. If this value multiplied by two, is greater (modulo 224) than the last CDcount in the FIFO, then the next picture to be decoded is associated with the time-stamp stored at this position of the FIFO.

The “time-stamp association” information is available in the PES_TSx register. The same mechanism is implemented for the “DSM trick mode” association (register PES_TMx).

15.4 Video decoding pipeline control

Note The video decoder only operates in bandwidth reduction mode.

The pipeline is the core of the decoder. It is that part of the circuit which converts the compressed bit-stream data for each picture into a decoded (or reconstructed) picture. These pictures can be frame or field pictures. The operation of the pipeline is controlled picture-by-picture. The decoding of a new picture can potentially start on every VSYNC, but usually the rate of decoding is faster than the VSYNC rate.

The pipeline is controlled by the pipeline controller. When the pipeline controller starts the decoding pipeline a DSYNC signal is issued and VID_STA.PSD is set.

This signal is also sent to the start code detector. When the pipeline has completed its decoding operation, a completion signal is sent to the pipeline controller, which is then able to launch another decoding operation, either immediately or when the next VSYNC occurs.

The pipeline controller interprets certain bits of the decoding instruction, which must be set up by the user before the start of each new task. The remaining bits of the instruction define the decoding task itself.

The pipeline receives its compressed data from the bit buffer. This data is first processed by the variable length decoder (VLD) which regenerates the run/level coded DCT coefficients and the motion vectors (if present) for each macroblock. The picture data is reconstructed by passing the run/level data through the inverse quantizer and inverse DCT blocks.

This is then added to the predictors which have been fetched from the memory taking into account the macroblock prediction modes and motion vectors.

Finally, the decoded picture is written back into the memory, from where it can be accessed by the display unit for output.

The pipeline is also able to skip through picture data for various reasons. The different possibilities are:

- Skip to Next Sequence. This occurs unconditionally on the first instruction execution after a hard or soft reset (see Reset on page 133). Compressed data is skipped until the first picture start code following a sequence start code is found. The pipeline then indicates task completion and waits for a new instruction.
- Skip to Next Picture. This occurs either after a pipeline reset (see Reset on page 133) or when the decoding instruction specifies that one or two pictures should be skipped (see Decoding task on page 144). In the first case compressed data is skipped until the next picture start code is found, after which the pipeline indicates task completion and waits for a new instruction. In the second case, after the skipping operation the decoding of the following picture is started immediately.
- Skip to Next Slice. This occurs after automatic error concealment (see Error recovery and missing macroblock concealment on page 145). Compressed data is skipped until the next slice start code in the picture is found, after which normal decoding resumes.
- Before starting to decode a sequence, certain static parameters must be set up. These are:
- MPEG-1 or MPEG-2 mode selection. Bit VID_PPR2.MP2 must be set for an MPEG-2 sequence, reset for an MPEG-1 sequence.
- Decoded picture size. Register VID_DFW must be set up with the picture width in macroblocks, and register VID_DFS must be set up with the number of macroblocks in the picture.

Decoding is enabled by setting bit VID_CTL.EDC.

15.5 Quantization table loading

The two quantization matrices (intra and non-intra) used by the inverse quantizer must be initialized by the user. There are no built-in quantization matrices. Therefore, they must be loaded either with default matrices or with those extracted from the bit-stream by the ST20.

The quantization tables are double-buffered. This enables one or both tables to be updated without disturbing the decoding task in progress.

The video decoder maintains two bits which record whether one or both of the tables have been modified. A modified table is automatically brought into operation at the start of the next decoding operation, i.e. when the next DSYNC occurs.

After a hard reset, the same pair of tables is always selected. The data previously loaded into the tables is not affected. Other types of reset have no effect on the quantization tables.

The quantization tables are written at the address held in the register VID_QMW. Bit VID_HDS.QMI is used to select the Intra or Non-Intra quantization table; when it is set, the Intra table is selected; when clear the Non Intra table is selected.

15.6 Memory mapping of data

Two types of external SDRAM can be mapped from the STi5518.

- 1 or 2 x 16-Mbit SDRAM
- 1 x 64-Mbit SDRAM

This section discusses video decoder memory (SDRAM) addressing, 32-bit word addressing for the CPU and 64-bit word addressing for FIFO for both of these types of external SDRAM.

15.6.1 Mapping 1 or 2 x 16-Mbit SDRAM

Video decoder memory SDRAM addressing (for 1 or 2 x 16-Mbit SDRAM)

The locations in an SDRAM are addressed row-by-row, bank A then bank B, as shown below:

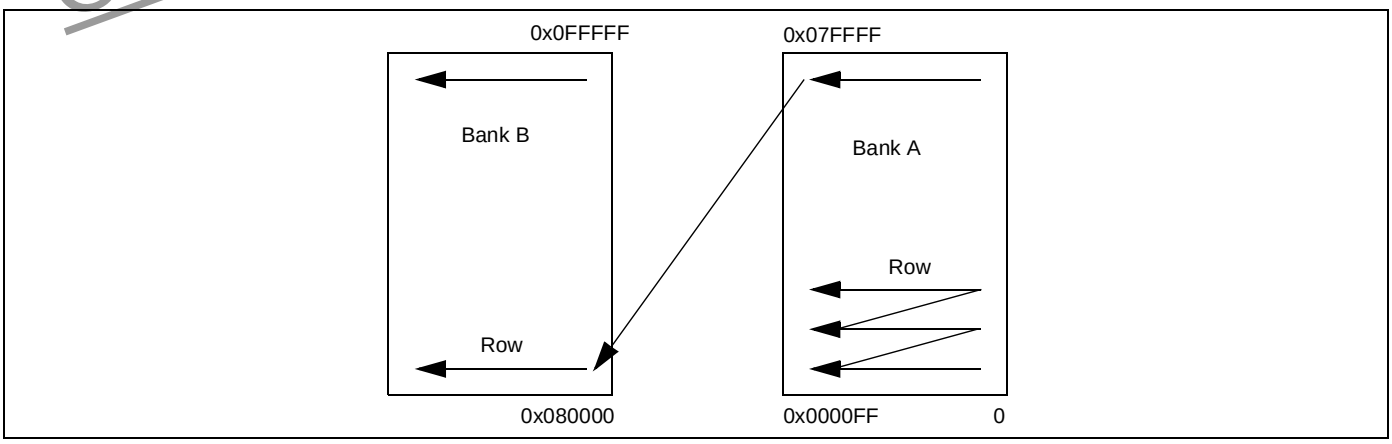


Figure 76 Standard addressing in a SDRAM (16-bit words) for 1 or 2 x 16-Mbit SDRAM

32-bit word addressing for the CPU (for 1 or 2 x 16-Mbit SDRAM)

The CPU accesses the SDRAM by using a 19-bit address for each 32-bit word. It is the task of the SDRAM memory controller to re-map the logical address space of the CPU onto the SDRAM address space.

The logical address map seen by the CPU is different from the one described above. For each row, both banks are used. The addresses seen by the CPU through the SDRAM interface are counted in the following order:

Bank A row0 --> Bank B row0 --> Bank A row1 --> Bank B row1 --> ... and so on, as illustrated in Figure 77:

When using a second SDRAM chip, addresses continue in a similar way, starting from the next address above the first SDRAM, as illustrated in Figure 77 on page 138. A maximum of two SDRAM chips is supported.

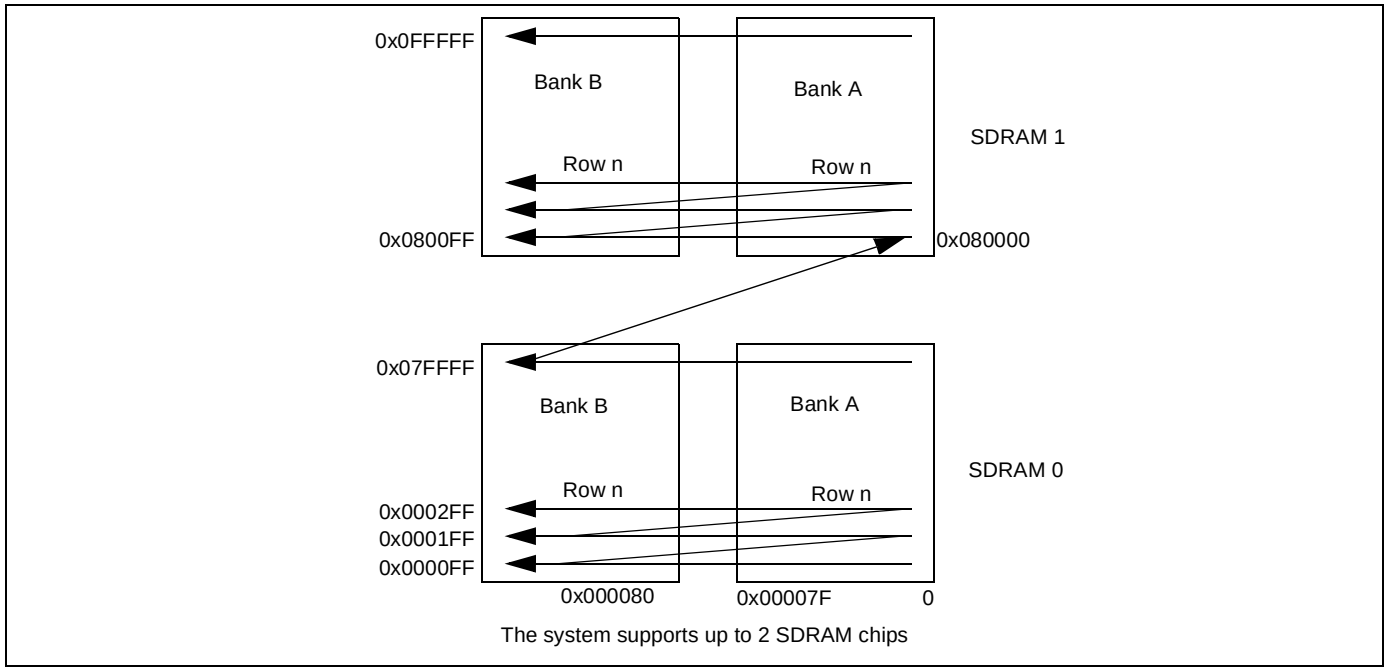


Figure 77 32-bit word addressing, as seen by the CPU for 1 or 2 x 16-Mbit SDRAM

64-bit word addressing for FIFO processes (for 1 or 2 x 16-Mbit SDRAM)

The video decoder uses circular buffers mapped into external SDRAM which act as software FIFOs. The processes pertaining to these circular buffers are managed with a 64-bit granularity. The memory mapping for these buffers is similar to that of the CPU and is shown in Figure 81 on page 141.

When using a second SDRAM chip, addresses continue in a similar way, starting from the next address above the first SDRAM. A maximum of two SDRAM chips is supported.

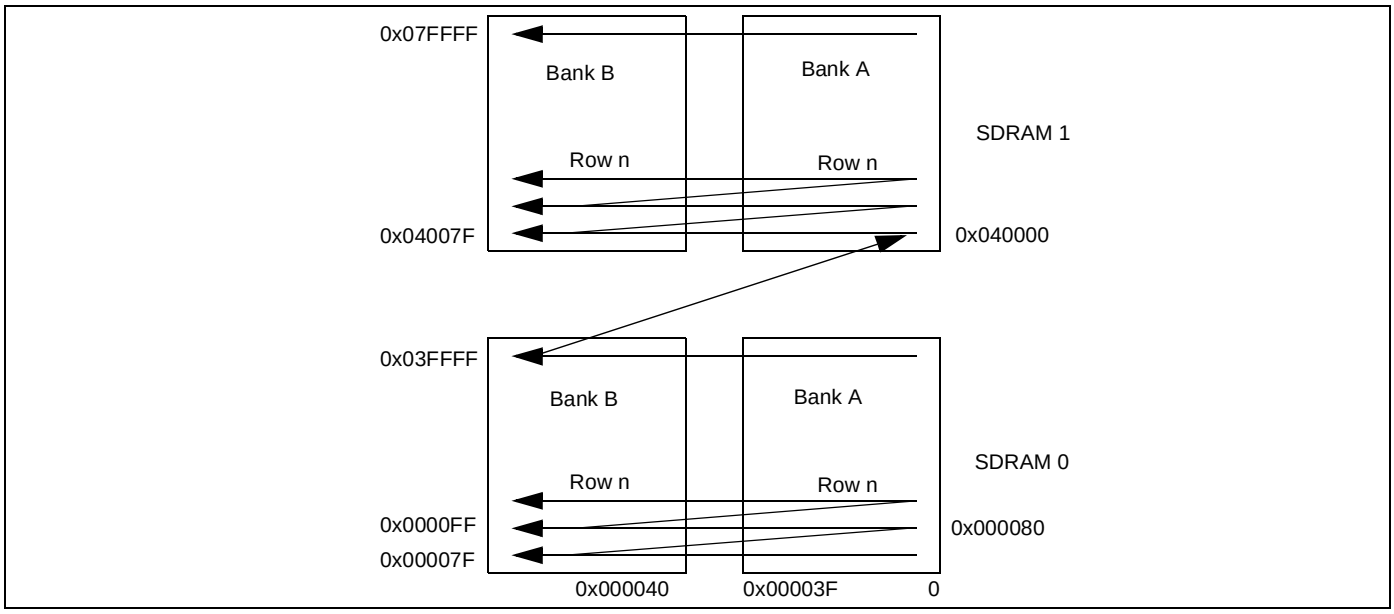


Figure 78 64-bit word addressing for FIFO processes for 1 or 2 x 16-Mbit SDRAM

15.6.2 Mapping 1 x 64-Mbit SDRAM

Video decoder memory SDRAM addressing (for 1 x 64-Mbit SDRAM)

The locations in an SDRAM are addressed row-by-row, bank A, B , C then bank D, as shown below:

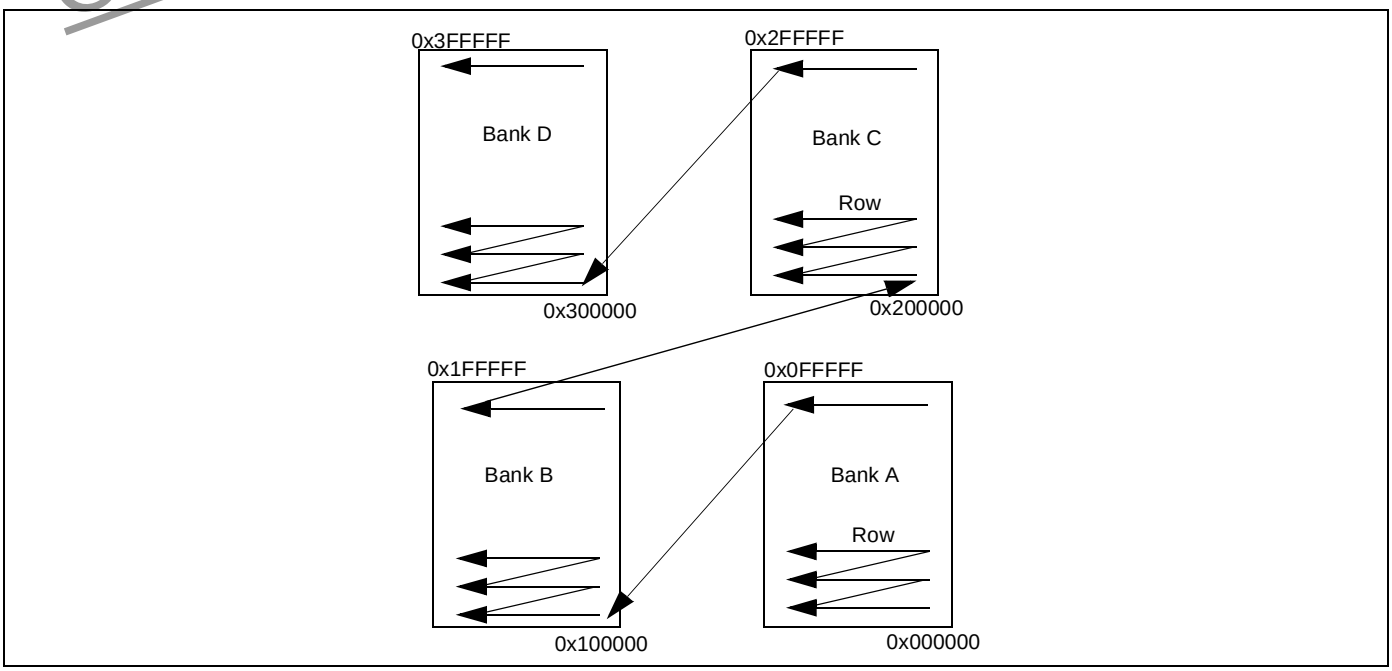


Figure 79 Standard addressing in a SDRAM (16-bit words) for 1 x 64-Mbit SDRAM

32-bit word addressing for the CPU (for 1 x 64-Mbit SDRAM)

The CPU accesses the SDRAM by using a 21-bit address for each 32-bit word. It is the task of the SDRAM memory controller to re-map the logical address space of the CPU onto the SDRAM address space.

The logical address map seen by the CPU is different from the one described above. For each row, both banks are used. The addresses seen by the CPU through the SDRAM interface are counted in the following order:

Bank A row0 --> Bank B row0 --> Bank A row1 --> Bank B row1 --> ... --> Bank A row 4005 --> Bank B row 4005, --> Bank C row 0 --> Bank D row 0... and so on, as illustrated in Figure 80 on page 141:

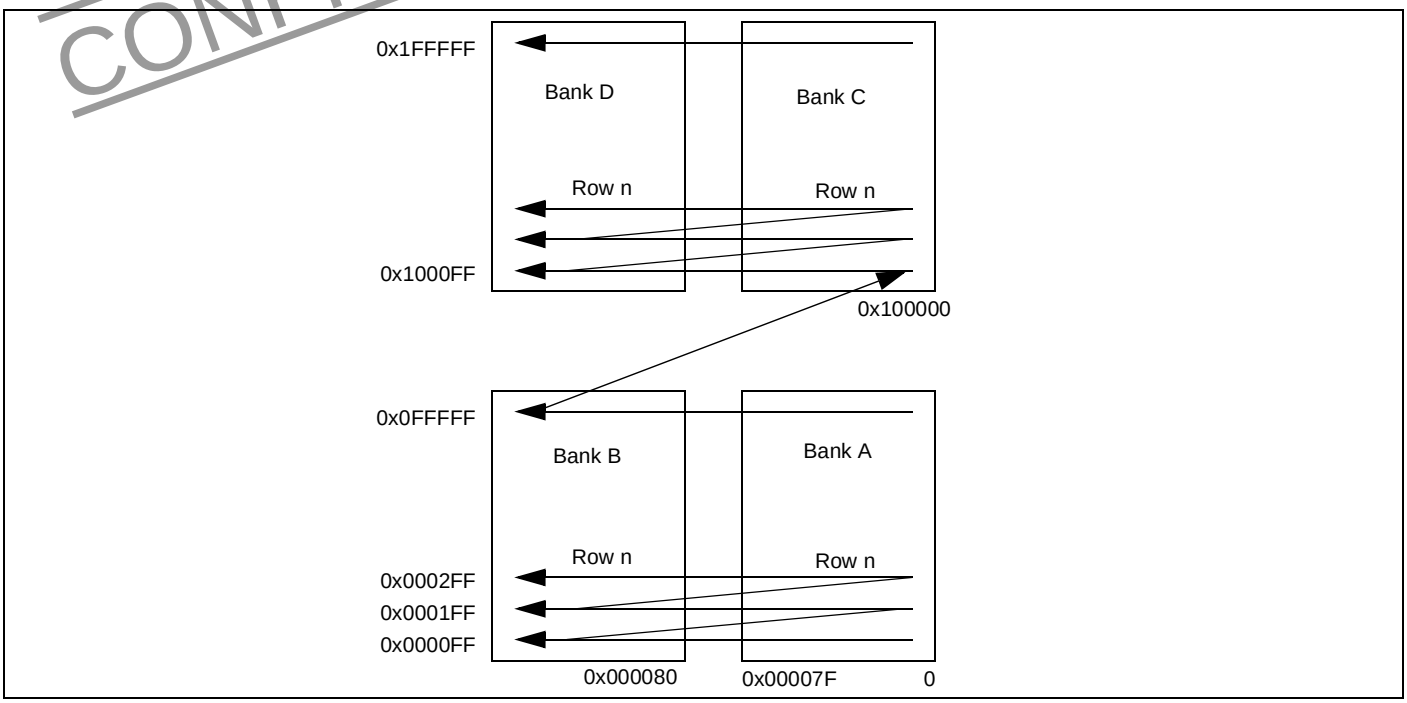


Figure 80 32-bit word addressing, as seen by the CPU for 1 x 64-Mbit SDRAM

64-bit word addressing for FIFO processes (for 1 x 64-Mbit SDRAM)

The video decoder uses circular buffers mapped into external SDRAM which act as software FIFOs. The processes pertaining to these circular buffers are managed with a 64-bit granularity. The memory mapping for these buffers is similar to that of the CPU and is shown in the figure below.

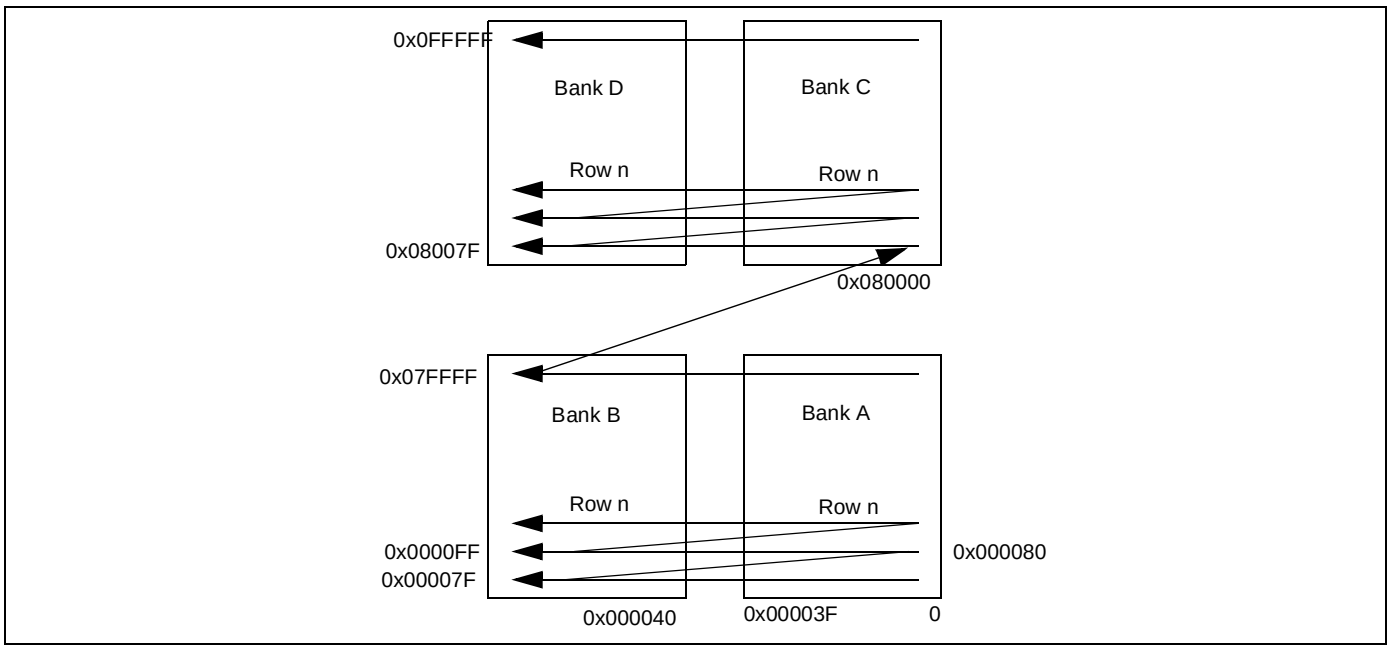


Figure 81 64-bit word addressing for FIFO processes for 1 x 64-Mbit SDRAM

15.6.3 Memory segments

The circular-buffer start and end pointers are programmed by the user, in segments, where each segment is 256 bytes. The values in the configuration registers are numbers of segments. For example a value of 4 means 4 x 256 bytes = 1kbyte or 128 x 64-bit words. This would result in a pointer pointing to a 64-bit word address of 128 (0x80). This address would be physically mapped to the first word in the second row of bank A of SDRAM 0, as shown below;

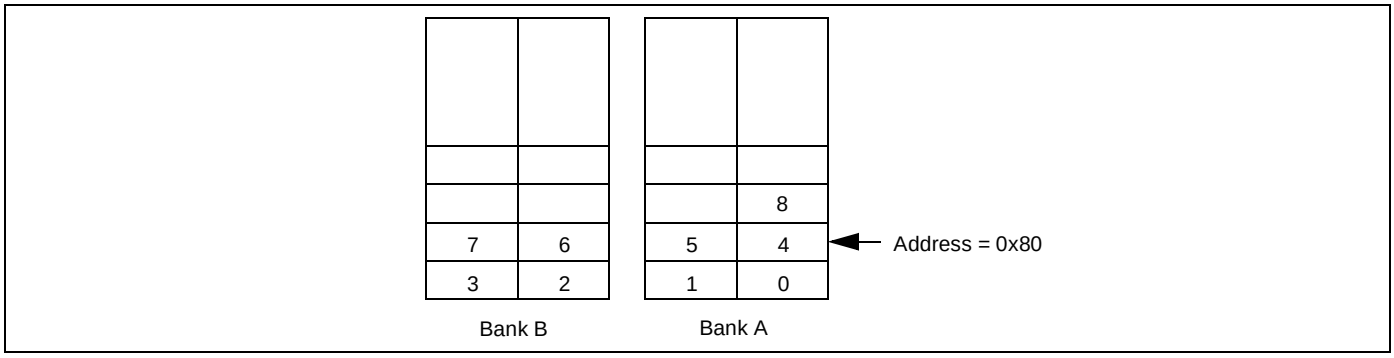


Figure 82 SDRAM segments as seen by the user

15.6.4 Arrangement of pixel-pairs inside a luma SDRAM row

Every SDRAM row in a luma frame contains 256 16-bit words and can store up to two luma macroblocks. Every 16-bit word contains a pair of horizontally adjacent luma pixels. The row itself stores a pair of horizontally adjacent luma macroblocks. The pixel pairs are arranged in line order; the first 16 words store the first line of pixels for the two macroblocks, the next 16 words the second line and so on, as shown below:

16-bit word addresses in SDRAM row	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Arrangement of luma pixel pairs	Y = 2 pixels	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
⋮	Macroblock 0								Macroblock 1							
16-bit word addresses in SDRAM row	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
Arrangement of luma pixel pairs	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 83 Arrangement of pixel pairs in a luma SDRAM row

15.6.5 Arrangement of pixel-pairs inside a chroma SDRAM row

Every SDRAM row in a chroma frame contains 256 16-bit words and can store up to four chroma macroblocks. Every 16-bit word contains a pair of horizontally adjacent 8-bit chroma pixels.

The row stores pixel pairs in line order for macroblocks 0 and 1 and then macroblocks 2 and 3. The Cb and Cr words are interleaved two by two in the linear addressing order, as shown below:

16-bit word addresses in SDRAM row	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Arrangement of luma pixel pairs	Cb = 2 pixels	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr
⋮	Macroblock 0								Macroblock 1							
16-bit word addresses in SDRAM row	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
Arrangement of luma pixel pairs	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr
16-bit word addresses in SDRAM row	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Arrangement of luma pixel pairs	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr
⋮	Macroblock 2								Macroblock 3							
16-bit word addresses in SDRAM row	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
Arrangement of luma pixel pairs	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr	Cb	Cb	Cr	Cr

Figure 84 Arrangement of pixel pairs in a chroma SDRAM row

15.7 Using picture pointers

Before decoding a picture, the following frame buffer pointers must be set up:

- VID_RFC, VID_RFP for reconstructed frame pointers for chroma and luma. These pointers define the memory buffer to which the decoded picture is written.
- VID_FFC, VID_FFP for forward prediction frame pointers for chroma and luma. These pointers define the areas in memory from which the predictors are fetched.
- VID_BFC, VID_BFP for backward prediction frame pointers for chroma and luma. These pointers define the areas in memory from which the predictors are fetched.

The displayed frame pointers, VID_DFC, VID_DFP, are described in Sub-picture display on page 153.

The following rules must be followed when using prediction-frame pointers:

1 Pictures are always stored as frames of interleaved lines. Therefore, to access a field (top or bottom), the starting address of the frame must be defined.

2 P-frame picture (frame, field or dual-prime prediction): VID_FFP and VID_FFC are set to the address of the predictor frame (in which the two predictor fields lie). VID_BFP and VID_BFC are not used.

3 B-frame picture (frame or field prediction): VID_FFP and VID_FFC are set to the address of the forward predictor frame (in which the two predictor fields lie). VID_BFP and VID_BFC are set to the address of the backward predictor frame (in which the two predictor fields lie).

4 P-field picture (field, 16 x 8 or dual-prime prediction): When decoding either field, VID_FFP and VID_FFC are set to the address of the previous decoded I or P frame. VID_BFP and VID_BFC are not used.

5 B-field picture (field or 16 x 8 prediction): VID_FFP and VID_FFC are set to the address of the frame in which the two forward predictor fields lie. VID_BFP and VID_BFC are set to the address of the frame in which the two backward predictor fields lie.

6 I-pictures: For I-picture decoding, no predictors are necessary, but VID_FFP and VID_FFC must be set to the address of the last decoded I- or P-picture for use by the automatic error concealment function.

15.8 Video pipeline

15.8.1 Decoding task

A task is a single picture decoding operation. A task is specified by the task description or instruction, which is set up before the decoding of each picture. A task starts when the internal signal DSYNC is generated. A task completes (the decoder becomes idle) when the picture is entirely reconstructed in the memory and the picture header of the following picture is detected by the pipeline. The instruction is double buffered, so that during execution of a decoding task, the instruction for the next task can be set up by the CPU. When the next instruction is activated, a DSYNC can be generated and the next decoding task started. The buffering mechanism is illustrated in the figure below. Note that some instruction bits are latched by VSYNC, others by a signal from the pipeline controller “new instruction”.

The instruction is written into registers VID_PPR1 and VID_PPR2. If a new instruction is not written, the task descriptor will be the same as the previous one.

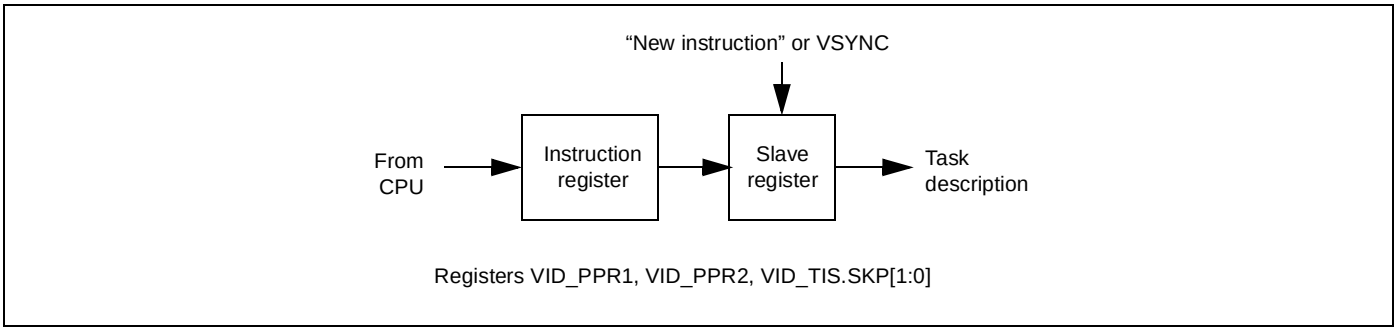


Figure 85 Instruction buffering

Normally, it is VSYNC starts the execution of a new instruction and, thus, the generation of DSYNC. If however, a VSYNC occurs before task completion (i.e. before the pipeline becomes idle), the start of the next task is delayed until the present one is completed. In this way, the picture decoding can extend beyond the nominal period by one or two VSYNC periods.

Three status bits (and thus interrupts) are associated with pipeline control:

- VID_STA.PSD indicates the occurrence of a DSYNC.
- VID_STA.PII indicates that the pipeline is idle.
- VID_STA.DEI indicates that the decoder is idle, i.e. the pipeline is idle and the next picture start code has been found.

The operation of the pipeline controller is shown in the diagram below. The abbreviations used in the diagram are explained in the following table.

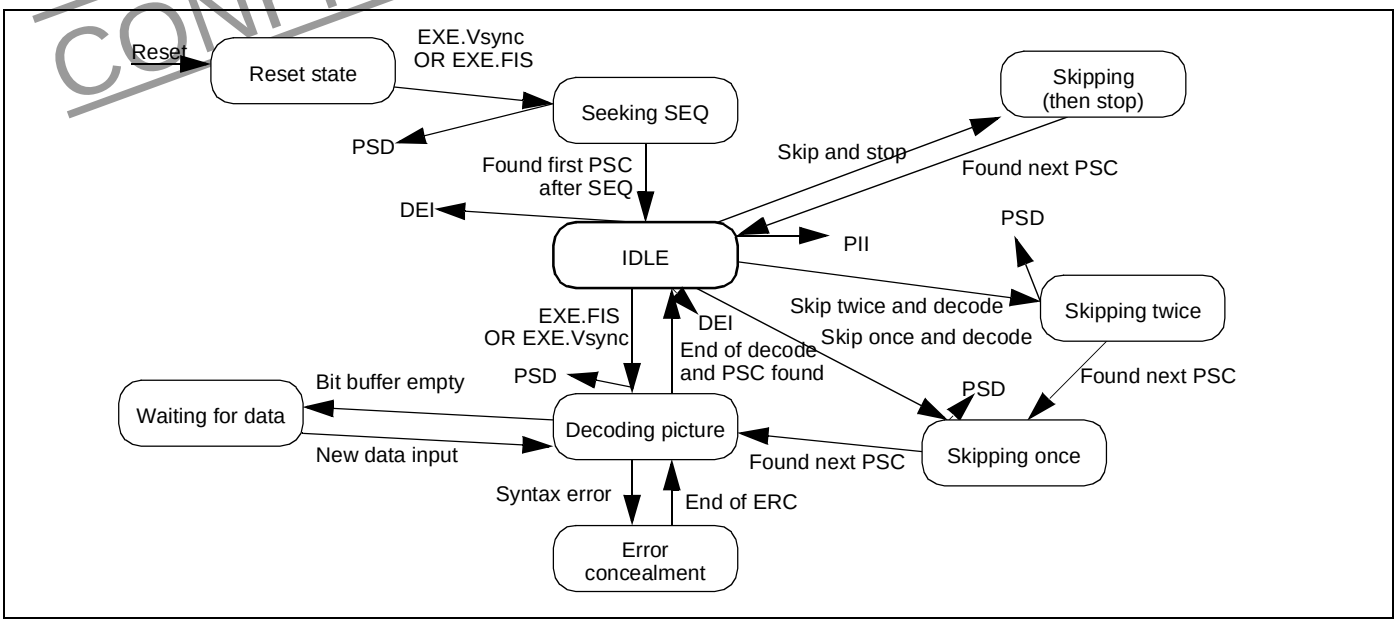


Figure 86 Task control state diagram

Abbreviation	Meaning
ERC	Automatic error concealment
EXE.FIS	Both VID_TIS.EXE and VID_TIS.FIS are set
EXE.Vsync	Bit VID_TIS.EXE set when external VSYNC occurs
PII	Pipeline idle
DEI	Decoder idle interrupt generated
PSC	Picture start code
PSD	Pipeline start decode interrupt generated
SEQ	Sequence start code
Skip and decode	VID_TIS = EXE | SKP[01] and VSYNC occurred
Skip and stop	VID_TIS = EXE | SKP[11] and VSYNC occurred
Skip twice and decode	VID_TIS = EXE | SKP[10] and VSYNC occurred

Table 72 State transition abbreviations

The instruction bits which affect state transitions are VID_TIS.EXE and VID_TIS.FIS. The events to which the controller responds are:

- VSYNC, which could be a *VSYNC top* or a *VSYNC bottom* and
- IDLE representing the idle state of the pipeline.

15.8.2 Error recovery and missing macroblock concealment

For the video decoder, there are four levels of error-detection and recovery:

- bit-stream syntax error detection with the option of automatic missing macroblock concealment;
- bit-stream semantic error detection with the option of automatic concealment or skip to the next picture;

- pipeline overflow or underflow error detection;
- user-initiated skip to next sequence using soft reset.

Syntax error detection and concealment

In normal operation of the STi5518, error concealment must always be enabled, i.e. VID_CTL.EDC should be reset.

If the VLD detects a syntax error in the bit-stream, the pipeline will copy macroblocks from the previous picture using the motion vectors reconstructed for the previous row of macroblocks in the current picture, while scanning the bit-stream until a slice start code is detected. At this point normal decoding resumes. If the slice in which the error occurred was the last one in the picture, concealment will continue until the end of the picture, at which time the pipeline stops normally (assuming that the following picture start code is intact).

The concealment macroblocks are accessed using the pointers VID_FFP and VID_BFP. Lost macroblocks in the first row are copied directly from the previous pictures (i.e. as P-macroblocks with zero motion vectors). If an intra picture is coded with concealment motion vectors, these will be used. If not, then the concealment will be a simple copy from the previous picture using zero vectors. Even in intra pictures, the pointer VID_FFP must be set up.

The table below gives the rules that are used for fetching concealment macroblocks.

Picture type	Macroblock type	Fetch rule
I-picture	I-macroblock without vectors	Copy with zero motion.
	I-macroblock with vectors	Copy as forward predicted macroblock.
P-picture	I-macroblock without vectors	Copy with zero motion.
	I-macroblock with vectors	Copy as forward predicted macroblock.
	P-macroblock	Copy using stored vector.
	P-field-macroblock	Copy in field mode using both vectors.
	Skipped macroblock	Copy with zero vector.
	Dual-prime macroblock	Copy using stored vector.
B-picture	I-macroblock without vectors	Copy with zero motion.
	I-macroblock with vectors	Copy as forward predicted macroblock.
	Forward macroblock	Copy using stored vector.
	Backward macroblock	Copy using stored backward vector.
	Bidirectional macroblock	Only the forward vectors are stored, concealed as forward macroblock.
	Skipped macroblock	Copy in frame mode using the same mode and vectors as the previous macroblock.

Table 73 Rules for fetching concealment macroblocks

If an error is detected in the bit-stream before it enters the parser, then an error start code can be inserted into the bit-stream in order to initiate concealment. However, when doing this there are certain restrictions on the placement of the error start code in order to avoid emulation of other start codes. An Application Note is available on this topic.

Overflow or underflow error

An overflow error occurs whenever the pipeline reconstructs more macroblocks than are defined by the decoded picture size, VID_DFS. This can occur when the input data to the decoder contains undetected errors. This condition is signalled by bit VID_STA.SER. Decoding is automatically halted when this error is detected. In order to restart decoding a pipeline reset must be performed.

An underflow error occurs whenever the pipeline reconstructs less macroblocks than are defined by the decoded picture size, VID_DFS. This condition is signalled by bit VID_STA.PDE. Decoding is automatically halted when this error occurs. In order to restart decoding a pipeline reset must be performed.

15.9 PES parser

Description

The PES parser is situated between the ST20 arbiter/bus and the compressed data FIFOs of the video/audio core. It has a 100 Mbits/sec (max burst) bit rate, and allows the following input streams:

- Packetized PES (MPEG-2), ISO 13818-1
- MPEG-1 system layer (ISO 11172-1)

The MPEG2 PES &MPEG1 system parser accepts PES streams in the same way that pure audio or video streams are accepted.

For packetized elementary stream data which is demultiplexed from a transport stream (MPEG-2), the data stream consists of concatenated, incomplete packets of audio, and video PES. To handle this configuration, the STi5518 contains two separate parsers: one for the audio (audio PES parser in audio decoder) and one for the video (MPEG2 PES & MPEG1 system parser).

As the audio or video data is input, it is demultiplexed by each parser and the audio / video streams are placed in their respective buffers. For program stream data or MPEG-1 systems stream data, the audio and video packets are complete so that a single parser (MPEG2 PES & MPEG1 system parser) can be used. The packets are internally separated into video and audio streams. If required, the two parsers can still be used but the packets must be separated by the ST20 (recommended mode). See the figure below.

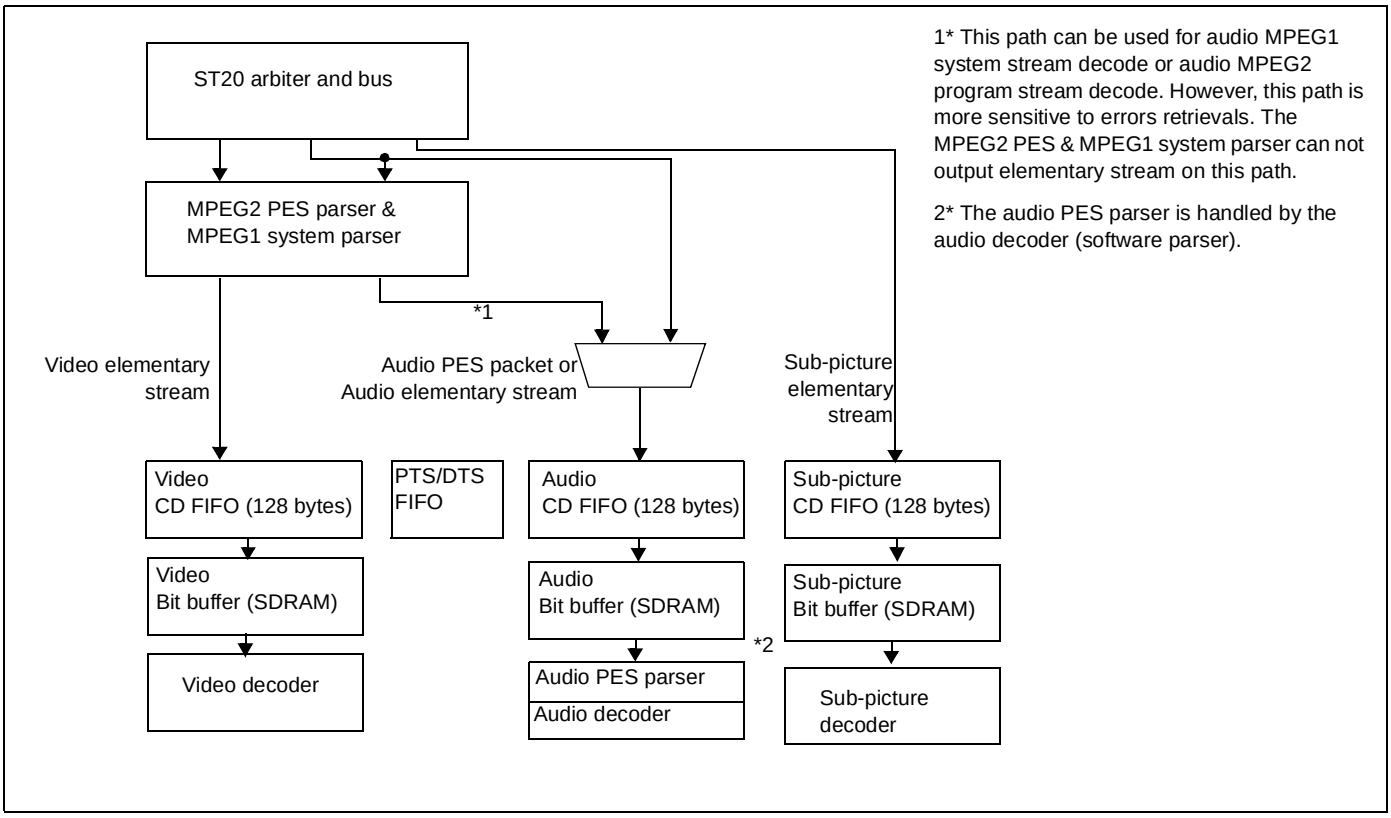


Figure 87 System parser internal architecture

When the MPEG2 PES & MPEG1 system parser is configured to accept MPEG-2 PES audio/video packets (mode 3), the parser extracts audio & video bit-streams in accordance with the programmed stream ID. For the audio stream this is contained in PES_CF1; for the video stream in PES_CF2. Any audio or video packets which are not selected for

decode (because their stream IDs do not match the programmed values) are discarded. The audio PES are output on path 1 (see Figure 87 on page 147).

When used for decoding program streams or MPEG-1 system streams, the audio, video and system level data are automatically separated internally to the MPEG2 PES & MPEG1 system parser. Time-stamp association is supported by the decoder.

During parsing, decode or display time-stamps (DTSs or PTSs, selected by PES_CF1.SDT) are stored in an internal FIFO. When the image corresponding to these time-stamps is decoded (or, in the case of video, about to be decoded) the corresponding time-stamp is made available and a flag or interrupt is given.

Functional modes

The parsers are enabled by setting register PES_CF2.SS for the video parser, and registers STREAMSEL/ DECODESEL for the audio parser. Depending on the required mode, one or both of the parsers are required.

Four different modes can be configured with the two mode bits of register PES_CF2[7:6]:

- Mode 0: Automatic configuration. The parser examines the incoming stream and self-configures for decode. The mode selected can be read back from PES_TM2[1].
- Mode 1: MPEG-1 system stream decode. Single data strobe input format. The audio elementary stream is extracted by the MPEG-2 PES PARSER & MPEG-1 SYSTEM PARSER block and sent to the CD audio FIFO.
- Mode 2: MPEG-2 PES decode. Twin data strobe input format. The video PES stream and the audio PES stream are sent separately and respectively to MPEG-2 PES PARSER & MPEG-1 SYSTEM PARSER block and audio CD FIFO. This is the most common way to enter data into the circuit.
- Mode 3: MPEG-2 whole PES Audio/Video Packets. Single data strobe input format. This is used to decode MPEG-2 program streams. The audio PES are output on path 1 (see Figure 87 on page 147) extracted by the MPEG-2 PES PARSER & MPEG-1 SYSTEM PARSER block and send to the CD audio FIFO.

The video parser is reset by setting PES_CF2.SS to 0.

15.10 Enhanced trick-modes

DVD trick-modes, especially backward-mode, require more video decoding flexibility than standard MPEG applications. The STi5518 supports the following trick-mode features:

- Programmable video CD (Compressed Data) FIFO pointer
- Programmable SCD (Start Code Detector) pointer
- Programmable VLD (Variable Length Decoder) pointer

The figure below illustrates the video decoder features which enhance DVD trick-modes.

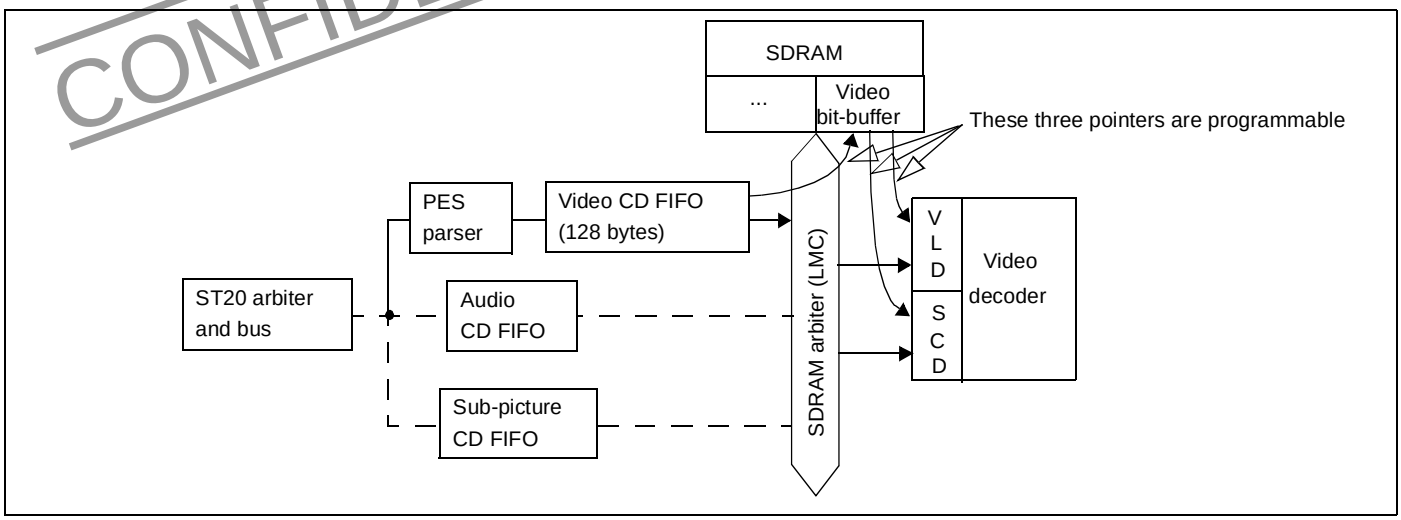


Figure 88 Enhanced trick-mode support

Programming a video CD FIFO pointer

The video CD FIFO destination is programmed inside the SDRAM shared memory (up to 64 Mbits).

1 Set register bit VID_TP_LDP.TM to “1”.

2 Flush the FIFO by writing 64 bytes (0xff) to the CD FIFO.

3 Write a new 20-bit CD pointer value to the VID_TP_CD register (this is a 3 x 8-bit register).

4 Set bit CWL of register VID_CWL to “1”. This starts the FIFO reset mechanism.

Status bit CWR of the VID_ITS register indicates that the CD FIFO is ready for transfer into SDRAM.

Register VID_TP_CALINIT can be used for overwrite protection.

Programming an SCD pointer

1 Set register bit VID_TP_LDP.TM to “1”.

2 Write a new 20-bit SCD pointer value to the VID_TP_SCD register (this is a 3 x 8-bit register).

3 Set bit STL of register VID_STL to “1”. This starts the FIFO reset mechanism.

Status bit SWR of the VID_ITS register indicates that the SCD FIFO is ready for transfer into SDRAM.

Programming a VLD pointer

1 Set register bit VID_TP_LDP.TM to “1”.

2 Set register VID_TRF with the temporal reference, and set register VID_TP_VLD with a new read-pointer.

3 Set bit TR_TML of register VID_TRF to “1”.

Status bit TR_OK of register VID_ITS indicates that the VLR read pointer has been loaded into the memory controller, and that the VLD is ready to decode the selected picture.

16 Sub-picture decoder

16.1 Introduction

A hardware sub-picture decoder is integrated in the STi5518. The sub-picture bit-buffer that contains sub-picture units (SPU) is integrated in SDRAM external memory and has a programmable size. Its position and size can be set in multiples of 2 Kbytes. The sub-picture bit buffer is set-up at power-up reset. During player operation, its size and location are constant.

Compressed data is input into the bit-buffer using a DMA, or by a CPU write. Once control is given to the sub-picture decoder, it runs autonomously until stopped by software control. The sub-picture decoder can decode complete sub-picture units - which consist of a sub-picture unit header, compressed pixel data and the display control sequence table - without any interaction from the CPU.

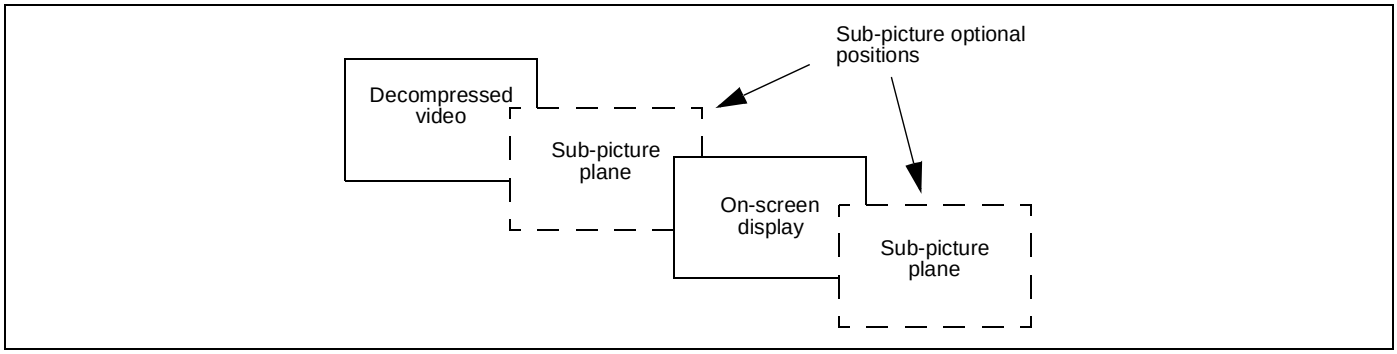


Figure 89 Display planes

The sub-picture decoder can also be used as a hardware cursor unit. The priority of the sub-picture is first raised by programming a register so it is in front of all the other display planes. A cursor can be defined using an optionally compressed (run-length encoded) bitmap stored in external SDRAM. The bitmap can be any size up to a full screen. Per-pixel alpha-blending factors can be defined for each cursor to provide anti-aliasing with the background. The cursor is then moved around using register writes into X and Y coordinate registers.

The figure below illustrates the sub-picture decoder architecture.

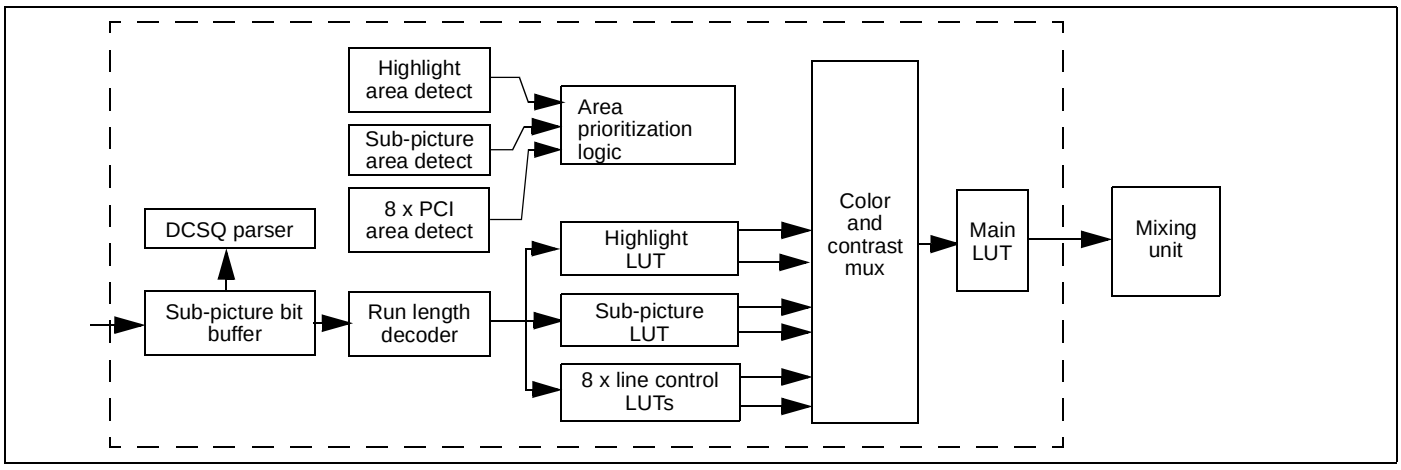


Figure 90 Sub-picture unit architecture

16.2 Buffer management and pointers

There are four registers which control the sub-picture bit buffer read and write processes, as shown in Figure 91:

- Bit buffer base address (VID_SPB). This is an offset relative to the ST20 SDRAM base address. It is programmed in units of 2 Kbytes.
- Bit buffer end address (VID_SPE). This address is an offset relative to the ST20 SDRAM base address. It is programmed in units of 2 Kbytes.
- Bit buffer read pointer (VID_SPRead). It is set by software for each sub-picture unit. This is done before control is given to the sub-picture hardware decoder. This register is double buffered. The shadow register is updated with each field VSYNC event. This pointer is an offset relative to the ST20 SDRAM base address. It is programmed in units of 64-bit words.
- Bit buffer write pointer (VID_SPWrite). It is set by the ST20 before transferring each sub-picture unit into the bit buffer. This pointer is an offset relative to the ST20 SDRAM base address. It is programmed in units of 64 bit words.

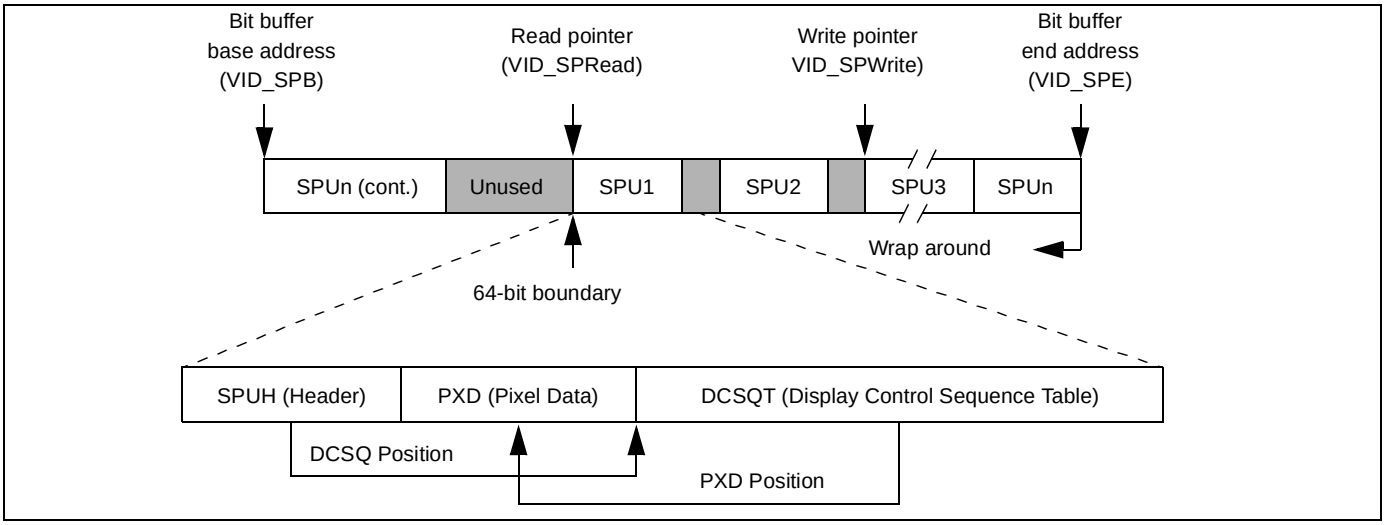


Figure 91 Buffer management

16.3 Operation

Each sub-picture unit data-buffer start-position is programmed using the register VID_SPWrite. Subsequently the sub-picture header, the pixel data, the display control sequences are sent via fifos to the sub-picture decoder. Write into fifos is done by DMA or by CPU write. Only data belonging to the sub-picture unit (SPUH, PXD, DCSQT) are transferred into the sub-picture bit buffer. Sub-picture pack headers are removed by the software demultiplexor.

The decoder reads the header of the first packet (see Figure 92) and jumps to the first display control sequence using the command pointer.

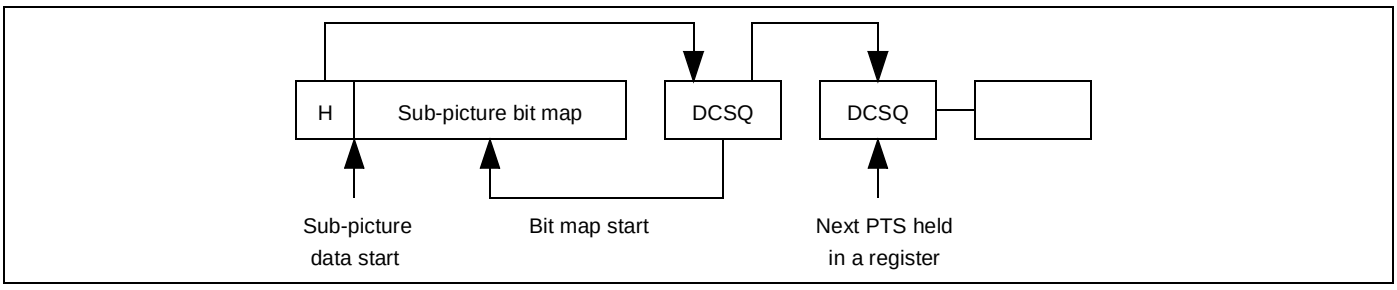


Figure 92 Sub-picture unit structure

The instructions found in the DCSQ packets enable the sub-picture unit to program the palettes, set mixing factors etc. for each region. The DCSQ packets also contain a time stamp which indicates to which image the sub-picture information refers.

This information is related to a local time for this sub-picture unit. The micro should enable a given sub-picture unit at the right global time via some registers: data buffer start position, start sub-picture unit status bit.

The overall control of the sub-picture decoder is performed by software.

The final information in the DCSQ packet is the region size (rectangle) and the relative position, in bytes, of the bit-map start.

A key point here is that the sub-picture decoder must read beyond the end of the DCSQ packet in order to verify the next PTS. With this information held in a register, the sub-picture decoder knows, in advance, when to change the DCSQ or bit-map information. The sub-picture unit simply executes the same DCSQ until the image corresponding to the next time-stamp is reached.

This is done at the beginning of every field so that the sub-picture decoder can load all the relevant information from DCSQ before the first sub-picture pixel is required.

The sub-picture region declaration is held in registers in the decoder so that the sub-picture decoder is turned on and off at the correct position on the screen (see Figure 93). The bit-map start-pointer indicates where, in the bit map data, to start decoding. When the correct image, corresponding to the local time stamp contained in the DCSQ, should be displayed the sub-picture controller enables the sub-picture decode for that image.

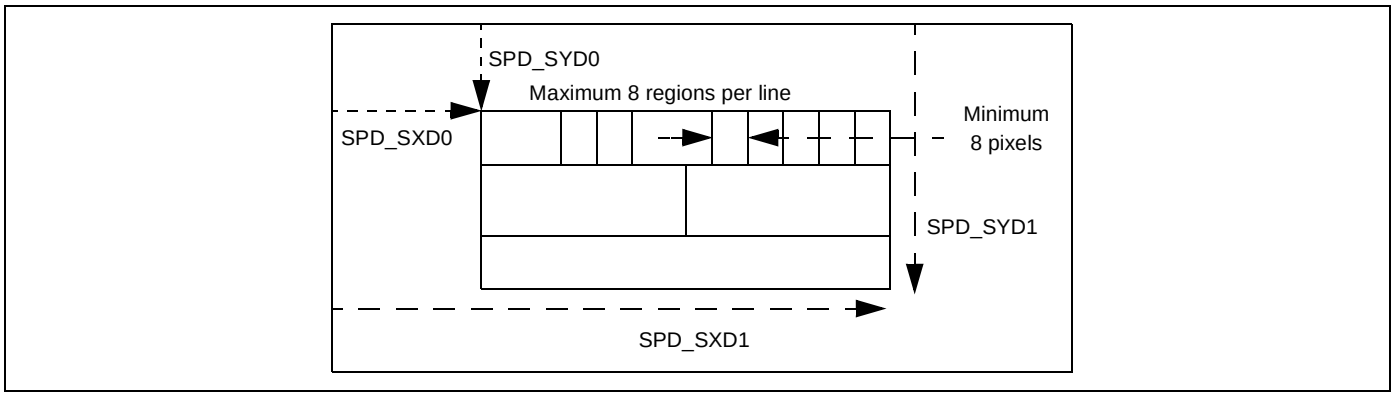


Figure 93 Sub-picture region declaration

A pause mode is defined in the sub-picture decoder. As explained previously, the sub-picture decoder is autonomous within a sub-picture unit.

This means that the DCSQ switching is timed automatically using an internal 90 kHz clock. During video trick modes, where the video stream may be frozen or slowed down the same thing should be possible with the sub-picture decoder in order to maintain the synchronization between the two streams.

A pause mode is implemented for the sub-picture decoder which stops the 90 kHz counter and therefore pauses the sub-picture decoder. This is controlled using the P field in the SPD_CTL1 register and is synchronized to the VSYNC signal. This control bit can therefore be used as a pause and a single step control bit.

The sub-picture decoder registers are put together in the sub-picture memory map except:

- sub-picture software reset (register SPD_SPR),
- sub-picture pause mode (SPD_CTL1.P bit),
- sub-picture FIFO full (bit 18 of VID_ITS and VID_STA register).

16.4 Sub-picture display

16.4.1 Look-up tables

There are 11 look-up tables inside the sub-picture decoder:

- 1 highlight LUT (2 bits to 4 bits mapping)
- 1 sub-picture LUT (2 bits to 4 bits mapping)
- 8 PCI LUTs (2 bits to 4 bits mapping)
- 1 main LUT (4 bits to 24 bits mapping)

The sub-picture and PCI LUTs are automatically supplied by the decoder itself (sub-picture commands contained in the SPU). The highlight and main LUTs need to be loaded by the ST20 (SPD_HCN, SPD_HCOL, SPD_LUT registers).

The output of the sub-picture main LUT is mixed with the other planes. The contrast value between these two sources is set by the SET_CONTR DCSQ command, by the PCINFs of a CHG_COLCON command or by a highlight color information (the highlight LUT has the highest priority, followed by the PCI LUTs. The sub-picture LUT has the lowest priority).

The mixed video is a 24 bits Y, Cr, Cb video where:

- $Y_{MIXED} = [Y_{PLANES} \times (16 - k) + Y_{SUBP} \times k] / 16$
- $Cr_{MIXED} = [Cr_{PLANES} \times (16 - k) + Cr_{SUBP} \times k] / 16$
- $Cb_{MIXED} = [Cb_{PLANES} \times (16 - k) + Cb_{SUBP} \times k] / 16$
- k = 0 if contrast value from high light, sub-picture, PCI LUTs = 0
- k = contrast value + 1 if contrast value > 0

16.4.2 Sub-picture areas

The active sub-picture decoding area can be 720 x 576 or 720 x 480 pixels. In order to align the sub-picture decoding area with the video decoding area, the upper left corner of the active sub-picture decoding area has to be set by software, using the registers SPD_XD0 and SPD_YD0. The same semantics have been defined as for the video decoder, as shown in Figure 94 . The active sub-picture display area is defined in a similar manner, using the SPD_SXD0, SPD_SYD0, SPD_SXD1 and SPD_SYD1 registers.

The highlighted area is defined by SPD_HLS, SPD_HLSY, SPD_HLEX, SPD_HLEY registers, and is set by software.

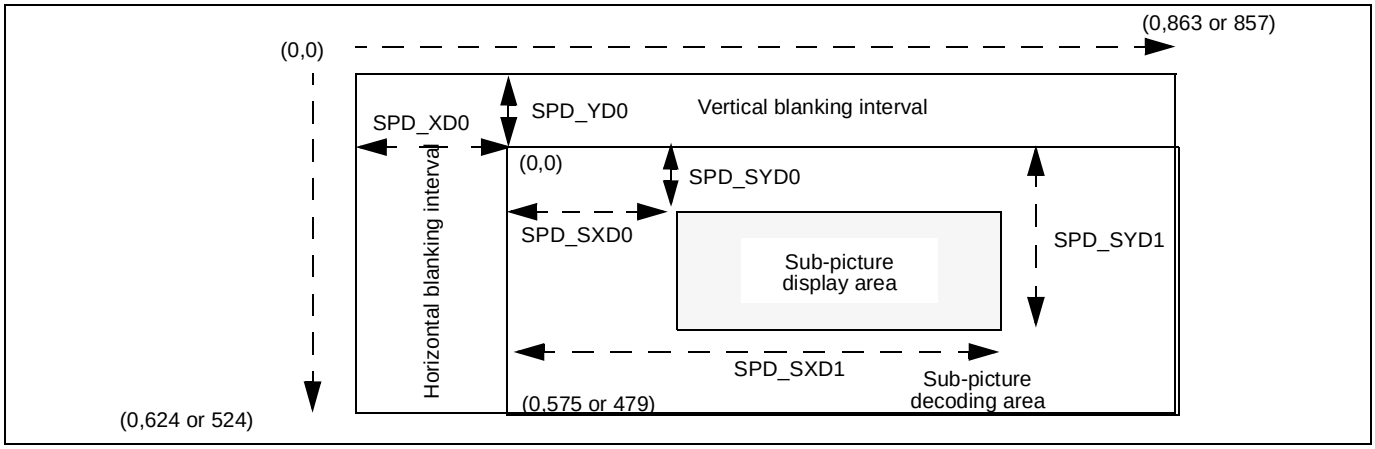


Figure 94 Sub-picture areas

17 Overlay graphics and texts

17.1 Introduction

The STi5518 has integrated OGT (overlay graphics and texts) hardware used for SVCD. The data stream is similar to the sub-picture stream. The OGT bit stream is made up of OGT pages, where each page contains a header sequence followed by a bitmap packet (1 picture/page). OGT compressed data are stored in a bit-buffer with programmable position and size (in any multiple of 2 Kbytes). The OGT decoder can decode a complete page of 2 fields in accordance with IEC SC100B/NP177/PTD-003 SVCD specifications.

The figure below illustrates the OGT model, with an OGT page placed in a screen and 2 highlight areas.

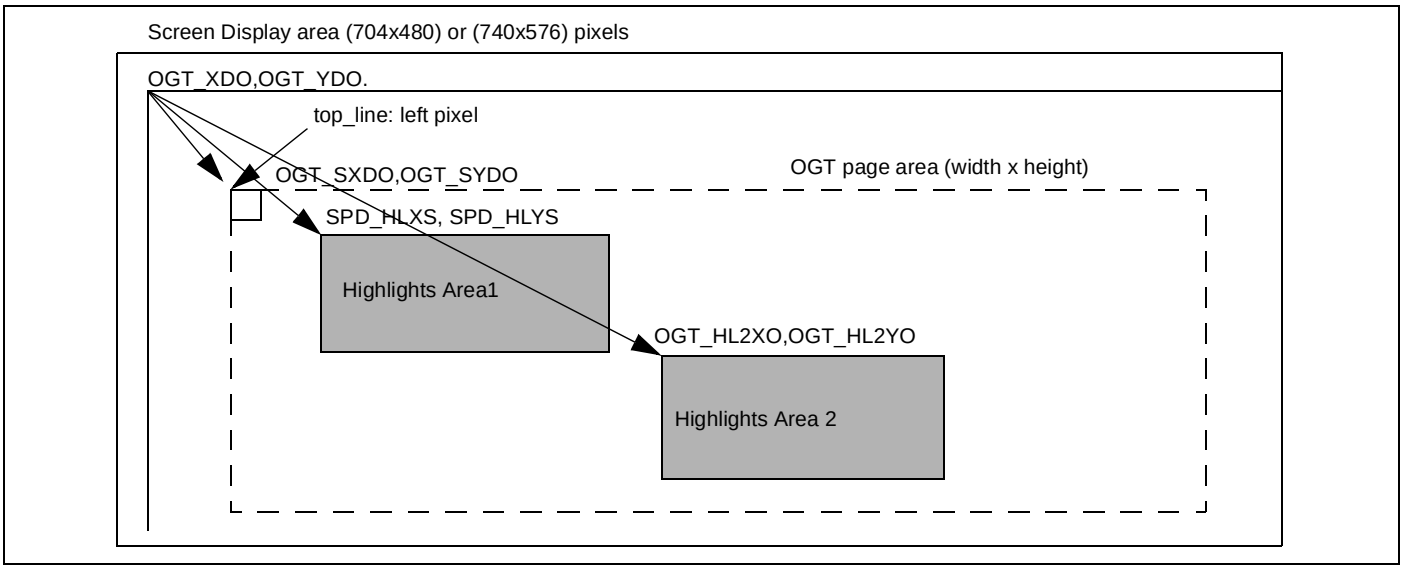


Figure 95 OGT display model

17.2 Buffer management

The bit-buffer is configured at reset. Its size is defined by registers VID_SPB and VID_SPE and is constant during the decoding process.The OGT uses the following 4 sub-picture registers to control the OGT bit-buffer read and write processes:

- VID_SPB bit-buffer base address, programmed in units of 2 kbytes.
- VID_SPE bit-buffer end address, programmed in units of 2 kbytes.
- VID_SPRead bit-buffer read pointer. This register must be updated with the **vsync_bottom** to re-decode the same OGT page, or to decode the next page. **vsync_bottom** is an offset to the ST20 SDRAM base address, it is programmed in units of 64-bit words.
- VID_SPWrite bit-buffer write pointer. This register must be set by the ST20 before transferring each OGT page into the bit-buffer. The address is an offset to the ST20 SDRAM base address, it is programmed in units of 64-bit words.

17.3 Operation

OGT page data are sent via the sub-picture data FIFO to the OGT decoder. The FIFO is written to by DMA or by CPU write. Only bit-map data are transferred into the bit-buffer. The parameters contained into the OGT header (areas, highlight, LUT,) are extracted by software and are held in the OGT registers.

Each OGT page contains an associated presentation time stamp (PTS) which controls when the OGT page is displayed. When the PTS is reached, the register OGT_CTL.S (start_decode) is set and register VID_SPRead is programmed on the vsync, before the **vsync_top** where the OGT page is displayed. The OGT decoder displays the same page until the next PTS, or until the display time (defined by the duration time) is reached.

To re-decode and re-display the same page, the read pointer must be reprogrammed and register bit OGT_CTL.S must be set to 1 before each vsync top.

To stop the display, OGT_CTL.S must be reset before the **vsync_top**, where the “PTS + duration” time is reached.

17.4 Display

The OGT decoder contains 3 look-up tables: one OGT and two highlight.

The CLUT defines 4 color and transparency values for the pixels of the OGT page. The CLUT can be changed from page to page. The highlight and main LUT are loaded by registers OGT_LUT, OGT_LUT_H1, OGT_LUT_H2.

The active OGT decoding area can be 704x576 or 70x480 pixels. To align the OGT decoding area with the video decoding area, the upper left corner of the active OGT area must be set by software using the registers OGT_XDI and OCT_YDI. The active OGT decoding area is defined by registers SPD_SXDO, SPD_SYDO, SPD_SXD1 and SPD_SYD1.

2 highlight areas can be defined in each OGT area. However, the bottom of the highlight1 area must always be above the top of the highlight2 area. The position of highlight1 is defined by registers SDP_HLSX, SPD_HLSY, SPD_HLEX, SPD_HLEY; The position of highlight2 is defined by registers OGT_HL2XO, OGT_HL2YO, OGT_HL2Y1, OGT_HL2X1.

18 Display planes

18.1 Overview

The graphics and display subsystem reads, processes, overlays and mixes pixel data stored in the various buffers of the SDRAM, and produces a combined image for display on a TV. The buffers are called display planes.

The graphics and display subsystem has four display planes as listed and illustrated below:

- Background color (Background color plane on page 157);
- MPEG video plane (MPEG video plane on page 158);
- On-screen display plane (On-screen display (OSD) on page 169);
- Sub-picture plane (Sub-picture or cursor plane on page 183).

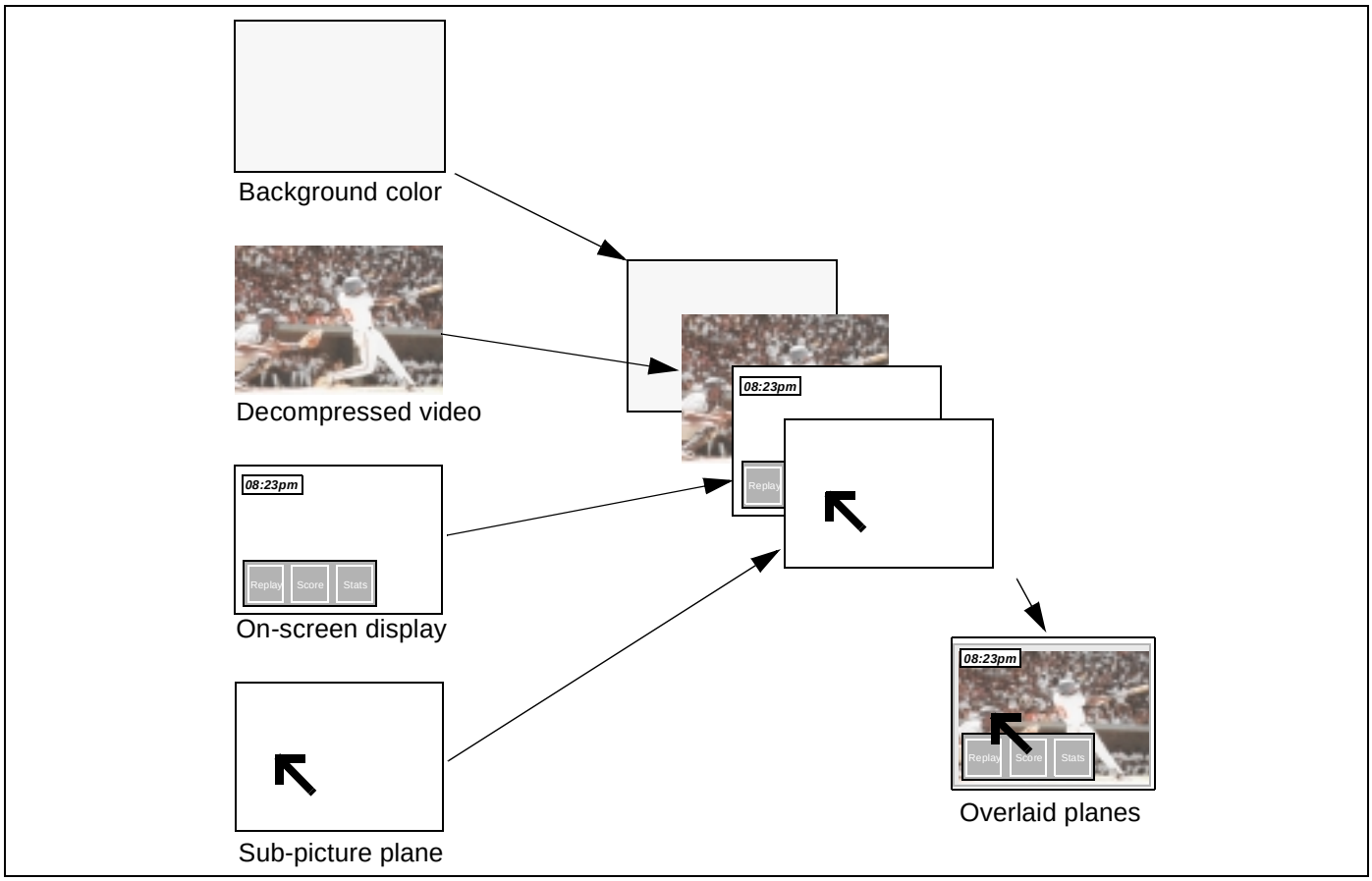


Figure 96 Display planes

The display planes are normally overlaid in the order shown above, with the background color at the back and the sub-picture used as a cursor plane at the front. The position of the sub-picture plane is programmable through bit VID_OUT.SPO. It can be configured to be:

- the most forward layer, as shown above;
  In this case it can be used as a cursor plane or a second on-screen display plane.
- behind the OSD plane, in front of the MPEG video.
  In this case it can be used as a second on-screen display plane.

Figure 97 shows a simplified block diagram of the graphics and display subsystem.

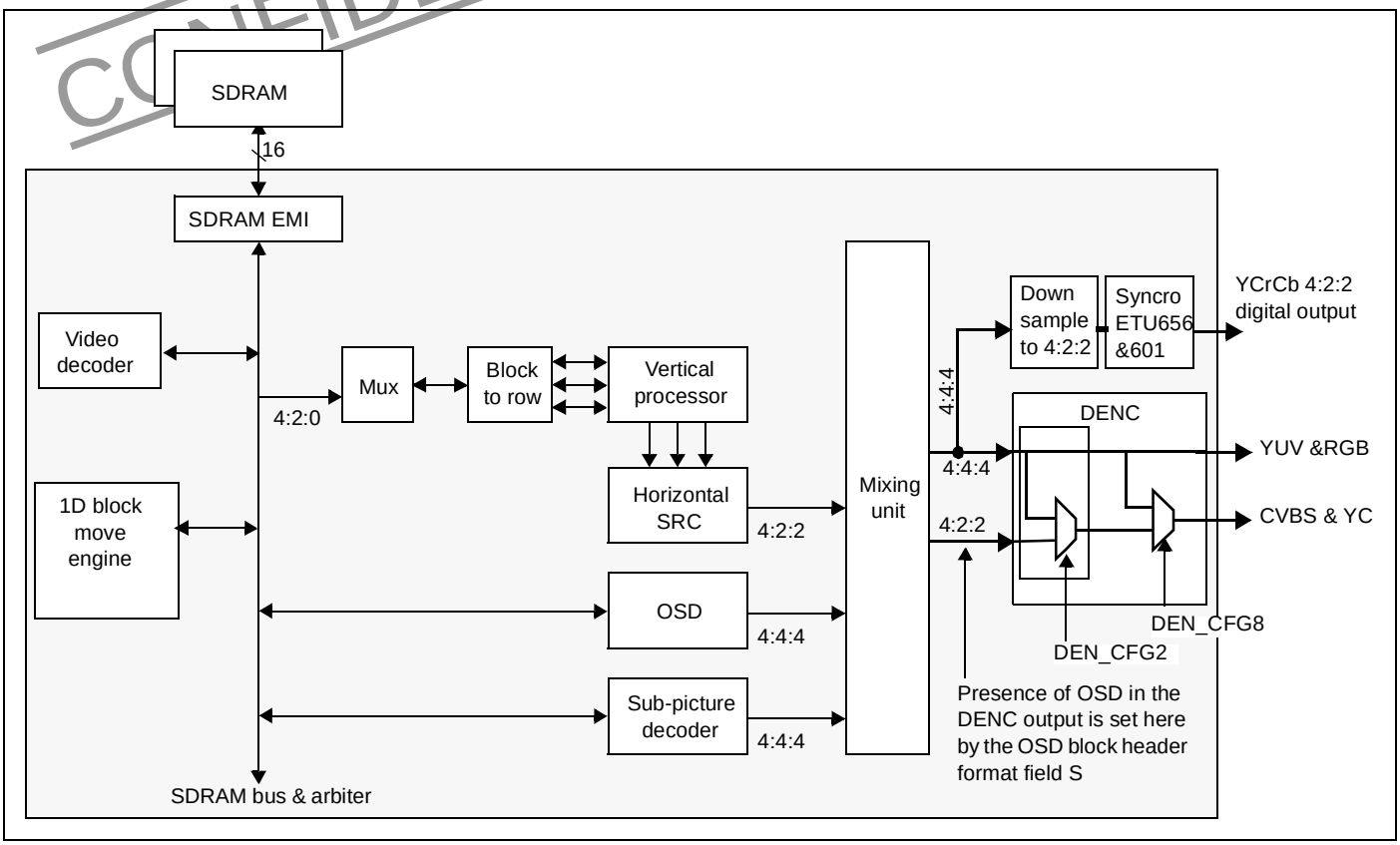


Figure 97 Graphics and display subsystem

The planes can be blended together using the mixing unit, as described in Mixing display planes on page 183.

The mixing unit has two outputs as illustrated in the figure above:

- A 4:2:2 output that is input to the DENC to generate CVBS and YC output. This format is generally used for VCR recording. The OSD and sub-picture display planes can be disabled from this output as described in on page 185.
- A 4:4:4 output, used in either of the following ways:
  - As an input to the DENC to generate the YUV and RGB signals. This is generally used for TV display, and includes ALL of the available display planes.
  - As a digital signal that bypasses the DENC and is output to the YUV pins “YC0 to YC7”. This signal is downsampled to 4:2:2 format by removing the chroma. It is used, for example, for connection to an external graphics device.

The routing of the 4:4:2 and 4:4:4 signals from the mixing unit, through the DENC to the DACs, is controlled by the DEN_CFG2 and DEN_CFG8 registers. When the YCrCb 4:4:2 signal is used, the presence of OSD in the DENC output is selected or deselected by the OSD block header format field S (see Table 85: *OSD block header format on page 177*). When the YCrCb 4:4:4 signal is used, OSD is always present in the DENC output.

18.2 Background color plane

The background color plane has the same size and position as the MPEG video plane. This plane is always at the back with all the other display planes on top. The color of the plane is defined by three registers: VID_BCK_Y, VID_BCK_U and VID_BCK_V.

18.3 MPEG video plane

The MPEG video plane displays a moving image decoded from an MPEG video stream by the MPEG video decoder. The display priority has the MPEG video as the third layer, on top of the background color. The sub-picture and OSD output are on top of the MPEG video plane.

The picture data is received either from the display frame buffer area of the external memory, or directly from the MPEG video decoder in the case of B frames in memory reduction mode.

The data is passed through three FIFOs (one for luminance and two for chrominance) into the block-to-row converter. The block-to-row converter generates a line based raster from the frame store, which is organized as MPEG macroblock. It also performs the pan/scan operation and vertical post-processing of the decoded video. The block-to-row converter is described in Section 18.3.3.

The output of the block-to-row converter is fed to the sample rate converter (SRC). The SRC is an 8-tap filter, which has two functions:

- up and down scaling of pel data when the displayed line length is greater or smaller than the decoded picture width, and implementation of the fractional part of the pan-scan horizontal offset.

The outputs from the SRC are upsampled lines each having equal numbers of luminance and chrominance samples. The SRC can be bypassed if desired.

The sample rate converter is described in Sample rate converter on page 159.

18.3.1 Setting-up the display

The VID_DFP and VID_DFC registers must be set up with the base address of the buffer containing the picture to be displayed. This register is double-buffered; when a new value is written it is taken into account on the occurrence of a VSYNC. Thus it is possible to write a new value for this pointer every field, although it would normally be updated only once per frame.

The picture stored in the buffer is always treated as a frame by the STi5518. If at any time no display is required, bit VID_DCF.EVD may be reset, in which case a constant black value is output.

The size and location of the display window is defined by the registers VID_XDO, VID_XDS, VID_YDO and VID_YDS. The values loaded into these registers define the horizontal and vertical boundaries of the displayed picture, as shown in Figure 98 .

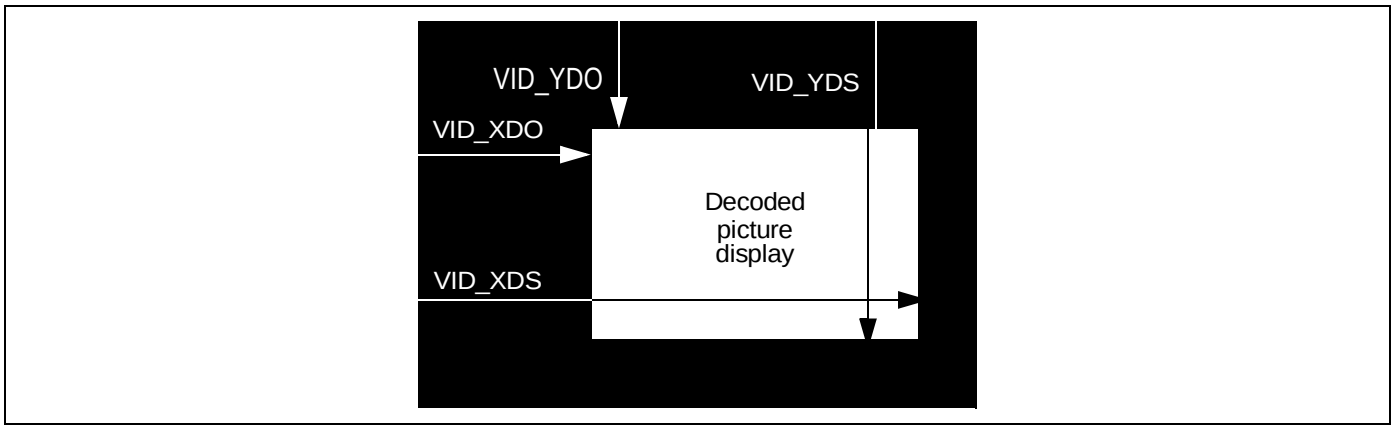


Figure 98 Display window positioning

18.3.2 Sample rate converter

The purpose of the sample rate converter (SRC) is to allow up or down sampling of picture data in order to increase or decrease the number of horizontal samples in a line. Upsampling is necessary if the horizontal size of the display is greater than the decoded picture width. For example if it is required to display a 720-pel wide 16:9 source image on a 4:3 display also of 720-pel width, then 540 pels selected from each source line must be upsampled to 720. Downsampling is required when the resolution of the display is less than that of the decoded image. For example when square pixels are required for an NTSC image the 720 pixel wide image decoded must be downsampled to 640 pixels.

To enable the SRC, bit **VID_DCF.DSR** must be reset. If this bit is set, the SRC is bypassed and the horizontal resolution of the decoded picture is not changed. The sample rate converter can change the sampling rate by a programmable factor. The upsampling ratio is limited to 8 and the downsampling to less than or equal to a factor of 2. The same filter is used both for upsampling and downsampling. As either of these limits is approached artifacts may appear in the displayed image. The SRC operates by directly interpolating samples required for the new sampling rate by using those of the decoded picture data read from the display buffer. This is performed by an 8-tap interpolation filter with the structure shown in Figure 99 .

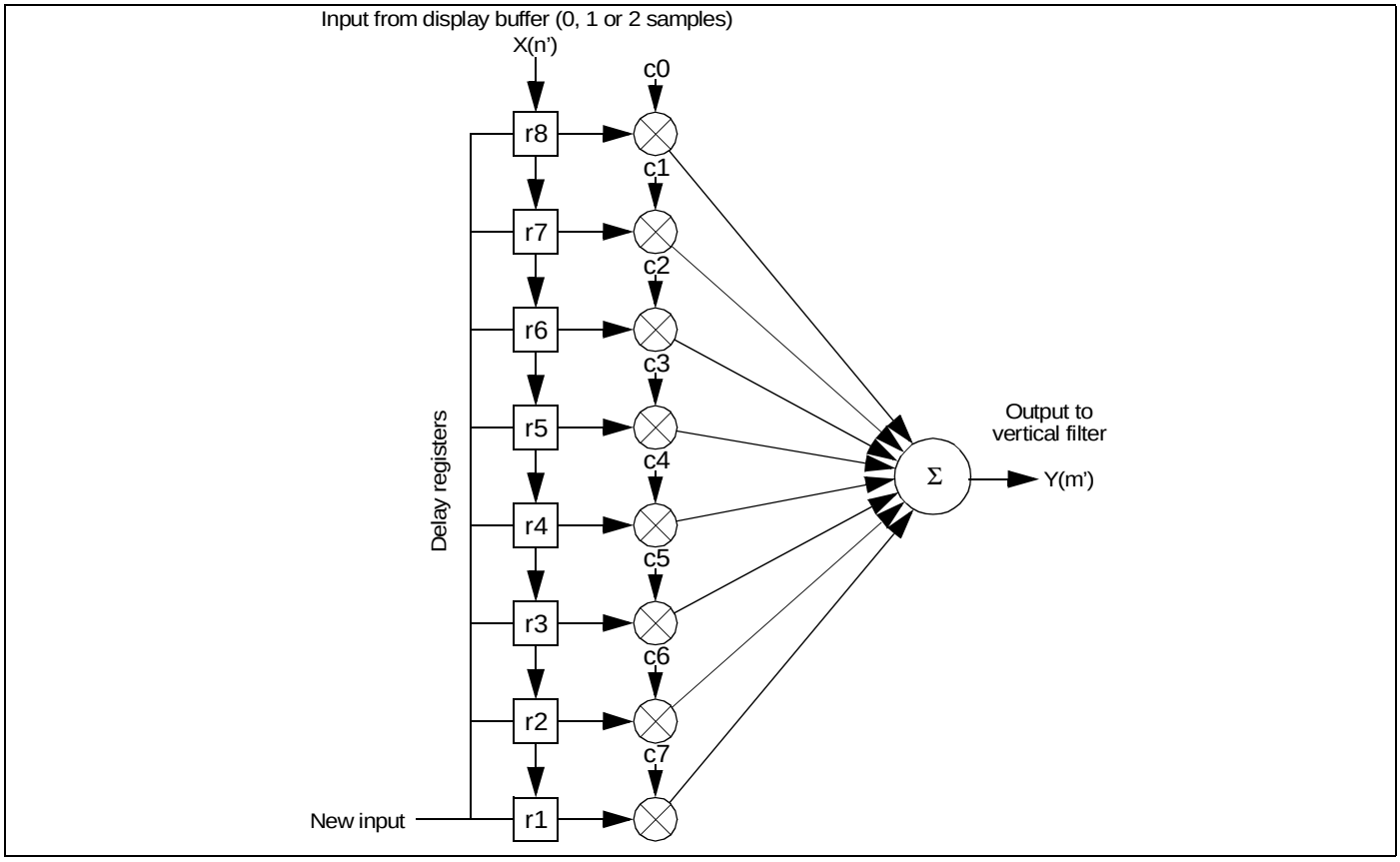


Figure 99 8-tap interpolation filter

The filter has three sets of delay registers multiplexed between the Y, C_B and C_R samples. It has 8 sets of coefficients, each set defining one of 8 sub-pel interpolation positions. Consider an upsampling example, for sub-pel position 0, the output is aligned with stored sample “r4”, for sub-pel position 1, the output corresponds to an interpolated pel position one eighth of the distance from sample “r4” to sample “r5”, and so on. The number of inputs clocked into the SRC is equal to the number of samples used in each line of the source image, and the number of outputs generated is equal to the number of samples displayed. Thus the rate of generation of outputs will be greater than the input data rate in the case of upsampling and less in the case of downsampling.

Operation of the SRC

The sample rate converter works in the following manner: The SRC takes block of M samples of the input signal denoted as x(n'), n' = 0,1,2,3,.... M-1. and computes a block of L output samples y(m'), m' = 0,1,2,. L-1.

For each output sample time m', m' = 0,1,2.., L-1 the 8 samples in the filter are multiplied with one of the 8 sets of filter coefficients the products are accumulated to give the output y(m'). Each time the quantity m'M/L increases by one, one sample from the input buffer is shifted into the filter.

The coefficient set used will depend on the position of the sample being generated relative to the original samples of the source image. Thus after L output values are computed M input samples have been shifted into the filter delay registers.

The SRC up/down sampling factor is set up in the **VID_LSR** register. The re-sampling factors for the luminance and chrominance components are exactly the same. The resampling factor is equal to L/M. The value programmed into **VID_LSR** is 256 × M/L. This value is used to determine both the rate of input of data into the filters and the sequence of sub-pel interpolation positions. The mechanism by which this is achieved is shown in Figure 100 .

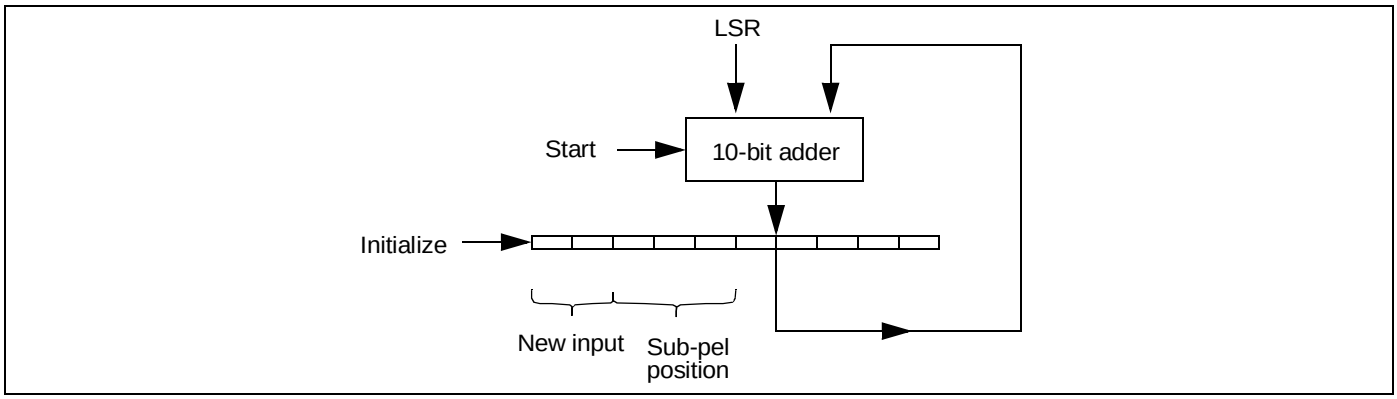


Figure 100

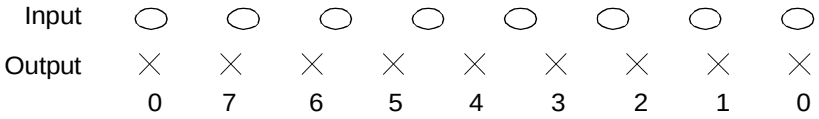
Upsampling example

The example in Figure 101 illustrates the operation of the sample rate converter when the upsampling ratio is 8:7. For every 8 samples clocked out of the filters, 7 samples are clocked in.

To illustrate the interpolation positions, at the right of Figure 101 are shown the outputs which would occur with a simple linear interpolation (i.e.a 2-tap filter). The actual SRC output values are the 8-tap filter outputs with coefficients

appropriate to sub-pel positions 0, 7, 6, 5, 4, 3, 2, 1, 0 etc. The SRC output is limited to lie within the range [1,254], so the codes 0x00 and 0xFF are never output, giving compatibility with ITU-R 656.

A - Relation of input and output samples



Sub-pel position

B - Filter operation

	r8	r7	r6	r5	r4	r3	r2	r1	
	n+7	n+6	n+5	n+4	n+3	n+2	n+1	n	n+3
No input sample read →	n+7	n+6	n+5	n+4	n+3	n+2	n+1	n	7/8 (n+4) + 1/8 (n+3)
	n+8	n+7	n+6	n+5	n+4	n+3	n+2	n+1	3/4 (n+5) +1/4 (n+4)
	n+9	n+8	n+7	n+6	n+5	n+4	n+3	n+2	5/8 (n+6) +3/8 (n+5)
	n+10	n+9	n+8	n+7	n+6	n+5	n+4	n+3	1/2 (n+7) +1/2 (n+6)
	n+11	n+10	n+9	n+8	n+7	n+6	n+5	n+4	3/8 (n+8) +5/8 (n+7)
	n+12	n+11	n+10	n+9	n+8	n+7	n+6	n+5	1/4 (n+9) +3/4 (n+8)
	n+13	n+12	n+11	n+10	n+9	n+8	n+7	n+6	1/8 (n+10) +7/8 (n+9)
	n+14	n+13	n+12	n+11	n+10	n+9	n+8	n+7	n+10

Delay register contents (One cycle per output clock cycle)

Output (shown interpolated linearly)

Figure 101 SRC example for 8:7 upsampling

The VID_LSR value is added into an accumulator register at a rate equal to the filter output rate. The top two bits indicate how many new inputs are to be loaded into the filter (0,1 or 2). The next three bits of the accumulator register are used to select the sub-pel position. For example, with an upsampling factor of 8:7, the **VID_LSR** value is (256/8) × 7 = 224. The sequence of values in the accumulator register will be as shown in Table 74 , assuming that it is initialized to zero.

Accumulator register	New input	Sub-pel position
0	yes	0
224	no	7
192	yes	6
160	yes	5
128	yes	4
96	yes	3
64	yes	2
32	yes	1
0	yes	0

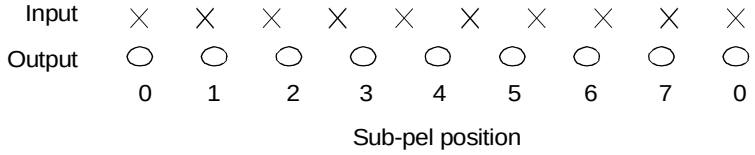
Table 74 Accumulator register sequence for upsampling example

The VID_LSR value thus defines a cycle of sub-pel positions as well as the rate of data input. If a value of less than 32 is loaded into VID_LSR, i.e. an upsampling ratio of greater than 8 is defined, there could be repeated values in the filter output. This may cause unacceptable display artifacts.

Downsampling example

The example shown in Figure 102 illustrates the operation of the sample rate converter when the downsampling ratio is 9:8 (720:640).

A - Relation of input and output samples



B - Filter operation

r8	r7	r6	r5	r4	r3	r2	r1	
n+7	n+6	n+5	n+4	n+3	n+2	n+1	n	n+3
n+8	n+7	n+6	n+5	n+4	n+3	n+2	n+1	1/8 (n+5) +7/8 (n+4)
n+9	n+8	n+7	n+6	n+5	n+4	n+3	n+2	1/4 (n+6) +3/4 (n+5)
n+10	n+9	n+8	n+7	n+6	n+5	n+4	n+3	3/8 (n+7) +5/8 (n+6)
n+11	n+10	n+9	n+8	n+7	n+6	n+5	n+4	1/2 (n+8) +1/2 (n+7)
n+12	n+11	n+10	n+9	n+8	n+7	n+6	n+5	5/8 (n+9) +3/8 (n+8)
n+13	n+12	n+11	n+10	n+9	n+8	n+7	n+6	3/4 (n+10) +1/4 (n+9)
n+14	n+13	n+12	n+11	n+10	n+9	n+8	n+7	7/8 (n+11) +1/8 (n+10)
n+15	n+14	n+13	n+12	n+11	n+10	n+9	n+8	n+11

Delay register contents (One cycle per output clock cycle)

Output (shown interpolated linearly)

Figure 102 SRC example for 9:8 downsampling

The **VID_LSR** value required for a downsampling ratio of 9:8 is 256 x 9 /8 = 288.

Accumulator register	Number of inputs	Sub-pel position
0	1	0
288	1	1
576	1	2
864	1	3
128	1	4
416	1	5
704	1	6
992	1	7
0	2	0

Table 75 Accumulator register sequence for downsampling example

At the start of a line, the 3 sets of delay registers r1, r2 and r3 are loaded with the black value (Y=16, C_B=C_R=128).

The first output is thus derived from the inputs stored in registers r4 to r8. At the end of a line, the last eight input samples are stored in registers r1 to r8.

The last valid interpolation is between the samples stored in r4 and r5. Correct Interpolation is not possible beyond this except in the case where the next output is in sub-pel position 0. This output is valid since coefficient C0 is zero for this position and the invalid sample beyond the end of the line is ignored.

There is thus no valid interpolation possible between the last four input samples. This is illustrated in Figure 103 in which 544 pels are upsampled to 721, in which the upsampling ratio is 4:3. The **VID_LSR** register would be loaded with the value 192.

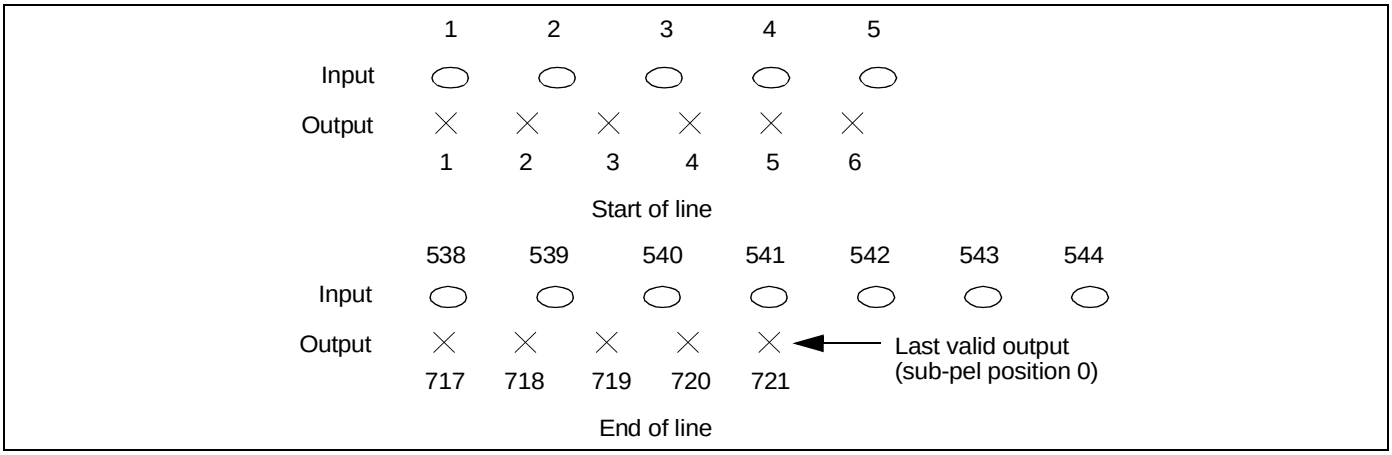


Figure 103 Downsampling example

The number of valid outputs generated can be calculated as follows:

The ratio between the number of input and output samples is 256:**VID_LSR**. Given that the last output sample cannot occupy a position beyond the fourth-last input sample, the following inequality is always true:

$$VID\_LSR\ (N-1) \leq 256\ (M-4)$$

where N is the number of output samples and M is the number of input samples. The value of N is thus given by:

$$N = \lfloor 256\ (M-4) / VID\_LSR + 1 \rfloor$$

where $\lfloor x \rfloor$ indicates the integer part of x.

The value programmed into the VID_XDS register must ensure that all samples beyond the last valid sample are masked.

18.3.3 Block-to-row converter

The block-to-row converter generates a line-based raster scan of individual video components (YCbCr) from an MPEG macroblock-organized frame store.

The block-to-row converter also performs:

- pan/scan;
- vertical post-processing of decoded video, such as:
  - vertical zoom-out x2, x3, x4
  - vertical programmable filtering with any zoom-in, and zoom-out up to x2;

- horizontal zoom-out x2 (for zoom-out by more than x2).

Pan/scan vectors

When the display window has a smaller horizontal dimension than the decoded picture, a vector can be programmed in order to define the starting point of the displayed picture, as shown in Figure 104 . The vertical component must be macroblock aligned, so the line number must be a multiple of 16.

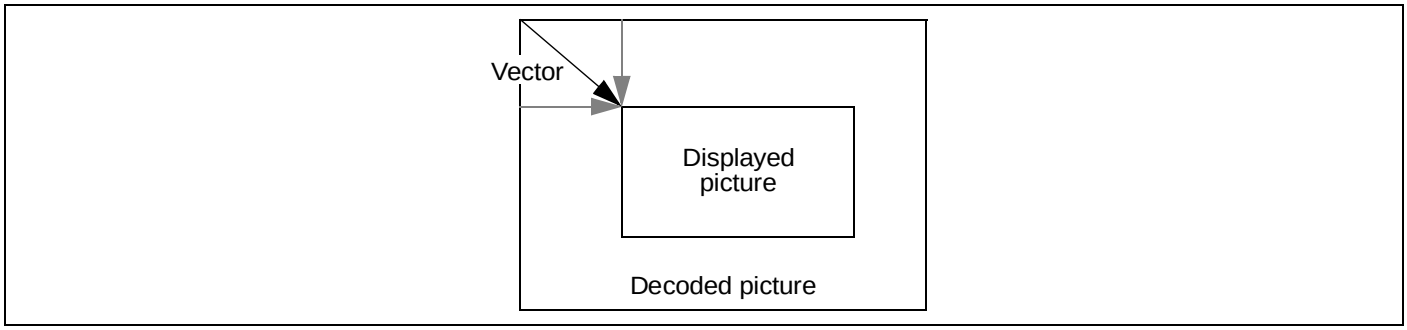


Figure 104 Pan/scan vector

This vector defines the point in the decoded picture which corresponds to the top-left-hand corner of the displayed picture. The displayed picture size and location is defined by the numbers programmed in registers VID_XDO, VID_XDS, VID_YDO and VID_YDS.

The pan/scan vector components are programmed into the registers VID_PAN, VID_LSO, VID_CSO and VID_SCN. These registers are double-buffered; when a new value is written it is taken into account on the occurrence of a VSYNC. Thus it is possible to write a new value of the pan/scan vector for every field.

The integer part of the horizontal component of the pan/scan vector is loaded into the VID_PAN register, and the fractional part defines the contents of the VID_LSO and VID_CSO registers. The relationship between these quantities is illustrated in Figure 105 .

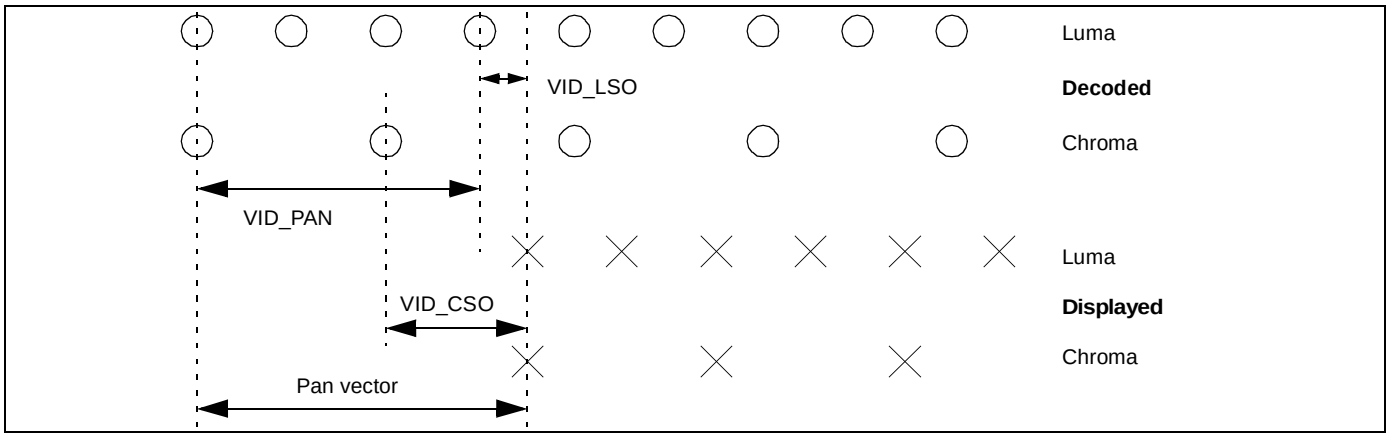


Figure 105 Components of the pan/scan vector

The numbers loaded into the VID_LSO and VID_CSO registers are used to initialize the luminance and chrominance upsampling control registers at the start of every line. VID_LSO is set up directly with the value of the fractional part of the pan/scan vector horizontal component. VID_CSO is set up with half of this number, plus 128 if the integer part is an odd number. The resolution to which the horizontal component can be defined is 1/8 pel.

The vertical component of the pan/scan vector is programmed into VID_SCN, in units of macroblock rows (i.e. units of 16 lines). Scanning can done line-by-line on any of the 8 lines in each field, by programing the OFFEVEN and OFFODD bits of the VID_VFCMODE and VID_VFLMODE registers.

Vertical filter

The block-to-row converter has an output filter for vertical post-processing of the video. This vertical filter performs the chroma reconstruction from 4:2:0 to 4:2:2 format, and upsamples and downsamples the luma and chroma components. Any zoom, from any zoom-in to zoom-out by 4, can be performed in addition to the following basic modes

- zoom-in by 2;
- zoom-out by 2, 3 or 4 (Zoom-out by 4 must be set by register bit VID_DCF.zoom4);
- zoom-out by n (where n lines are produced for each line stored, $1/16 \leq n \leq 2$);
- 16:9 and 14:9 letter box filtering.

The programmable PAL/NTSC vertical filter optimizes video quality according to the type of source - full or half resolution, interlaced or progressive. One luma filter operation and one chroma filter operation can run at the same time. The chroma filter mode is programmed in the VID_VFCMODE register, and the luma filter mode is programmed in the VID_VFLMODE register. This table describes the function of these registers and Table 76 on page 167 lists the recommended configurations.

Bitfield	Description
INCREMENT	The INCREMENT bit is used for downsampling up to zoom-out x2, and in conjunction with ZOOMOUT for fractional zoom-out between x2-x3, or x3-x4, for example x2.1, x2.2... etc. Define the vertical ratio of the zoom, and given by the formula: $increment = \left(\frac{framestoresize}{desiredsize} \times 512\right) - 1$ Where *framestoresize* is the number of lines in the framestore, and *desiredsize* is the number of lines in the display region. Ex, to display a 525 line framestore onto a 625 display region: $increment = \left(\frac{525}{625} \times 512\right) - 1 = 429$
OFFEVEN	Vertical scans for even fields. Adjusts and superposes the luma and chroma filtering. The offset is a variable distance from the first line.
OFFODD	Vertical scans for odd fields. Adjusts and superposes the luma and chroma filtering. The offset is a variable distance from the first line.
ZOOMOUT	The INCREMENT bit is used for downsampling up to zoom-out x2, for zoom-out x3 and x4, ZOOMOUT must be used in as below. For zoom-out between x2-x3, or x3-x4, for example x2.1, x2.2... etc. ZOOMOUT must be used in conjunction with INCREMENT. 00: zoom-out up to x2 01: zoom-out by 2 10: zoom-out by 3 11: zoom-out by 4
INTP	Interpolation field forces line repeat instead of integration (if 0, the top-line value is reproduced).
HORIZDN	Horizontal downsample by 2; neighboring samples are averaged. NOTE: That no other post processing filter can be used at the same time as this function. The lines in the macroblock buffer RAM are read out as they are, however, the line offset part of start_offset is still used.

Here are the VID_VFLMODE and VID_VFCMODE I program for the zoomout by 3 in full progressive and half progressive source resolution.

The interface between the block-to-row converter and the next block (horizontal Sample Rate Converter (SRC)), has three video components. The video data output is in 4:2:2 format.

Successive samples are presented at the relevant video component port (Y, C_r or C_b) synchronously with the relevant output sample clocking signal.

Horizontal compression

For zoom-out by three or four, a frame store must not only be compressed vertically by three or four, but also horizontally by three or four. The SRC is able to downsample up to a factor of two.

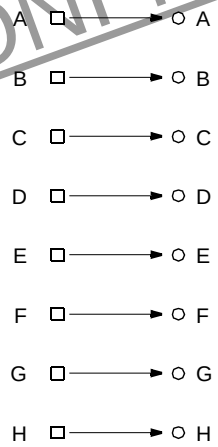
For zoom-out by three or four horizontally, the block-to-row converter must pre-process its output data for the SRC. This is set by bit 39 of the VID_VFCMODE and VID_VFLMODE registers.

Filter modes

The vertical filter supports an unlimited number of configurations, however, recommended configurations are listed in the table below.

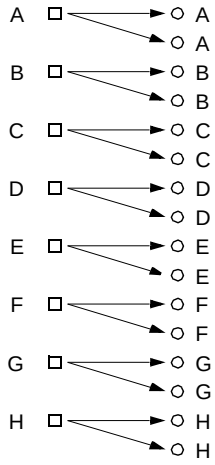
Display	Source Resolution		Luminance reg. (VID_VFLMODE)						Chrominance reg. (VID_VFCMODE)						VID_DCF	
			[39]	[38]	[37:36]	[35:23]	[22:10]	[9:0]	[39]	[38]	[37:36]	[35:23]	[22:10]	[9:0]	[1]	[0]
	Spacial	Temporal	horiz down	interpolate	zoom-out	offset odd	offset even	increment	horiz down	interpolate	zoom-out	offset odd	offset even	increment	LFB	CFB
Full screen	Full	Interlaced or progressive	0	1	00	255	255	509	0	1	00	0	0	254	0	0
		Interlaced or progressive	0	0	00	0	0	511	0	0	00	0	0	255	0	0
		Progressive	0	1	00	0	0	511	0	1	00	0	0	511	0	1
	Half	Interlaced or Progressive	0	1	00	255	127	253	0	1	00	0	0	126	0	0
		Progressive	0	0	00	127	0	511	0	0	00	0	0	255	1	1
16:9 letter box	Full	Interlaced or progressive	0	1	00	255	255	679	0	1	00	0	0	339	0	0
		Progressive	0	1	01	255	0	679	0	1	01	0	0	339	1	1
	Half	Interlaced	0	1	00	255	127	339	0	1	00	0	0	169	0	0
		Progressive	0	1	00	127	0	679	0	1	00	0	0	339	1	1
14:9 letter box	Full	Interlaced (field based)	0	1	00	0	36	583	0	1	00	0	0	291	0	0
480 to 576	Full	Interlaced/Pr.	0	1	00	512	469	425	0	1	00	192	42	212	0	0
576 to 480	Full	Interlaced/Pr.	0	1	00	512	563	611	0	1	00	192	90	305	0	0
zoom-in x 2	Full	Interlaced or progressive	0	1	00	0	127	255	0	1	00	192	0	127	0	0
	Full	Progressive	0	1	00	256	191	511	0	1	00	0	0	255	1	1
zoom-in x4	Full	Interlaced	0	1	00	255	511	127	0	1	00	0	0	63	0	0
zoom-in x4	Half	Interlaced	0	1	00	255	288	63	0	1	00	0	0	31	0	0
zoom-in x2	Half	Progressive	0	1	00	128	0	255	0	1	00	0	0	127	1	1
zoom-out x2	Full	Interlaced/ progressive	0	1	00	0	0	1019	0	1	00	0	0	509	0	0
	Half	Interlaced	0	1	00	255	255	509	0	1	00	0	0	253	0	0
	Half	Progressive	0	1	01	127	0	509	0	1	00	0	0	509	1	1
zoom-out x3	Full	Interlaced/ Progressive	1	0	10	0	0	573	1	0	01	0	0	382	0	0
	Half	Interlaced	0	1	01	0	0	380	0	1	00	0	0	380	0	0
	Half	Progressive	0	0	10	0	0	573	0	0	01	0	0	382	1	1
zoom-out x4	Full	Interlaced/ Progressive	1	0	11	0	0	509	1	0	11	0	0	254	0	0
	Half	Interlaced	0	1	00	0	0	1019	0	1	00	0	0	509	0	0
	Half	Progressive	0	0	11	255	64	507	0	0	11	0	0	253	1	1

Table 76 Vertical filter modes



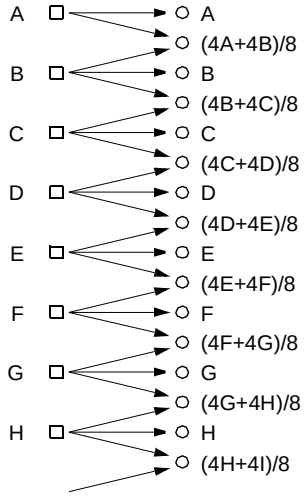
No zoom-out

parameter	value
start_offset_odd	000000000
start_offset_even	000000000
increment	0111111111
interpolate	don't care[1]
zoom_out_mode	00
horizontal_downsample	0



increment 255 offset 0 no interpolation

parameter	value
start_offset_odd	000000000
start_offset_even	000000000
increment	0011111111
interpolate	0
zoom_out_mode	00
horizontal_downsample	0



increment 255 offset 0

parameter	value
start_offset_odd	000000000
start_offset_even	000000000
increment	0011111111
interpolate	1
zoom_out_mode	00
horizontal_downsample	0

Figure 106 Filter mode examples

18.3.4 Degradation mode

In certain situations the system constraints may justify use of the STi5518 in a configuration where the available bandwidth on the SDRAM interface is limited. There could be many reasons for these constraints, such as a low clock frequency to use cheaper SDRAMS, or the processor making heavy use of the SDRAM. MPEG decode and display, being a real-time process and also a heavy user of SDRAM memory bandwidth, will then require a graceful degradation mode.

The effective distance (in pixels) between the display process and the decode process is measured by hardware. In conditions of limited bandwidth the decoder will become late and therefore may get caught by the real-time limited display process.

Degradation mode can be enabled or disabled using the panic threshold register VID_PTH. A threshold or minimum allowable distance between the decode and display processes can be set. If this threshold is crossed, the decoder will automatically ensure that any bidirectionally predicted macroblock access will result in only a single prediction access to external memory, thus reducing the bandwidth required by the decoder, and allowing recovery.

18.4 On-screen display (OSD)

The STi5518 has an integrated On-Screen Display (OSD) unit. This can be used to overlay the video picture with graphics generated by software. The display priority puts the OSD in front of the MPEG video. It can be configured to be either in front of or behind the sub-picture plane by clearing or setting VID_OUT.SPO respectively. The OSD can be enabled by setting OSD_CFG.ENA. The OSD bit-map is defined with respect to the display area and is independent of the decoded picture size and any pan/scan offset. The output from the OSD is in 4:4:4 format.

The OSD of the STi5518 has the following special features:

- Linked-list memory management;
- Selectable 2, 4 or 8-bits per pixel palette modes giving 4, 16 or 256 palette colors;
- Either 6-bit luma resolution and 4-bit chroma resolution per component or 8-bit luma resolution and 8-bit chroma resolution;
- Programmable 4-bit mixing factor for each OSD region to blend the video plane and OSD data;
- When anti-aliasing is enabled, each color in an OSD region can be assigned a separate 6-bit mixing factor for mixing with video;
- Optional anti-flicker and anti-flutter filters;
- Half resolution mode.

These features are described in the following sections.

The OSD unit uses color *look-up-tables* (LUTs), also called *palettes*, with 2-bit, 4-bit or 8-bit input. The LUT means that memory is used efficiently when only a few colors are needed. A 2-bit LUT means that four colors can be used at once, and each pixel of the bit-map occupies only two bits of memory. A 4-bit LUT gives 16 colors and an 8-bit LUT gives 256 colors. The palette of 4, 16 or 256 predefined colors is loaded into the SDRAM by software using the shared memory interface. The palette modes are described in Section 18.4.5.

The output from the LUT can be 14-bit pixels (6-bit Y, 4-bit Cb, 4-bit Cr) or 24-bit pixels (8-bit Y, 8-bit Cb, 8-bit Cr) plus one bit or six bits for transparency control. The color modes are described in Section 18.4.5.

The OSD can consist of a number of display regions, each with its own palette and characteristics. The number of OSD regions resident in memory at any time is limited only by the amount of memory available. Each region has a specification, stored in memory, which contains a header, possibly including a palette, and a bit-map. The specifications for the regions are linked in a list structure. The bit-map data in each specification is contiguous with the palette

information, as shown in Figure 112: *OSD specification* on page 173. The bit-map refers to the 2-, 4- or 8-bit color definitions in the palette to create the required picture.

During the display of an image a small state machine first picks up the palette from SDRAM and loads it into the LUT then the OSD region start and stop addresses are read. When the display reaches the OSD start position (defined in the bit-map) the bit-map is sent pixel by pixel to the LUT and the display switches from video to the output of the LUT or a mixture of both.

This process continues until the defined stop position. Thus, for the defined OSD region, the video display is overlaid by the colors which are defined by a combination of the LUT and the bit-map.

18.4.1 Using the OSD

The OSD is enabled if bit OSD_CFG.ENA is set. The starting address in memory of the OSD specification for the top field is defined by register VID_OTP, and that for the bottom field is defined by register VID_OBP.

The line numbers used to define the top and bottom of an OSD region are the internal (field) line numbers defined in Figure 107 . It is thus possible to share the same OSD specification for both fields of a frame. In this case the VID_OTP and VID_OBP registers would be loaded with the same address.

OSD specifications can be written into the SDRAM using the ST20 or the block move DMA. They can be rapidly moved within SDRAM using the SDRAM block move function.

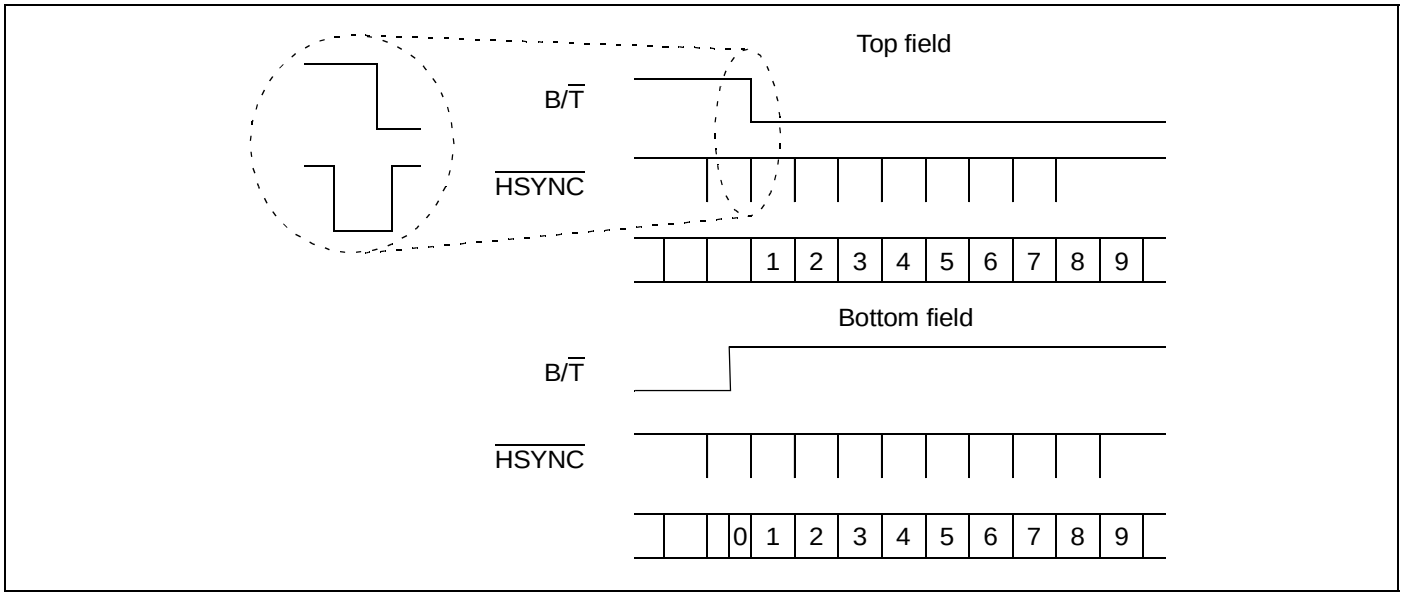


Figure 107 Internal line numbering

18.4.2 OSD regions

The OSD function can be used to display a user-defined bit-map over any part of the displayable (i.e. non-blanked) screen, independent of the size and location of the active video area (defined by VID_XDO, VID_XDS, VID_YDO, VID_YDS). This bit-map can be defined independently for each field.

The OSD consists of one or more regions in the display. Each region is a rectangle, and can have its own palette and other properties. Figure 108 shows examples of OSD regions. Region 3 shows that the OSD can be outside the active video area.

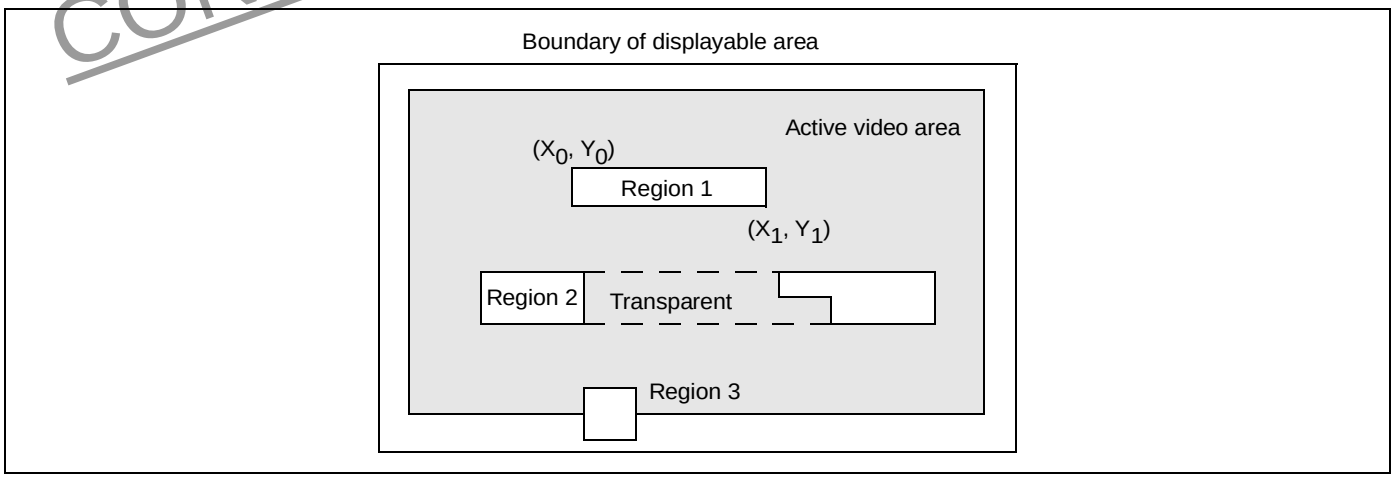


Figure 108 OSD regions

No display line can be included in more than one active OSD region, so only one OSD region can be active on a line. If two areas of OSD are required which include the same display line then one region must be defined which includes both areas. For example, Figure 109 shows two areas, marked A and B, with some display lines used in both areas. If these areas are to be active at the same time then one region, marked C, must be defined, which includes both areas. The area of C outside A and B can be defined as transparent.

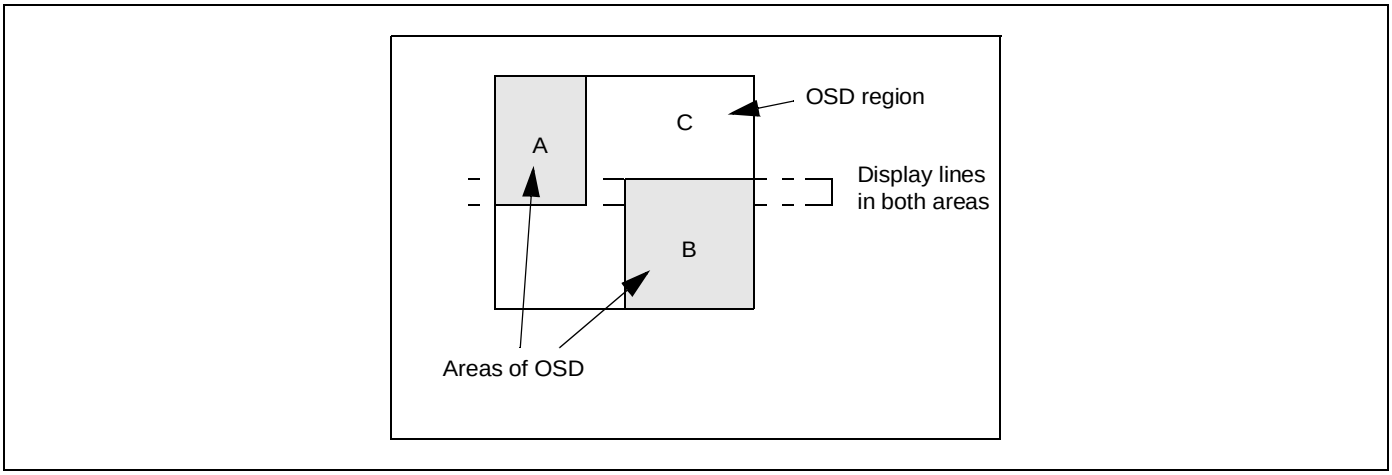


Figure 109

18.4.3 OSD specification

An OSD specification is two lists of blocks of 64-bit words, stored in SDRAM. One list is for the top field and one for the bottom. Generally the lists are linked lists, as shown in Figure 110 . The order of the blocks in the list is the order of the regions from top to bottom of the display. The last block in an OSD specification must point to an invalid header, which

gives a starting line beyond the displayable area (example OxFFFFFFFFFFFFFFFF). This invalid header will end the OSD.

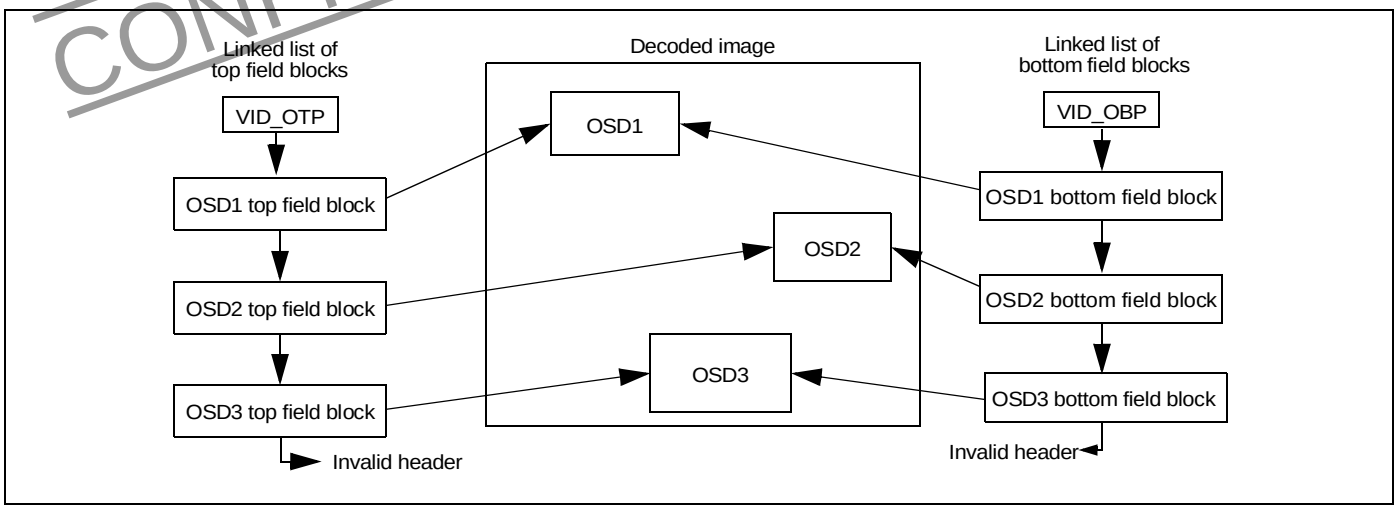


Figure 110 Linked list structure for OSD data

The only case in which linked lists are not used is when the palette mode is 2 bits for 2 pels, i.e. the palette mode flags are set to M=1, Q=1, E=0, as described in Section 18.4.5. In this case the blocks must be contiguous in memory, so the start of each block must immediately follow in memory the end of the previous block, as shown in Figure 111 . No linked list is allowed after a 2 bits for 2 pels zone.

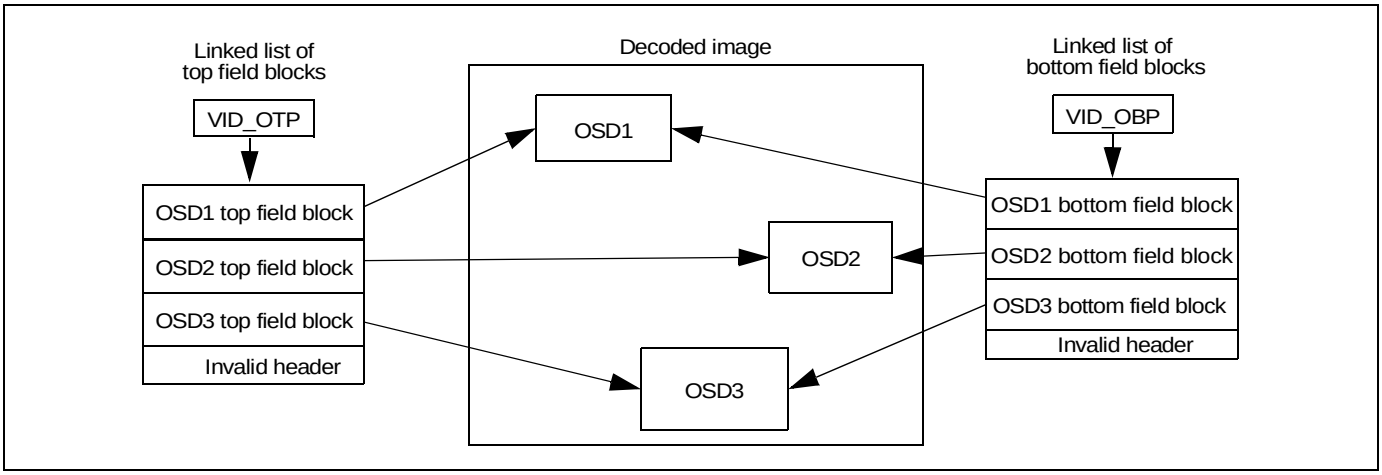


Figure 111 Block structure for OSD data in 2 bits per 2 pixels mode

Each block defines one field of one region and includes a header, an optional palette and a bit-map. Each block must be aligned on a 64-bit boundary and the first block of each field must be aligned on a 256-bit boundary. Figure 112 shows a linked list of two OSD blocks.

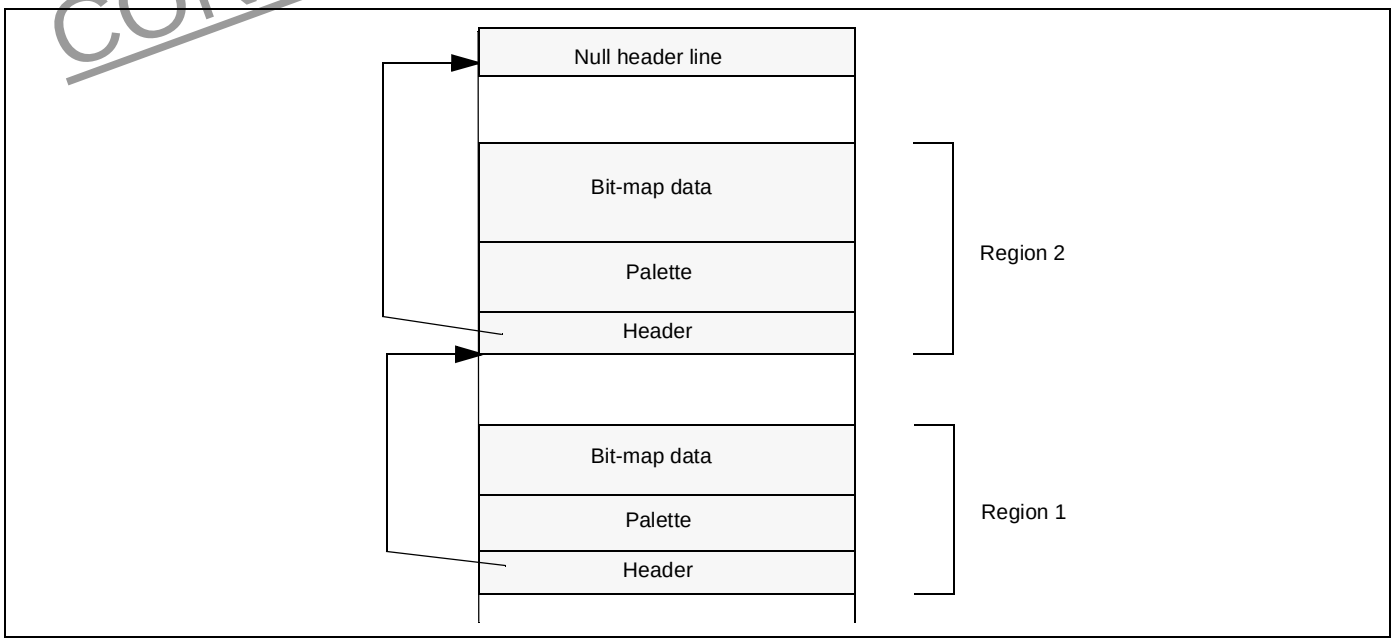


Figure 112 OSD specification

Each region has associated with it a palette defining 4, 16 or 256 colors, used by the bit-map. If required, one of these colors can be “transparent”, allowing the background to show through. Each region may have its own palette, or if a sequence of regions uses the same palette then the palette need only be defined in the first region of the sequence.

The header of each block contains a definition of the boundaries of the region, a pointer to the next region and other control information. The format of the palette depends on the palette mode, as described in Section 18.4.5. The formats are given in Section 18.4.7.

18.4.4 OSD region position

The position of each region of the OSD is defined in the header of the specification block. The positions of the left and right edge samples of an OSD region are defined as follows, in units of PIXCLK cycles from the falling edge of HSYNC:

left edge position = (2 × *X_left*) + 8

right edge position = (2 × *X_right*) + 9

where *X_left* and *X_right* are the values defined in the header of the OSD region specification. This is illustrated in Figure 113 and Figure 114 . The first sample output in an OSD region is always a C_B value.

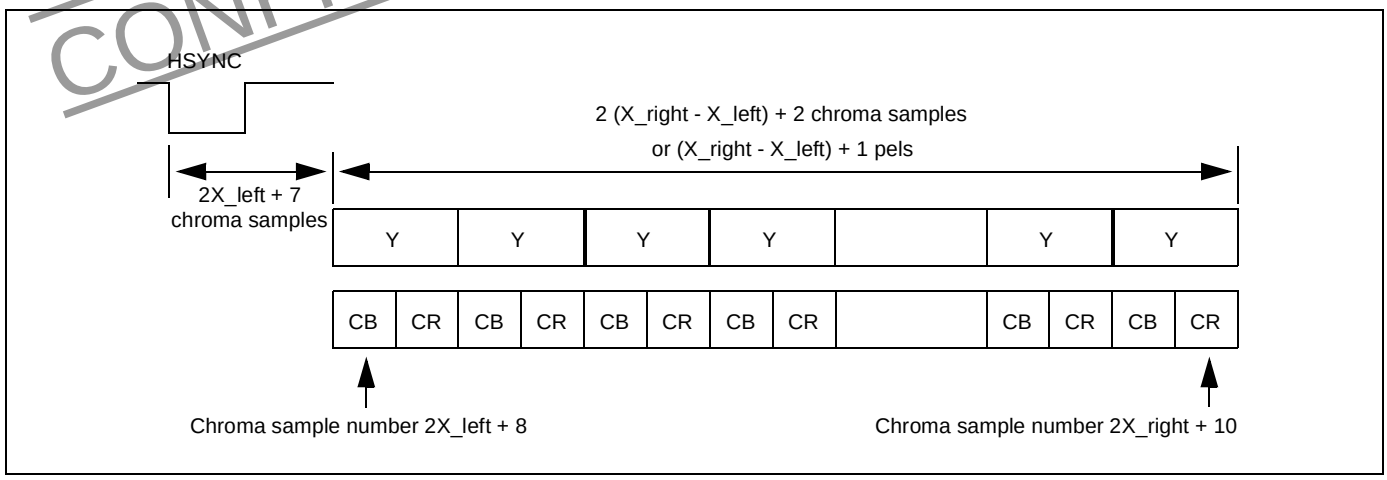


Figure 113 OSD region horizontal positioning in 4:4:4 output

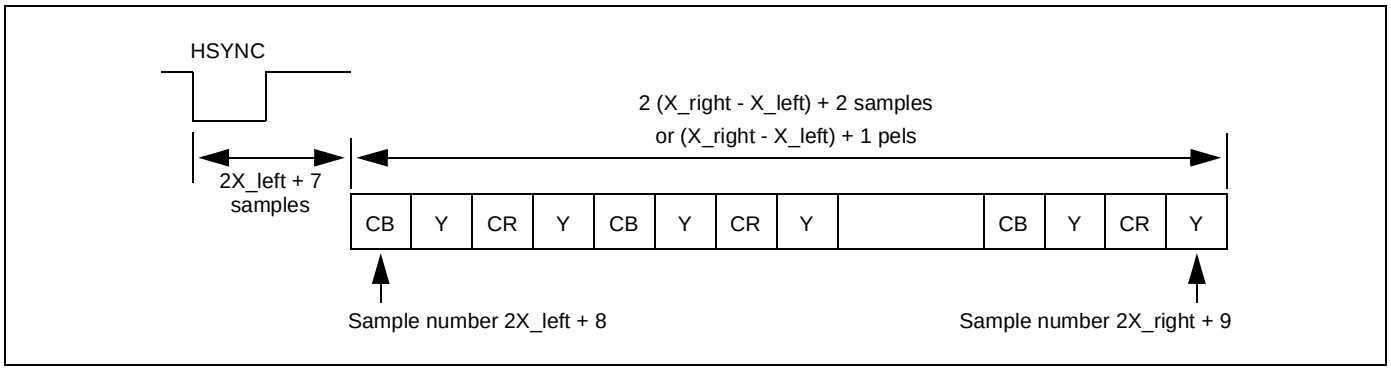


Figure 114 OSD region horizontal positioning in 4:2:2 output

The top and bottom of the region are defined by the values *Y_top* and *Y_bottom*, which are also in the block header. These values are specified in units of display lines. The top line specified in the first word of an OSD region specification must be greater than or equal to 3.

18.4.5 Color palette

Each specification block after the first can either define a new palette or use the same palette as the preceding region. If a new palette is defined then it is held in SDRAM immediately after the header and before the bit-map. The P flag in the header defines whether the palette follows the header, as shown in Table 77

P	Palette
0	The palette for the region is immediately after header.
1	The palette is the same as for the preceding region.

Table 77 Palette as before flag

Palette modes

The palette mode defines the bits per pel in the bit-map and the pixel resolution. The palette mode can be different for each OSD region, and is defined by the M, Q and E flags in the OSD region specification header. Q defines the pixel

resolution, allowing half resolution modes to save memory while retaining the color resolution. The meaning of each combination of these flags is given in the table below

M	Q	E	Bits per pixel	No. of colors	Resolution
0	0	0	2	4	1 pel
1	1	0	2	4	2 pels
1	0	0	4	16	1 pel
0	1	0	4	16	2 pels
0	0	1	8	256	1 pel
1	1	1	8	256	2 pels
1	0	1	Reserved		
0	1	1	Reserved		

Table 78 M, Q and E palette mode header flags

To reduce the size of the bit maps while retaining the color resolution, a half resolution mode is provided, as shown in In half resolution, each pel in the bit-map defines the color of two adjacent pixels in the same line in the display.

Color modes

The pixel color mode defines the format of the output from the palette. Three pixel color modes are supported, as listed in in the table below. This table shows how many bits are used for each color element and how many bits for the mixing factor MixWeight, which determines the effect of overlaying the picture. Anti-aliasing is supported only with 24-bit color.

24-bit or 14-bit color for the region field is selected by the bit Tc in the header of the specification block. Anti-aliasing is selected by the bit AA in the header.

Mode	Color resolution	Mixing	Reference
24-bit color with anti-aliasing	Y[7:0], Cb[7:0], Cr[7:0]	MixWeight[5:0] defined for each color.	Table 80
24-bit color	Y[7:0], Cb[7:0], Cr[7:0]	MixWeight[3:0] defined for the region.	Table 80
14-bit color	Y[5:0], Cb[3:0], Cr[3:0]	MixWeight[3:0] defined for the region.	Table 81

Table 79 OSD color modes

The format of each line of the palette depends on the color mode. The table below gives the format of the palette lines for each color mode.

Field	Bits	Description
Cr[7:0]	7:0	Cr chroma value
Cb[7:0]	15:8	Cb chroma value
Y[7:0]	23:16	Y luma value
W[5:0]	29:24	Mix weight. See Section 18.4.9.
Reserved	31:30	Reserved. Write 0.

Table 80 Palette line format in 24-bit color with anti-aliasing

Field	Bits	Description
Cr[7:0]	7:0	Cr chroma value
Cb[7:0]	15:8	Cb chroma value
Y[7:0]	23:16	Y luma value

Table 81 Palette line format in 24-bit color without anti-aliasing

Field	Bits	Description
T	24	Transparency 0 Do NOT blend video with OSD for this color. 1 Blend video with OSD for this color using the mix weight.
Reserved	31:25	Reserved. Write 0

Table 81 Palette line format in 24-bit color without anti-aliasing

Field	Bits	Description
Cr[3:0]	3:0	Cr chroma value
Cb[3:0]	7:4	Cb chroma value
T	8	Transparency: 0 Do NOT blend video with OSD for this color. 1 Blend video with OSD for his color using the mix weight.
Reserved	9	Reserved. Write 0.
Y[5:0]	15:10	Y luma value

Table 82 Palette line format in 14-bit color mode

Standard colors

The table below shows the 14-bit Y, C_R and C_B values nearest to the standard color bar colors.

Standard color	Y	C_R	C_B
White	0x3B	0x8	0x8
Black	0x04	0x8	0x8
Red	0x10	0xD	0x6
Green	0x1C	0x4	0x4
Blue	0x9	0x7	0xD
Yellow	0x28	0x9	0x3
Cyan	0x22	0x3	0xA
Magenta	0x15	0xC	0xD

Table 83 Standard colors in 14-bit color

The table below shows the 24-bit Y, C_R and C_B values nearest to the standard color bar colors.

Standard color	Y	C_R	C_B
White	0xEC	0x80	0x80
Black	0x10	0x80	0x80
Red	0x40	0xD4	0x64
Green	0x70	0x40	0x48
Blue	0x24	0x74	0xD4
Yellow	0xA0	0x8C	0x2C
Cyan	0x88	0x2C	0x9C
Magenta	0x54	0xC8	0xB8

Table 84 Standard colors in 24-bit color

18.4.6 OSD bit-map

The bit-map for an OSD region follows the palette if defined or the header if no palette is defined.

The bit-map defines the OSD pixels in left to right order within lines, and the lines in top to bottom order. The number of bits per pixel may be 2, 4 or 8 depending on the palette mode. The value for each pixel gives the line of the palette which defines the color for the pixel.

As all the different sources are mixed in 4:4:4 format, there is no risk of mis-colored boundary between OSD, video and sub-picture. An homogeneous decimation is applied to avoid the problem on the 4:2:2 format.

18.4.7 OSD block header format

Table 81 shows the layout of the header, which occupies one 64-bit word. Table 85 shows the layout in graphical form, with each line representing a quarter of a 64-bit word.

Field	Size	Bits	Meaning	Ref.
M	1	63	Palette mode.	Table 78
Q	1	62		
E	1	61		
Tc	1	60	0: Select 14-bit color mode 1: Select 24-bit color mode	Table 79
P	1	59	0: A new palette follows the header. 1: The palette is the same as the previous region.	Table 77
AA	1	58	Select anti-aliasing. Anti-aliasing can only be used with 24-bit color.	Section 18.4.5
S	1	57	OSD region not included in CVBS output in dual output mode.	Below
Y_top	9	56:48	Position of the top of the OSD region.	Section 18.4.4.
MixWeight	4	47:44	Mixing weight $\alpha 2$ with planes behind when anti-aliasing is disabled. When anti-aliasing is enabled, write 0.	Section 18.4.9
OSDp[6:4]	3	43:41	Pointer to the next region specification.	Below
Y_bottom	9	40:32	Position of the bottom of the OSD region.	Section 18.4.4.
OSDp[12:7]	6	31:26	Pointer to the next region specification.	Below
X_left	10	25:16	Position of the left of the OSD region.	Section 18.4.4.
OSDp[18:13]	6	15:10	Pointer to the next region specification.	Below
X_right	10	9:0	Position of the right of the OSD region.	Section 18.4.4.

Table 85 OSD block header format

M	Q	E	Tc	P	AA	S	Y_top
MixWeight				OSDp[6:4]			Y_bottom
OSDp[12:7]						X_left	
OSDp[18:13]						X_right	

Table 86 OSD region specification header

The header contains the pointer OSDp[18:0]. This pointer defines the address of the next block in the linked list to load from memory, as described in Section 18.4.3. The blocks can be anywhere in SDRAM and the pointer is given in units of 64-bit words. The block must be 16-word aligned, so the pointer OSDp[18:0] must be a multiple of 16. Thus the least significant 4 bits of OSDp are always zero, and are not included in the header.

The location of the first OSD specification block of a field is defined by the VID_OBP or VID_OTP registers in units of 256 bytes. This means the full address of the first block must be a multiple of 64.

The bit S in Table 87 on page 178 controls the presence of the OSD on a region basis for the 4:2:2 path to the Digital Encoder, from which YC and CVBS signals are generated. If this bit is 1, the OSD region will not be present; if it is set to 0, the OSD region will be present provided the OSD is included in the 4:2:2 output, as defined by **VID_OUT**. This is to allow selective recording of OSD regions. This bit does not affect the 4:4:4 path to the Digital Encoder from which the RGB and YCbCr signals are generated.

18.4.8 OSD specification block examples

This section shows the format for some complete specification blocks.

The table below shows a specification using 2 bits per pixel in the bit-map with 1 pel resolution and 14-bit color. Only the first 8 pixels of the bit-map are shown. The palette occupies one 64-bit word, and the bit-map occupies one 64-bit word for every 32 pixels.

<table>
<tr><th colspan="16">Bits within a 16-bit quarter-word</th><th rowspan="2">Description</th></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">M=0</td><td>Q=0</td><td>E=0</td><td>Tc=0</td><td colspan="2">P=0</td><td>A=0</td><td>S</td><td colspan="8">Y_top</td><td>Word 0</td></tr>
<tr><td colspan="4">MixWeight</td><td colspan="3">OSDp[6:4]</td><td colspan="9">Y_bottom</td></tr>
<tr><td colspan="7">OSDp[12:7]</td><td colspan="9">X_left</td></tr>
<tr><td colspan="7">OSDp[18:13]</td><td colspan="9">X_right</td></tr>
<tr><td colspan="7">Palette0 Y</td><td>0</td><td>T0</td><td colspan="3">Palette0 Cb</td><td colspan="3">Palette0 Cr</td><td>Word 1</td></tr>
<tr><td colspan="7">Palette1 Y</td><td>0</td><td>T1</td><td colspan="3">Palette1 Cb</td><td colspan="3">Palette1 Cr</td></tr>
<tr><td colspan="7">Palette2 Y</td><td>0</td><td>T2</td><td colspan="3">Palette2 Cb</td><td colspan="3">Palette2 Cr</td></tr>
<tr><td colspan="7">Palette3 Y</td><td>0</td><td>T3</td><td colspan="3">Palette3 Cb</td><td colspan="3">Palette3 Cr</td></tr>
<tr><td colspan="16">Bit-map for 8 OSD pixels</td><td>Bit-map word</td></tr>
</table>

Table 87 2 bits per pixel, 14-bit color OSD region specification

The table below shows a specification using 4 bits per pixel in the bit-map with 2 pel resolution and 14-bit color. Only the first 4 pixels of the bit-map are shown. Each entry in the bit-map uses 4 bits, but defines two display pels of the same color. The palette occupies four 64-bit words, and the bit-map occupies one 64-bit word for every 16 bit-map pixels.

Bits within a 16-bit quarter-word																**Description**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
M=1	Q=1	E=0	Tc=0	P=0	AA=0	S	Y_top									Word 0
MixWeight				OSDp[6:4]			Y_bottom									
OSDp[12:7]						X_left										
OSDp[18:13]						X_right										
Palette0 Y						0	T0	Palette0 Cb				Palette0 Cr				Word 1
Palette1 Y						0	T1	Palette1 Cb				Palette1 Cr				
Palette2 Y						0	T2	Palette2 Cb				Palette2 Cr				
Palette3 Y						0	T3	Palette3 Cb				Palette3 Cr				
Palette4 Y						0	T4	Palette4 Cb				Palette4 Cr				Word 3
Palette5 Y						0	T5	Palette5 Cb				Palette5 Cr				
Palette6 Y						0	T6	Palette6 Cb				Palette6 Cr				
Palette7 Y						0	T7	Palette7 Cb				Palette7 Cr				
Palette8 Y						0	T8	Palette8 Cb				Palette8 Cr				Word 4
Palette9 Y						0	T9	Palette9 Cb				Palette9 Cr				
Palette10 Y						0	T10	Palette10 Cb				Palette10 Cr				
Palette11 Y						0	T11	Palette11 Cb				Palette11 Cr				
Palette12 Y						0	T12	Palette12 Cb				Palette12 Cr				Word 5
Palette13 Y						0	T13	Palette13 Cb				Palette13 Cr				
Palette14 Y						0	T14	Palette14 Cb				Palette14 Cr				
Palette15 Y						0	T15	Palette15 Cb				Palette15 Cr				
Bit-map for 4 OSD pixels																Bit-map word

Table 88 4 bits per pixel, 2-pel resolution 14-bit color OSD region specification

The table below shows a specification using 8 bits per pixel in the bit-map with full resolution and 14-bit color. Only the first 2 pixels of the bit-map are shown. Each pixel in the bit-map uses 8 bits. The palette occupies 64 64-bit words (i.e. 512 bytes), and the bit-map occupies one 64-bit word for every 8 bit-map pixels.

Bits within a 16-bit quarter-word																**Description**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
M=0	Q=0	E=1	Tc=0	P=0	AA=0	S	Y_top									Word 0
MixWeight				OSDp[6:4]			Y_bottom									
OSDp[12:7]						X_left										
OSDp[18:13]						X_right										
Palette0 Y						0	T0	Palette0 Cb				Palette0 Cr				Word 1
Palette1 Y						0	T1	Palette1 Cb				Palette1 Cr				
Palette2 Y						0	T2	Palette2 Cb				Palette2 Cr				
Palette3 Y						0	T3	Palette3 Cb				Palette3 Cr				
Palette4 Y						0	T4	Palette4 Cb				Palette4 Cr				Word 2
...								...				...				...
Palette252 Y						0	T12	Palette252 Cb				Palette252 Cr				Word 64
Palette253 Y						0	T13	Palette253 Cb				Palette253 Cr				
Palette254 Y						0	T14	Palette254 Cb				Palette254 Cr				
Palette255 Y						0	T15	Palette255 Cb				Palette255 Cr				
Bit-map for 2 OSD pixels with 8 bits per pel or 8 bits per 2 pel																Bit-map Word

Table 89 8 bits per pixel, 14-bit color OSD region specification

The table below shows a specification using 2 bits per pixel in the bit-map with full resolution and 24-bit color, without anti-aliasing. Only the first 8 pixels of the bit-map are shown. Each entry in the bit-map uses 2 bits. The palette occupies two 64-bit words, and the bit-map occupies one 64-bit word for every 32 bit-map pixels. The palette for 4-bit and 8-bit colors would be similar, but with 16 or 256 color lines in the palette instead of 4.

Bits within a 16-bit quarter-word																**Description**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
M=0	Q=0	E=0	Tc=1	P=0	AA=0	S	Y_top									Word 0
MixWeight				OSDp[6:4]			Y_bottom									
OSDp[12:7]						X_left										
OSDp[18:13]						X_right										
0 (reserved)							T0	Palette0 Y								Word 1
Palette0 Cb								Palette0 Cr								
0 (reserved)							T1	Palette1 Y								
Palette1 Cb								Palette1 Cr								
0 (reserved)							T2	Palette2 Y								Word 2
Palette2 Cb								Palette2 Cr								
0 (reserved)							T3	Palette3 Y								
Palette3 Cb								Palette3 Cr								
Bit-map for 8 OSD pixels																Bit-map word

Table 90 2 bits per pixel 24-bit color without anti-aliasing OSD region specification

Table 91 shows a specification using 2 bits per pixel in the bit-map with full resolution and 24-bit color with anti-aliasing. Only the first 8 pixels of the bit-map are shown. Each entry in the bit-map uses 2 bits. The palette and bit-map occupy the same memory as without anti-aliasing. The palette for 4-bit and 8-bit colors would be similar, but with 16 or 256 color lines in the palette instead of 4.

Bits within a 16-bit quarter-word																**Description**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
M=0	Q=0	E=0	Tc=1	P=0	AA=1	S	Y_top									Word 0
0 (reserved)				OSDp[6:4]			Y_bottom									
OSDp[12:7]						X_left										
OSDp[18:13]						X_right										
0		MixWeight0					Palette0 Y									Word 1
Palette0 Cb							Palette0 Cr									
0		MixWeight1					Palette1 Y									
Palette1 Cb							Palette1 Cr									
0		MixWeight2					Palette2 Y									Word 2
Palette2 Cb							Palette2 Cr									
0		MixWeight3					Palette3 Y									
Palette3 Cb							Palette3 Cr									
Bit-map for 8 OSD pixels																Bit-map word

Table 91 2 bits per pixel 24-bit color with anti-aliasing OSD region specification

18.4.9 Mixing OSD with video

The mixing function allows each OSD pixel to be blended with the corresponding pixel generated by the planes behind the OSD. This is shown as α2 in Figure 117 and Figure 118. The mix weight is a programmable parameter and can be set for each OSD region, or for each color when anti-aliasing is enabled.

When anti-aliasing is disabled, the mix weight is set for each region in the region header. The mix weight is a 4-bit number allowing mixing ratios from 0 to 1 with a resolution of 1/15. The resulting pixel can be completely transparent (weighting of 0/15) or can completely cover the video (15/15). Each individual color in the palette can be specified to be used with or without mixing by setting the transparency (T) bit of the palette. A T bit equal to 0 means no mixing for the particular color, and a T of 1 means that mixing should be used.

When anti-aliasing is enabled, a mix weight is defined for each color in each region in the LUT. The mix weight is a 6-bit number allowing mixing ratios from 0 to 1 with a resolution of 1/63. The resulting pixel can be completely transparent (weighting of 0/63) or can completely cover the video (63/63).

Palette color zero can also be set to be transparent by setting the Y, Cb and Cr values to zero. Only palette color zero can be used in this way.

18.4.10 Anti-flicker and anti-flutter filters

Flicker and flutter effects are visual problems due to interlaced television displays. Flutter effects can occur if the same OSD image is displayed in both fields within one frame. Anti-flicker and anti-flutter filtering are provided as part of the display unit.

At every VSYNC pulse the values in the register OSD_CFG are taken in account. The table below shows the register configurations for anti-flitter/ anti-flutter filtering.

OSD_CFG.NOR	OSD_CFG.FIL	Filtering
0	0 or 1	none (normal mode)
1	0	anti-flicker filter applied
1	1	anti-flutter filter applied

Table 92 Register configurations for anti-flitter/ anti-flutter filtering

For flicker or flutter filtering the OSD boundary weight can be specified in the register **OSD_BDW**. This weight will be taken into account at OSD top and bottom borders in filter modes.

The anti-flicker filter can be described by the following expressions, where T is a top-field pixel, B is a bottom field pixel and n is the line number:

$$T = \frac{(B[n-1] + 2T[n] + B[n])}{4} \qquad B = \frac{(T[n] + 2B[n] + T[n+1])}{4}$$

The OSD Boundary Weight register (OSD_BDW) represents the value of mix_weight at the horizontal border of an OSD region or at a horizontal border of a transparent region within an OSD region. The OSD_BDW value can be used if flicker problems occur at OSD-video borders.

The anti-flutter filter can be described by the following expression, where T is a top-field pixel, B is a bottom field pixel and n is the line number:

$$T = T[n] \qquad B = \frac{(T[n] + T[n+1])}{2}$$

18.4.11 OSD active signal

The OSD active signal can be used in two modes. The mode is controlled using **OSD_ACT.OAM**. In the first mode the OSD active signal is configured as an output. In this mode the OSD active signal denotes when an active OSD pixel (non transparent) is on the YC output bus, as in Table 93 . The signal, in this mode, has a programmable delay controlled by **OSD_ACT.OAD**. This delay can be set such that the OSD active signal is set as much as two clocks before or 1 to 64 clocks after the actual pixel.

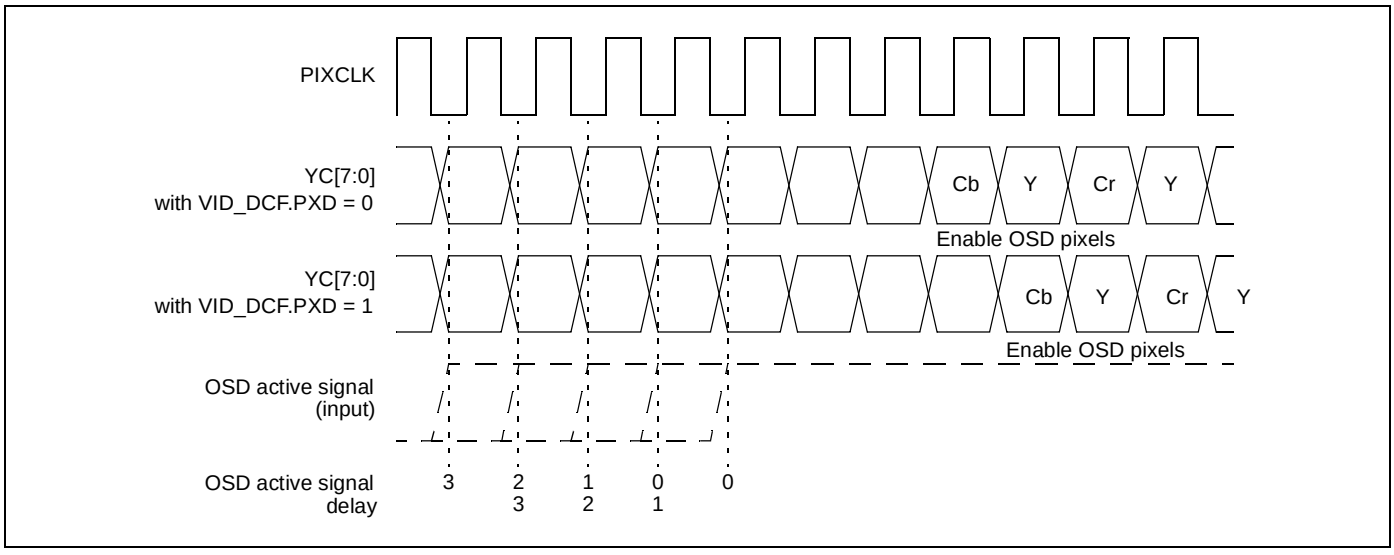


Figure 115 OSD active timing when OSD_ACT.OAM = 1

In the second mode of operation, the OSD active signal is configured as an input and is used to disable the OSD. When the signal goes high, the OSD will be placed on the YC bus if the OSD is enabled. When this signal is low then no OSD will be placed on the bus even if OSD is enabled. The programmable delay is used in the same way as for the input signal.

OSD active mode	OSD active signal	Meaning
0	0	Signal is an output. Video Pixels only on display bus.
0	1	Signal is an output. OSD Pixels on the display bus.
1	0	Signal is an input. Disable the OSD output.
1	1	Signal is an input. Enable OSD output if available.

Table 93 OSD active signal operation

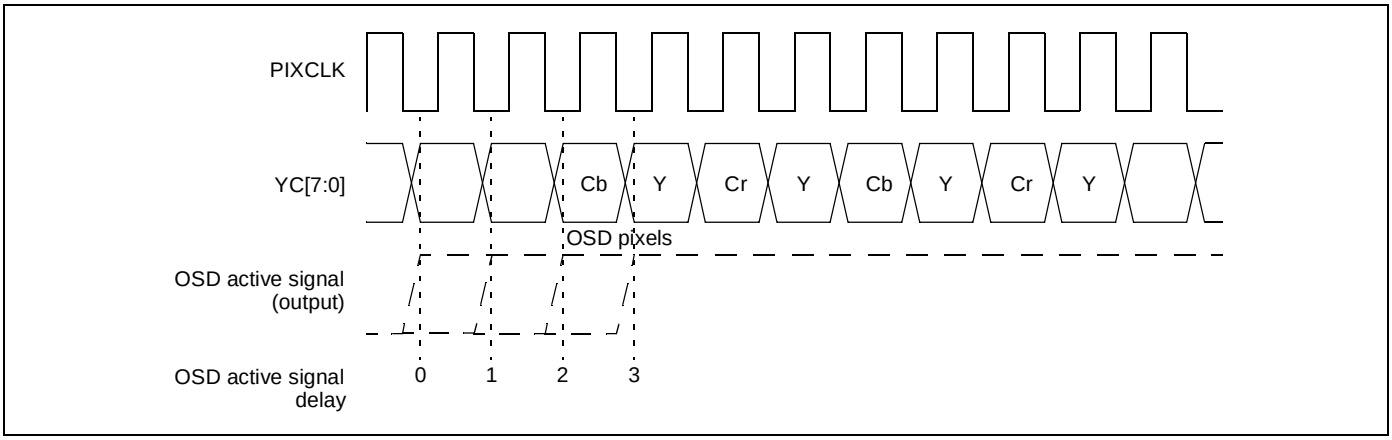


Figure 116 OSD active timing when OSD_ACT.OAM = 0

18.5 Sub-picture or cursor plane

The sub-picture or cursor plane displays the output from the sub-picture decoder, described in Sub-picture decoder on page 150. The sub-picture can be either in front of or behind the OSD, depending on whether the bit **VID_OUT.SPO** is 1 or 0 respectively.

The sub-picture decoder can also be used as a hardware cursor unit. The sub-picture should be configured to be in front of the OSD. A cursor can be defined using an optionally compressed (run-length encoded) bitmap stored in external SDRAM. The bitmap can be any size up to a full screen. Per-pixel alpha-blending factors can be defined for each cursor to provide anti-aliasing with the background. The cursor is then moved around using register writes into X and Y coordinate registers.

18.6 Mixing display planes

The blending of the elements of the final picture is performed by a mixing unit, which is shown in Table 94 and Figure 117 . The mixing of the display planes is controlled by five mix weights, $\alpha 1$ to $\alpha 5$.

The mix weights value (α1) controls the mixing of the the background color plane and the video plane. It is an 8-bit values held in the VID_MWV register, as shown in Table 94 . The two back planes are generated and mixed in 4:2:2 format and then converted to 4:4:4 for mixing with the sub-picture and OSD.

Factor	Register	Mixed planes	Plane displayed when mix weight is 0	Plane displayed when mix weight is 0xFF
α1	VID_MWV	Background color and MPEG video	MPEG video	Background color

Table 94 Control of mixing factor α1

The resultant of mixing the back three planes can be blended in turn with the sub-picture and OSD. The mixing formula is: display=(1-α) x source1 + α x source2.
The mixing order depends on whether the sub-picture is in front of or behind the OSD.

The OSD mix weight is α2 and is defined in the OSD palette. If anti-aliasing is disabled, then the mix weight is a 4-bit value, defined for each OSD region, and mixing can be enabled or disabled for each color. If anti-aliasing is enabled, then the mix weight is a 6-bit value defined for each color. Blending the OSD is described in Section 18.4.9.

The sub-picture mix weight is α3, which is a 4-bit value defined per pixel type. These values can be controlled by the sub-picture bit stream except in the highlight area which is controlled by SPD_HCN.

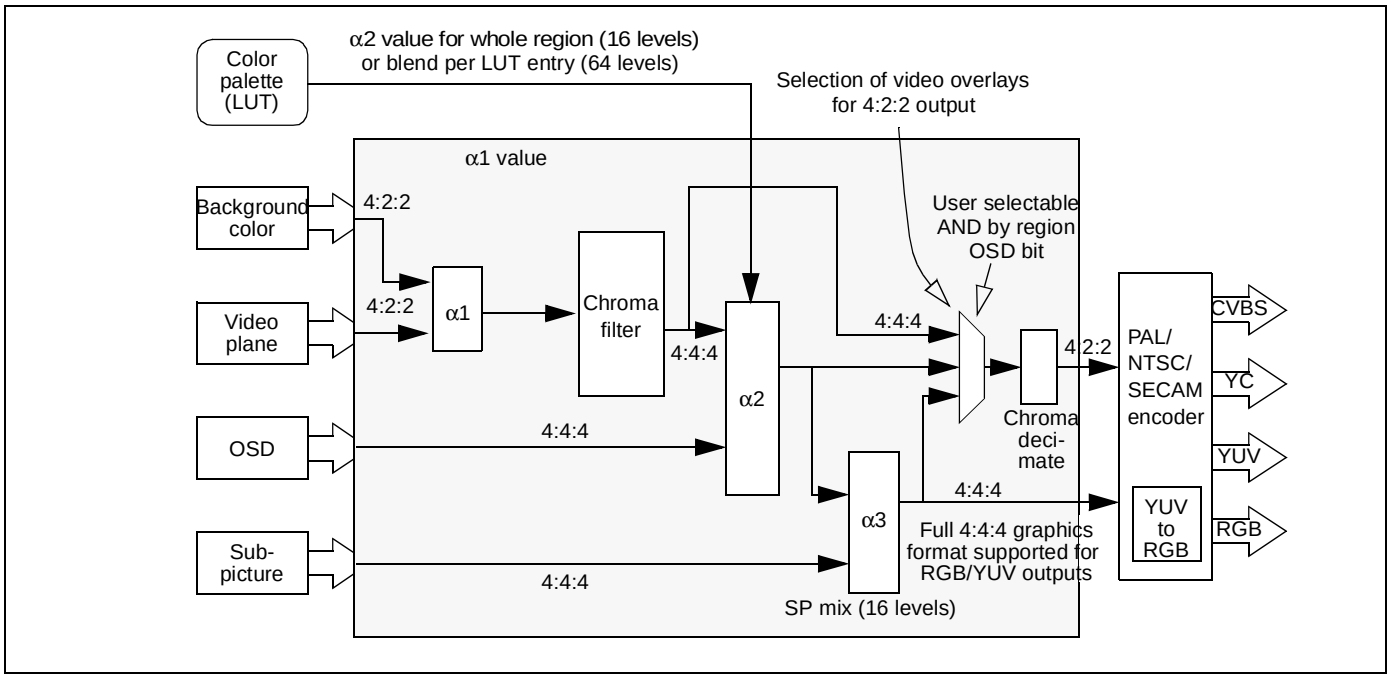


Figure 117 Mixing with the sub-picture in front

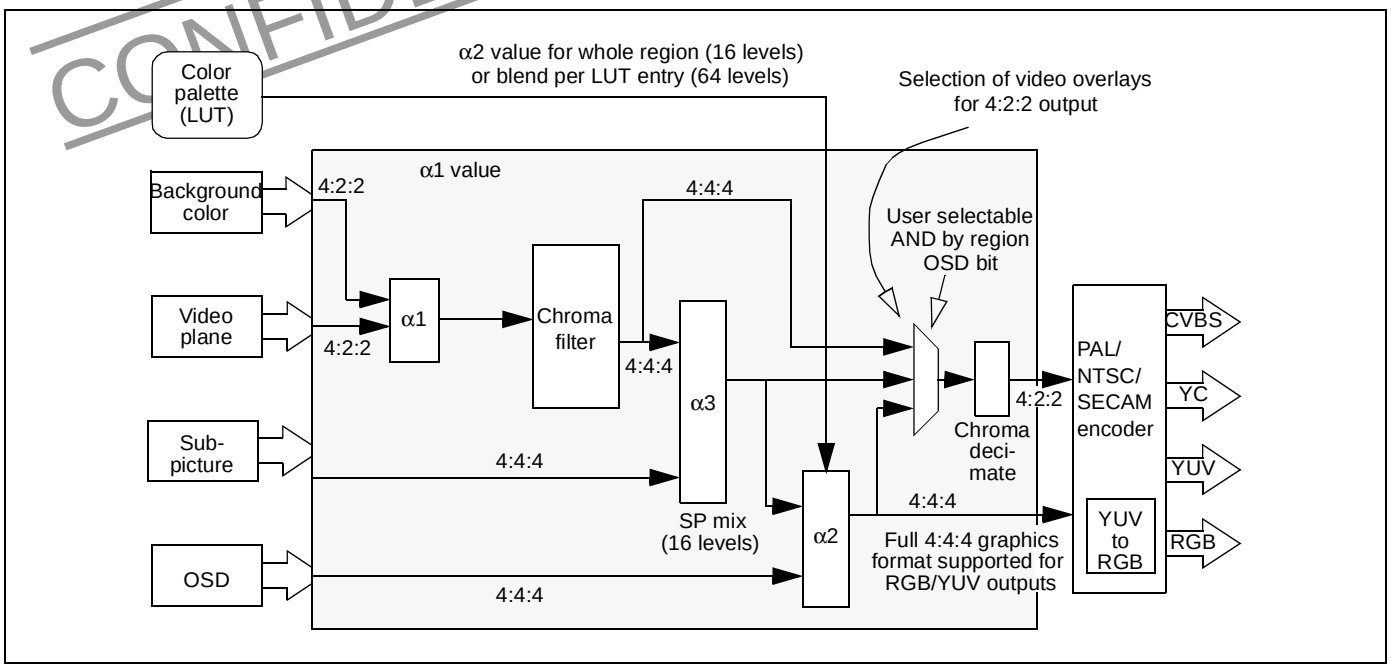


Figure 118 Mixing with the OSD in front

18.6.1 4:2:2 Output control

The 4:2:2 output is used by the DENC to generate CVBS and YC output, and is generally used for recording by VCR. The OSD and sub-picture can be omitted from this output, as controlled by the register VID_OUT and by the S bits in the OSD region headers. The combined actions of these fields depend on whether the OSD is behind or in front of the sub-picture. The 4:4:4 output is unaffected.

VID_OUT.LAY defines the number of planes in front of the video which are initially included. VID_OUT.NOS turns on or off the OSD, as shown in Table 95 . If VID_OUT.NOS=0 (OSD present) then the S bit in an OSD region header can alternatively be used to turn off that OSD region. When the OSD and Sub-picture planes are present,VID_OUT.SPO defines which plane is in front. The comibation of bit functions is given in the table below.

LAY	NOS	Sub-picture in front (SP0=1)	OSD in front (SP0=0)
00-01	Any	Video only	Video only
10	0	Video + OSD	Video + sub-picture
	1	Video only	Video + sub-picture
11	0	Video + OSD + sub-picture	Video + sub-picture + OSD
	1	Video + sub-picture	Video + sub-picture

Table 95 Encoding of LAY and NOS fields of VID_OUT

19 SDRAM block move

This SDRAM Block Move module, copies blocks of data from one byte address within the SDRAM to another. The source address, destination address and the number of bytes to be transferred are specified by the SDRAM block move registers listed in Table 96.The registers are all serial read/write registers in the peripheral address space.

To perform a SDRAM block move from one SDRAM memory buffer to another, the block move size, destination address and read address must be set by the registers listed in the table below. Registers USD_BMS and USD_BWP must be written before USD_BRP, as the third write to USD_BRP initiates the block move. While a block move is in progress, all other accesses to the SDRAM are disabled. The progress of the block move can be monitored using register bit VID_STA.BMI. This bit is set while a block move is in progress, and reset when the block move engine is idle. When the block move is finished, an interrupt is generated if the mask bit VID_ITM.BMI is set.

Register	Bits	Name
USD_BMS	15:0	Block move size
USD_BWP	19:0	Memory write pointer
USD_BRP	19:0	Memory read pointer

Table 96 SDRAM block move registers

20 Digital encoder

20.1 Introduction

This the final stage of the video pipeline of the device is a high performance PAL/SECAM/NTSC digital encoder referred to as the DENC. The DENC converts a 4:2:2 digital video stream into a standard analog baseband PAL/ SECAM/NTSC signal and into RGB and YUV analog components.

The DENC can perform closed-captions, CGMS, WSS, Teletext and VPS encoding and allows Macrovision^{TM} 7.01/ 6.1 copy protection. Six analog output pins are available, on which it is possible to output either (S-VHS(Y/C) + CVBS + RGB) or (S-VHS(Y/C)+ CVBS + V + Y + U) or (Y1 + C1 + CVBS1 + C2 + Y2 + CVBS2).

The encoder can operate in master mode or in one of several slave modes, where it locks onto incoming sync signals. An auto-test mode is also provided.

The main functions are controlled using an 8-bit register interface with the CPU. The registers are described in the *STi5518 Register Manual*.

20.2 Video timing

The burst sequences are internally generated, subcarrier generation being performed numerically with CKREF as reference. 4-frame bursts are generated for PAL or 2-frame bursts for NTSC. Rise and fall times of synchronization tips and burst envelope are internally controlled according to the relevant ITU-R and SMPTE recommendations. The 6-frame subcarrier phase sequence is generated in SECAM (see Subcarrier insertion (SECAM) on page 200).

Figure 121, Figure 122 and Figure 123 depict typical VBI (vertical blanking interval) waveforms.

It is possible to encode incoming YCrCb data on those lines of the VBI that do not bear line sync pulses or pre/post-equalization pulses (see Figures 121, 122 and 123). This mode of operation is referred to as *partial blanking* and is the default set-up. It allows the encoded waveform to keep any VBI data present in digitized form in the incoming YCrCb stream (e.g. supplementary closed-caption lines or *StarSight* data, etc.). In SECAM mode, only Y data are encoded, Cr and Cb are ignored. Alternatively, the complete VBI may be *fully blanked*, so no incoming YCrCb data is encoded on these lines. Full or partial blanking is set by register bit DEN_CFG1.blkli.

For 525/60 systems, with the SMPTE line numbering convention:

- complete VBI consists of lines 1 to 19 and the second half of lines 263 to 282;
- partial VBI consists of lines 1 to 9 and the second half of lines 263 to 272;
- line 282 is either fully blanked or fully active.

For 625/50 systems, with the CCIR line numbering convention:

- complete VBI consists of the second half of lines 623 to 22 and lines 311 to 335;
- partial VBI consists of the second half of lines 623 to 5 and lines 311 to 318;
- line 23 is always fully active.

In an ITU-R656-compliant digital TV line, the active portion of the digital line is the portion included between the SAV (Start of Active Video) and EAV (End of Active Video) words. However, this digital active line starts somewhat earlier and may end slightly later than the active line usually defined by analog standards.

The DENC allows two approaches:

- Encodes the full digital line (720 pixels / 1440 clock cycles). In this case, the output waveform will reflect the full YCrCb stream included between SAV and EAV.

- Drops some YCrCb samples at the extremities of the digital line so that the encoded analog line fits within the analog ITU-R/SMPTE specifications.

In all cases, the transitions between horizontal blanking and active video are shaped to avoid too steep edges within the active video. Figure 124 gives typical timings concerning the horizontal blanking interval and the active video interval.

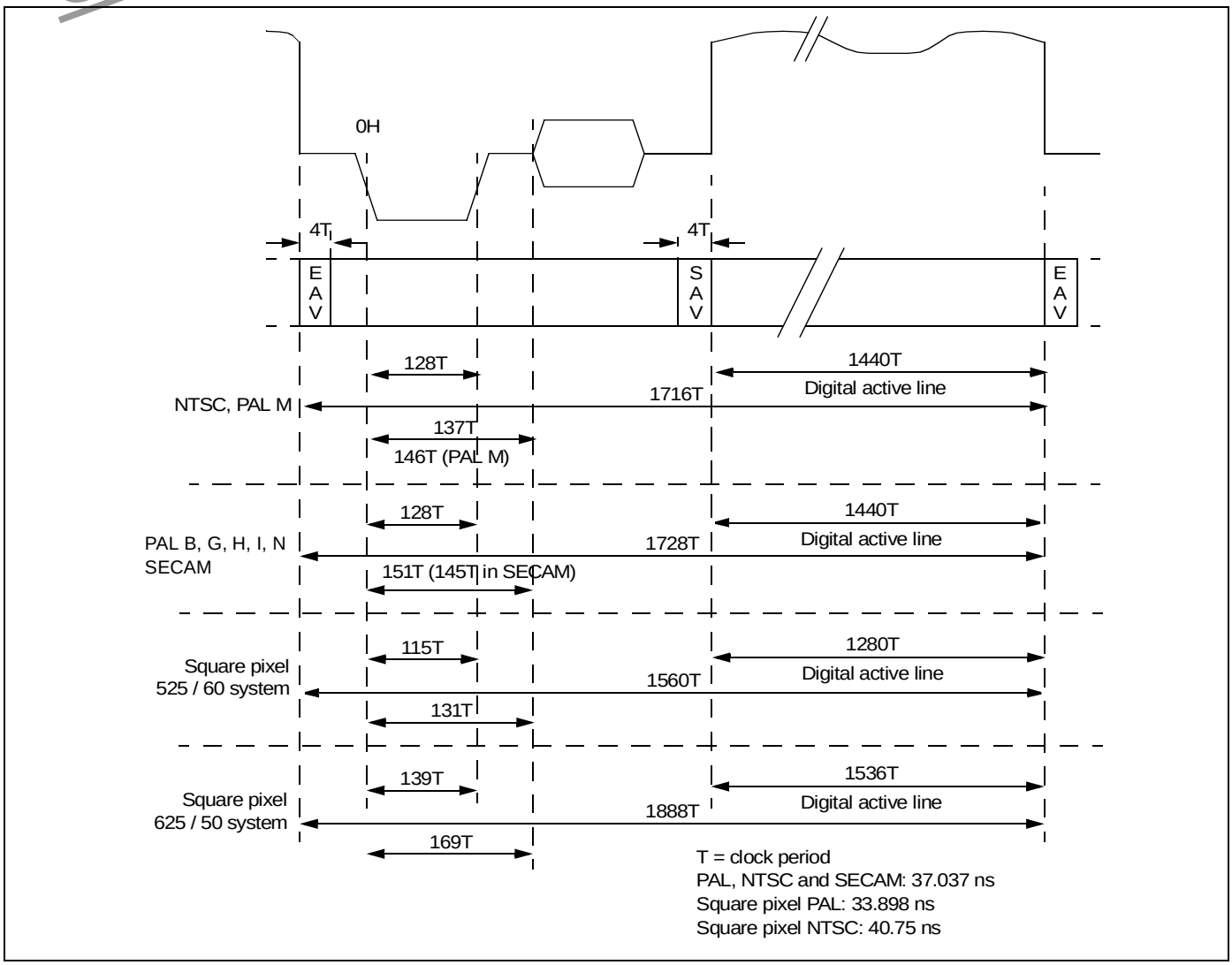


Figure 119 Input data format (ITU-R656 /D1 4:2:2)

Note The burst envelope shown here indicates the location from which the first subcarrier positive zero crossing is sought (with respect to the 0_H reference). The normal burst always starts with such a positive zero crossing.

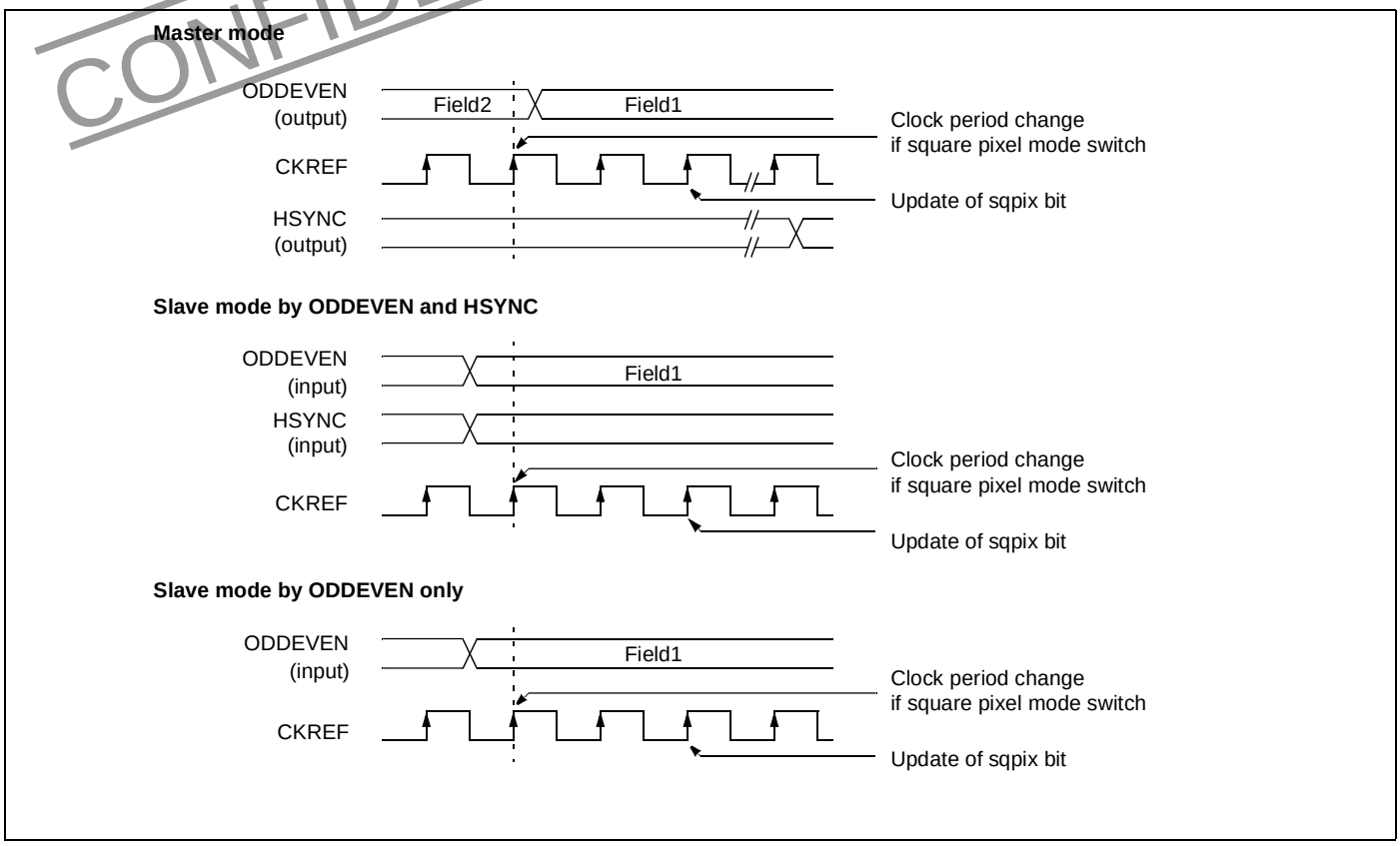


Figure 120 Square pixel mode switch

Note 1. These diagrams are valid with contents of "delay" and "synchro-delay" register fields equal to the default values.
2. If on-the-fly format changing is required, clock switching must be synchronized onto the start of frame as shown in the above waveform. Internally, sqpix bit update is taken into account on the beginning of a new frame.

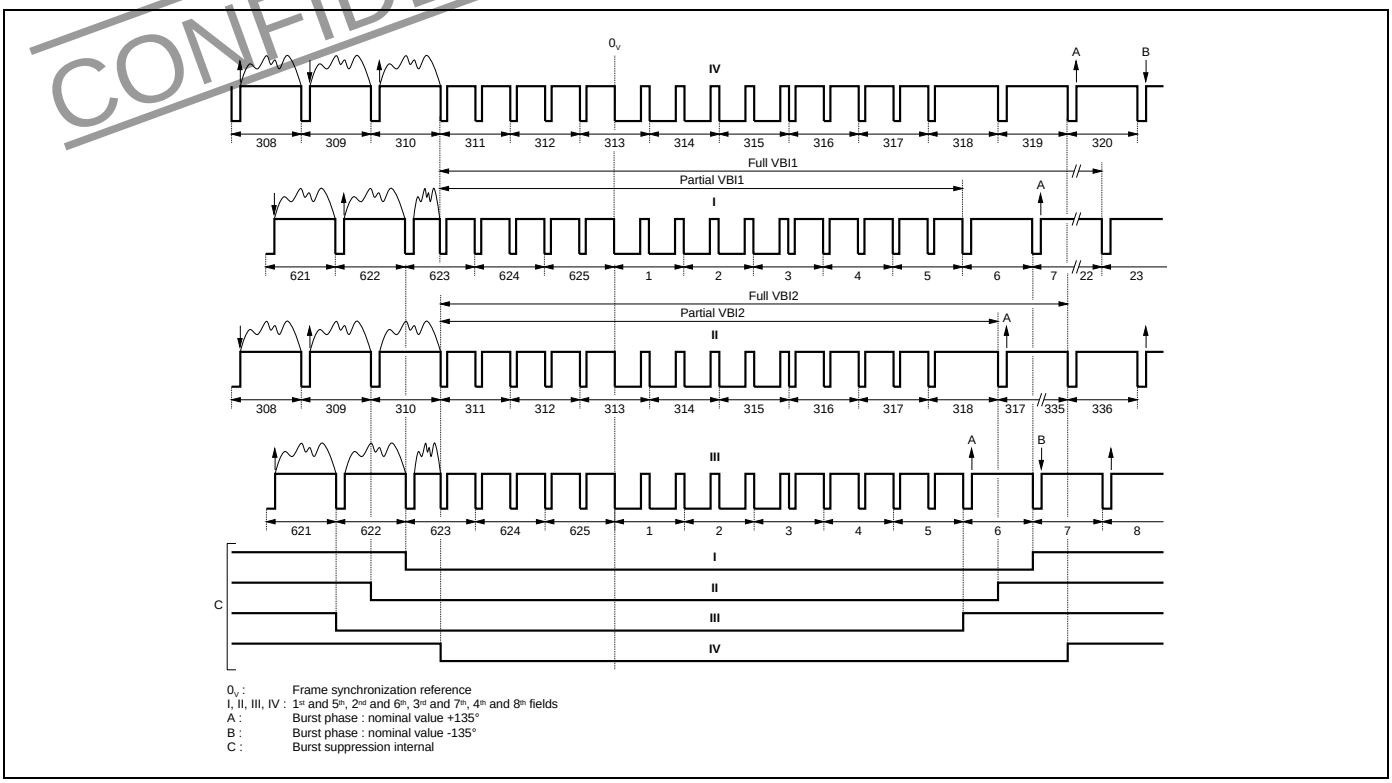


Figure 121 PAL-BDGHI, PAL-N typical VBI waveform, interlaced mode (ITU-R625 line numbering)

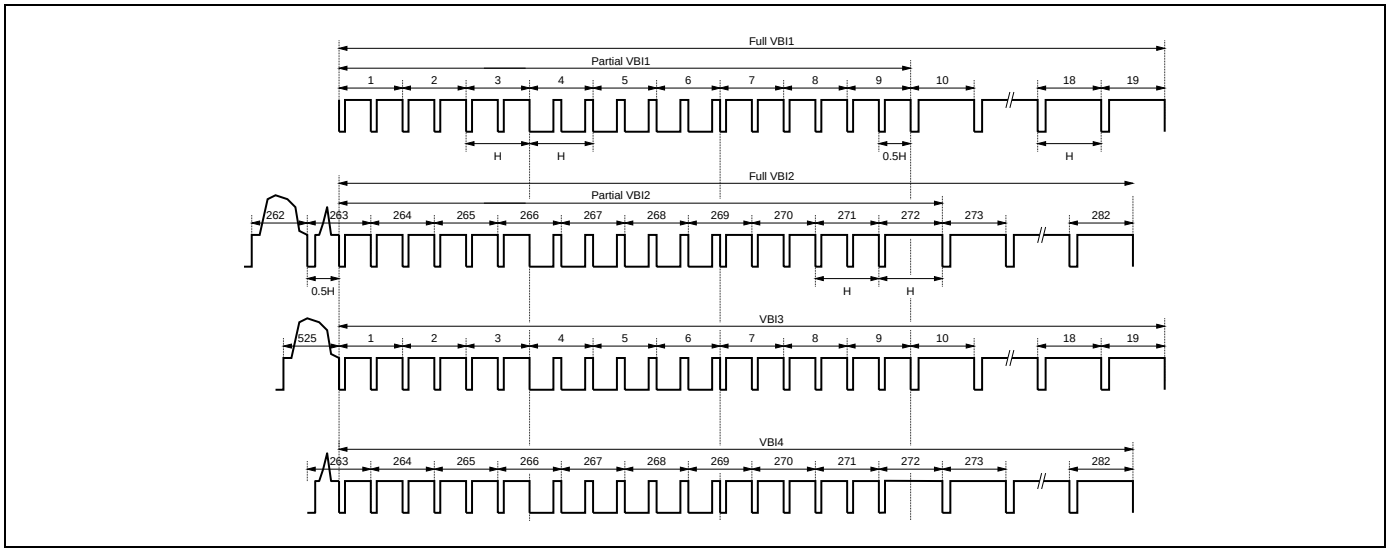


Figure 122 NTSC-M typical VBI waveforms, interlaced mode (SMPTE-525 line numbering)

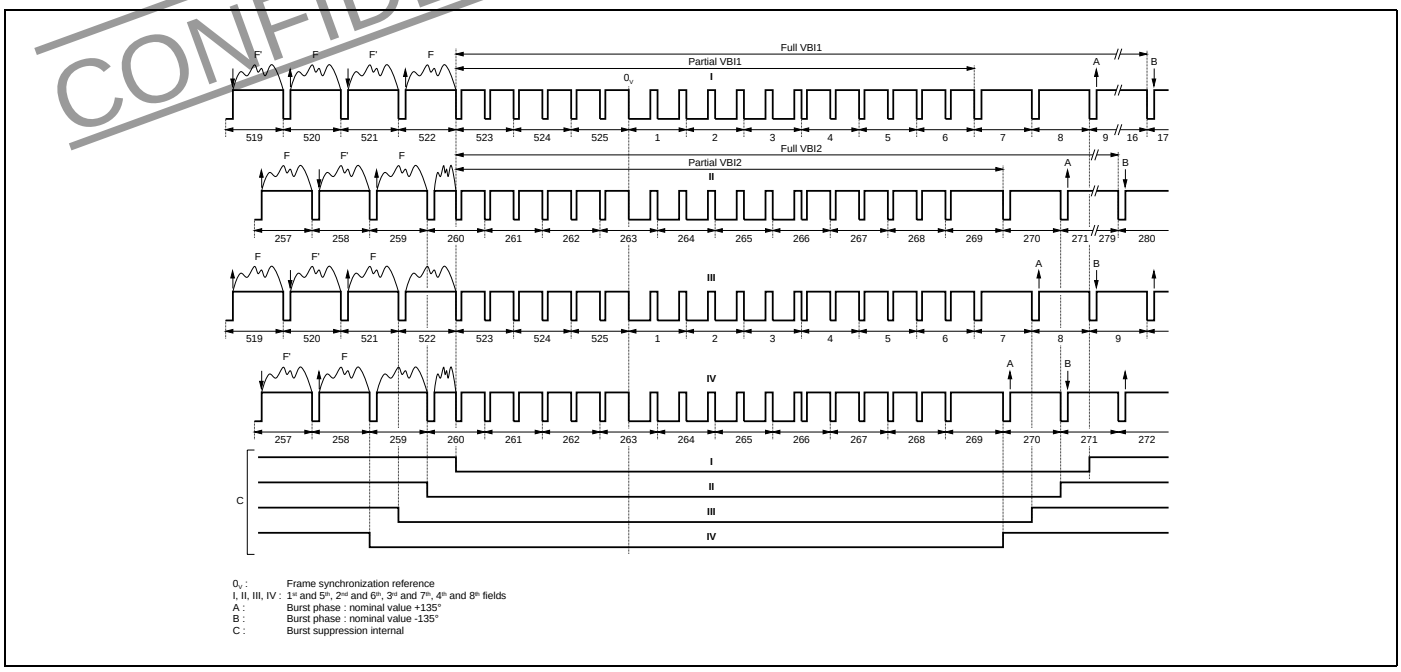


Figure 123 PAL-M typical VBI waveforms, interlaced mode (ITU-R/CCIR-525 line numbering)

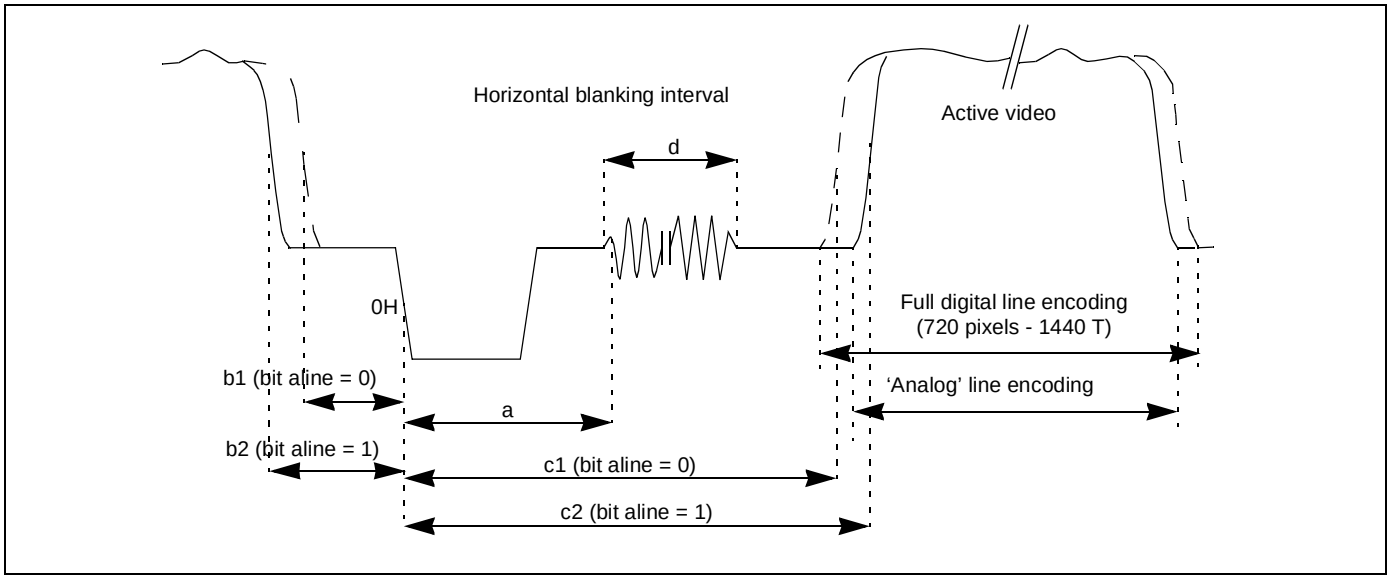


Figure 124 Horizontal blanking interval and active video timings

	NTSC-M	PAL-BDGHI	PAL-N	PAL-M	SECAM
a[1]	5.38 µs (even lines) 5.52 µs (odd lines)	5.54 µs (A-type) 5.66 µs (B-type)	5.54 µs (A-type) 5.66 µs (B-type)	5.73 µs (A-type) 5.87 µs (B-type)	5.60 µs
b1	1.56 µs	1.3 µs	1.3 µs	1.56 µs	1.0 µs
b2	1.56 µs	1.52 µs	1.52 µs	1.56 µs	1.52 µs
c1	8.8 µs	9.6 µs	9.6 µs	8.8 µs	9.9 µs

Table 97 Typical timing values for Figure 124

	NTSC-M	PAL-BDGHI	PAL-N	PAL-M	SECAM
c2	9.41 µs	10.48 µs	10.48 µs	9.41 µs	10.48 µs
d	9 cycles of 3.58 MHz	10 cycles of 4.43 MHz	9 cycles of 3.58 MHz	9 cycles of 3.58 MHz	-

Table 97 Typical timing values for Figure 124

1. These are typical values, actual values will depend of the static offset programmed for subcarrier generation.

20.3 Reset procedure

A hardware reset sets the DENC in HSYNC+ODDEVEN (line-locked) slave mode, for NTSC-M, interlaced ITU-R601 encoding closed-captioning, WSS, VPS, Teletext and CGMS encoding are all disabled.

The configuration can then be customized by writing into the appropriate registers. A few registers are never reset, their contents are unknown until the first loading (see the *STi5518* Register Manual).

It is also possible to perform a software reset by setting the 7th bit in the DEN_CFG6 register. The device responds in a similar way as after a hardware reset except that the configuration registers and a few other registers are not altered.

20.4 Master mode

In this mode, the encoder supplies HSYNC and ODDEV sync signals (with independently programmable polarities) to drive other blocks. Refer to the following figures for timings and waveforms.

The encoder starts encoding and counting clock cycles as soon as the master mode has been loaded into the control register DEN_CFG0.

Configuration bits syncout_ad[1:0] (register DEN_CFG4) shift the relative position of the sync signals by up to 3 clock cycles to cope with any YCrCb phasing.

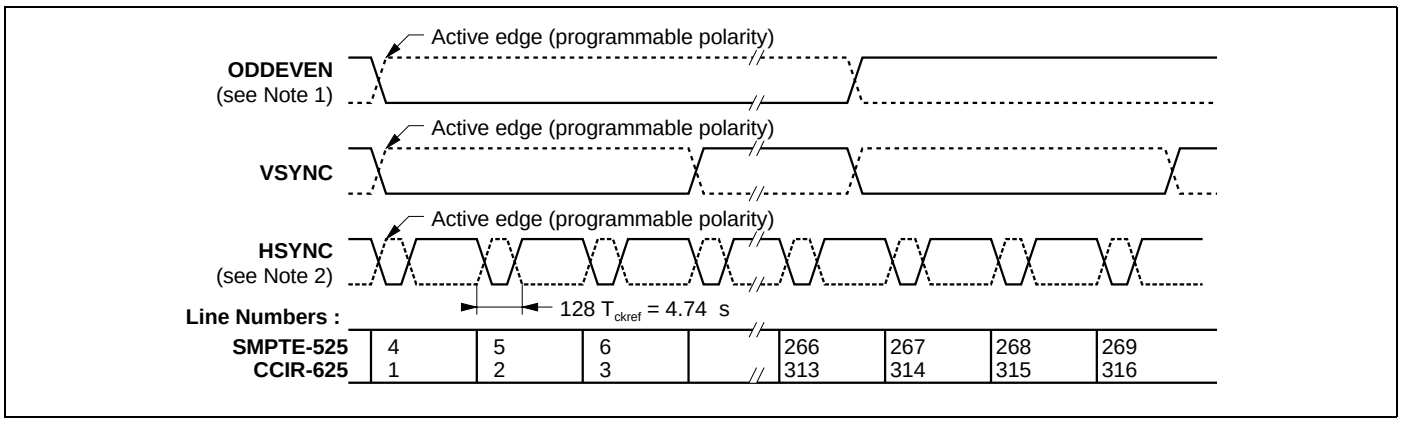


Figure 125 ODDEVEN, VSYNC and HSYNC waveforms

Note 1. When ODDEV is a sync input, only one edge ("the active edge") of the incoming ODDEV is taken into account for synchronization. The "non-active" edge (2nd edge on this drawing) is not critical and its position may differ by ±H/2 from the location shown.

Note 2. The HSYNC pulse width indicated is valid when the DENC supplies HSYNC. In those slave modes where it receives HSYNC, only the edge defined as active is relevant, and the width of the HSYNC pulse it receives is not critical.

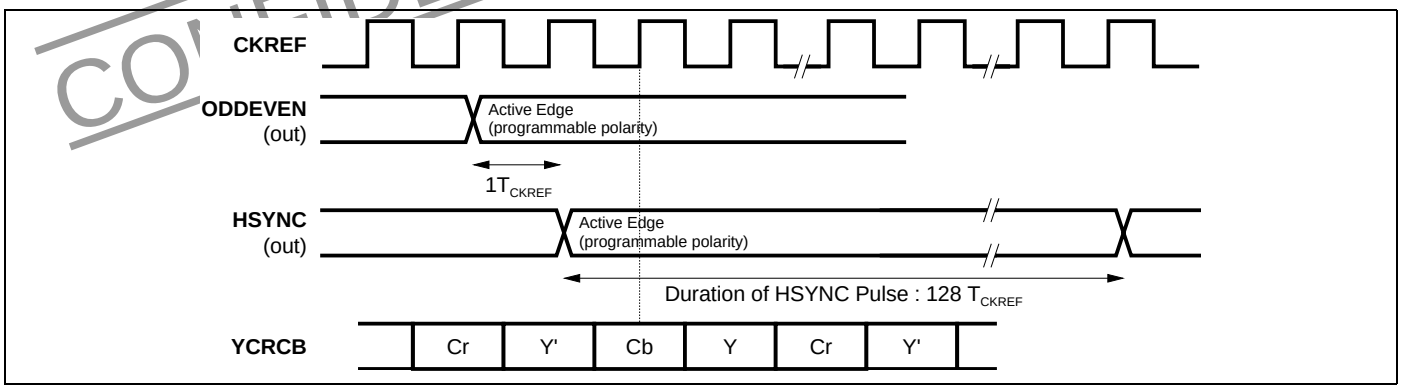


Figure 126 Master mode sync signals

Note This figure is valid for bits "syncout_ad[1:0]"= default

20.5 Slave modes

20.5.1 Introduction

The following slave modes are available:

- ODDEVEN(VSYNC) + HSYNC based (line-based sync),
- ODDEVEN(VSYNC)-only based (frame-based sync),
- sync-in-data based (line locked or frame locked).

ODDEVEN refers to an odd/even field flag, also known as BottomTop. HSYNC is a line sync signal and VSYNC is a vertical sync signal. Their waveforms are depicted in Figure 127 . The polarities of HSYNC and ODDEVEN(VSYNC) are independently programmable in all slave modes. In all slave modes, ODDEVEN(VSYNC) and/or HSYNC signals must be related to CKREF, the principal DENC clock. In other words, there is *NO* genlocking performed by the DENC.

20.5.2 Line-based synchronization

ODDEVEN+HSYNC based synchronization

Synchronization is performed on a line-by-line basis by locking onto incoming ODDEVEN and HSYNC signals. Refer to Figure 127 for waveforms and timings. The polarities of the active edges of HSYNC and ODDEVEN are programmable and independent. The first active edge of ODDEVEN initializes the internal line counter but encoding of the first line does not start until an HSYNC active edge is detected (at the earliest, an HSYNC transition may be at the same time as ODDEVEN). At that point, the internal sample counter is initialized and encoding of the first line starts. Then, encoding of each subsequent line is individually triggered by HSYNC active edges. The phase relationship between HSYNC and the incoming YCrCb data is normally such that the first clock rising edge following the HSYNC active edge samples Cb (i.e. a blue chroma sample within the YCrCb stream). It is however possible to internally delay the incoming sync signals (HSYNC+ODDEVEN) by up to 3 clock cycles to cope with different data/sync phases, using configuration bits syncin_ad in DEN_CFG4. The DENC is thus fully slaved to the HSYNC signal, which means that lines may contain more or less samples than usual.

- If the digital line is shorter than its nominal value, the sample counter is re-initialized when the "early" HSYNC arrives and all internal synchronization signals are re-initialized.

- If the digital line is longer than its nominal value, the sample counter is stopped when it reaches its nominal end-of-line value and waits for the “late” HSYNC before re-initializing.

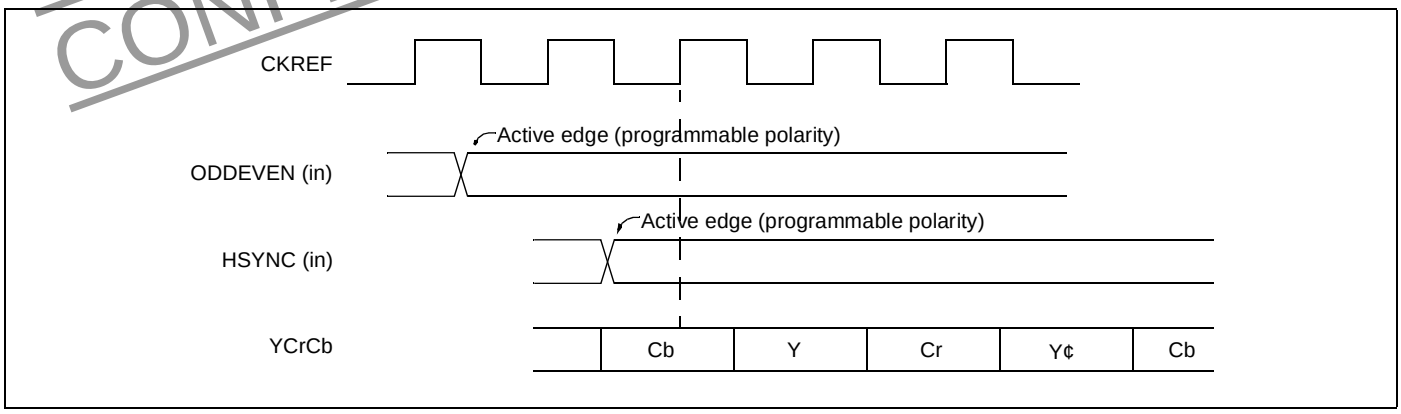


Figure 127 HSYNC + ODDEVEN based slave mode sync signals

*Note This figure is valid for bits **syncin_ad[1:0]** = default*

HSYNC+VSYNC based synchronization

Synchronization is performed on a line-by-line basis by locking onto incoming VSYNC and HSYNC signals. Refer to Figure 128 for waveforms and timings. The polarities of HSYNC and VSYNC are programmable and independent.

The incoming VSYNC signal is immediately transformed into a waveform identical to the odd/even waveform of an ODDEVEN signal, therefore, the behavior with this synchronization is identical to that described above for ODDEVEN+HSYNC based synchronization. Again, the phase relationship between HSYNC and the incoming YCrCb data is normally such that the first clock rising edge following the HSYNC active edge samples “Cb” (i.e. a 'blue' chroma sample within the YCrCb stream). It is however possible to internally delay the incoming sync signals (HSYNC+VSYNC) by up to 3 clock cycles to cope with different data/sync phasing, using configuration bits syncin_ad (DEN_CFG4).

20.5.3 Frame-based synchronization

ODDEVEN-only based synchronization

Synchronization is performed on a frame-by-frame basis by locking onto an incoming ODDEVEN signal. A line sync signal is derived internally and is also issued to the outside as HSYNC. Refer to Figure 129 for waveforms and timings. The phase relationship between ODDEVEN and the incoming YCrCb data is normally such that the first clock rising edge following the ODDEVEN active edge samples “Cb” (i.e. a “blue” chroma sample within the YCrCb stream). It is

however possible to internally delay the incoming ODDEVEN signal by up to 3 clock cycles to cope with different data/ sync phasing, using configuration bits syncin_ad in DEN_CFG4.

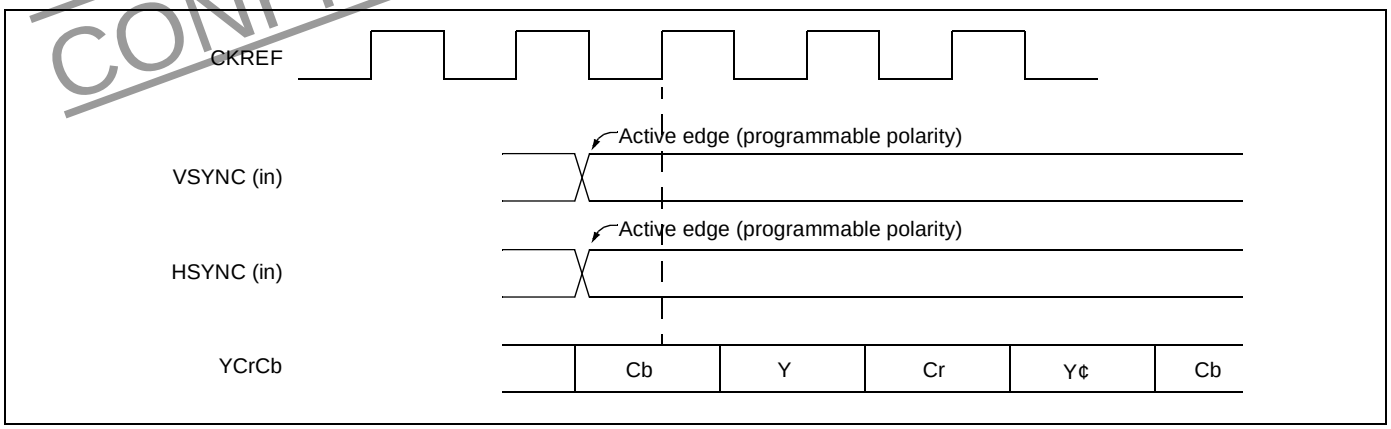


Figure 128 HSYNC + VSYNC based slave mode sync signals

Note 1. This figure is valid for bits "syncin_ad[1:0]" = default.
2. The active edges of HSYNC and VSYNC should normally be simultaneous. It is permissible that HSYNC transitions before VSYNC, but VSYNC must not transition before HSYNC.

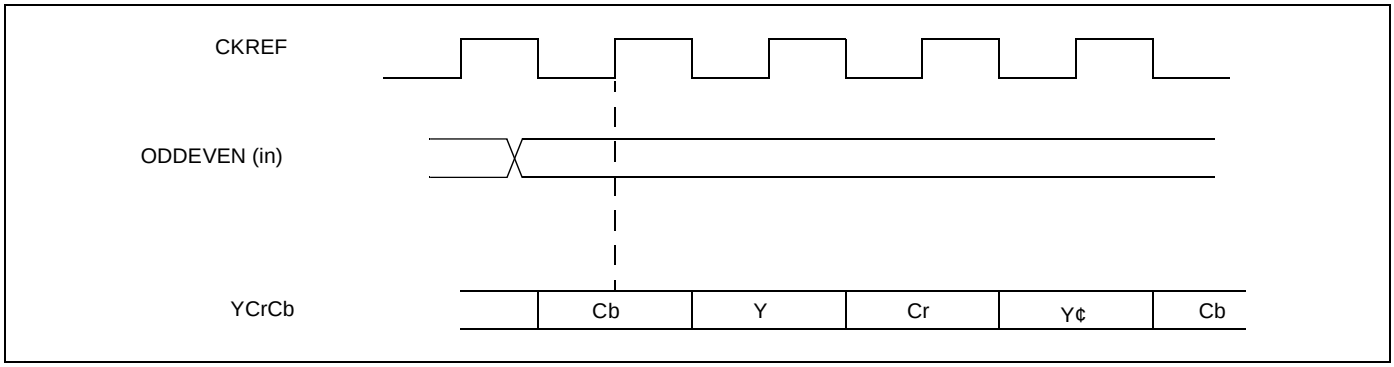


Figure 129 ODDEVEN based slave mode sync signals

Note This figure is valid for bits syncin_ad[1:0] = default

The first active edge of ODDEVEN triggers generation of the analog sync signals and encoding of the incoming video data. Frames being supposed to be of constant duration, the next ODDEVEN active transition is expected at a precise time after the last ODDEVEN detected.

So, once an active ODDEVEN edge has been detected, checks that the following ODDEVEN are present at the expected instants are performed.

Encoding and analog sync generation carry on unless three successive fails of these checks occur.

In that case, three behaviors are possible, according to the configuration programmed in registers DEN_CFG1-2:

- If freerun is enabled, the DENC carries on outputting the digital line sync HSYNC and generating analog video just as though the expected ODDEVEN edge had been present. However, it will re-synchronize onto the next ODDEVEN active edge detected, whatever its location.
- If freerun is disabled but bit syncok is set in the configuration registers, the DENC sets the active portion of the TV line to black level but carries on outputting the analog sync tips (on Ys and CVBS) and the digital line sync signal HSYNC. When programmed, Macrovision^{TM} pseudo-sync pulses and AGC pulses are also present in the analog sync waveform.

- If freerun is disabled and bit syncok is not set, all analog video is at black level and neither analog sync tips nor digital line sync are output.

This mode is a frame-based sync mode, as opposed to a field-based sync mode. This means that only one type of edge (rising or falling, according to programming) is of interest to the DENC; the other one is ignored.

VSYNC-only based synchronization

Synchronization is performed on a frame-by-frame basis by locking onto an incoming VSYNC signal.

An auxiliary line sync signal HSYNC must also be fed to the DENC, which uses it to reconstruct from VSYNC and HSYNC information an internal odd/ even waveform identical to that of an ODDEV signal. Therefore, the behavior with this synchronization is identical to that described above for ODDEV-only based synchronization (except that nothing is output on HSYNC pin since it is an input port in this mode).

Note that HSYNC is an input but has no other use than allowing the DENC to decide whether an incoming VSYNC pulse flags an odd or an even field. In other words, the DENC does not lock onto HSYNC in this mode since this is NOT a line-locked mode.

The phase relationship between VSYNC and the incoming YCrCb data is normally such that the first clock rising edge following the VSYNC active edge samples “Cb” (i.e. a 'blue' chroma sample within the YCrCb stream). It is however possible to internally delay the incoming sync signals (VSYNC+HSYNC) by up to 3 clock cycles to cope with different data/sync phasing, using configuration bits Syncin_ad (DEN_CFG4).

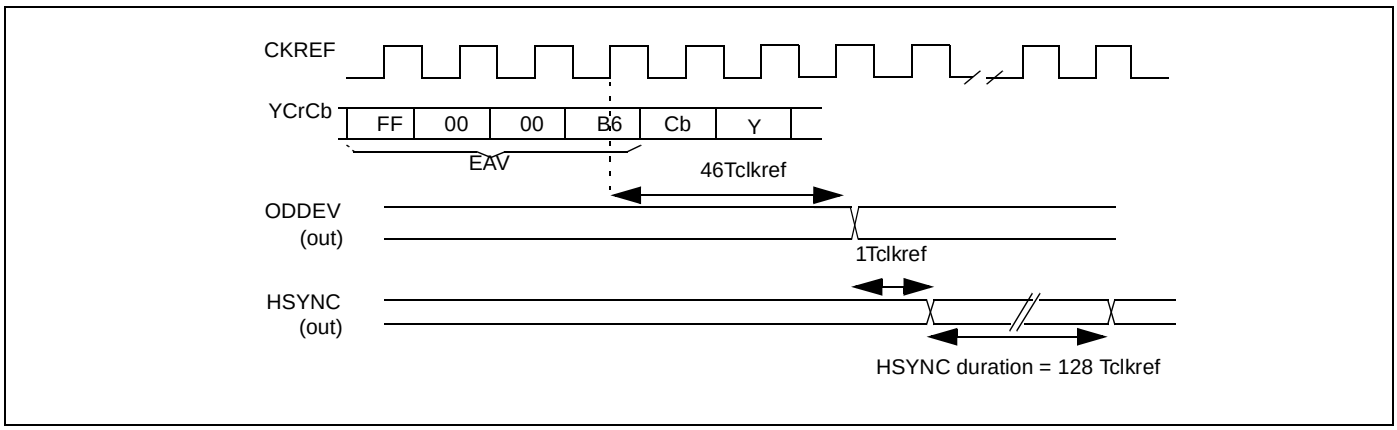


Figure 130 Data (EAV) based slave mode sync signals

20.5.4 Sync-in-data based synchronization

“End-of-frame” word-based synchronization

Synchronization is performed by extracting the 1-to-0 transitions of the ’F’ flag (end-of-frame) from the ’EAV’ (End-of-Active Video) sequence embedded within ITU-R656 / D1 compliant digital video streams. Both a frame sync signal and a line sync signal are derived and are made available externally as ODDEV and HSYNC. Refer to Figure 130 for waveforms and timings.

The first successful detection of the ’F’ flag triggers generation of the analog sync signals and encoding of the incoming video data. Frames being supposed to be of constant duration, the next EAV word containing the ’F’ flag is expected at a precise time after the latest detection.

So, once an active ’F’ flag has been detected, checks that the following flags are present within the incoming video stream at the expected times are performed.

Encoding and analog sync generation carry on unless there are three successive fails of these checks. Then, depending on the programmed configuration, one of the following three events occurs:

- If free-run is enabled, the DENC carries on generating the digital frame and line syncs (ODDEV and HSYNC) and generating analog video just as though the expected F flag had been present. However, it will re-synchronize onto the next F flag detected within the incoming ITU-R656/ D1 video stream.
- If free-run is disabled but the bit syncok is set in the configuration registers, the DENC sets the active portion of the TV line to black level but carries on outputting the analog sync tips (on Ys and CVBS) and the digital frame and line sync signals ODDEV and HSYNC. (When programmed, MacrovisionTM pseudo-sync pulses and AGC pulses are also present in the analog sync waveform).
- If free-run is disabled and the bit syncok is not set, all analog video is at black level and neither analog sync tips nor digital frame/line sync are output.

The SAV and EAV words are Hamming-decoded.

After detection of two successive errors, a bit is set in the status register to inform the micro-controller of the poor transmission quality.

'End-of-line' word-based synchronization

Synchronization is performed by extracting the F and 'H' flags from the SAV (Start of Active Video) and EAV (End of Active Video) words embedded within ITU-R656/D1 compliant digital video streams. A line sync signal and a frame sync signal are derived from these flags and are issued to the outside on the HSYNC and ODDEVEN/VSYNC pins in output mode. These signals are also used by the DENC, which treats them as incoming ODDEVEN and HSYNC signals in HSYNC+ODDEVEN based synchronization.

Auto-test mode

An auto-test mode is available, which causes the DENC to produce a color bar pattern, in the appropriate standard, independently from the video input.

The auto-test mode is started by setting to 7 the 3-bit field sync in the register DEN_CFG0. As this mode sets the DENC in master mode, VSYNC/ODDEVEN and HSYNC signals are in output mode. In table below, the decimal values of Y, Cr and Cb are shown corresponding to the auto-test color bar.

	Y	Cr	Cb
Black	16	128	128
Blue	36	116	212
Red	64	212	100
Magenta	84	200	184
Green	112	56	72
Cyan	136	44	156
Yellow	160	140	44
White	236	128	128

Table 98 Auto-test colors

The corresponding decimal output values just before the DACs are shown graphically in Figure 132 and Figure 133 . Both figures show the static values corresponding to the input values in Table 98 .

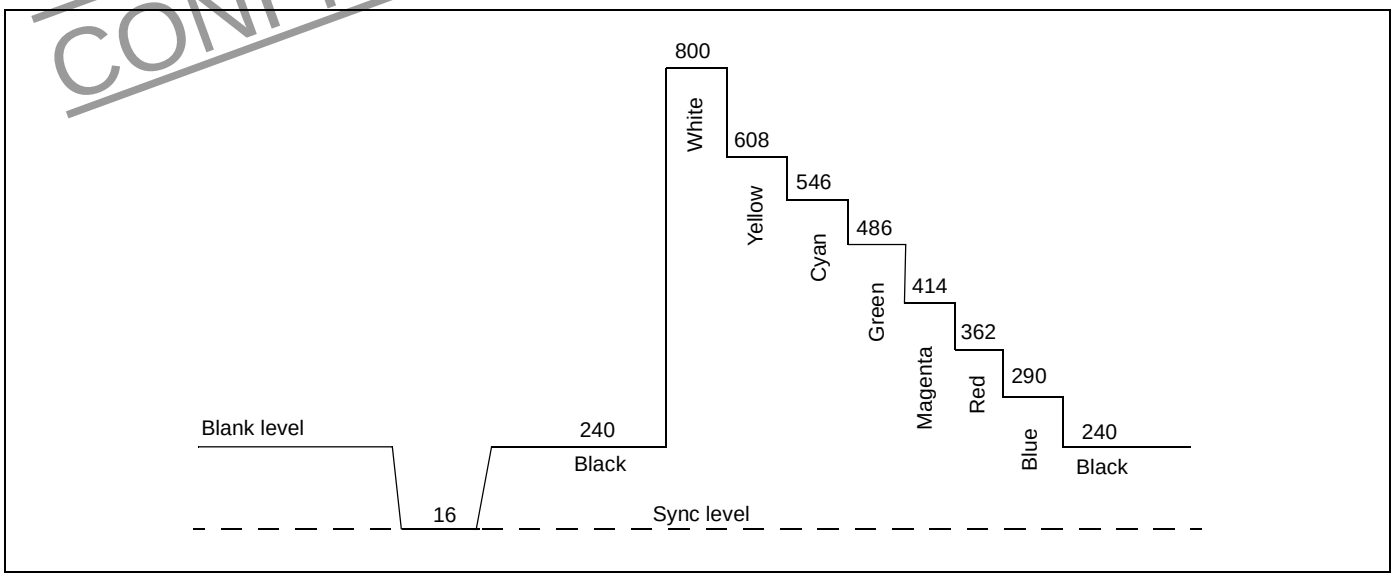


Figure 131 Luminance output levels in auto-test for NTSC without set-up

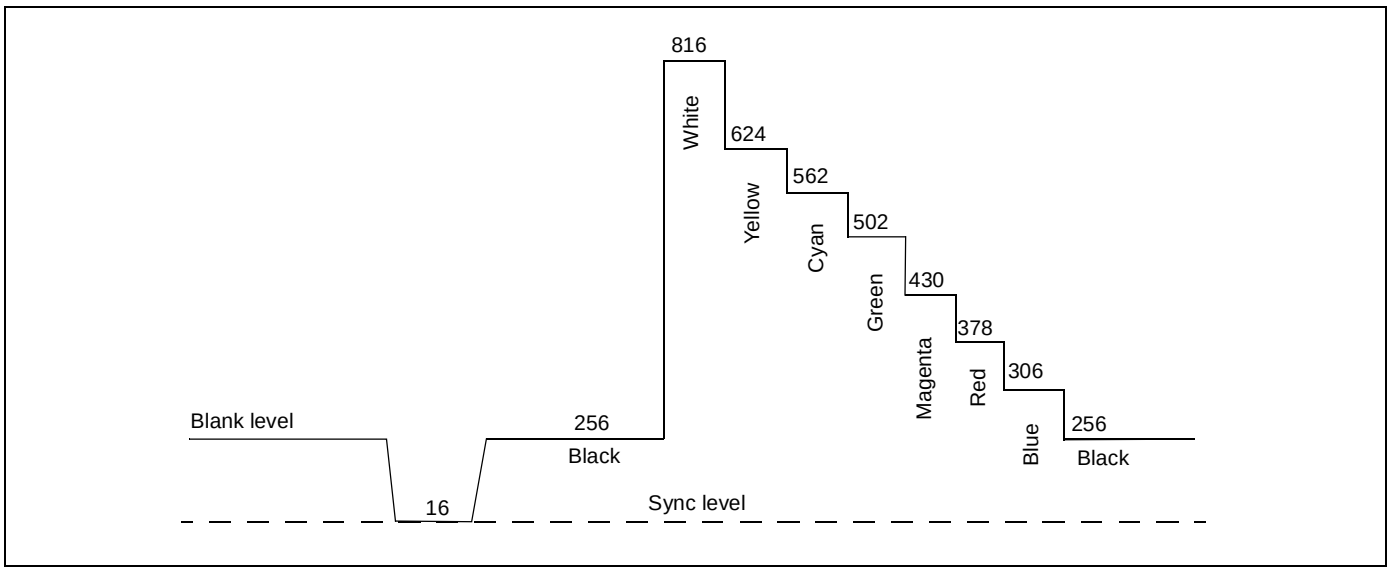


Figure 132 Luminance output levels in auto-test for PAL (BGHI) and SECAM

20.6 Input demultiplexor

The incoming YCrCb data, as well as Y4 and CrCb in 4:4:4 mode, is demultiplexed into a “blue-difference” chroma information stream, a “red-difference” chroma information stream and a luma information stream. Incoming data bits are treated as blue, red or luma samples according to their relative position with respect to the sync signals in use and the contents of configuration bits syncin_ad (slave modes) or syncout_ad (master mode). Brightness, saturation and contrast are then performed on demultiplexed data, refer to the Register Manual registers DEN_REG_69, DEN_REG_70 and DEN_REG_71.

The ITU-R601 recommendation defines the black luma level as Y=16 and the maximum white luma level as Y = 235. Similarly, it defines 225 quantification levels for the color difference components (Cr, Cb), centered around 128. After

the saturation, brightness and contrast stage, the incoming YCrCB samples can be saturated in the input multiplexer with the following rules:

- For Cr or Cb samples: Cr,Cb > 240 => Cr,Cb saturated at 240
  Cr,Cb < 16 => Cr,Cb saturated at 16
- For Y samples: Y > 235 => Y saturated at 235
  Y < 16 => Y saturated at 16

This avoids having to heavily saturate the composite video codes before digital-to-analog conversion in case erroneous or unrealistic YCrCb samples are input to the encoder (there may otherwise be overflow errors in the codes driving the DACs), and therefore avoids generating a distorted output waveform.

However, in some applications, it may be desirable to let extreme YCrCb codes pass through the demultiplexor. This is controlled using bit maxdyn in register DEN_CFG6. In this case, only codes 0x00 and 0xFF are overridden; if such codes are found in the active video samples, they are forced to 0x01 and 0xFE.

In any case, the YCrCb codes are not overridden for EAV/SAV decoder.

20.7 Subcarrier generation

A Direct Digital Frequency Synthesizer (DDFS) generates the required color subcarrier frequency using a 24-bit phase accumulator. This oscillator feeds a quadrature modulator which modulates the base-band chrominance components.

The subcarrier frequency is obtained from the following equation:

$$Fsc = (Increment\_Word / 2^{24}) \times CKREF$$

where *Increment _Word* is a 24-bit value.

Hard-wired *Increment_Word* values are available for each standard and can be automatically selected. Alternatively (according to bit selrst_inc in DEN_CFG5), the frequency can be fully customized by programming other values into a dedicated *Increment_Word* register, DEN_IDFS. This allows, for instance, the encoding of NTSC-4.43 or PAL-M-4.43.

This is done with the following procedure:

- Program the required increment in DEN_IDFS.
- Set bit selrst_inc to 1 in register DEN_CFG5.
- Perform a software reset using register DEN_CFG6. This sets all bits in all DENC registers except DEN_CFGn to their default value.
  Alternatively, set DEN_CFG8 bits ph_rst_mode[1:0] to 01. Then the frequency (and phase) update is done on the beginning of the next video line.

Warning: if a standard change occurs after the software reset, the increment value is automatically re-initialized with the hard wired or loaded value according to bit selrst

The reset phase of the color subcarrier can also be software-controlled by register DEN_PDFS.

The subcarrier phase can be periodically reset to its nominal value to compensate for any drift introduced by the finite accuracy of the calculations. In PAL and NTSC subcarrier phase adjustment can be performed every line, every eight fields, every four fields, or every two fields (DEN_CFG2 bits valrst[1:0]). If SECAM is performed, the subcarrier phase is reset every line.

20.8 Burst insertion (PAL and NTSC)

The color reference burst is inserted so as to always start with a positive zero crossing of the subcarrier sine wave. The first and last half-cycles have a reduced amplitude so that the burst envelope starts and ends smoothly.

The burst contains 9 or 10 sine cycles of 4.43361875 MHZ or 3.579545 MHz (depending on the standard programmed in the register DEN_CFG0) as follows:

NTSC-M	9	cycles of	3.579545	MHz
PAL-BDGHI	10	cycles of	4.43361875	MHz
PAL-M	9	cycles of	3.579545	MHz
PAL-N	9	cycles of	3.579545	MHz

The burst can be turned off (no burst insertion) by setting DEN_CFG2 configuration bit bursten to 0.

Burst insertion is performed by always starting the burst with a positive-going zero crossing. This guarantees a smooth start and end of burst with a maximum of undistorted burst cycles and can only be beneficial to chroma decoders.

This avoids an uncontrolled initial burst phase, and guarantees a start on a positive-going zero crossing with the consequence that two burst start locations are visible over successive lines, according to the line parity. This is normal and explained below.

In NTSC, the relation between subcarrier frequency and line length creates a 180^o subcarrier phase difference (*with respect to the horizontal sync*) from one line to the next according to the line parity. So if the burst always starts with the same phase (positive-going zero crossing), this means the burst will be inserted at time X or at time $X+T_{NTSC}/2$ after the horizontal sync tip according to the line parity, where T_{NTSC} is the duration of one cycle of the NTSC burst.

With PAL, a similar rationale holds, and again there will be two possible burst start locations. The subcarrier phase difference (*with respect to the horizontal sync*) from one line to the next in that case is either 0 or 180^o with the following series: A-A-B-B-A-A-...-etc. where A denotes “A-type” bursts and B denotes “B-type” bursts, A-type and B-type being 180^o out of phase with respect to the horizontal sync. So two locations are possible, one for A-type, the other for B-type.

This assumes a periodic reset of the subcarrier is automatically performed (see bits valrst[1:0] in DEN_CFG2). Otherwise, over several frames, the start of burst will drift within an interval of half a subcarrier’s cycle. THIS IS NORMAL and means the burst is correctly locked to the colors encoded. The equivalent effect with a gated burst approach would be the following: the start location would be fixed but the phase with which the burst starts (*with respect to the horizontal sync*) would be drifting.

20.9 Subcarrier insertion (SECAM)

subcarrier frequency in SECAM mode depends on Cr and Cb values (frequency modulation). The color subcarrier frequency is 4,250,000 Hz for Cb=128 (on blue lines) and 4,406,249 Hz for Cr=128 (on red lines). Frequency clipping values are 3,900,000 Hz and 4,756,250 Hz.

The insertion point of the non-modulated subcarrier is shown in the figure below.

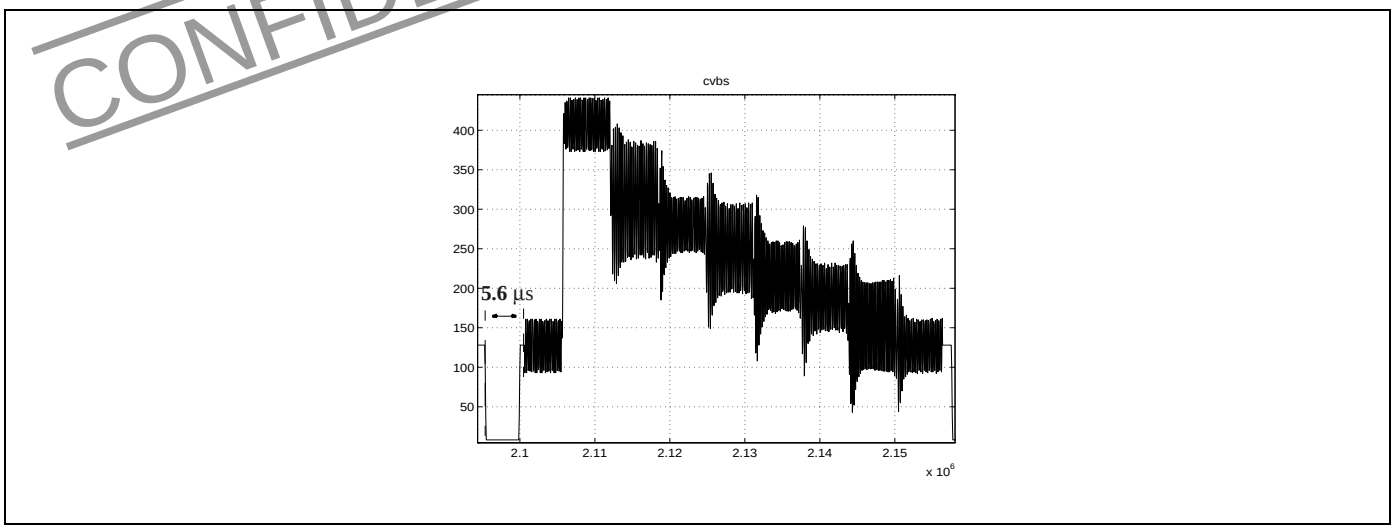


Figure 133 SECAM color bar pattern (blue line)

In odd fields the phase of subcarrier follows the sequence: 0, 0, π, 0, 0, π, 0, 0, π, ... comparing to a sine wave starting at the same point - 5.6 µs after horizontal synch pulse (inverted on one line out of every three and also at each frame). This sequence begins from line 1 or line 23 of the first field (see gen_secam bit of register DEN_CFG7). DEN_CFG7 bit inv_phi_secam allows the inversion of this sequence (π, π, 0, π, π, 0,... instead of 0, 0, π, 0, 0, π,...), in odd fields. In even fields the sequence of subcarrier is always inverted with respect to the odd field one.

To enable SECAM mode, program a 1 in DEN_CFG7.7 (MSB) and then perform a soft-reset or loading of DEN_CFG0.

20.10 Luminance encoding

The demultiplexed Y samples are band-limited and interpolated at CKREF clock rate. The resulting luminance signal is properly scaled before insertion of any closed-captions, CGMS, VPS, Teletext or WSS data and synchronization pulses.

The interpolation filter compensates for the sin(x)/ x attenuation inherent in D/A conversion and greatly simplifies the output stage filter. See Figures 134, 135 and 136 for characteristic curves.

In addition, the luminance that is added to the chrominance to create the composite CVBS signal can be trap-filtered at 3.58 MHz (NTSC) or 4.43 MHz (PAL). This supports applications oriented towards low-end TV sets which are subject to cross-color if the digital source has a wide luminance bandwidth (e.g. some DVD sources). Note that the trap filter does not affect the S-VHS luminance output nor the RGB outputs. If SECAM is performed, enable the trap filter with 4.43 MHz cut-off frequency on the luma part of the CVBS signal (see DEN_CFG3 bits entrap and trap_4.43).

A 7.5 IRE pedestal can be programmed if needed with all standards (see registers DEN_CFG1 and DEN_CFG7). This allows in particular to encode Argentinian and non-Argentinian PAL-N, or Japanese NTSC (NTSC with no set-up).

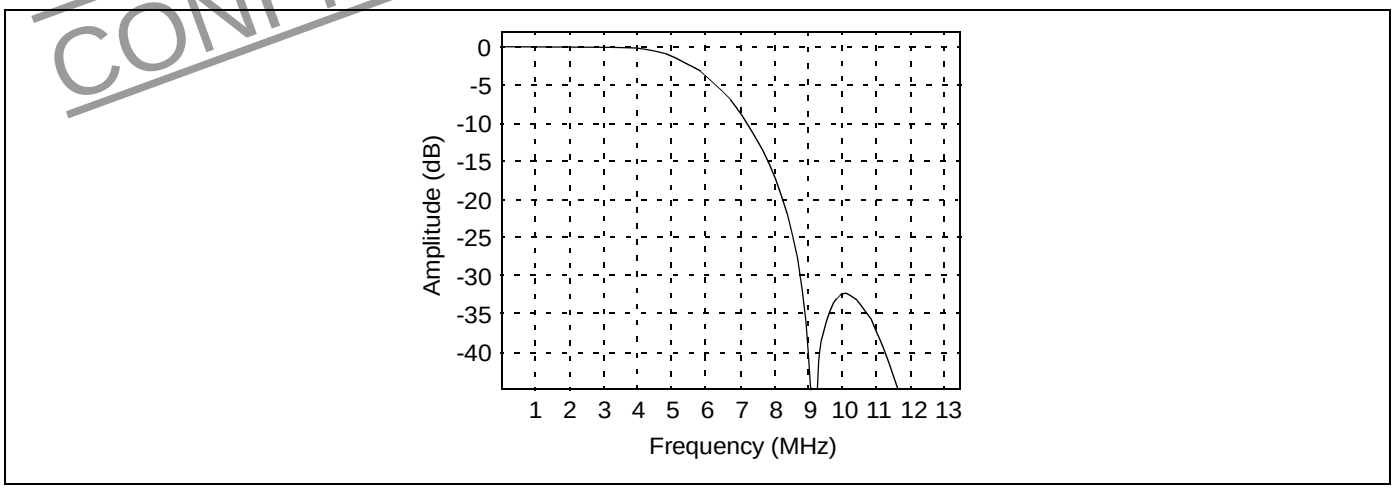


Figure 134 Luma filtering including DAC attenuation

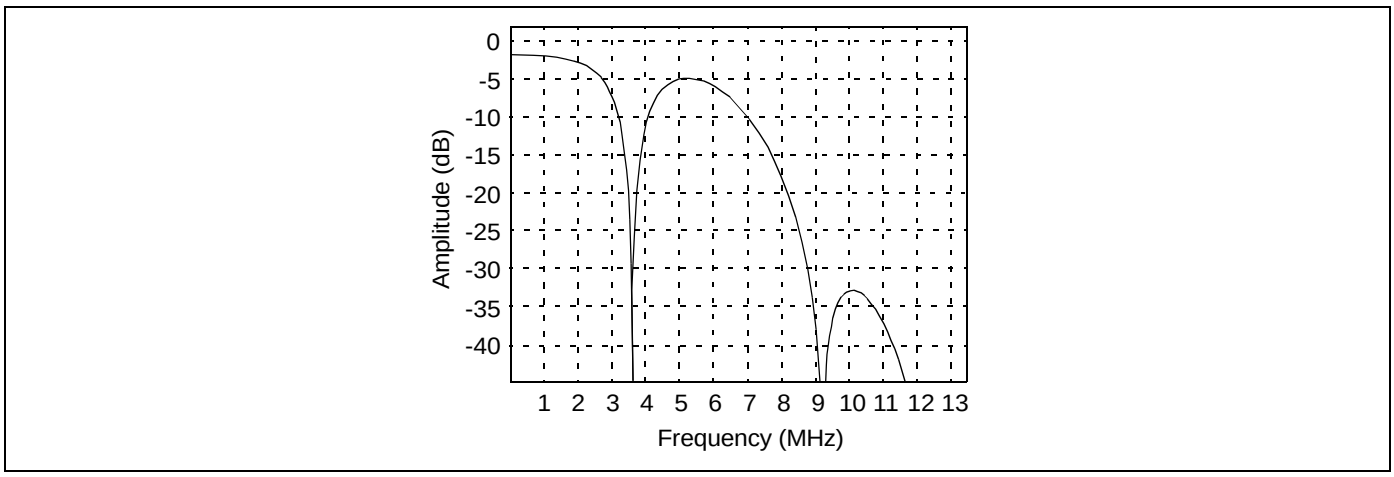


Figure 135 Luma filtering with 3.58 MHz trap, including DAC attenuation

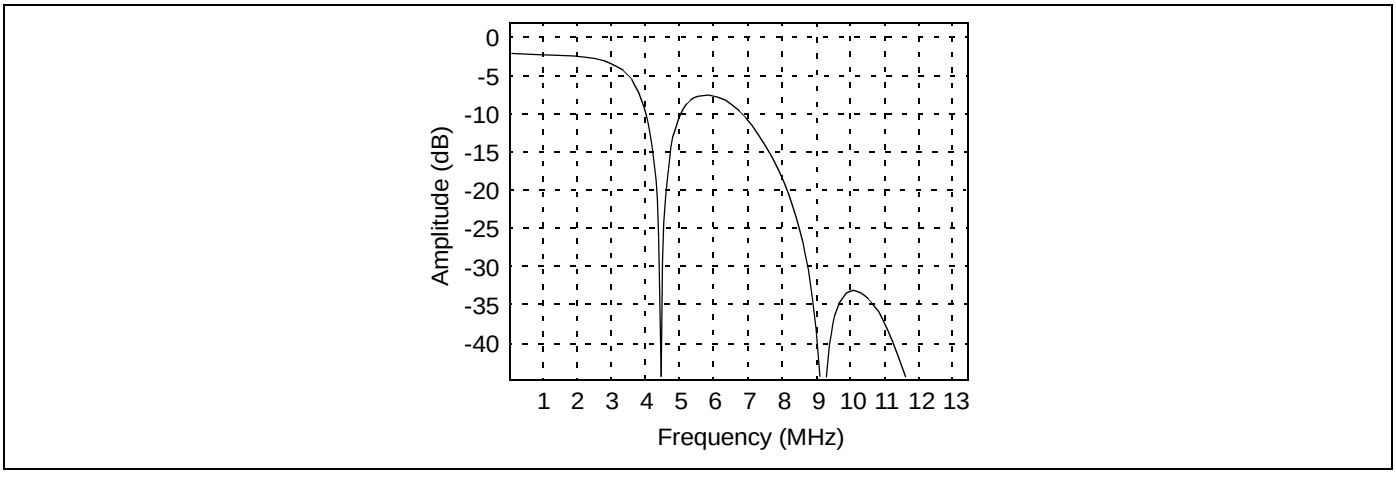


Figure 136 Luma filtering with 4.43 MHz trap, including DAC attenuation

The luma processing as well as line and field timings in SECAM mode are identical to PAL BDGHI ones.

20.11 Chrominance encoding

U, V (PAL and NTSC) and Dr, Db (SECAM) chroma components are computed from demultiplexed Cb, Cr samples. Before modulating the subcarrier, these are band-limited and interpolated at CKREF clock rate. This processing eases the filtering following D/A conversion and allows more accurate encoding.

A set of 4 different filters is available in PAL and NTSC for chroma filtering to fit a wide variety of applications in the different standards and include filters recommended by ITU-R 624-4 and SMPTE170-M.

The available 3-dB bandwidths are 1.1, 1.3, 1.6 or 1.9 MHz. See Figures 139, 140, 141, 142 and 143 for the various frequency responses and register DEN_CFG1 for programming. The narrower bandwidths are useful against cross-luminance artifacts, the wider bandwidths allow higher chroma contents.

In SECAM, 1.3 MHz low-pass and pre-emphasis filtering are performed on Dr and Db chroma components, before the frequency modulation, according to ITU-R Rec624-4.

Refer to Figure 137 for frequency response of these filters. Bell filtering is performed at the end of frequency modulation stage.

Peak to peak amplitude of modulated chrominance signal at the central frequency (4 279.7 kHz) is 22,88% of the black-white interval (22.88 IRE).

Refer to Figure 138 for frequency response of bell filter with subcarrier frequencies and clipping values.

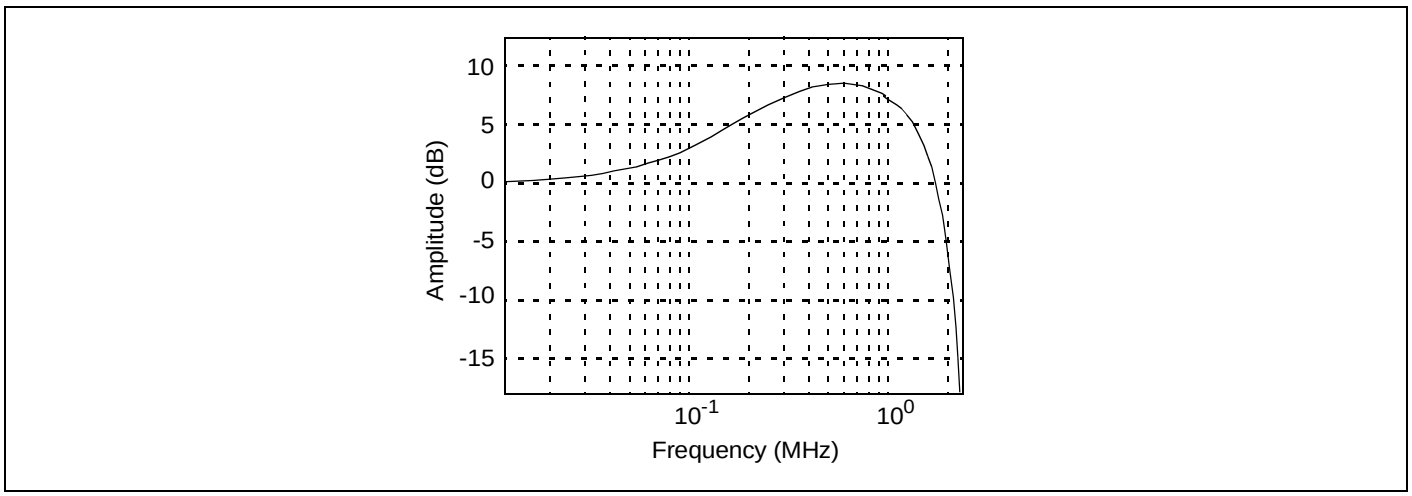


Figure 137 SECAM chroma filtering (pre-emphasis and 1.3 MHz low pass filtering)

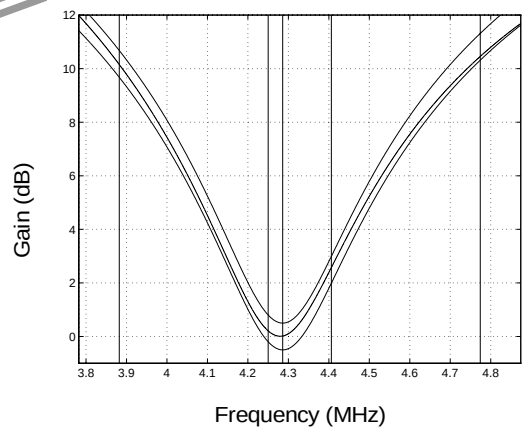


Figure 138 SECAM high-frequency subcarrier pre-emphasis (Bell filtering), including DAC attenuation

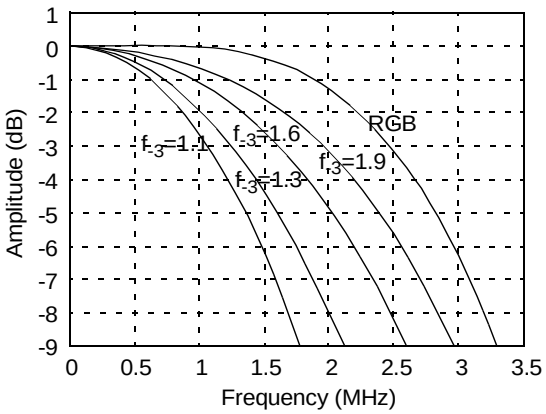


Figure 139 Various chroma filters available and RGB filter

20.12 Composite video signal generation

The composite video signal is created by adding the luminance (after trap filtering - optional in PAL and NTSC, see register DEN_CFG3) and the chrominance components. A saturation function is included in the adder to avoid overflow errors should extreme luminance levels be modulated with highly saturated colors. This does not occur with natural colors but may be generated by computers or graphics engines.

A “color killing” function is available, whereby the composite signal contains no chrominance, i.e. replicates the trap-filtered luminance. This function does not suppress the chrominance on the S/VHS outputs, but suppressing the S-VHS chrominance is possible using bit bkdac*n* in DEN_CFG5, where the chrominance signal is outputted on DAC *n*.

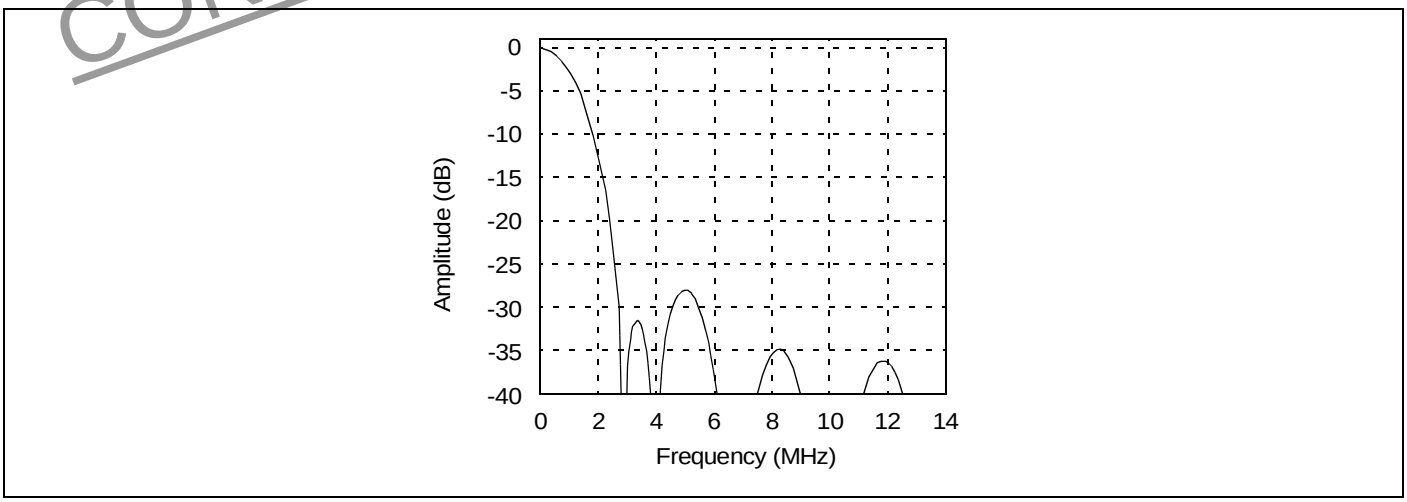


Figure 140 1.1 MHz chroma filter

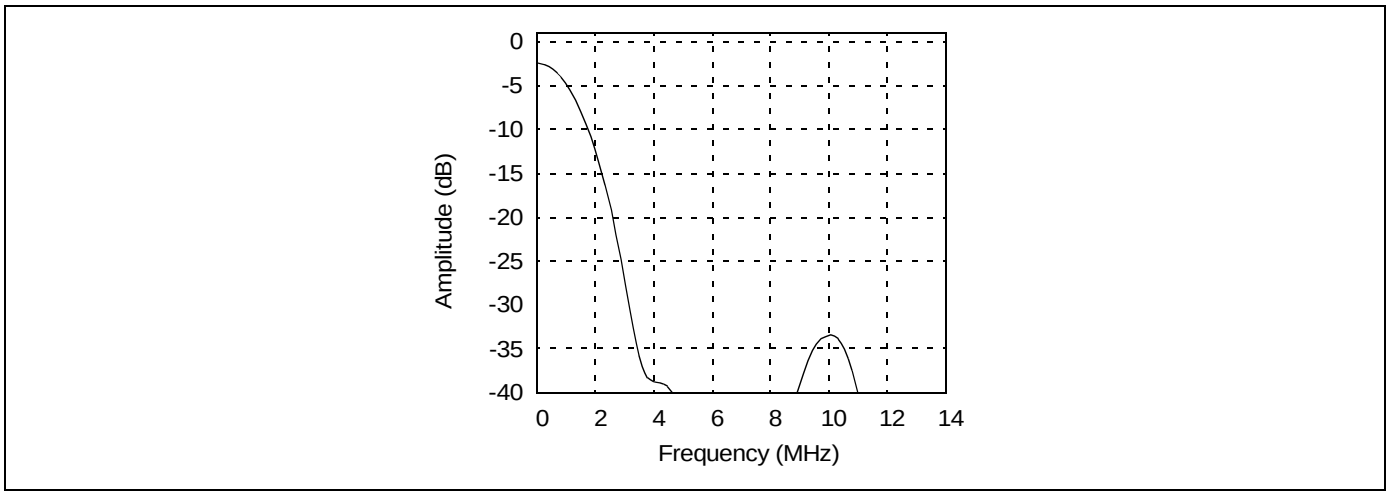


Figure 141 1.3 MHz chroma filter

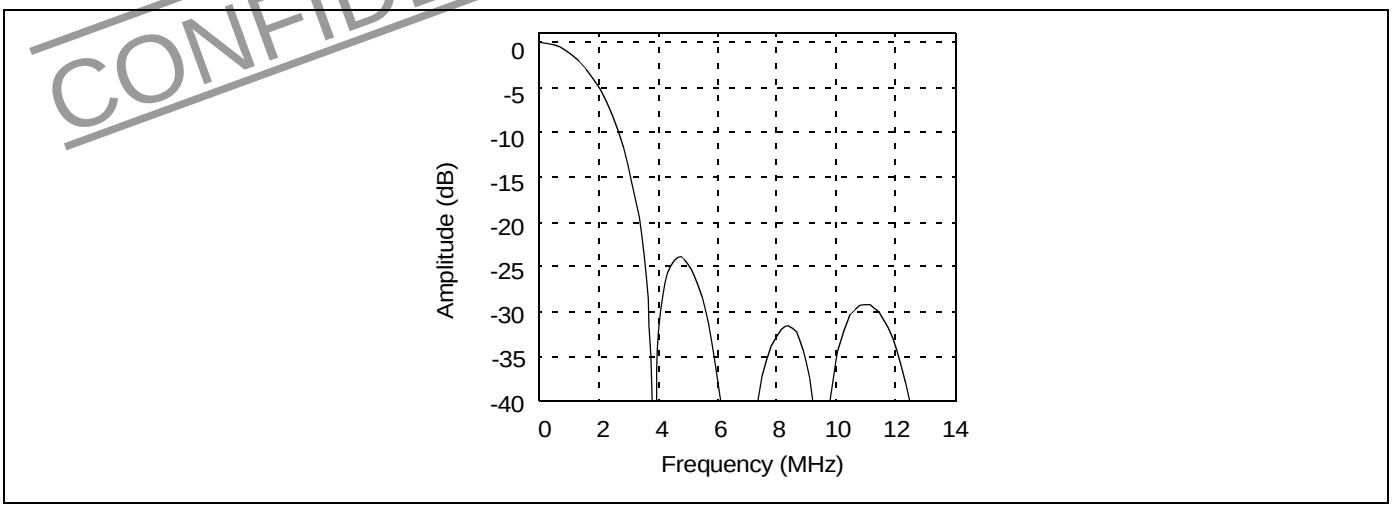


Figure 142 1.6 MHz chroma filter

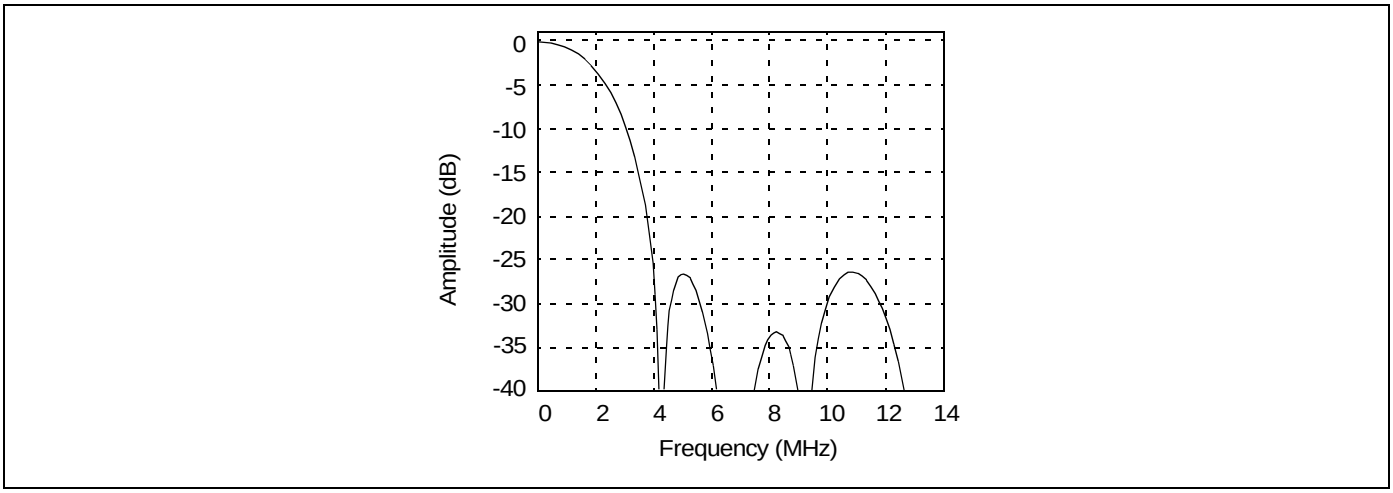


Figure 143 1.9 MHz chroma filter

20.13 RGB and UV encoding

After demultiplexing, the Cr and Cb samples feed a 4 times interpolation filter. The resulting base-band chroma signal has a 2.45 MHz bandwidth (Figure 144) and is combined with the filtered luma component to generate R,G,B or U,V samples at 27 MHz.

If Y4 and CrCb inputs are used, the filtering identical to luma filtering (see Figure 134) is performed on all components (Y4, Cr and Cb). In this case DAC5 output data encoded from Y4 input if YUV configuration is used (see **DEN_CFG8** bits **conf_out1** and **conf_out0**).

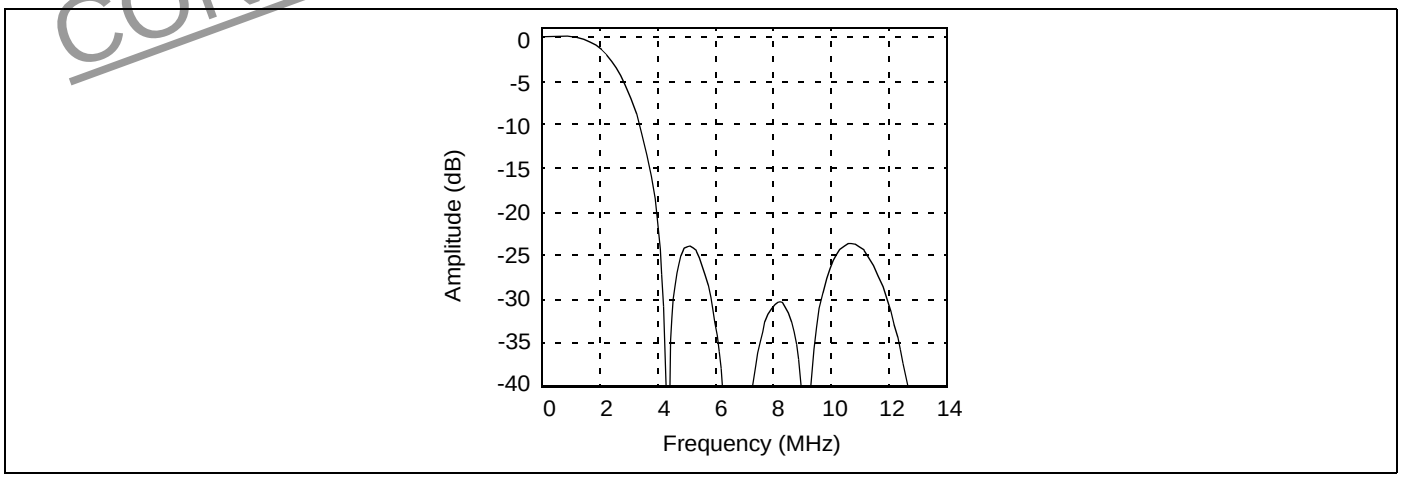


Figure 144 RGB - chroma filtering

20.14 Closed-captioning

Closed-captions (or data from an Extended Data Service as defined by the closed-captions specification) can be encoded by the circuit. The closed-caption data is delivered to the circuit through the register interface. Two dedicated pairs of bytes (two bytes per field), each pair preceded by a clock run-in and a start bit can be encoded and inserted on the luminance path on a selected TV line. The Clock Run-In and Start code are generated by the DENC.

Closed-caption data registers are double-buffered so that loading can be performed anytime, even during line 21/284 or any other selected line.

User register DEN_CCF1 and DEN_CCF2 each contain the first and second byte to send (LSB first) after the start bit on the appropriate TV line, where DEN_CCF1 refers to field 1 and DEN_CCF2 to field 2. The TV line number where data is to be encoded is programmable using registers DEN_CLF1 and DEN_CLF2. Lines that may be selected include those used by the *StarSight* data broadcast system. Closed-caption data has priority over any CGMS signals programmed for the same line.

The internal Clock Run-In generator is based on a Direct Digital Frequency Synthesizer. The nominal instantaneous data rate is 503,496 kHz (i.e. 32 times the NTSC line rate). Data LOW corresponds nominally to 0 IRE, data HIGH corresponds to 50 IRE at the DAC outputs.

When closed-captioning is on (bits cc1 and cc2 in DEN_CFG1), the CPU should load the relevant registers (DEN_CCF1 or DEN_CCF2) once every frame at most (although there is in fact some margin due to the double-

buffering). Two bits are set in the DEN_STA register in case of attempts to load the closed-caption data registers too frequently; these can be used to regulate the loading rate.

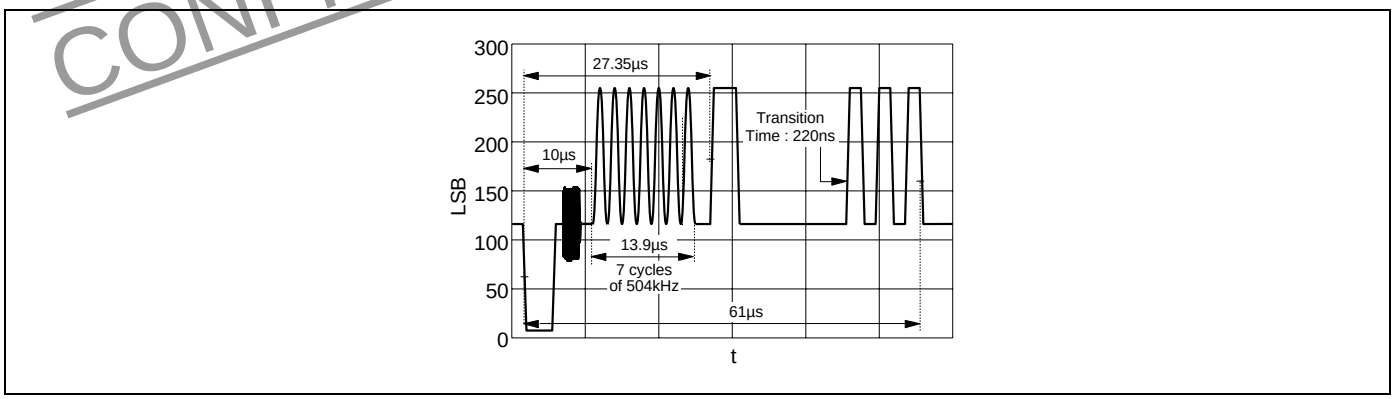


Figure 145 Example of closed-caption waveform

The closed-caption encoder considers that closed-caption data has been loaded and is valid on completion of the write operation into DEN_CCF1 for field1, or DEN_CCF2 for field 2. If closed-caption encoding has been enabled and no new data bytes have been written into the closed-caption data registers when the closed-caption window starts on the appropriate TV line, then the circuit outputs two US-ASCII NULL characters with odd parity after the start bit.

20.15 CGMS encoding

CGMS stands for *Copy Generation Management System*, and is also known as VBID and described by standard *CPX-1204* of *EIAJ*. CGMS data can be encoded by the digital encoder.

Three bytes, containing 20 significant bits, are delivered to the chip via the register interface. Two reference bits (1 then 0) are encoded first, followed by 20 bits of CGMS data. This includes a Cyclic Redundancy Check sequence, which is not computed by the device but is supplied to it as part of the 20 data bits. The reference bits are generated locally by the DENC. Figure 146 shows a typical CGMS waveform.

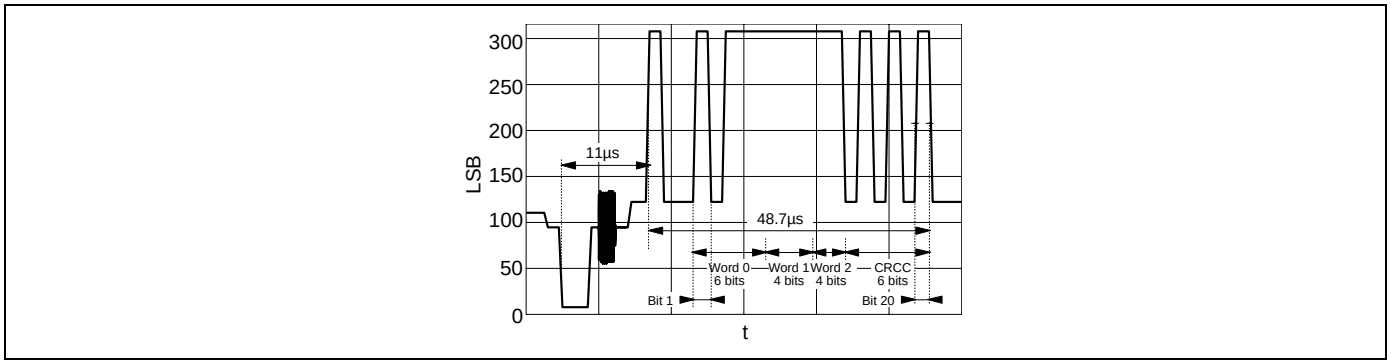


Figure 146 Example of CGMS waveform

CGMS encoding is enabled by setting bit encgms in register DEN_CFG3. When enabled, the CGMS waveform is present once in each field, on lines 20 and 283 (SMPTE-525 line numbering).

The CGMS data register is double-buffered, which means that it can be loaded at any time (even during line 20/283) without any risk of corrupting CGMS data that could be in the process of being encoded. The CGMS encoder considers that new CGMS data has been loaded and is valid on completion of the write operation into register **DEN_CGMS**.

20.16 WSS encoding

The digital encoder allows WSS (Wide Screen Signalling) in 625-line format, complying with the ETS 300 294 standard. Two bytes are delivered to the circuit through the register interface into two dedicated registers (see register DEN_WSS).

WSS encoding is enabled using bit enwss in register DEN_CFG3. When WSS encoding is enabled, a waveform is present on the first half of line 23 of each frame. Data is preceded by a run-in sequence and a start code generated locally by the DENC.

20.17 VPS encoding

VPS data encoding is defined by *ETS 300 231 communication, June 1993*. VPS data can be encoded by the DENC on line 16 (CCIR) for 625-line PAL and SECAM television systems. The VPS data is delivered to the circuit using registers DEN_VPS. The data transmission is preceded by a clock run-in and a start code generated by the DENC. The clock frequency is 5 MHz. This feature is enabled by setting the envps bit of register DEN_CFG7. Figure 147 shows an example of VPS waveform.

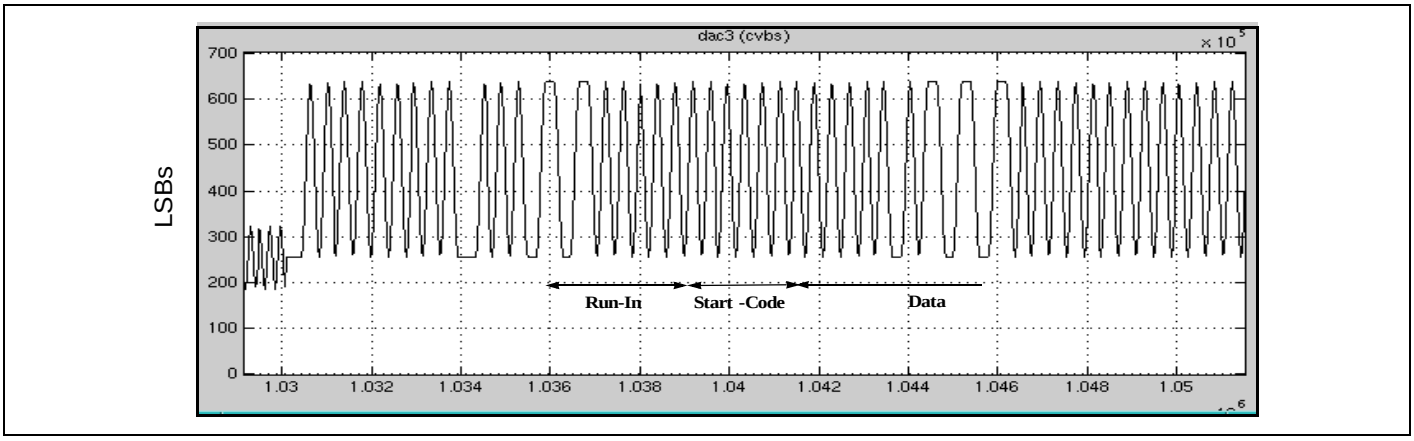


Figure 147 Example of VPS waveform

20.18 Teletext encoding

The DENC can encode Teletext according to the “CCIR/ITU-R Broadcast Teletext System B” specification (also known as World System Teletext), and NABTS (North American Basic Teletext Specification) EIA-516.

In DVB applications, Teletext data is embedded within DVB streams as MPEG data packets. The Transport Layer Processing IC (ST20) sorts incoming data packets and stores Teletext packets in a buffer. It then passes them to the DENC on request.

Signal exchange

The DENC and the Teletext buffer exchange 2 signals: TTXS (Teletext Synchronization) going from the DENC to the Teletext Buffer and TTXD (Teletext Data) going from the Teletext Buffer to the DENC.

The TTXS signal is a request signal generated on selected lines. In response to this signal, the Teletext buffer is expected to send Teletext bits to the DENC for insertion of a Teletext line into the analog video signal. The number of Teletext bits sent, depends on the Teletext system being used (selected by register bit DEN_REG_64.ttxt_abcd) 360 bits are sent for Teletext B - WST in PAL and SECAM, or 288 for Teletext C - NABTS in NTSC.

The duration of the TTXS window corresponds to the number of bits being sent (see Transmission Protocol below).

- For Teletext B and 625 line systems, the TTXS window duration is 1402 reference clock periods (corresponding to 360 bits).
- For Teletext C and 525 line systems (NABTS), this duration is 1121 master clock periods.

Following the TTXS rising edge, the encoder expects data from the Teletext buffer after a programmable number (2 to 9) of 27 MHz master clock periods. Data is transmitted synchronously with the master clock at an average rate of 6.9375 Mbit/s according to the protocol described below. In order of transmission, it consists of: 16 Clock Run-In bits, 8 Framing Code bits and one Teletext packet of 336 or 228 bits (depending on the Teletext system being used). If more than one packet of bits (336 or 228) are transmitted, they are ignored by the DENC. By default, register bit DEN_REG_65.ttx_mask_off masks the two bits of Teletext framing code, allowing the code to be set by the DENC according to the selected Teletext standard.

Transmission protocol

In order to transmit the Teletext data bits at an average rate of 6.9375 Mbit/s, which is about 1 / 3.89 times the master clock frequency, the following scheme is adopted:

The 360-bit packet is regarded as nine 37-bit sequences plus one 27-bit sequence. In every sequence, each Teletext data bit is transmitted as a succession of *four* identical samples at 27 Msample/s, except for the *10th*, *19th*, *28th* and *37th* bits of the sequence which are transmitted as a succession of *three* identical samples.

Programming “TTXS rising” to “first valid sample”

The encoder expects the Teletext buffer to clock-out the first Teletext data sample on the $(2+N)^{th}$ rising edge of the master clock following the rising edge of TTXS. Figure 148 depicts this graphically for N=0.

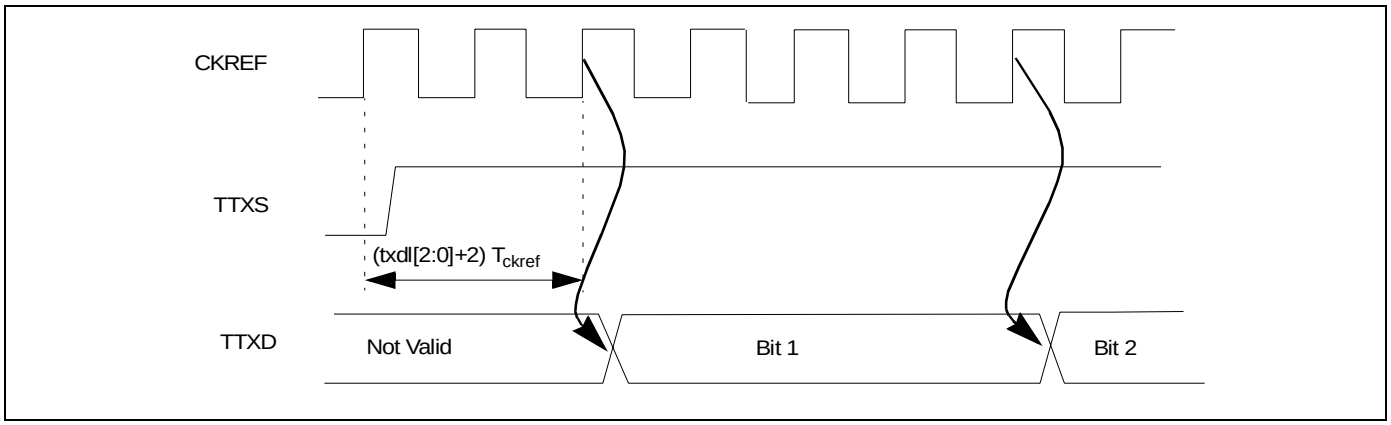


Figure 148 *TTXT Rising* to *First Valid Sample* delay for txdl[2:0] = 0

N is programmable from 0 to 7 by register bits DEN_TTX1.ttxdel[2:0]. The value written in txdl[2:0] is 2 less than the overall delay in CKREF cycles, so a value of 0 for txdl[2:0] corresponds to an overall delay of 2 cycles, and a value of 7 corresponds to a delay of 9 cycles.

Programming teletext line selection

Five dedicated registers, DEN_TTX1-5, program Teletext encoding in various lines in the Vertical Blanking Interval (VBI) of each field. In this way, each line in VBI can be selected independently.

Full-page teletext encoding is set by register bit DEN_TTX1.fp_ttxt. Teletext is encoded on lines 24 to 311 and 336 to 623 (ITU-R line numbering). This is in addition to the lines already programmed in the VBI. When full page teletext is performed, no video data is encoded (YCrCb, Y4 and CrCb input streams are ignored).

Teletext pulse shape

The shape and amplitude of a single Teletext pulse is shown in Figure 149 . Its relative power spectral density is shown in Figure 150 and Figure 151 It is zero at frequencies above 5 MHz, as required by the World System Teletext specification.

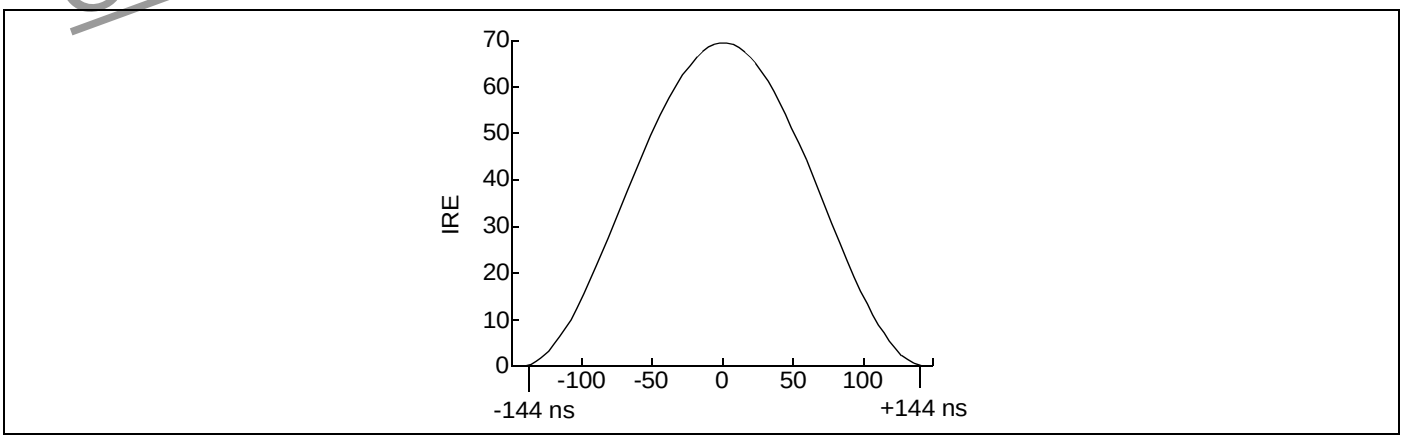


Figure 149 Shape and amplitude of a single teletext symbol

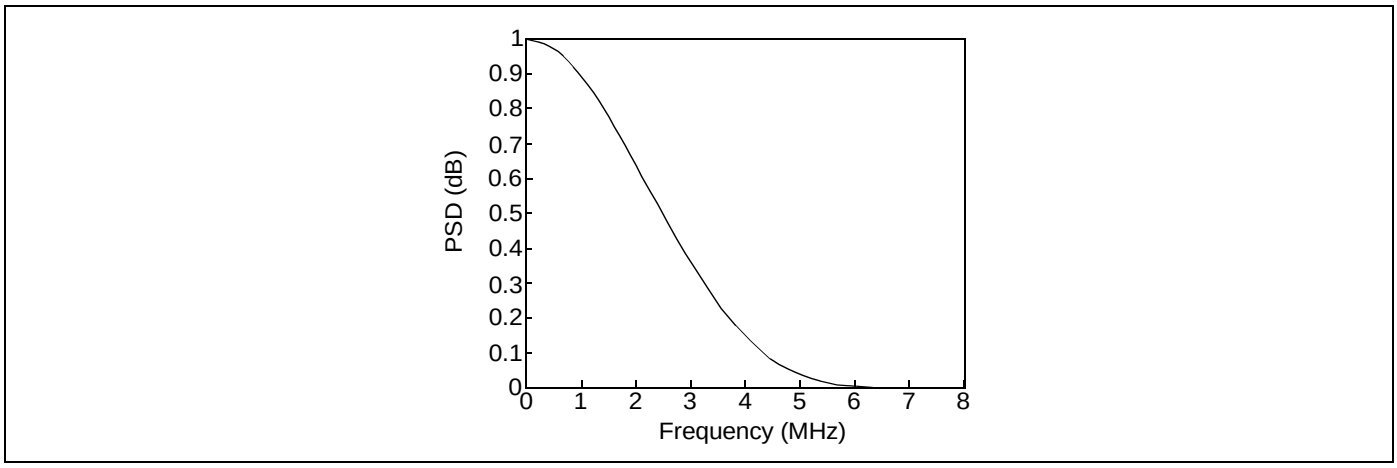


Figure 150 Linear PSD scale

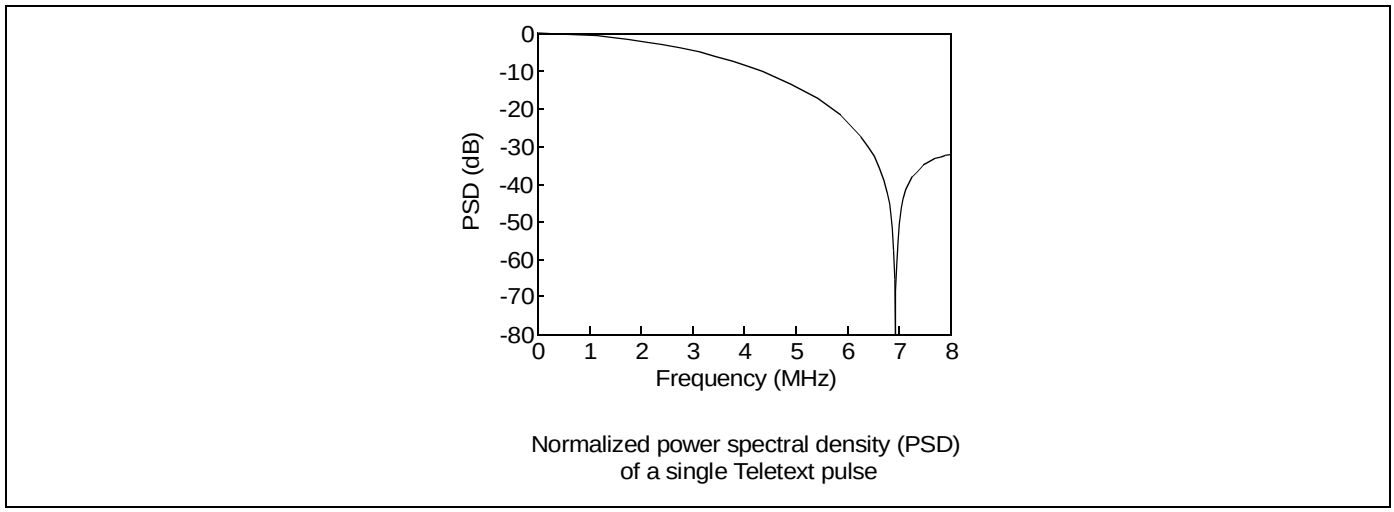


Figure 151 Logarithmic PSD scale

20.19 Line skip and line insert capability

This patented feature of the DENC offers the possibility to cut the cost of the application by suppressing the need for a VCXO.

Ideally, the master clock used on the application board and fed to the MPEG decoding IC would have exactly same frequency as the clock that was used when the MPEG data was encoded. Obviously this is not realistic; up to now a solution commonly used was to dynamically adjust the clock on the board as close to the “ideal” clock as possible with the help of time stamps embedded within the MPEG stream. Such a kind of tracking often involves the use of a VCXO: when the MPEG data buffer fills up to more than some threshold the clock frequency is increased, when it empties down to some other threshold the clock frequency is lowered.

The DENC offers an alternative, cost-saving solution: by programming two bits in register DEN_CFG6, the DENC is able to reduce or increase the length of some frames in a way that will not introduce visible artifacts (even if comb-filtering is used). These bits should be set according to the level of the MPEG data buffer.

Operation with the DENC as sync master is as follows:

- If the MPEG data buffers fills up too much, set bit jump to 1 and bit dec_ninc to 1. The DENC will reduce the length of the current frame. Bit jump will then automatically reset to 0.
- If the MPEG data buffers empties too much, set bit jump to 1 and bit dec_ninc to 0. The DENC will increase the length of the current frame. Bit jump will then automatically reset to 0.

These operations can be repeated until the MPEG data buffer is inside its fixed limits.

Line skip and line insert can be used in slave mode; the sync signals supplied to the DENC must be in accordance with the programmed frame lengths.

20.20 CVBS, S-VHS, RGB and UV outputs

Six out of eight video signals can be directed to six analog output pins through a 10-bit D/A converters operating at the reference clock frequency. The available combinations are:

S-VHS (Y/C) + CVBS + RGB, or S-VHS (Y/C) + CVBS + U + Y2 + V, or Y1 + C1 + CVBS1 + C2 + Y2 + CVBS2.

These combinations are controlled by bits conf_out1 and conf_out0 in register DEN_CFG8, as shown in the table below;

conf_out1	conf_out0	dac1	dac2	dac3	dac4	dac5	dac6	Notes
0	0	Y	C	CVBS	C	Y	CVBS	
0	1	Y	C	CVBS	V	Y	U	
1	x	Y	C	CVBS	R	G	B	Default

Table 99 Encoding of conf_out

The C to Y peak to peak amplitude ratio can be modified in both CVBS and VHS (Y/C) outputs (see mult_rgb_c).

Default peak to peak amplitude of UV and RGB outputs is set to 70% of Y or CVBS peak to peak amplitude, for 100/0/100/0 color bar pattern, and can be modified using the multiplying factors in registers DEN_DAC45 and DEN_DAC6C.

If DEN_CFG7[2] bit uv_lev is 0 (default value) U and V outputs have 0.7V peak to peak amplitude if 100/0/100/0 color bar pattern is inputted. If this bit is 1 U and V outputs are those defined by ITU-R 624-4 for PAL and NTSC standards (Vpp/Upp = 1.4). In that case U peak to peak amplitude is 0.61V (0.57V if DEN_CFG7 bit setupYUV is set) and V peak to peak amplitude is 0.86 V (0.80 V if register DEN_CFG7 bit setupYUV is set). In all these cases UV outputs can be multiplied by 0.75 to 1.22 factor according to register bits DEN_DAC45.dac4_mult and DEN_DAC6C.dac6_mult.

A single external analog power supply pair is used for all DACs, but two independent pairs of current and voltage references are needed. A resistor must be connected between each I_REF and V_REF pins.

The internal current sources are independent from the positive supply, and the consumption of the DACs is constant whatever the codes converted.

Any unused DAC may be independently disabled by software, in which case its output is at neutral level (blanking for luma and composite outputs, no color for chroma output, black for RGB and UV outputs). For applications where a single CVBS output is required, the RGB/CVBS+S-VHS/UV Triple DAC should be disabled, pin I_REF_DAC_RGB should be tied to analog power supply and pin V_REF_DAC_RGB should be left unconnected.

Due to the 2.5V power supply used, the output swing of the DACs is about 1Vp-p. Therefore some external gain may be required, which, combined with the recommended output filtering stage, means active filtering. For this active filtering stage to be very simple, it is possible to “invert” the DAC outputs by programming a bit of DEN_CFG5. Code N becomes code 1024-N, i.e. the resulting waveform undergoes a symmetry around the mid-swing code.

21 Teletext DMA

21.1 Introduction

Teletext data is retrieved from memory, serialized and transferred to the DENC by a dedicated teletext DMA. The DENC encodes Teletext data according to the “CCIR/ITU-R Broadcast Teletext System B” specification (also known as “World System Teletext”).

21.2 Teletext packet format

One teletext packet (otherwise called a teletext line) is a stream of 360 bits, transferred at an average frequency of 6.9375 MHz. The data format is the same as the contents of the PES data packet as defined in the ETSI specification. The DMA reads-in multiples of 46 bytes and transfers lines of 45 bytes to the DENC.

Each teletext packet is composed of the clock run-in and the data-field, as illustrated in Figure 152.

- The clock run-in is composed of two bytes, each with the hexadecimal value #AA (binary value ‘10101010’).
- The data-field consists of three fields: framing code, magazine and packet address, and data block fields. These three fields provide the block of teletext data.

The framing code is a single byte of hexadecimal value #E4[1]. The data is transmitted in order, from the LSB to the MSB of each byte in memory.

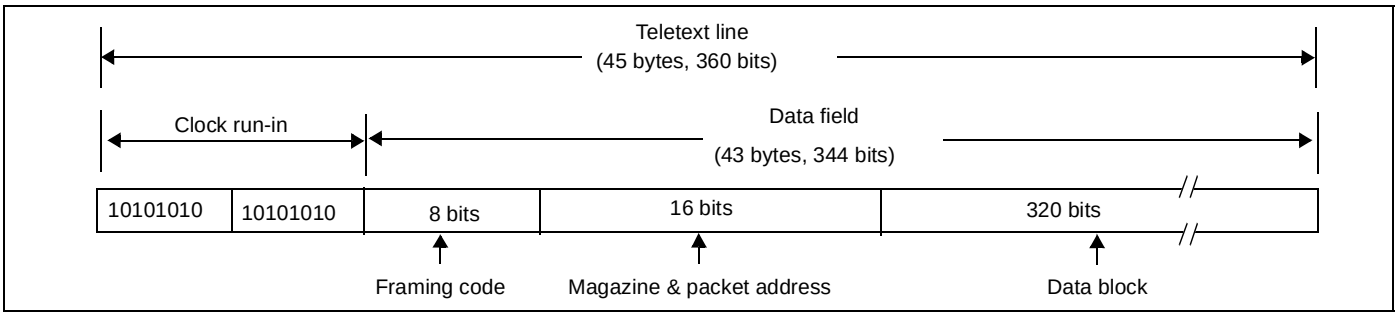


Figure 152 Teletext packet format

21.3 Data transfer sequence

The DENC issues a teletext request signal to the teletext DMA, this is shown by the rising edge of signal *TtxtRequest* in Figure 153. After a delay, programmable from 2 to 9 master-clock periods, the teletext DMA transmits the first valid teletext data bit of the teletext packet.

1. Specification for conveying ITU-R Systems B Teletext in Digital Video Broadcasting (DVB) bit-streams.

The 360 bits of output data are defined as nine 37-bit sequences, ending with one 27-bit sequence. Within each sequence, each bit is transmitted in four 27 MHz cycles, except bits 10, 19, 28 and 37, which are transmitted in three 27 MHz cycles. This is illustrated in the figure below for bits 0 to 10.

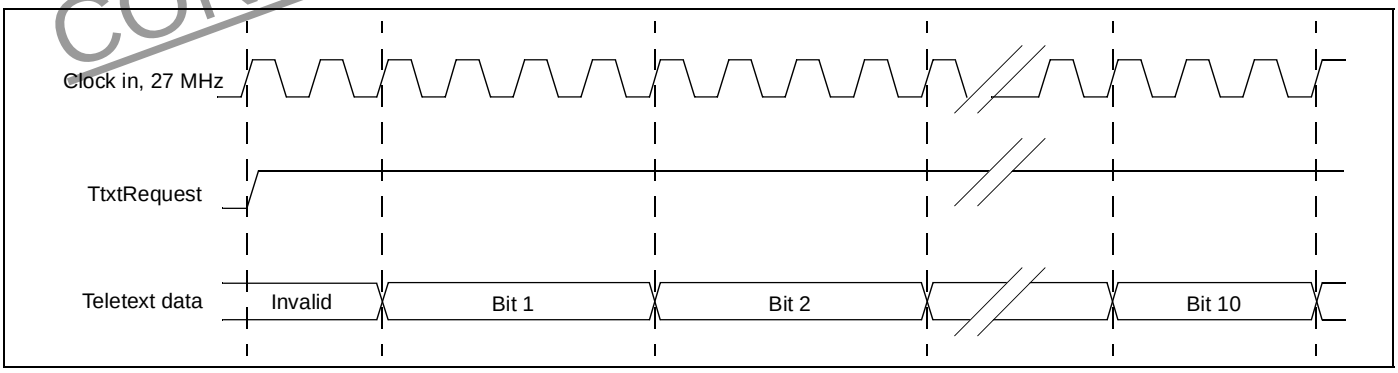


Figure 153 Teletext data transfer sequence

The duration of the TTXS window is 1402 reference clock periods (51.926 µs), which corresponds to the duration of 360 Teletext bits (see Transmission Protocol below).

The delay between signal *TtxtRequest* becoming high and the transfer of the first bit of the teletext packet is between 2 and 9, 27 MHz clock cycles. This delay is programmed by register bits DEN_TTX1.ttxdel[2:0]. The value written to this register is increased by two 27 MHz clock cycles, so the value 0 corresponds to an overall delay of 2x27 MHz clock cycles, and the value 7 corresponds to a delay of 9x27 MHz clock cycles.

21.4 Interrupt control

Teletext interrupts can be programmed by the Ttxt_IntEnable register to interrupt the CPU whenever one of the following occurs:

- A teletext data transfer is complete;
- the current video frame toggles odd-to-even or even-to-odd.

The interrupt status is given by the Ttxt_IntStatus register and masked by the Ttxt_IntEnable register. The interrupt bits are reset when the CPU writes to the acknowledge register, or when a DMA operation is completed.

21.5 Teletext registers

Five dedicated DENC registers program the teletext encoding in various areas of the Vertical Blanking Interval (VBI) of each field. Four of these areas (i.e. blocks of contiguous Teletext lines) can *independently* be defined within the two VBIs of one frame (e.g. 2 blocks in each VBI, or 3 blocks in field1 VBI and one in field2 VBI, etc.). In certain circumstances it is possible to define up to 4 areas in each VBI.

Full-page teletext encoding is enabled by register bit DEN_TTX1.fp_ttxt. In this case, teletext is encoded on lines 7 to 311 and 320 to 623 (ITU-R line numbering). If bit fp_ttxt_all is set, teletext encoding is also enabled on lines 6, 318 and 319. When full page teletext is performed, no video data is encoded (YCrCb, Y4 and CrCb input streams are ignored).

22 Double triple video DAC

22.1 Description

There are two on-chip 3x10-bit digital-to-analog converters (triple DACs) used for video output. One provides output signals in CVBS, Y, C, and the other in RGB. The figure below shows the DAC schematic.

An external reference resistor is associated with the bandgap voltage to generate a reference current. This resistor is connected between the V_REF pin of the bandgap and a dedicated I_REF pin to achieve a higher noise immunity.

The global segmented architecture is presented in the figure below. Current-sources provide an output range of 1.45V maximum with good linearity. Sampled data are available on video outputs after 1 clock period (on the next rising clock edge).

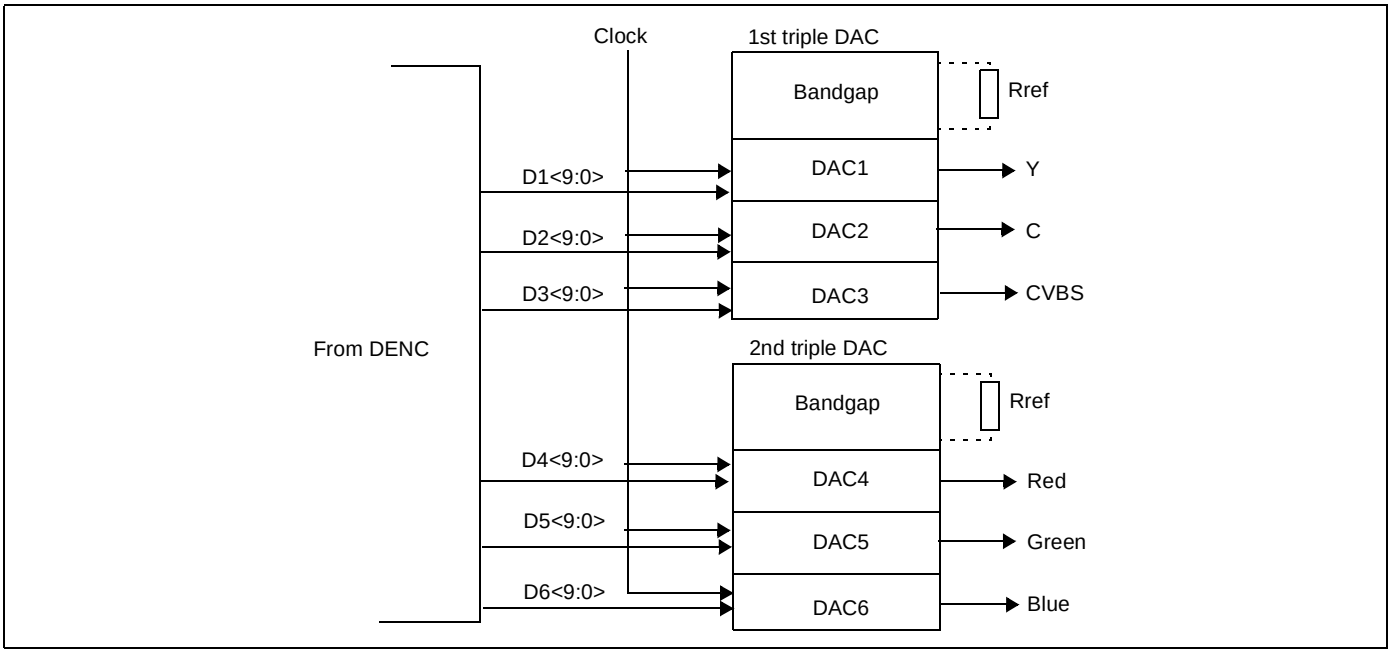


Figure 154 Double triple video DAC schematic

The triple video DAC has its own 2.5V power and ground supplies for noise reduction. To guarantee good frequency response at high frequencies, these power and ground supplies are not connected to digital power supplies inside the chip.

22.2 Input codes for video application

The table below lists the reference input codes generated by the DENC depending on the configuration for the DACS and the standard.

	Y-PAL/SECAM	Y-NTSC	RGB
WHITE(235)	816.00	802.00	602.00
BLACK(16)	256.00	240.00	41.00
SYNC TIP	16.00	16.00	n.a.

Table 100 Reference input codes

Note that CVBS = Y+C, so chrominance component has no effect on CVBS signal (C is null)

22.3 Video output voltage level

The resistor Rref connected to the bandgap has a direct effect on the output current which flows from the DAC's outputs. For the maximum code (1023 in decimal):

Iout (max) = 80.704 / Rref

For example, with a typical value of Rref = 20kohms, Iout (max) = 4.04mA for each DAC.

The value of Rref must be carefully chosen: Iout should always be lower than 5mA, otherwise, DAC linearity is not guaranteed.

Because of the sensitive relationship between the DAC and Rref, the tolerance on the Rref value must be small. Typically, the Rref resistor must be a 1% resistor.

The output voltage on the RGB output pins depends on the external load resistor Rload (connected between the DAC output and ground):

*Vout (max) = Rload * Iout (max)*

For example, with a typical load value Rload = 274ohms and Iout(max) = 4.04mA, Vout (max) = 1.11V.

For any given digital input code, the output voltage of the DAC will be given by the following formula:

*Vout = Din / 1023 * Vout (max) = (Din * Rload * 80.704) / (Rref * 1023)*

*Vout = Din * Rload * 0.079 / Rref*

For example, with Rload = 274ohms, Rref = 20kohms and Din = 526, Vout = 0.57V

22.4 Video specifications and DAC setup

In Y-PAL/SECAM, the output video range between code 16 (synchronization level) and code 816 (white level) should be: Vout (816) - Vout (16) = 1V. The output video range between code 256 (black level) and code 816 (white level) should be 700mV. This must be respected for all applications.

The value of the Rref resistor must be chosen according to the value of Rload and the previous formula, to achieve the standard output video range. The minimum value for Rref is 18kohms.

According to video specifications ITU-R BT 601, the nominal sampling frequency is 27 MHz. The clock for the DACs is the same as the one for the DENC but buffered. This clock is usually generated externally by a VCXO. It must be as clean as possible to achieve a good signal-to-noise ratio.

22.5 Output-stage adaptation and amplification

A schematic of the output stage is shown below. The purpose of the output stage depends on the application and the required price-to-performance ratio. The output stage is connected directly to a scart connector or to other components (in which case the level and output impedance of the output signal may be different).

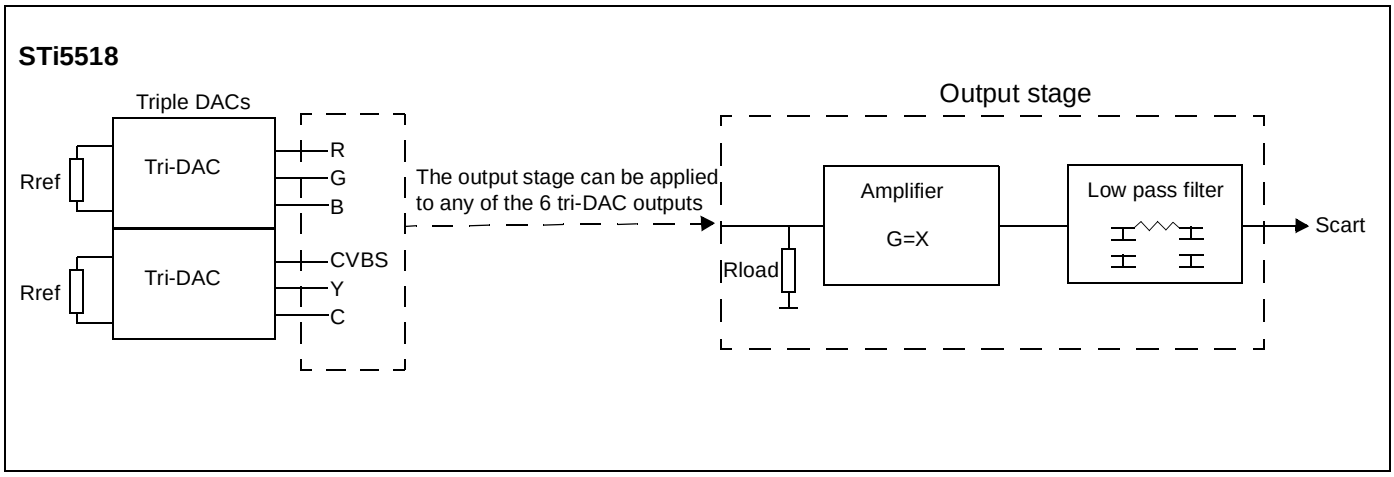


Figure 155 Output stage schematic

The amplifier gain must be in accordance with the one of the tri-DACs (defined by Rload and Rref). If the amplifier gain cannot be set to the standard output video range by Rref and Rload, it can be tuned. In common applications (with Rref=20Kohms and Rload=274ohms), a video amplifier should be adequate. The ideal input impedance of the output stage should be greater than Rload.

The tri-dacs have no cut-off frequency, therefore, a low-pass filter (around 10 MHz) must be applied to remove harmonics (of mainly 27 MHz). If additional attenuation is applied by the filter due to imperfection of the amplifier (generally degrading the C/L ratio), correction must be applied to preserve a good performance. Also, to guarantee a good frequency behavior at high frequency, the analog power supply must be separate from the digital power supply. If this is not the case, an additional correction may be required.

23 Audio decoder

23.1 Features

Input formats

The audio decoder accepts: Dolby Digital, MPEG-1 layers I and II, MPEG-2 layer II 6-channel, PCM, CDDA data formats; MPEG2 PES streams for MPEG-2, MPEG-1, Dolby Digital, MP3, and Linear PCM (LPCM).

SPDIF input data (IEC-60958 or IEC-61937 standards) is accepted if an external circuitry extracts the PCM clock from the stream.

Audio/video synchronization

Skip frame, repeat blocks and soft mute frame features can be used to synchronize audio and video data. PTS audio extraction is also supported.

Output formats

The device outputs up to 6 channels of PCM data and appropriate clocks for external digital-to-analog converters.

- 6 PCM data on three outputs: left/right, centre/subwoofer and left surround/right surround: DAC_PCMOUT2-0;
- three clocks for the external DACs: DAC_PCMCLK, DAC_SCLK and DAC_LRCLK;

Programmable downmix enables 1, 2, 3 or 4 channel outputs. Data can be output in either I²S format or Sony format. The decoder can format output data according to IEC-60958 standard (for non compressed data: L/R channels, 16, 18, 20 and 24-bits) or IEC-61937 standard (for compressed data), for F_S = 96 kHz, 48 kHz, 44.1 kHz or 32 kHz.

Sampling frequencies

Sampling frequencies (set by register AUD_SFREQ) of 96 kHz, 48 kHz, 44.1 kHz, 32 kHz and half sampling frequencies are supported. A down-sampling filter (96 kHz/48 kHz) is available.

Special modes

The decoder supports dual mode for MPEG and Dolby Digital. It includes a Dolby surround compatible downmix and a ProLogic decoder.

A pink noise generator enables the accurate positioning of speakers for optimal surround sound setup.

PCM beep tone is a special mode used for set-top box. It generates a triangular signal, of variable frequency and amplitude, on the left and right channels.

In global mute mode, the decoder decodes the incoming bitstream normally but the PCM and SPDIF outputs are softmuted. This mode is used to prepare a period of decoding mode, to synchronize audio and video data without hearing the audio.

Virtual Surround

The 24-bit audio DSP cell supports TruSurround, SRS Labs' virtual technology for two-speaker playback of multi-channel audio. This reduces 6 channels of either Dolby Digital (AC-3) or MPEG Multichannel audio to two channels only, providing to the user virtual multichannel sound effect. Information on how to use this feature is sent to SRS Labs Licensees only.

Trick modes

Slow-forward and fast-forward trick modes are available for compressed and non-compressed data.

Control interface

The control interface of the decoder is activated via memory mapped registers in the ST20 address space.

23.2 Architecture overview

Data flow

The audio decoder has a programmable core, which is optimized for audio decoding algorithms. Dedicated hardware performs bitstream depacking and IEC data formatting.

The figure below illustrates the audio decoder data flow. The compressed bitstream is transferred from the audio bit buffer (which is mapped into external (SDRAM) memory) to the audio decoder, via the MPEG DMA, which filters a 64-byte FIFO. When the FIFO is filled, data are transmitted from the FIFO to the audio decoder.

- The input processor (composed of a packet parser and an audio parser) unpacks the bitstream (packet parser) and verifies the syntax of the incoming stream (audio parser).
- The compressed audio frames with their associated information (PTS) are stored into the circular frame buffer.
- While a second frame is stored in the circular frame buffer, the first frame is extracted by the audio core decoder and decoded into audio samples.
- The PCM-unit converts the samples to PCM format, and controls the channel delay buffer so that each channel can be delayed independently.
- Simultaneously, the IEC unit transmits non-compressed data or compressed data.
  - In compressed mode, data is extracted directly from the circular buffer and formatted according to the IEC-61937 standard.
  - In non-compressed mode, the left and right PCM channels formatted by the PCM unit are output by the IEC unit, according to the IEC-60958 standard.

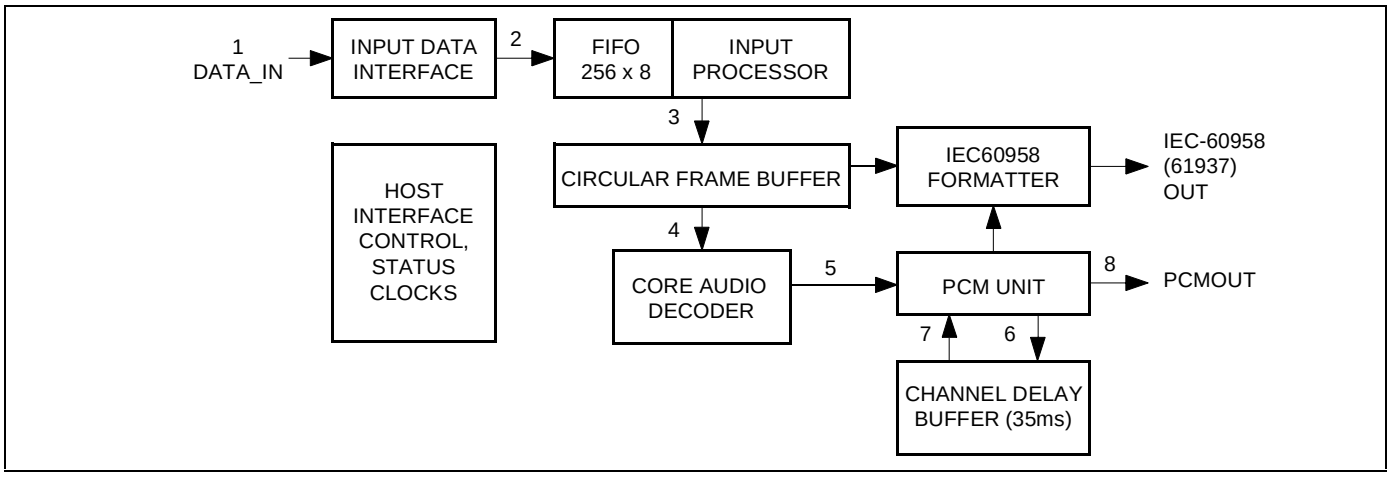


Figure 156 Architecture and data flow

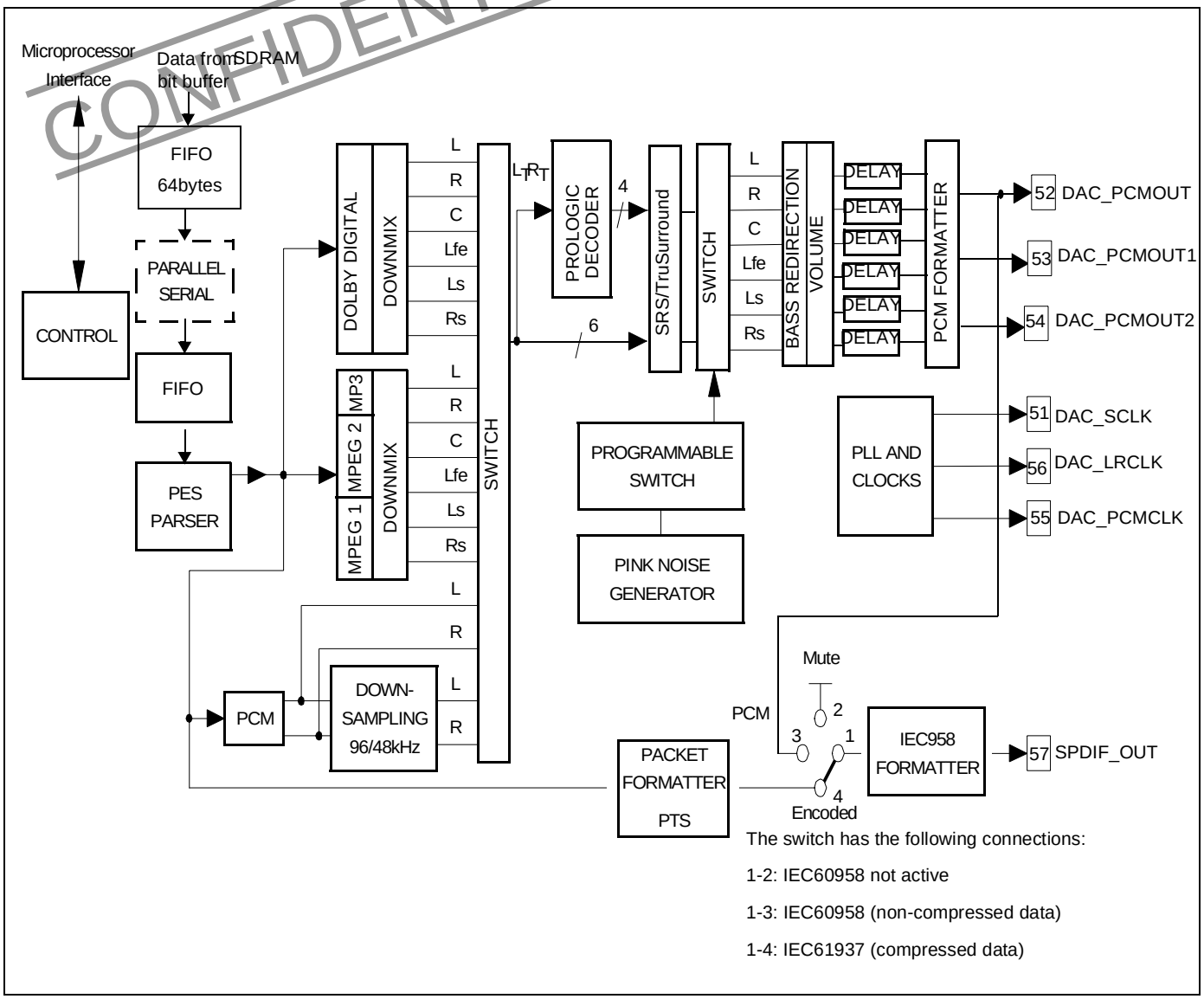


Figure 157 Audio decoder block-diagram

23.3 Decoding process

The decoding is performed in the following stages. Each stage can be activated or bypassed by the configuration registers:

Parsing	Bitstream parsing (performed by the input processor) discards all of the non audio information so that only the audio elementary stream (Dolby Digital, MPEG1/2, LPCM, PCM, DTS, MP3) is transmitted to the next stage (the circular frame buffer).The parsing stage operates in two phases: the packet parser unpacks the stream, the audio parser checks the syntax of the bitstream.
Main decoding	An elementary stream is input and decoded samples are output from this stage. The number of output channels is defined by the AUD_DOWNMIX register (1 channel up to 6channels). Dolby Digital, MPEG1 layers I and II, MPEG2 layer II, LPCM, CDDA, MP3 decoding formats are supported. The appropriate stream format must be set by registers AUD_STREAMSEL and AUD_DECODSEL before running the decoder.
Post decoding	Post decoding includes specific PCM processing: DC filter, de-emphasis filter, downsampling filter. These filters can be independently enabled or disabled by the AUD_DWSMODE register. Post decoding also provides a ProLogic decoder, described in ProLogic decoding modes on page 226. The decoder output can also be processed according to SRS Labs TruSurround algorithm.
Bass redirection	This stage redirects low-frequency signals to the subwoofer. The subwoofer is extracted from the channels L, R, C, Ls, Rs, LFe. There are six configurations for subwoofer channel extraction, these are set by the AUD_OCFG register. This is discussed in Output configurations on page 229.
Volume control	The volume is controlled in steps of 1dB, independently for each channel by the AUD_CHAN_IDx, AUD_VOLUME0 and AUD_VOLUME1 registers.

Table 101 Audio decoding stages

23.4 Operation

Reset

The audio decoder can be reset by software with the following two commands:

- SOFTRESET: to reset the audio decoder, ’1’ must be written in the register AUD_SOFTRESET (0x10). The interrupt related registers (AUD_INTE, AUD_INT, AUD_ERROR) and command registers (AUD_SOFTRESET, AUD_RUN, AUD_PLAY, AUD_MUTE, AUD_SKIP_MUTE_CMD, AUD_SKIP_MUTE_VALUE) are reset to zero. The volume registers are reset to 0, no other decoding configurations are changed. The DSP returns to idle mode.
- REBOOT: a 1 must be written to bit REB of the AUD_SKIP_MUTE_CMD register to reboot the audio decoder. Registers AUD_RUN, AUD_PLAY, AUD_MUTE and AUD_SKIP_MUTE_CMD are reset to 0. The DSP returns to idle mode. The decoding configurations are unchanged but 2 frames have to be sent to the decoder in order to perform the reboot.

Clocks

The following clocks are used by the audio decoder:

- The PCM clock (DAC_PCMCLK signal) used by the external DACs to convert DAC_PCMOUT0, 1, 2. It is usually generated by an embedded PLL from the 27 MHz clock input. If necessary, it can be also be generated by an external PLL. The internal frequency synthesizer can generate 256*fs or 384*fs where fs= 12, 16, 22.05, 24, 32, 44.1, 48, 96 or 192 kHz.
- Audio system PLL The system PLL creates the audio system clock from the 27 MHz input clock

- Bit clock DAC_SCLK

  The PCM serial clock is the bit clock. It provides clocks for each time slot (16 cycles for each channel in 16-bit mode, 32 cycles for each channel in 18-, 20-, 24-bit modes). The frequency of DAC_SCLK is, therefore, fixed to 2 x Nb time slots x Fs, where Fs is the sample frequency.

  The clock is derived from DAC_PCMCLK. The register AUD_PCMDIVIDER must be configured according to the selected output precision and the frequency of DAC_PCMCLK, so that the device can construct DAC_SCLK:Fsclk = Fpcmclk / (2 x (AUD_PCMDIVIDER+1)) giving: AUD_PCMDIVIDER = (Fpcmclk / (2xFsclk)) -1.

- Word Clock DAC_LRCLK

  The frequency of DAC_LRCLK is given by:

  - Flrclk = Fsclk/32; for 16 bit PCM output,
  - Flrclk = Fsclk/64; for 18, 20 or 24 bits PCM output.

  No special configuration is required. The polarity can be changed by the AUD_PCMCONF register bit INV (see PCM output on page 229).

23.5 Decoding states

There are two decoder states: idle state and decode state (see the figure below). The register AUD_RUN changes the state.

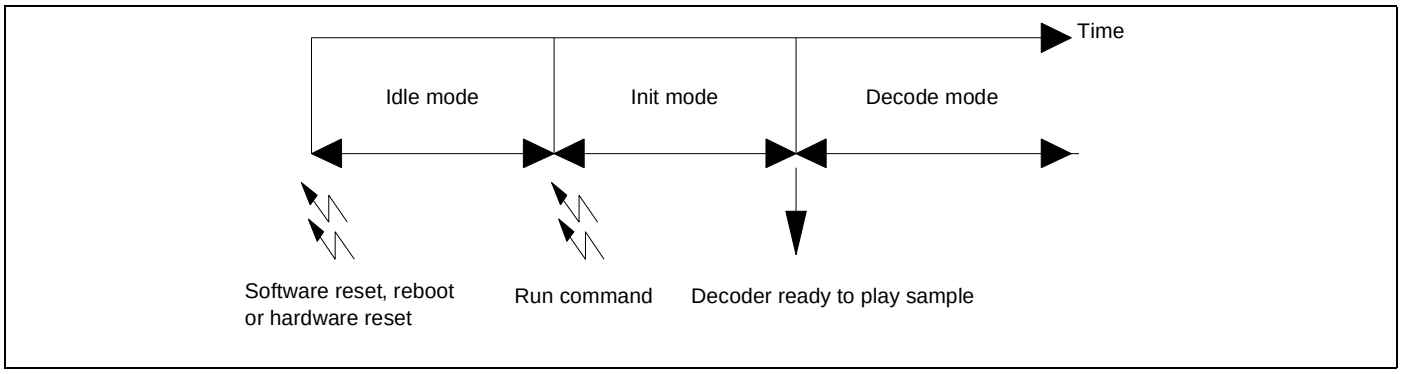


Figure 158 Decoding states

Idle mode

Idle mode is entered after a hardware or software reset. In this mode, the embedded DSP does not decode, i.e. no data are processed, the chip is waiting for the RUN command. During this mode all configuration registers must be initialized. In idle mode, even if the chip is not processing data, the DACs clocks can be output, enabling the set-up of the external DACs. Once the DAC_PCMCLK, DAC_SCLK and DAC_LRCLK clocks are configured, they can be output by setting the AUD_MUTE register.

Play	Mute	Clock (DAC_SCLK, DAC_ LRCLK) state	PCM output
X	0	Not running if PLAY is set to 0 after reset.	0
X	1	Running	0

Table 102 Idle mode, play and mute command effects

Note The PLAY command has no effect in this state, as the decoder is not running. It can, however, be sent and it will be taken into account as soon as the decoder enters the decode state.

Decode mode

This state is entered after the RUN command has been sent (i.e. AUD_RUN register = 1). In this mode, data is processed; the decoder can play sound, or mute the outputs by using the AUD_PLAY and AUD_MUTE registers:

To decode and output streams, the AUD_PLAY register must be set. If the AUD_MUTE register is reset, the sound is sent to outputs; if the AUD_MUTE register is set, the outputs are muted.

Reg. AUD_PLAY	Reg. AUD_MUTE	Clock state	PCM output	Decoding
0	0	Not running	0	No
0	1	Running	0	No
1	0	Running	Decoded Samples	Yes
1	1	Running	0	Yes

Table 103 Decode mode. play and mute commands effects

Note *It is not possible to change configuration registers in this state, the chip must be soft reset beforehand. Only the following registers can be changed "on-the-fly":* AUD_CHAN_IDX, AUD_VOLUME0, AUD_VOLUME1, AUD_*OCFG,* AUD_*DOWNMIX registers.*

23.6 Stream parsers

The synchronization status of both parsers is provided in the register AUD_SYNC_STATUS. Each time the synchronization status of one of the two parsers changes, the interrupt SYN is generated (if enabled) and the status can be read in AUD_SYNC_STATUS.

Packet parser

The packet parser unpacks stream, sorts packets and transmits data to the audio parser. Before unpacking packets and transmitting data, the packet parser must detect the packet-start by recognizing the packet synchronization word.

The parser can be set to search for two packet synchronization words before starting to unpack and transmit, by setting the register AUD_PACKET_LOCK to 1. Otherwise, the packet parser will start handling the stream once it has detected information matching the packet synchronization word.

The packet parser is also able to perform selective decoding, it can decode audio packets that match a specified ID. This ID is specified in AUD_ID and AUD_ID_EXT registers, the function is enabled by setting the AUD_ID_EN register.

Audio parser

The audio parser verifies the stream syntax, extracts non audio data and sends audio data to the frame buffer. The audio parser must detect the audio synchronization word corresponding to the type of stream to be decoded.

The audio parser can be set to detect more than one synchronization word before parsing, by setting the AUD_SYNC_LOCK register to a value between 1 and 3. This number represents the number of supplementary sync words to detect before considering to be synchronized.

23.7 Decoding modes

Dolby Digital decoding modes

The decoder must be programmed to specify the stream format as Dolby Digital encoded (in register AUD_DECODSEL=0).

The following modes refer to different implementations of the dialog normalization and dynamic range control features. The mode is selected by programming the register AUD_AC3_COMP_MOD.

- Line Mode: In Line Mode (AUD_AC3_COMP_MOD = 2), the dialog normalization is always enabled. It is done by the decoder itself and the dialog is reproduced at a constant level.

  The dynamic range control variable encoded in the bitstream is used and can be scaled by the two scaling registers AUD_AC3_HDR (for high-level cut compression) and AUD_AC3_LDR (for low-level boost compression). For 2/0 downmix, the high-level cut compression is not scalable.

- RF Mode: In RF Mode (AUD_AC3_COMP_MOD=3), the dialog normalization is always performed by the decoder. The dialog is reproduced at a constant level.

  The dynamic range control and heavy compression variables encoded in the bitstream are used, but compression scaling is not allowed. This means that the AUD_AC3_HDR and AUD_AC3_LDR registers can not be used in this mode. An eleven dB gain shift is applied on the output channels.

- Custom 0 Mode: In Custom 0 mode (AUD_AC3_COMP_MOD=0), the dialog normalization is not performed by the decoder and must be done by another circuit, externally.

  The dynamic range control variable encoded in the bitstream is used and can be scaled by the two scaling registers AUD_AC3_HDR (for high-level cut compression) and AUD_AC3_LDR (for low-level boost compression).

- Custom 1 Mode: In Custom1 mode (AUD_AC3_COMP_MOD=1), the dialog normalization is performed by the decoder. The dynamic range control variable encoded in the bitstream is used and can be scaled by the two scaling registers AUD_AC3_HDR (for high-level cut compression) and AUD_AC3_LDR (for low-level boost compression).

MPEG decoding modes

MPEG-1 layer1 and layer2 encoded data are decoded, as well as MPEG-2 layer2 data with or without extension (i.e. 6-channel streams). The MPEG input format must be specified in the AUD_DECODSEL register: where AUD_DECODSEL=1 for MPEG1 and AUD_DECODSEL=2 for MPEG2. The dataflow is show in the figure below.

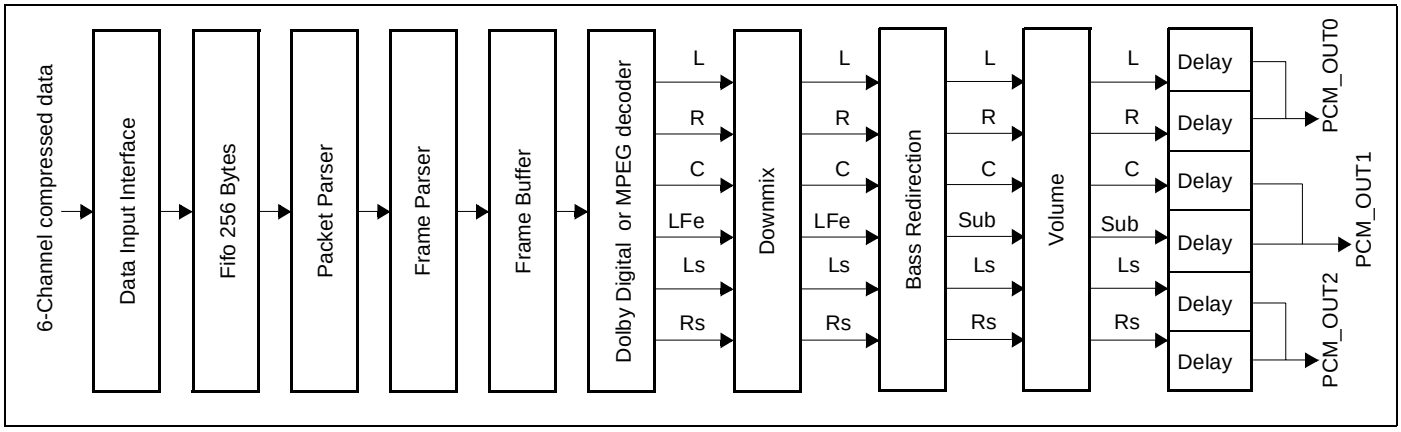


Figure 159 6-channel compressed data decoding flow

Dual-mode decoding modes

In dual-mode, two completely independent mono program channels (e.g. bilingual) are encoded in the bitstream, referred to as channel 1 and channel 2. The left/right output is set to the following options by the AUD_MP_DUALMODE register in MPEG format, and by the AUD_AC3_DUALMODE register in Dolby Digital format:

- output channel 1 on both L/R outputs;

- output channel 2 on both L/R outputs;
- mix channels 1 and 2 to monophonic and output on both L/R;
- output channel 1 on Left output, and channel 2 on right output.

PCM/LPCM decoding modes

The decoder supports PCM and LPCM multi-channel streams, set by the register AUD_DECODSEL=3.

When decoding PCM/LPCM streams encoded at 96 kHz, register AUD_DWSMODE configures the filter that downsamples the stream from 96 kHz to 48 kHz.

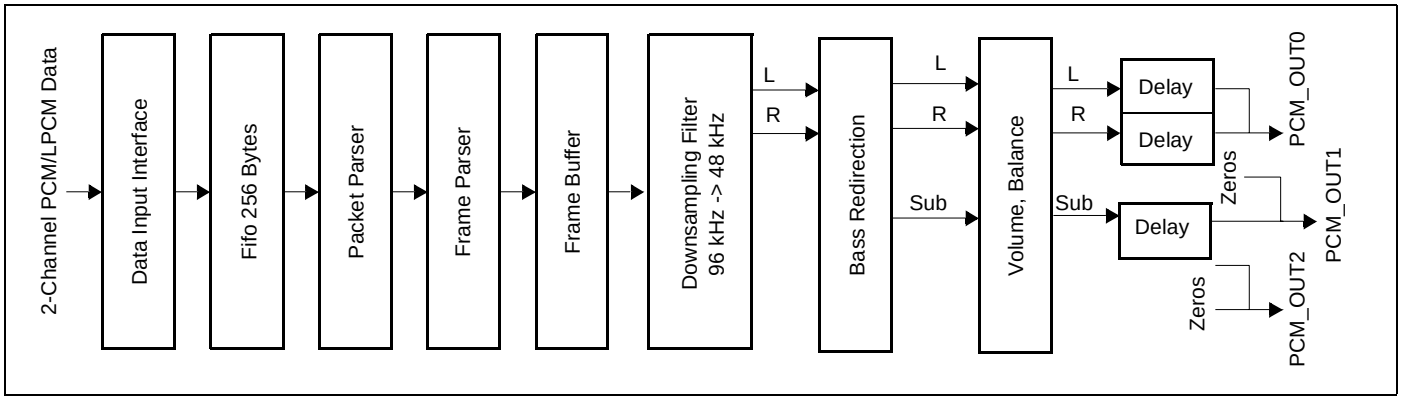


Figure 160 PCM/LPCM decoding flow

Note The device decodes the 6 channels of the DVD-LPCM stream, however, no downmix is possible.

Note For an 8-channel DVD-LPCM input file, only the 6 first channels are handled by the chip. The information contained in the 2 last channels is lost.

ProLogic decoding modes

ProLogic Compatible Downmix: A multichannel bitstream can be decoded and downmixed to provide a 2-channel ProLogic compatible output (Lt, Rt). This downmix is selected by the register AUD_DOWNMIX. The 2 channels can be used as the input of a ProLogic decoder and player (e.g. home theatre).

ProLogic Decoding: A 2-channel ProLogic bitstream can be decoded. The 2 channels could come from a Dolby Digital 2-channel bitstream, a LPCM or an MPEG1 bitstream. The 2-channel bitstream can be converted into a 4-channel output (L, R, C, S). The surround (S) is simultaneously sent on Ls and Rs channels. A ProLogic downmix enables to configure which channels to output on PCM data. This is done through the register AUD_PL_DWN.

An auto-balance feature is available and activated through AUD_PL_AB register. The delay on surround channel is configurable with the AUD_LSDLY register (while resetting the AUD_RSDLY register).

The bass redirection is performed after the ProLogic decode. The same bass redirection configuration than those available in non-ProLogic modes can be used except that the surround channels will not be added to the bass

redirection. In the case of Dolby Digital or MPEG, the Dolby Digital or MPEG stream can be decoded before the ProLogic decode.

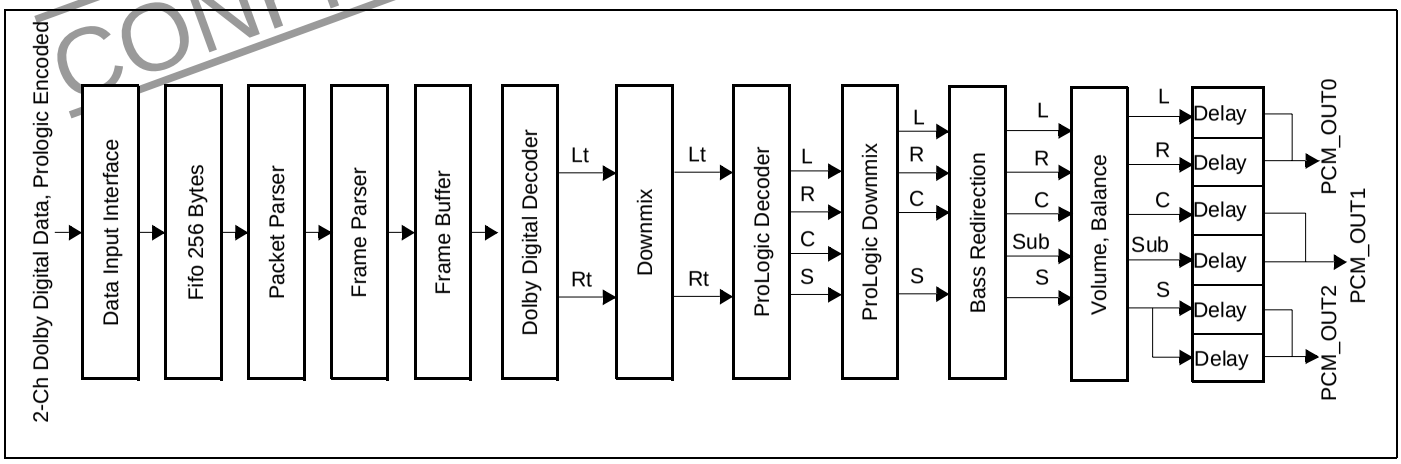


Figure 161 Dolby Digital & ProLogic decoding flow

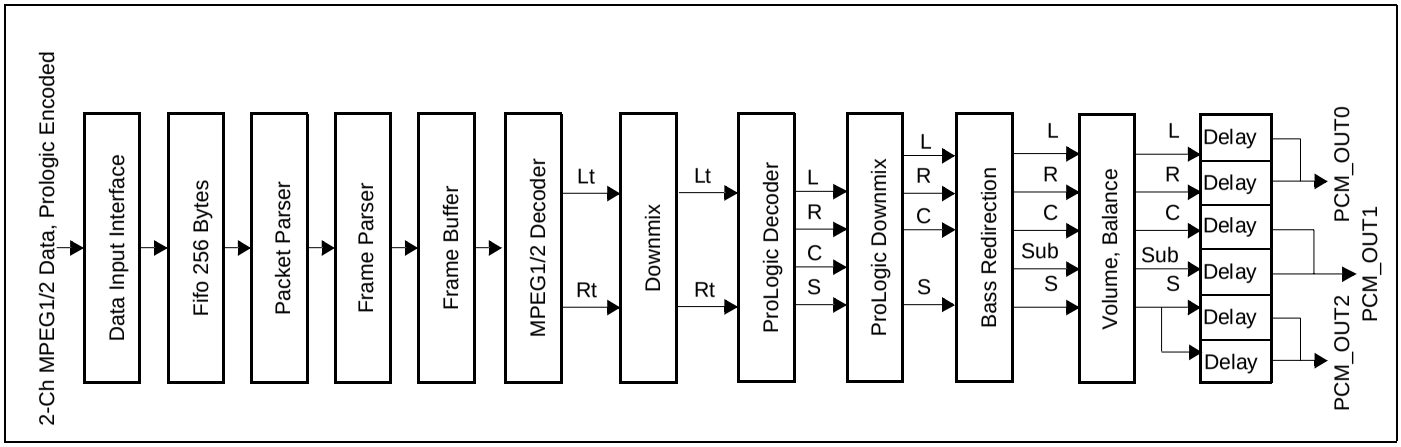


Figure 162 MPEG & ProLogic decoding flow

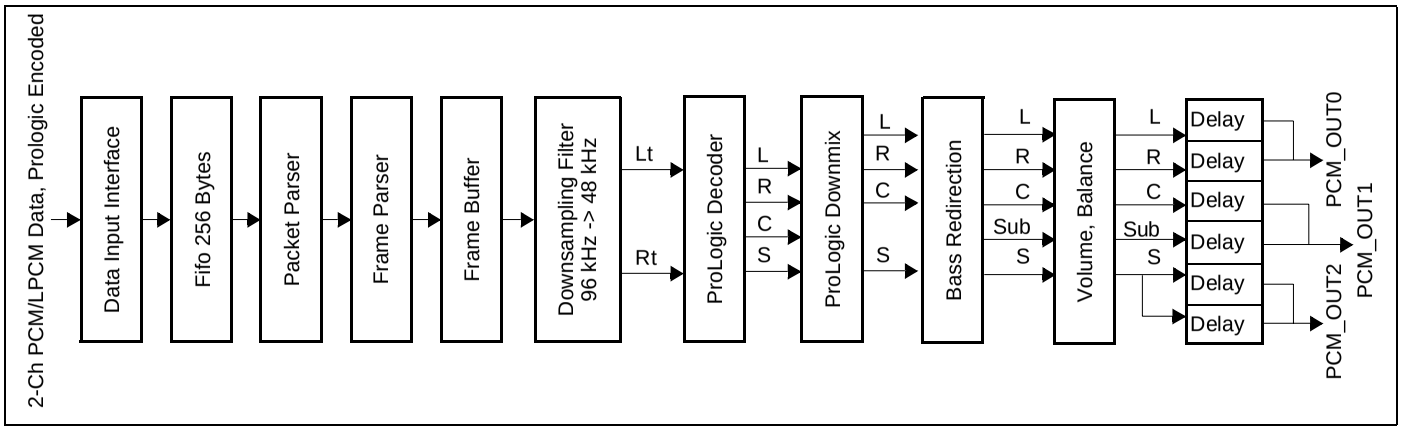


Figure 163 PCM/LPCM & ProLogic decoding flow

Pink-noise decoding modes

The pink noise generator is used to position the speakers in the listening room for optimum sound quality.

The decoder is programmed to generate pink noise by writing the value 4 in the AUD_DECODSEL register. The AUD_DOWNMIX register selects the pink noise output channels.

For pink noise generation, the register configuration should be: AUD_OCFG=0 and AUD_PCM_SCALE=0.

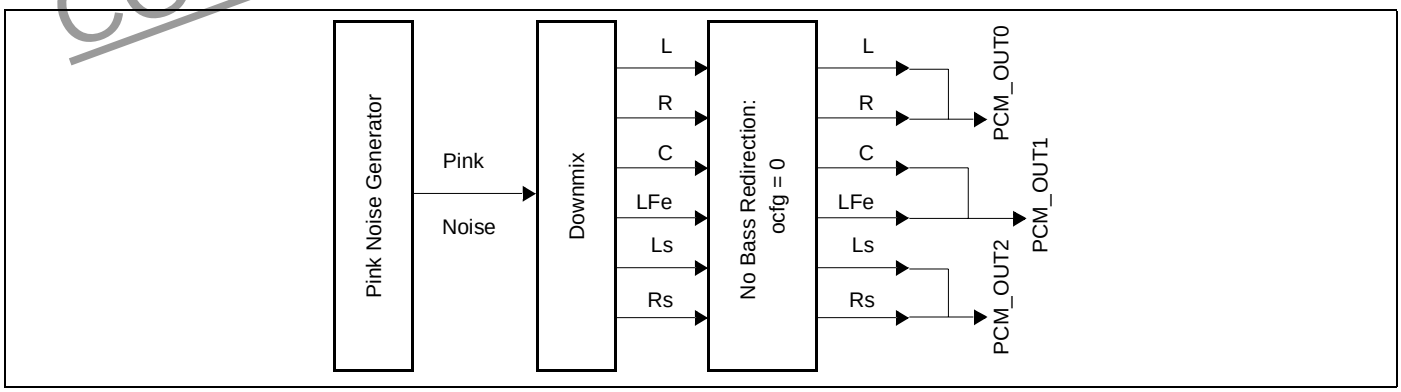


Figure 164 Pink noise decoding flow

Note The appropriate pink noise level is obtained by attenuating all the outputs by 10dB through volume registers.

MP3 decoding mode

MP3 supports the following frequencies in kHz: 12,16, 22.05, 24, 32, 44.1 and 48.

Downmix, PostProcessing, PCM delay and audio trick modes are not supported in MP3 mode.

Register AUD_SKIP_MUTE_CMD cannot be used in MP3 mode.

Volume control is possible in this mode if register AUD_OCFG is set to zero.

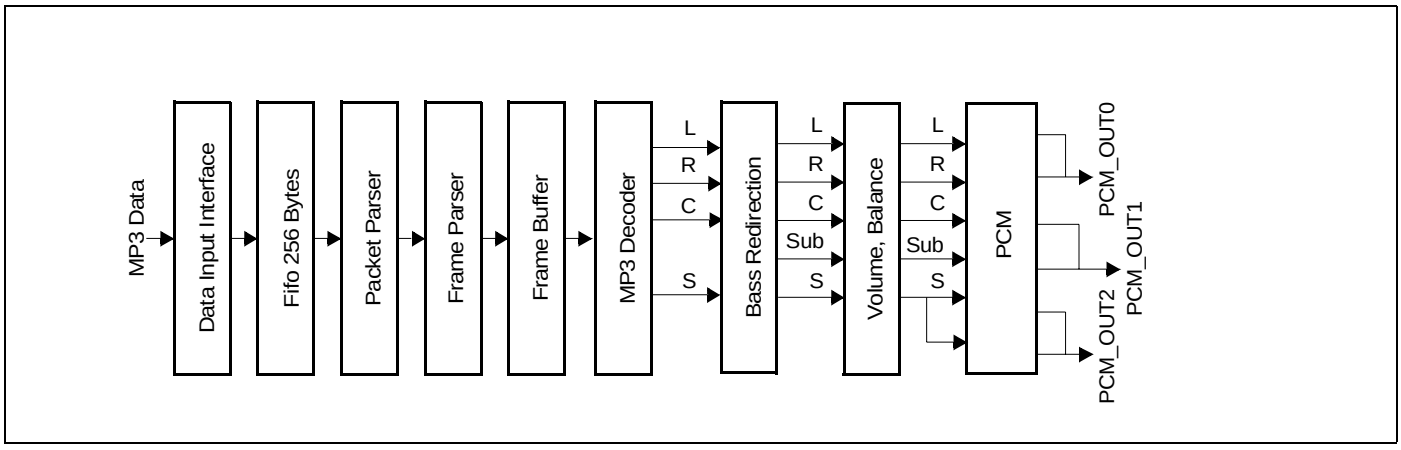


Figure 165 MP3 decoding flow

23.8 PCM output

Output configurations

Figure 166 shows the different configurations supported by the PCM output stage. The configuration is set by the AUD_OCFG register.

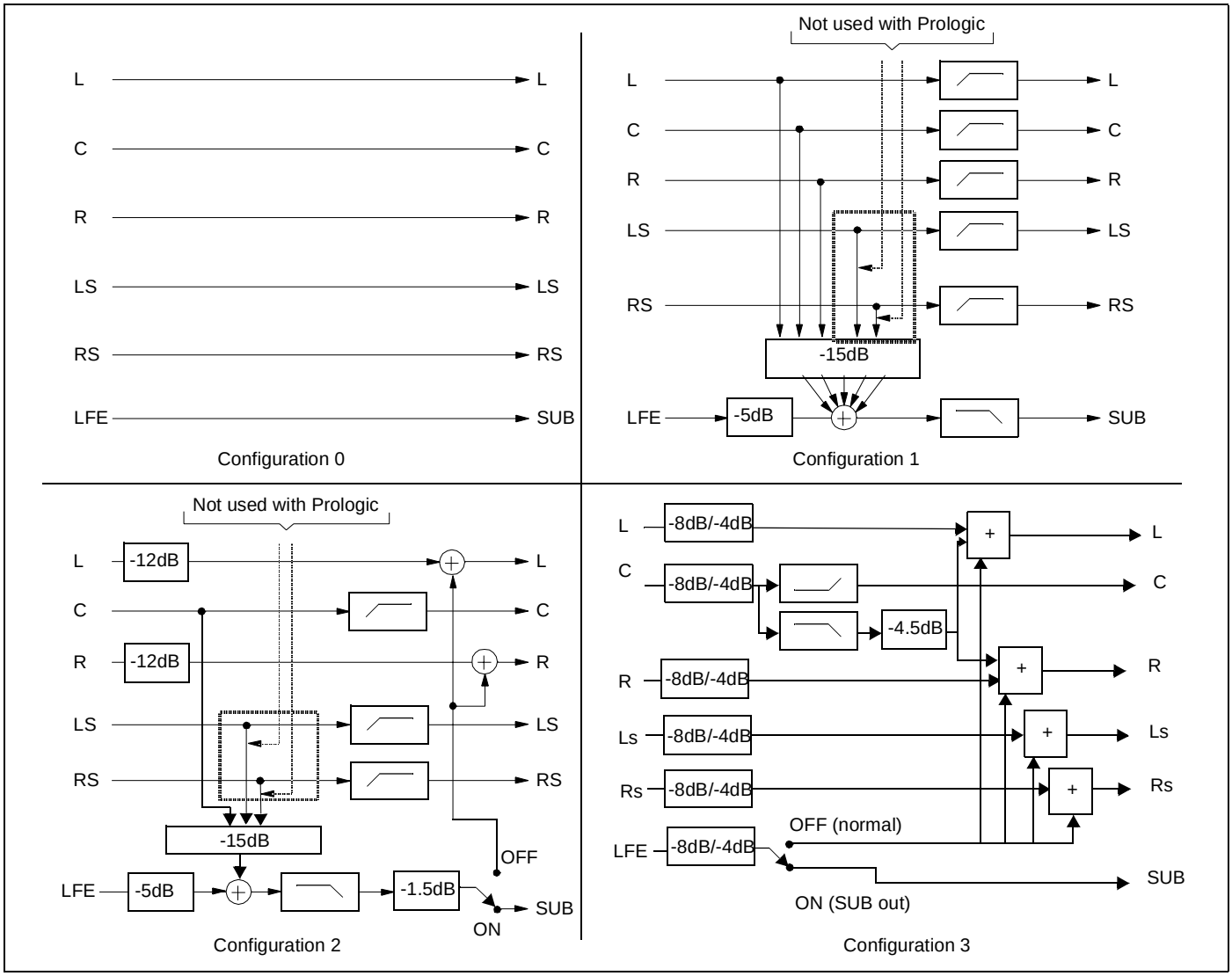


Figure 166 PCM output configurations

- In configuration 0, outputs are only scaled and rounded (see PCM scaling on page 230).
- In configuration 1, the main channels are attenuated by 15dB, and the LFE by 5dB before summing. After digital/analog conversion, the subwoofer pre-amplifier has to compensate for the different gains of the main channels and subwoofer.
- In configuration 2, the sub-woofer is optionally output. The signal for the sub-woofer is the processed sum of the centre and the surround channels (attenuated by 15dB) and the LFE (attenuated by 5dB). If it is not output, it is summed to the left and right channels. The left and right channels must be boosted by 12dB either internally (register bit AUD_OCFG.6) or externally (external amplifier).

- In configuration 3, the subwoofer is optionally output. If it is not output, all of the six input channels are attenuated by 8dB. If the subwoofer is output, the attenuation is reduced to 4dB. The outputs must then be boosted, by the amount of attenuation, either internally (register bit AUD_OCFG.6) or externally (external amplifier).

The same configurations are used for a decoded ProLogic program, with the exception that the surround channels are not added to the bass redirection (the surround channels of a ProLogic program are band limited and bass is considered as leakage).

PCM scaling

PCM scaling is required for every decoding mode. It is applied at the end of the filtering steps, before PCM output, allowing maximum effective word width for most of the signal processing before.

Independent volume for each channel is implemented for PCM scaling (registers AUD_CHAN_IDX, AUD_VOLUME0, AUD_VOLUME1).

Output quantization

For 16/18/20-bit DACs, a quantization with rounding is applied together with the PCM scaling. The sample value is multiplied by a rounding factor and rounded to 24 bits. The result is then left-shifted (4/6/8) for PCM output. The output precision is selectable from the 16bits/word to 24 bits/word by configuring register AUD_PCMCONF[1:0].

Interface and output formats

The decoded audio data are output in serial PCM format. The interface consists of the following signals:

DAC_PCMOUT0, 1, 2	PCM data - output
DAC_SCLK	Bit clock (or serial clock) - output
DAC_LRCLK	Word clock (or Left/Right channel select clock) - output
DAC_PCMCLK	PCM clock - input or output

Output precision and format selection

The PCM output is set in the AUD_PCMCONF register.

- Output precision is set from 16 bits/word to 24 bits/word by register bit AUD_PCMCONF.PREC.
- In 16-bit mode, data can be output either with the MSB or LSB first, by setting register bit AUD_PCMCONF.ORD.
- When AUD_PCMCONF.PREC is set for more than 16 bits, 32 bits are output for each channel.
- In this configuration, register bit AUD_PCMCONF.FOR selects either Sony or I²S compatible format, and register bit AUD_PCMCONF.DIF positions the 18, 20 or 24 bits either at the beginning or at the end of each 32-bit frame.

The following figure and table describe the different output formats, and then 2 configuration examples are given:

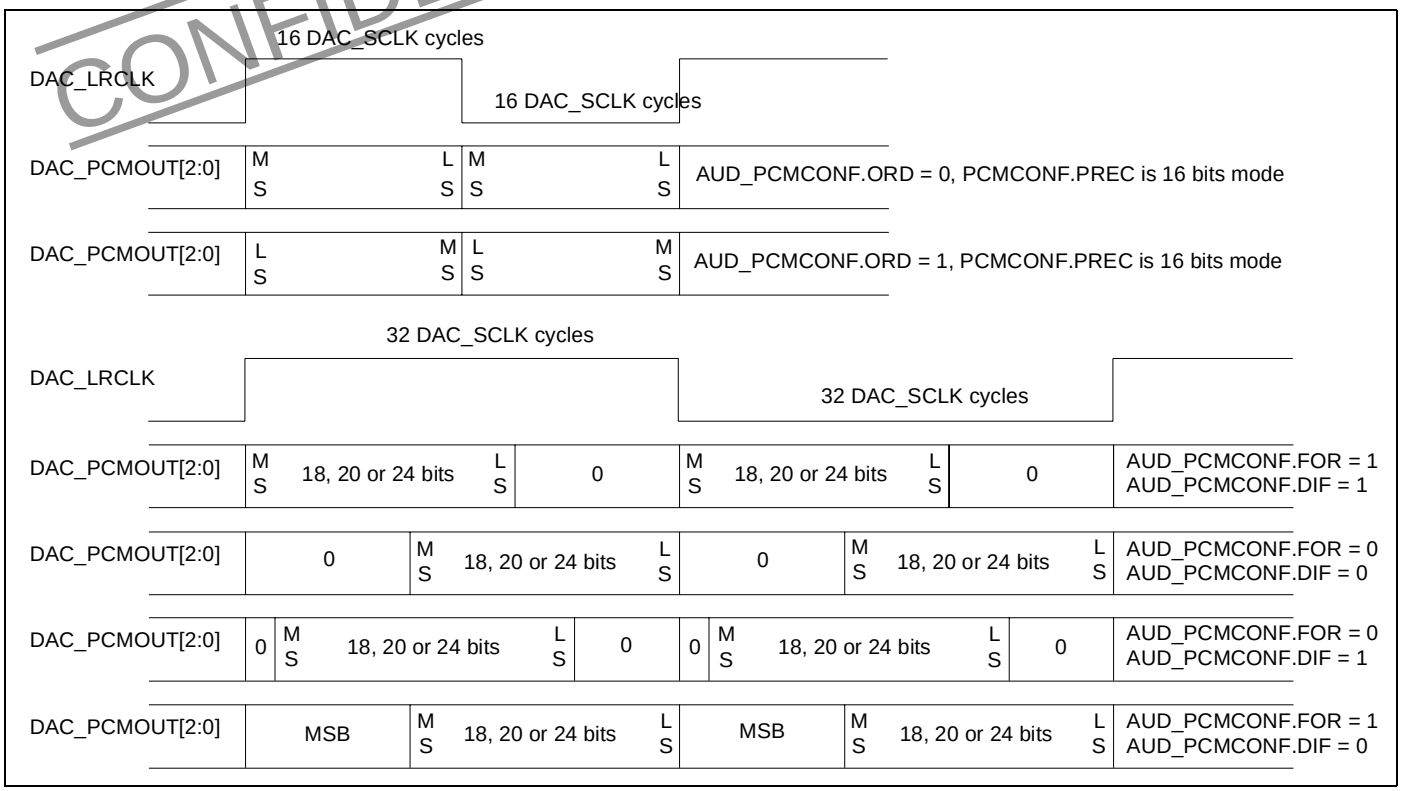


Figure 167 Output formats

AUD_PCMCONF register settings				DATA IN SAMPLE MEMORY DATA [23:0][1]	DATA SENT ON THE PCM SERIAL OUTPUT (LEFT BIT FIRST)
AUD_PCMCONF. PREC	AUD_PCMCONF. ORD	AUD_PCMCONF. FOR	AUD_PCMCONF. DIF		
0:16-bit mode	1	NA	NA	{d23-d8}-{8*0}	{d8-d23}: 16 bits
0:16-bit mode	0	NA	NA	{d23-d8}-{8*0}	{d23-d8}: 16 bits
1:18-bit mode	NA	0	0	{d23-d6}-{6*0}	{13*0}{0}{d23-d6}: 32 bits
1:18-bit mode	NA	0	1	{d23-d6}-{6*0}	{0}{d23-d6}{13*0}: 32 bits
1:18-bit mode	NA	1	0	{d23-d6}-{6*0}	{14*d23}{d26*d6}: 32 bits
1:18-bit mode	NA	1	1	{d23-d6}-{6*0}	{d23-d6}{14*0}: 32 bits
2:20-bit mode	NA	0	0	{d23-d4}-{4*0}	{11*0}{0}{d23-d4}: 32 bits
2:20-bit mode	NA	0	1	{d23-d4}-{4*0}	{0}{d23-d4}{11*0}: 32 bits
2:20-bit mode	NA	1	0	{d23-d4}-{4*0}	{12*d23}{d23-d4}: 32 bits
2:20-bit mode	NA	1	1	{d23-d4}-{4*0}	{d23-d4}{12*0}: 32 bits
3:24-bit mode	NA	0	0	{d23-d0}	{6*0}{0}{d23-d0}: 32 bits
3:24-bit mode	NA	0	1	{d23-d0}	{0}{d23-d0}{7*0}: 32 bits
3:24-bit mode	NA	1	0	{d23-d0}	{8*d23}{d23-d0}: 32 bits
3:24-bit mode	NA	1	1	{d23-d0}	{d23-d0}{8*0}: 32 bits

Table 104 PCM output formats

1. The internal 24-bit decoded, scaled and rounded audio samples are listed as they are stored in memory. These 24 bits are referred to as d23, d22,..., d0, where MSB=d23, LSB=d0.

Configuration example 1: in 16-bit mode, with AUD_PCMCONF.ORD=1: In memory, 24 bits are stored, where only the 16 MSB bits (d23, d22,... to d8) are significant and the 8 remaining bits are 0. This is noted: {d23-d8} {8*0}. The data are sent LSB first, i.e. d8 is sent first and d23 is sent last. This is noted {d8-d23}. 16 bits only are transmitted per channel.

Configuration example 2: in 20-bit mode (AUD_PCMCONF.ORD field is meaningless in this mode), with AUD_PCMCONF.FOR=1 and AUD_PCMCONF.DIF=0: In memory, 24 bits are stored, where only the 20 MSB (d23 to d4) are significant and the remaining 4 LSB are 0.This is noted: {d23-d4} {4*0}. 32 bits are transmitted per channel on the PCM outputs: the 12 first transmitted bits are d23, the last bits are d23 to d4, where d23 is transmitted first. This is noted: {12*d23} {d23-d4}.

Clock polarity

The polarity of the PCM serial output clock (DAC_SCLK) and the PCM word clock (DAC_LRCLK) are selected by the fields SCL and INV respectively, of the AUD_PCMCONF register.

Figure 168 shows the polarities of DAC_SCLK and DAC_LRCKL. The DAC samples DAC_LRCLK and DAC_PCMOUT on the rising edge of DAC_SCLK when AUD_PCMCONF.SCL=0, and on the falling edge of DAC_SCLK when AUD_PCMCONF.SCL=1.

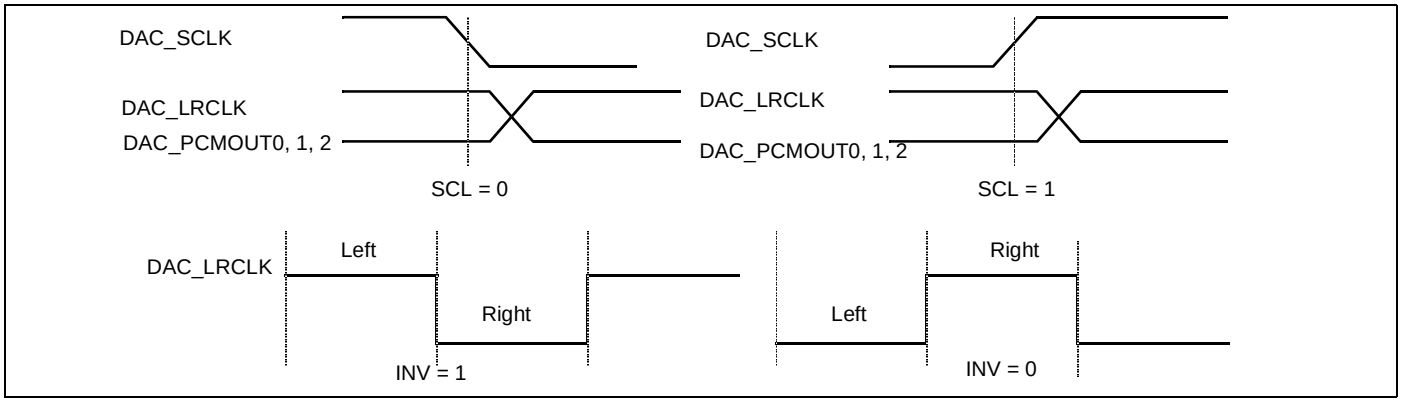


Figure 168 DAC_SCLK and DAC_LRCLK polarity selection

Register configuration	I²S format compatible outputs	Sony format compatible outputs
AUD_PCMCONF.DIF	1: not right padded	
AUD_PCMCONF.FOR	0: I²S format	1: Sony format
AUD_PCMCONF.INV	0: do not invert DAC_LRCLK	1: Invert DAC_LRCLK
AUD_PCMCONF.SCL	0: do not invert DAC_SCLK	0: do not invert DAC_SCLK

Table 105 PCM configuration for I^2S and Sony compatible outputs

23.9 SPDIF output

Overview

The SPDIF output pad is a TTL output pad with slew rate control. The output DC capability is 4 mA and the voltage drop is 3V. This output must be connected to a TTL driver before being connected to a transformer.

The SPDIF output supports IEC-60958 and IEC-61937 standards. The following registers must be initialized to configure the SPDIF output:

- The category code must be entered in the AUD_SPDIF_CAT register. It is related to the type of application. The category code is specified in the Digital Output Interface standard.
- The status bits that will be transmitted on the SPDIF output, must be programmed in the AUD_SPDIF_STATUS register.
- IEC clock setting must be specified in the AUD_SPDIF_CONF register.
- The data type dependent information can be specified in the AUD_SPDIF_DTDI register.
- The SPDIF type is selected through the AUD_SPDIF_CMD register: the IEC unit can output decoded data (PCM mode), encoded data, null data or pause bursts.

When configured in IEC-60958 mode, the SPDIF output is used to transmit the decoded left and right channels. The selection is done by choosing the PCM mode in the register AUD_SPDIF_CMD and resetting the COM status bit in AUD_SPDIF_STATUS register. If register bit AUD_PCMCONF[7] is set to ’1’, 16 bits of data are sent, and if set to ’0’, 24 bits are sent.

When configured in IEC-61937 mode, the SPDIF output is used to transmit encoded data taken directly from the frame buffer. The selection is done by choosing the encoded mode (ENC mode) in the register AUD_SPDIF_CMD and setting the bit COM in AUD_SPDIF_STATUS register. The decompressed data are output simultaneously on the PCM_OUT outputs except in DTS format for which only encoded data are transmitted.

When choosing to output encoded SPDIF data, a latency is automatically inserted between SPDIF output and PCM outputs. The PCM outputs are delayed compared to the SPDIF output. The latency value is defined by standards and applied when the auto-latency mode is selected.

When configured in muted mode (in the AUD_SPDIF_CMD register), the outputs are PCM null data. This can be used to synchronize the external IEC receiver. Register AUD_SKIP_MUTE_CMD bit MUT is used to transmit bursts of pause frames in IEC-61937 format.

Subcode into IEC60958 user data

User Data bits are specified in the IEC60958 specification. Each IEC60958 sub-frame contains 1 user-bit which can be used for subcode insertion in CD-DA mode. A CD-DA frame audio is 2352 bytes (PCM data) = 98 subframes of 24 pcm_data bytes associated with 98 subcode bytes.

A subcode byte is defined by P,Q,R,S,T,U,V,W bits with P bit always set to 1. Alternatively P,Q,R,S,T,U,V,W is included in IEC60958 user data.

The input rate on IEC60958 is one bit of user-data for 20 bits of PCM data, as illustrated in the figure below.

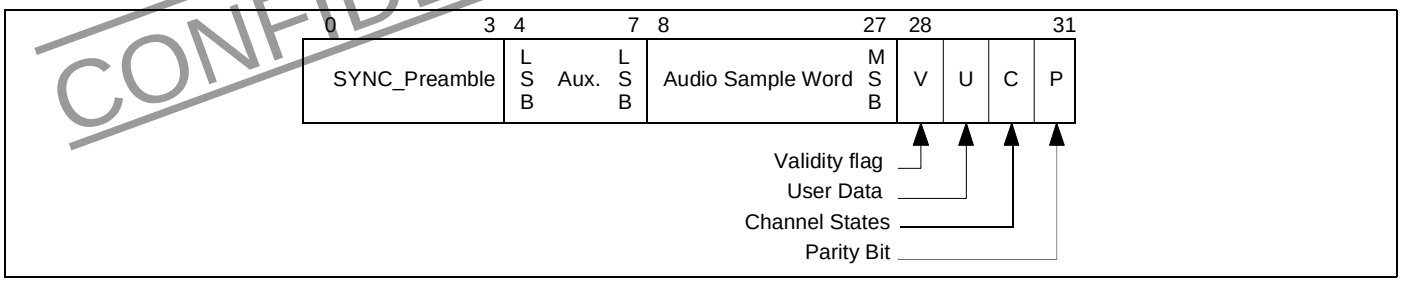


Figure 169 IEC60958 sub-frame format

The inclusion format of the subcodes into the user data is shown in the figure below.

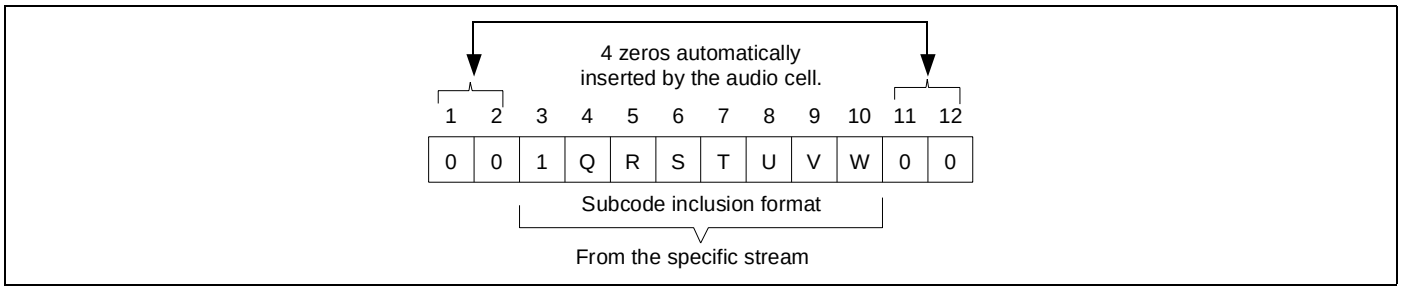


Figure 170 Subcode insertion in IEC6958

There are 8 bits of subcode for 12 bits of user data. That corresponds to 24 bytes of pcm-data if only 16 bits out of the 20 available bits are used. In VCD mode, subcodes are stored in the memory but not included on SPDIF output.

Data flow

When the sector processor is used, it provides 96 subcodes bytes which are collected into a 128x16 bits word buffer. 16 bits of subcodes can be read through the FEI_SUB register.

At the end one sector transfer (signalled with IT EOS), 96 bytes of subcodes can be accessed by reading the subcodes register address 48 times. These 48 values of 16-bits can be stored in the memory-subcode-buffer.

The register FEI_SFF detects the filling level of the FIFO (in the number of 16 bits word).

Note To prevent ST20 hang-up, verify that the subcode buffer is not empty by using FEI_SFF before reading the subcodes with register FEI_SUB.

After processing one sector, the track buffer contains linear PCM samples (DMA transferred) and the memory subcode buffer contains related subcodes (micro transferred).

To synchronize the PCM and subcode, the ST20 reads PCM data from track buffer, and subcode from the memory subcode buffer. It then interleaves 98 bytes of subcode and 2352 PCM bytes into the audio bit buffer.

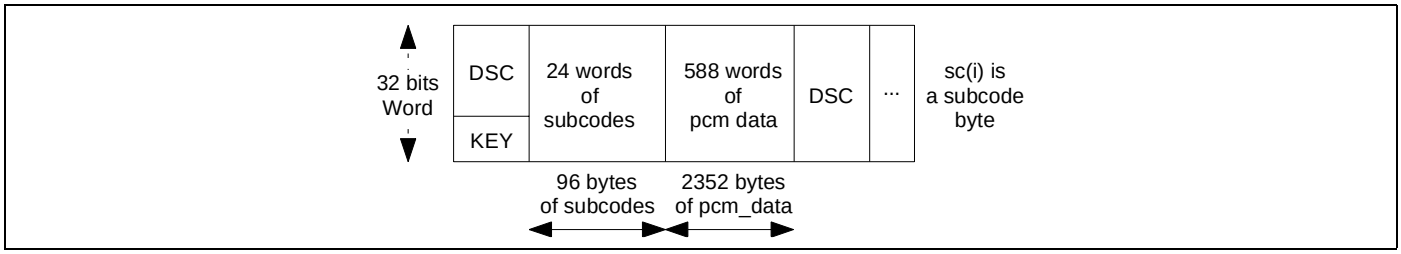


Figure 171 Audio bit buffer content

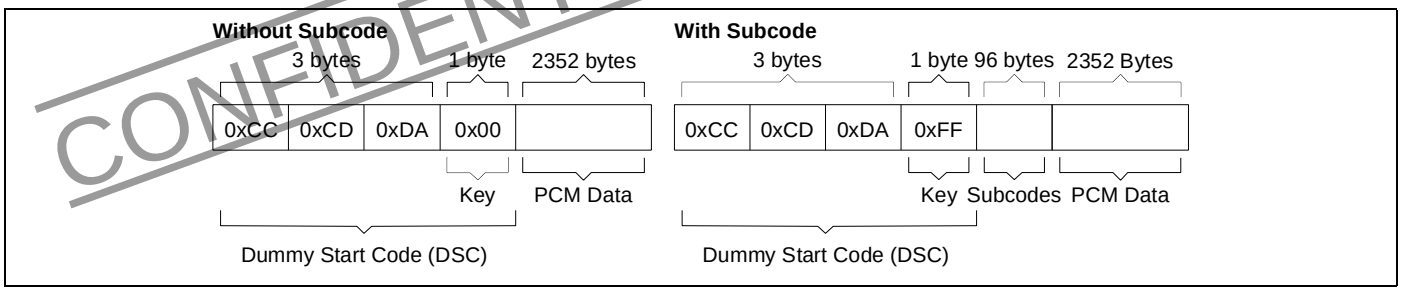


Figure 172 Specific audio input stream

A dummy start code (DSC) of 24 + 8 bits is inserted (by the ST20) into the audio bit buffer to secure the input in audio macrocell. In this case the first word which follows this DSC is guaranteed to be a subcode word.

- Assumption1: The first PCM sample is supposed to be always a left sample.
- Assumption2: A CD-DA PCM sample is 16 bit.

PCM_data in DVD: a normal mode without subcodes insertion is available to input pcm_data into the macrocell (for pcm_data in DVD mode). This is the global mechanism to input subcodes into the macro-cell (see figure below)

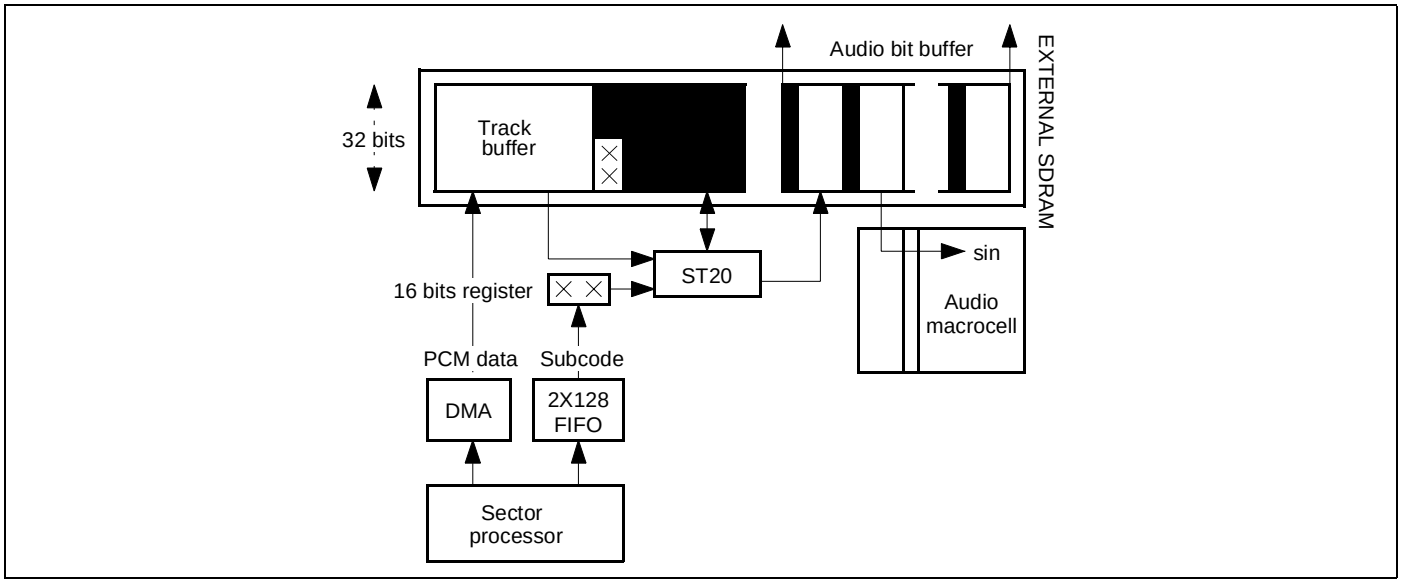


Figure 173 CD_DA and subcode data flow

23.10 Interrupts

Interrupt register

The audio decoder contains a 16 bit interrupt register AUD_INT associated with a 16 bit “enable” register AUD_INTE. A bit set in register AUD_INTE enables the corresponding interrupt. The interrupt associated with each bit is given in the register AUD_INT description.

According to the type of interrupt, other information such as stream header, type of error detected, PTS value, can be obtained by reading associated registers

Error concealment

Errors are signaled as interrupts by the audio core. Most of the errors are automatically handled by the core, but some require that software change. Error categories are defined in the AUD_ERROR register description in the device Register Manual.

Dolby Digital decoding errors are signaled in the AUD_ERROR register but handled directly by the core. These errors cannot be changed by software. Dolby Digital decoding errors signal that something went wrong during decoding. The core soft-mutes the frame and continues to decode.

MPEG decoding errors are signaled in the ERROR register but are handled directly by the core. Nothing can be done by the software. They signal that something wrong happened during the decoding. The core soft-mutes the frame and continues to decode. Only one error in this category indicates a programing error: if triggering the MPEG_EXT_CRC_ERROR, the bit MC_OFF must be set. This indicates that the decoder tries to decode more than 2 channels whereas the incoming stream contains only 2 channels.

Packet and audio synchronization errors are handled internally and usually indicate that the incoming bitstream is incorrect or that it has been incorrectly input to the chip. In these cases, the decoder resets the corresponding parsing stage (packet or audio parser) then searches for the next correct frame.

Miscellaneous errors such as the LATENCY_TOO_BIG error indicate a problem of latency programming which is superior to the maximum authorized value. The latency value should be changed or a switch made to auto-latency mode. Other miscellaneous errors are handled internally.

23.11 Audio/video synchronization

Presentation time stamp detection

When enabled through the INTE register, the interrupt PTS is generated when a PTS is present in the frame that is being output on DAC_PCMOUT (the interrupt is fired when the first decoded samples of the first block of the frame is output).

Pause frames capability

The number of audio blocks for the audio decoder to pause must be programmed in register AUD_SKIP_MUTE_VALUE. Then bit BLK of the AUD_SKIP_MUTE_CMD register must be set. The audio decoder will finish decoding the current frame, softmute the next frame, and pause for the number of blocks specified in AUD_SKIP_MUTE_VALUE. When the pause is finished, decoding continues.

Skip frames capability

The number of frames to skip must be programmed in register AUD_SKIP_MUTE_VALUE. Then bit SKP of the AUD_SKIP_MUTE_CMD register must be set. The audio decoder will finish decoding the current frame, softmute the next frame, and skip the number of frames specified in AUD_SKIP_MUTE_VALUE. After skipping, it resumes decoding from the next incoming frame.

Pause burst capability

To synchronize video and audio outputs, the audio cell must be able to insert a pause on the output when required. This means that the audio decoder has to stop before decoding a new frame and the output of the audio has to be muted for a period of time as illustrated below.

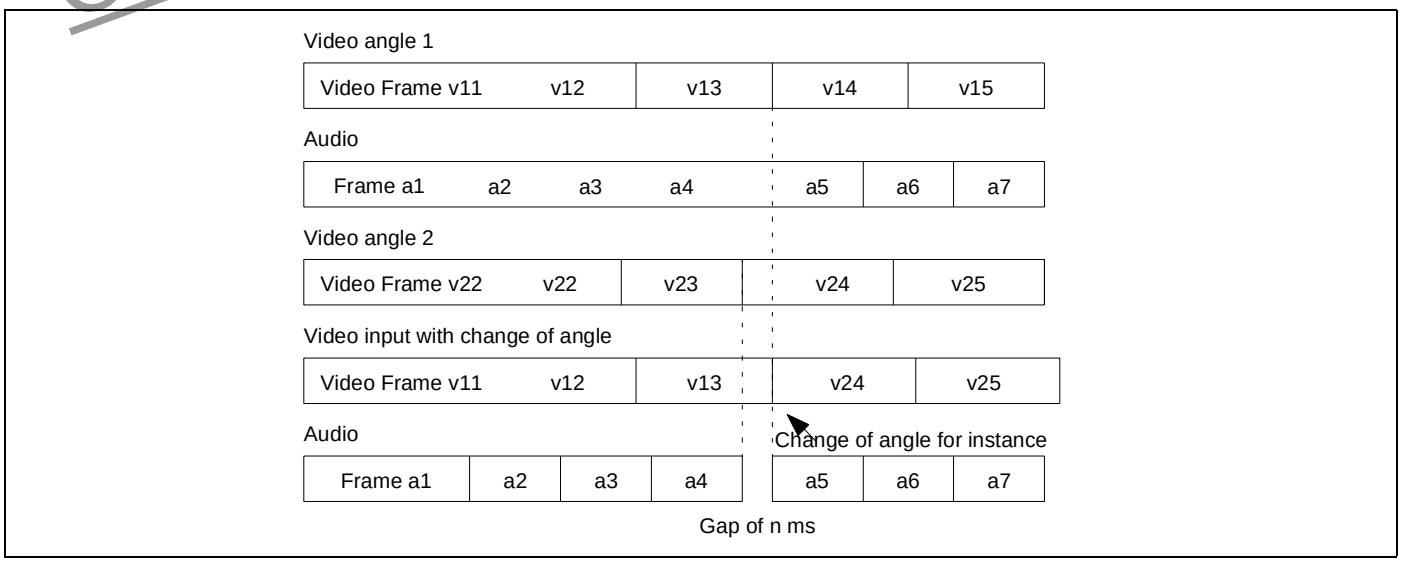


Figure 174 Pause burst capability illustration

A pause is initiated by register AUD_SKIP_MUTE_CMD:

- If bit SKIP_MUTE_CMD.PAU is set, a pause is inserted until bit PAU is reset.
- If register bit AUD_SKIP_MUTE_CMD.BLK is set, a pause burst is inserted for a duration set by the value in register AUD_SKP_MUTE_VALUE.

The granularity of the gap defined by this mechanism is:

- 256 sampling periods for AC-3 (5.3ms at 48 KHz - 5.8ms at 44.1 KHz)
- 96 sampling periods for MPEG (2ms at 48 KHz)

23.12 PCM beep tone

Description

PCM beep tone is a special mode used for Set Top Box. It generates a triangular signal of variable frequency and amplitude on the left and right channels.

Activating PCM beep tone mode

To active this mode:

- Reset the DSP
- Set-up the registers AUD_DECODESEL (0x4D) = 7 and AUD_STREAMSEL (0x4C) = 3
- Restart the DSP by asserting register AUD_RUN and AUD_PLAY

Changing the frequency

Set register AUD_PCM_BTONE (0x68) according to the equation below:

Beep_tone frequency = (Fs/2) / (register_value + 1)

Changing the amplitude

The amplitude of the PCM beep-tone is 0dB by default, to change the amplitude set the registers below:

- AUD_OCFG (0x66) = 0
- AUD_CHAN_IDX (0x67) = 0 (to select the channel pair (left and right)
- AUD_VOLUME0 (0x4E) = Attenuation value (step of -1dB) on left channel
- AUD_VOLUME1 (0x63) = Attenuation value (step of -1dB) on right channel

The PCM beep-tone can be sent to the SPDIF output when the SPDIF output is configured in PCM mode.

23.13 Audio trick modes

23.13.1 Description

Audio trick modes accelerate or slow-down the audio in analogue audio systems. Slow and fast forward are described in this section.

23.13.2 Slow forward

Audio play-back is slowed-down by copying the same sample two or three times into the RAM of the PCM output block.

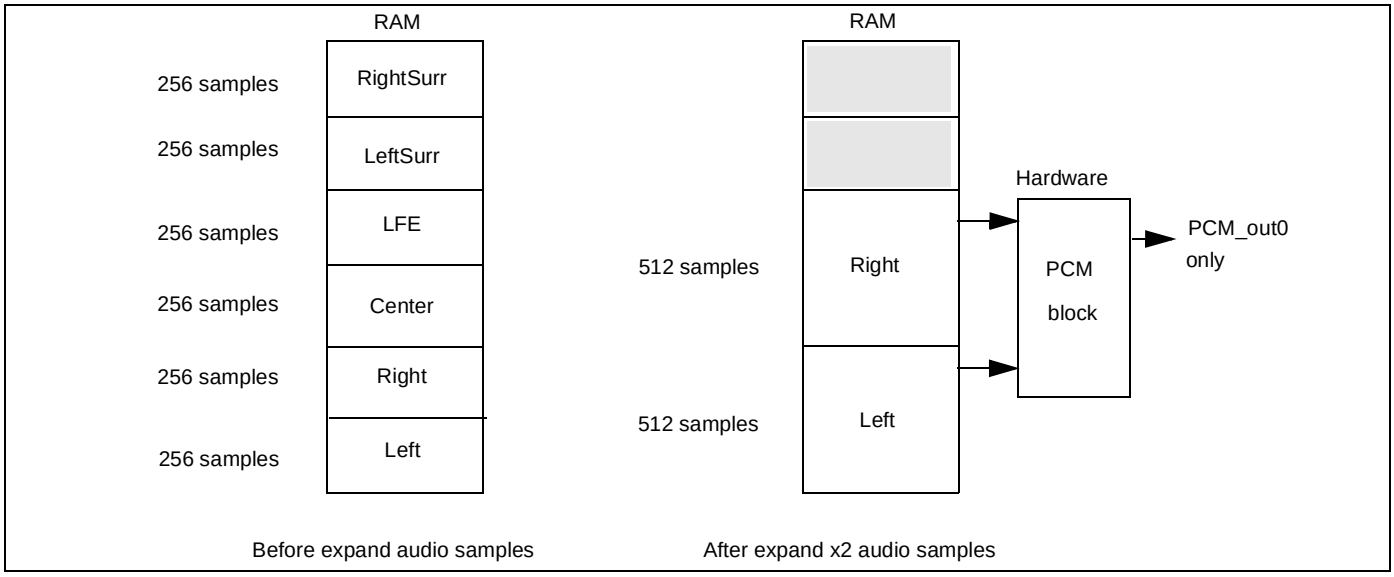


Figure 175 Expanding audio samples for the trick-mode “slow forward”

This method is managed by the embedded software and is independent of the DSP speed. However, it is dependent on RAM size and, as shown in Figure 175, only the left and right channels can be processed for slow forward.

To obtain all of the audio information on the left and right channels, a Dolby Surround compatible downmix must be done before the trick mode is carried out (configuration 2/0 Dolby Surround of the downmix).

Due to the change of PCM block size, Prologic decoding and SRS process must be disabled in slow-forward mode. Volume control and bass redirection are allowed. The AUD_TM_SPEED register can be configured in the following ways:

- AUD_TM_SPEED = 0: normal speed,
- AUD_TM_SPEED = 1: slow forward (twice slower),
- AUD_TM_SPEED = 2: very slow forward (three times slower).

23.13.3 Fast forward

This mode is implemented differently for compressed and non-compressed data. The fast-forward register configuration is identical for non-compressed and compressed algorithms.

Fast forward on compressed algorithms (AC3, MPEG1&2 and DTS)

For compressed algorithms, the audio cell receives data in units of frames which correspond to a duration of approximately 30ms on the PCM output after decoding. The data flow is illustrated below:

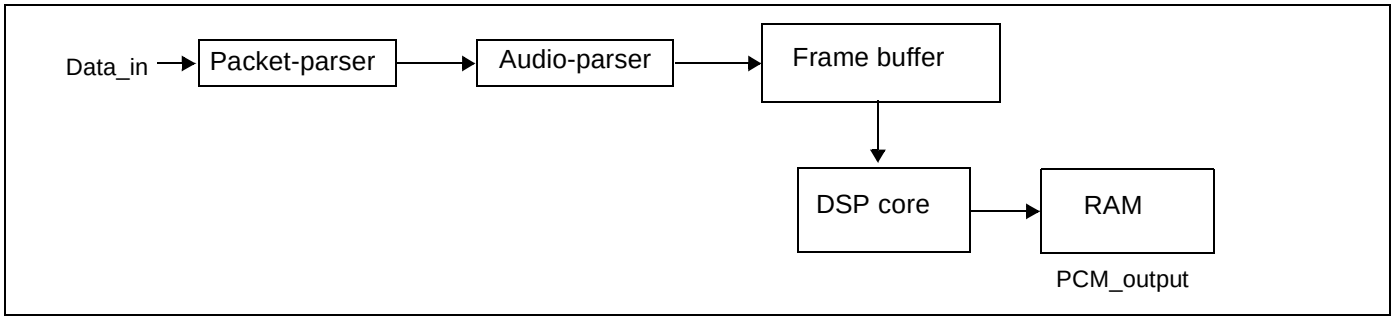


Figure 176 Data flow fast-forward mode on compressed audio algorithms

In normal mode (non fast-forward mode), the Audio Parser sends all of the complete frames to the Frame Buffer. In fast-forward mode, the Audio Parser can be configured to send alternate frames (1 in 2) or every third frame (1 in 3) to the Frame Buffer to be decoded. Fast-forward mode creates discontinuity between each PCM block of samples because the decoder looses the dependency of the missing frames. Post-processing is used to harmonize consecutive frames.

Fast forward on non-compressed algorithms (LPCM, PCM & CDDA)

For these formats, whole frames are handled but samples are skipped at the output. To play 2x faster, 1 sample out to 2 is played; to play 3x faster, 1 sample out of 3 is played.

Register configurations

The speed is set by the AUD_TM_SPEED register configurations below:

- AUD_TM_SPEED = 0: normal speed,
- AUD_TM_SPEED = 0x80: fast forward (two times faster),
- AUD_TM_SPEED = 0x40: very fast forward (three times faster).

To update this mode in the DSP, write the value 2 in the AUD_UPDATE register.

23.13.4 SPDIF output for audio trick modes

The table below summarizes the SPDIF output for audio trick modes:

SPDIF output mode	Non-compressed data	Compressed data
AC3, MPEG1&2, DTS in Slow mode	Ok	Does not work
AC3, MPEG1&2, DTS in Fast mode	Ok	Ok
LPCM, PCM, CDDA in Slow mode	Ok	NA
LPCM, PCM, CDDA in Fast mode	Ok	NA

Table 106 SPDIF output for audio trick modes

24 External audio decoder interface

The STi5518 can be connected to an external audio decoder development platform via the external audio decoder interface.

The interface to an external audio decoder is composed of two buses:

- a synchronous serial interface for compressed data transfer;
- a control interface through the I^2C and Programmable CPU Interface.

This chapter describes the synchronous serial interface. Its four signals are described in the table below, and a schematic of the external audio decoder interface is shown in the figure below..

External audio decoder interface signal name	Signal name	Pin no	Type	Description
EXT_AUD_DATA	DAC_PCMOUT0	52	Out	Packet data
EXT_AUD_CLK	DAC_SCLK	51	Out	Packet strobe
EXT_AUD_REQ	DAC_PCMOUT1	53	In	Data request
EXT_AUD_WCLK	DAC_LRCLK	56	Out	Word clock

Table 107 External audio decoder interface signals

EXT_AUD_REQ is active when the external audio decoder is capable of accepting data and EXT_AUD_CLK is used to strobe the data into the audio decoder on the rising edge. The signal EXT_AUD_WCLK is the EXT_AUD_CLK signal divided by 32. It is phased so that the transition coincides with a byte boundary. This signal can be used as a framing signal for certain external audio decoders. When the external audio decoder interface is crossed, there is no internal limitation on the format of the data that is transferred from the audio bit-buffer to the external decoder.

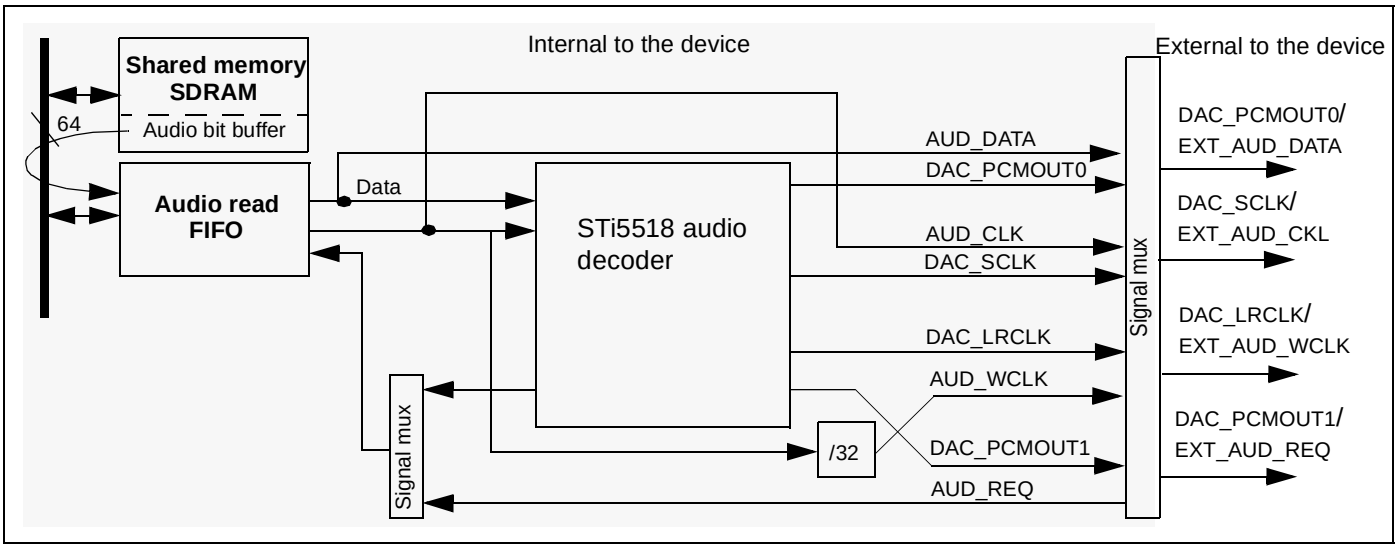


Figure 177 External audio decoder interface schematic

The EXT_AUD_CLK frequency can be either internal clock CLOCK2 or internal clock CLOCK3 (equal to CLOCK2/2). This is set by register VID_CFG_GCF bit SCK.

25 Clock generator

25.1 Introduction

All of the clocks are generated in this clock generator block, and can be defined in the following groups:

- System clocks based on a single PLL. The system PLL multiplies the 27 MHz input clock to generate a common multiple frequency for the ST20 processor, DVD I/F block (Link and FEI), MMDSP audio block and the video block (including the SDRAM clock).
- PCM clock, generated by a digital frequency synthesizer. This is part of the clock generator block, although situated in the audio block for optimum performance.
- SmartCard clock, based on a digital frequency synthesizer included in the Clock Generator.
- Auxiliary clock, provided by a digital frequency synthesizer included in the Clock Generator.
- Low-power, watchdog and power-down.

The system clock frequencies given in this chapter are the default frequencies. For selecting other operating frequencies see the applications note "STi5518 clock management and over-clocking".

The figure below illustrates PLL and Frequencies synthesizer configurations and the device clock distribution.

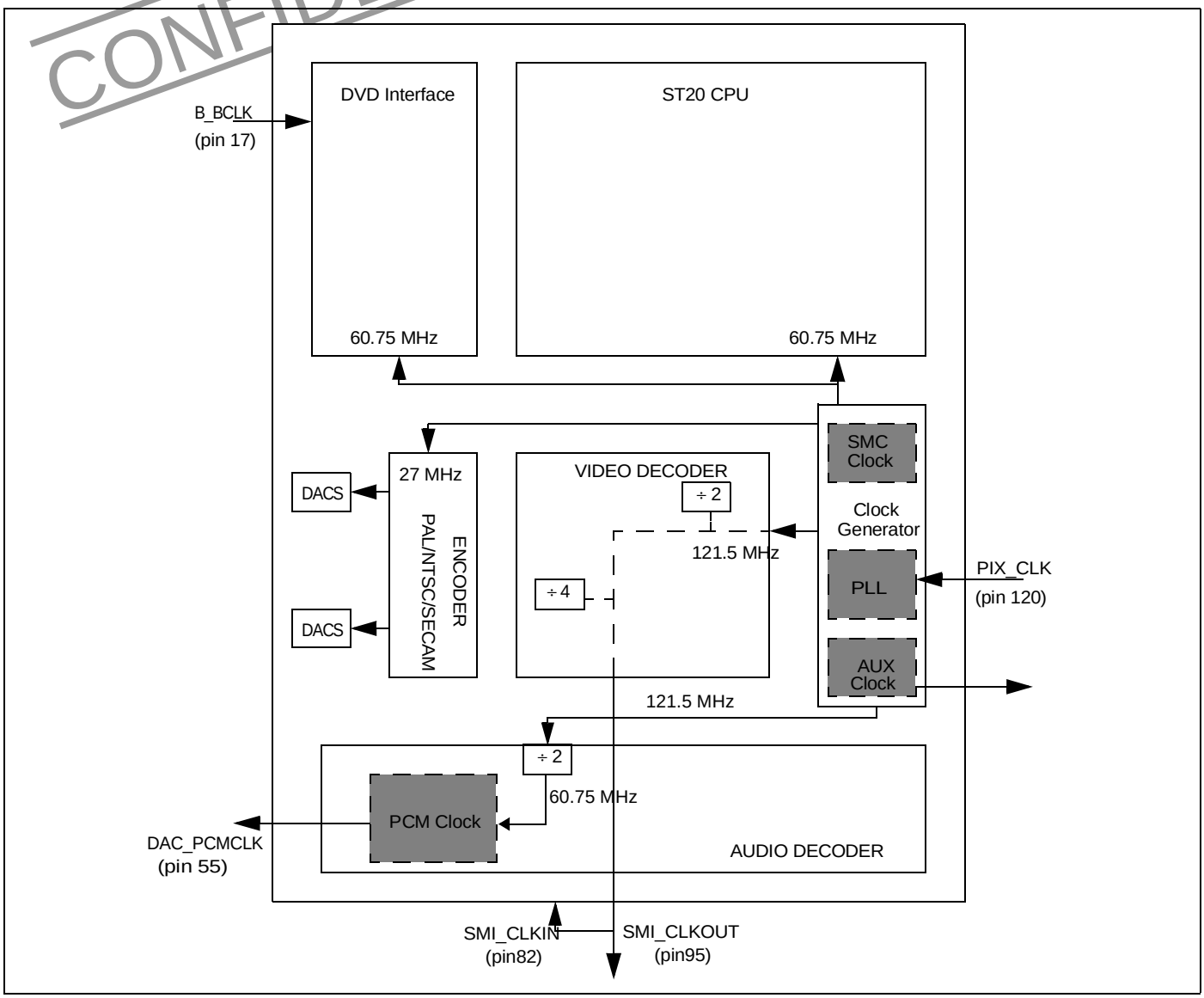


Figure 178 STi5518 PLL and frequency synthesizer configuration

25.2 System clocks

All of the system clocks are generated from the system PLL and integer dividers, with no need for external dividers and PLL circuitry. The reference input frequency is the 27 MHz clock. This reference is multiplied by an integrated PLL, and the PLL output is steered to a bank of 5 dividers. The table below summarizes the system clocks.

Clock	Default value (MHz)	Common value (MHz)	Comment
Input clock	27	27	x 9 by internal PLL = 243 MHz
Video	27	60.75	Generate internal Clk2 and Clk3
SDRAM	27	121.5	Programmable between 100 and 125 MHz to improved Band Width
TPMAC	60.75	60.75	Programmable between 60 and 81 MHz.

Table 108 System clocks summary

Clock	Default value (MHz)	Common value (MHz)	Comment
FEI	60.75	60.75	Same reference as TPMAC clock
Link core	60.75	60.75	Same reference as TPMAC clock
DENC	27	27	From input clock
UART	27	60.75	Normally derived from CPU Clock (ST20 internal divider).
Audio (MMDSP)	27	60.75	
Low power clock	212 kHz	212 kHz	27 MHz divided

Table 108 System clocks summary

Each system clock can be bypassed (output clock = bypass clock), enabled (the output clock is turned off), and divided by 2, separately. This is determined by the value programmed in the respective control registers; the table below gives the recommended divider values.

Clock (PLL frequency, $F_{(clockout)}$ = 243 MHz)	Reset value	Frequency	Divider register value
SDRAM Clock (CKG_DIV_MCK)	27	121.5	0x01
ST20(CKG_CNT_ST20, CKG_DIV_ST20), FEI & LINK	60.75	60.75	0x02
MMDSP (CKG_DIV_AUD)	27	121.5	0x01

Table 109 Recommended divider values

The PLL lock state is readable, and the PLL reset is programmable.

The system PLL multiplies the 27 MHz input clock, the output frequency is calculated as below, where N=162, M=18, P=1 for Fpll = 243 MHz:

$$F_{(clockout)} = \frac{2 \times N}{M \times 2^P} \times F_{(clockin)}$$

Where the values of M, N and P must satisfy the following constraints:

$$1 \le M \le 255,\ 1 \le N \le 255,\ 0 \le P \le 5$$

$$1MHz \le \frac{F_{(clockin)}}{M} \le 2MHz$$

$$F_{(clockin)} \le 200MHz$$

$$200MHz \le \left(\frac{2 \times N}{M}\right) \times F_{(clockin)} \le 622MHz$$

25.3 PCM clock

The PCM clock frequency synthesizer generates the DAC clocks for the audio decoder. After hard reset, the PCM clock pins are inputs to the device. When the AUD_PLLPCM register is set, the PCMCLK clock becomes an output.

The table below shows the values which must be written by the ST20 to obtain the PCMCLK.

The audio clock frequency synthesizer uses the registers CKG_SFREQAUD_SDIV, CKG_SFREQAUD_PE and CKG_SFREQAUD_MD and AUD_PLLMASK to program the frequency. Table 110 lists the register settings versus audio frequency values. Register CKG_SFREQAUD_CNT selects which controller is used.

Frequency	Register values in HEX			AUD_PLLMASK bit HALF_FS
	CKG_SFREQAUD_SDIV (0x1E4)	CKG_SFREQAUD_MD (0x1E7)	CKG_SFREQAUD_PE (0x1E5, 0x1E6)	
384 x 32 kHz	80	88	3600	0
384 x 44.1 kHz	60	C8	3EB2	0
384 x 48 kHz	60	B8	4800	0
256 x 32 kHz	80	D0	5100	0
256 x 44.1 kHz	80	98	6F05	0
256 x 48 kHz	80	88	3600	0
256 x 96 kHz	60	88	3600	0
384 x 96 kHz	40	B8	4800	0
256 x 12 kHz	C0	88	3600	0
384 x 12 kHz	A0	B8	4800	0
384 x 16 kHz	80	88	3600	1
384 x 22.05 kHz	60	C8	3EB2	1
384 x 24 kHz	60	B8	4800	1
256 x 16 kHz	80	D0	5100	1
256 x 22.05 kHz	80	98	6F05	1
256 x 24 kHz	80	88	3600	1

Table 110 PCM frequency values and register settings

25.4 SmartCard clocks

The DIRECTV SmartCard frequency is 18.436 MHz, the register values given in the table below must be used to achieve this frequency.

Frequency (MHz)	CKG_SFREQSMC_SDIV (hex)	CKG_SFREQSMC_MD (hex)	CKG_SFREQSMC_PE (hex)
18.436	3	17	48A7

25.5 Auxiliary clock

The auxiliary clock operates over the frequency range 1-216 kHz, in 1 kHz steps. The table below gives example values for programing the auxiliary clock.

Frequency (MHz)	CKG_SFREQAUX_SDIV (hex)	CKG_SFREQAUX_MD (hex)	CKG_SFREQAUX_PE (hex)
1	7	1A	0
2	6	1A	0
3	6	12	8000
4	5	1A	0

Table 111 Auxiliary clock programming values

Frequency (MHz)	CKG_SFREQAUX_SDIV (hex)	CKG_SFREQAUX_MD (hex)	CKG_SFREQAUX_PE (hex)
5	5	15	3333
10	4	15	3333
15	3	1C	199A
20	3	15	3333
25	3	11	5C29
30	2	1C	199A
35	2	18	283B
40	2	15	3333
45	2	13	6666
50	2	11	5C29
55	1	1F	4A79
60	1	1C	199A

Table 111 Auxiliary clock programming values

25.6 Low-power, watchdog and power-down

Low-power

The low-power timer is a 64bit counter which is always clocked, even when the other internal clocks are stopped. The low-power clock is generated from the 27 MHz input and is divided by the value (2 multiplied by the value programmed into register CKG_DIV_LPC). The LPM_TimerStart register, starts the low-power timer controller.

Watchdog counter

The low-power alarm counter can be used as a watchdog timer if register LPM_WDENABLE bit 0 is set. This makes it impossible to enter low-power mode when starting the low-power alarm counter.

To trigger the watchdog, the low-power alarm is programmed and started as normal. When the low-power alarm counts down to the value #1, the circuit resets.The LPM_WDFLAG register is set when a watchdog reset occurs.

Power-down

In power-down mode the internal clocks are turned off, the processor and ll of the peripherals, including the external memory controller and optionally the PLL, are stopped. Effectively, the internal clock is stopped and functional operation is stalled. On restart, the clock is restarted and the chip resumes normal operation. The PLL is turned on and off using the LPM_SysPLL register

Provided that there are no active external interrupts, power-down is entered when low-power alarm counter LPM_AlarmStart is programmed and started.

Power-down is exited when an enabled external interrupt becomes active, or when the low-power alarm counter reaches zero.

The low-power alarm counter is a 40-bit counter which triggers power-down mode. A write to the LPM_AlarmStart register starts the low-power alarm counter and the device enters low-power mode. When the counter has counted down to zero, and assuming no other valid wake-up sources occur first, the device exits low-power mode and the global clocks are turned back on.

In power-down mode the ST20 PLL can be left running, it can be partially turned off (power and reference still on) or it can be completely turned off. This is determined by the value in the LPM_SYSPLL register. The MPEG PLL can be turned off if required during power-down mode.

26 MPEGDMA controller

The MPEGDMA copies blocks of data from one memory address to an internal or external MPEG device. The source address, destination address and the number of bytes must be specified in the MPEGDMA registers. There are two groups of MPEGDMA registers used for video, audio and subpicture data transfers, MPEGDMA0 and MPEGDMA1.

An MPEGDMA data transfer is initiated by placing source address and destination device values into the MPEGDMAn_SRCADD, MPEGDMAn_BURSTSIZE and MPEGDMAn_WHICHDEC registers respectively, and then writing a byte count value into MPEGDMAn_BLSIZE register to start the data transfer process.

When the data transfer is complete an interrupt is generated, its value can be observed in the MPEG_STATUS register. This interrupt can be enabled onto the external *per_interrupt* bristle for transmission to an interrupt controller, etc. by setting bit 0 in MPEGDMAn_CNTRL register. No further data transfers can be started until the interrupt has been cleared by writing to the MPEGDMAn_INTACK register.

While a data transfer is in progress, the MPEGDMAn_CNTRL and MPEGDMAn_STATUS registers can be accessed, and no further operations can be started by writing to MPEGDMAn_BLSIZE.

At any time during a data transfer operation the process can be stopped by writing to the MPEGDMAn_ABORT register. This stops the data transfer and resets the DMA engine. The *Busy* flag in the MPEGDMAn_STATUS register can be polled to determine whether the DMA engine is ready for further instructions.

The MPEGDMA registers are accessed by bits 2 to 5 of the *dmacnt_and_peraddr* (peripheral address) inclusively. The table below summarizes the MPEGDMA registers, these are described in detail in the STi5518 Register Manual.

Register	Address	Width	Access	Notes
MPEGDMAn_BURSTSIZE	BASE + 0x00	5	W	The number of bytes to be transferred in one burst
MPEGDMAn_HOLDOFF	BASE + 0x04	5	W	Holdoff for MPEG decoder: range 0 to 31, where 0=0 delay cycles
MPEGDMAn_ABORT	BASE + 0x08	1	W	Abort all operation
MPEGDMAn_WHICHDEC	BASE+ 0x0C	2	W	DMA destination pointer
MPEGDMAn_STATUS	BASE + 0x10	2	R	Interrupt status register
MPEGDMAn_INTACK	BASE + 0x14	1	W	Interrupt acknowledge register
MPEGDMAn_SRCADD	BASE + 0x18	32	W	DMA source pointer
MPEGDMAn_CNTRL	BASE + 0x1C	2	R/W	Interrupt control register
MPEGDMAn_BLSIZE	BASE + 0x20	16	W	Data block dimension to be transferred

Table 112 MPEGDMA registers

27 Block move DMA

This module copies blocks of data from one byte address in memory to another. The module can only access memory. A source address, a destination address and a count of the number of bytes to be transferred must be specified.

The interface between the CPU and the block move module is provided using a set of registers and an interrupt. The interrupt signals when a DMA transfer has completed.

To perform a DMA block move from one memory buffer to another, the block move module must first be initialized with the source and destination addresses and then a byte count written to the BMDMA_COUNT register, to specify the amount of data to transfer and start the DMA operation.

The source and destination addresses are the bases of the source and destination areas and can be any byte addresses. The transfer size can be any value in the range of 1 to 65535 bytes. If the source area overlaps with the destination area, then the result is undefined.

At the end of the block move operation the BMDMA_STATUS register will signal that an interrupt is pending. If the interrupt enable bit of the BMDMA_INTEN register is set to 1, this will cause an interrupt. The interrupt pending bit must be reset by software which writes to the BMDMA_INTACK register, before any further block move operations can be performed.

A DMA block move can be aborted by writing to the BMDMA_ABORT register.

28 PWM and counter module

This module provides three PWM encoder outputs, three PWM decoder (capture) inputs and four programmable timers. Each capture input can be programmed to detect rising edge, falling edge, both edges or niether edge (disabled). These facilities are clocked by two independent clocks, one for PWM outputs and one for capture inputs/ timers.

The module generates a single interrupt signal. The exact event which caused an interrupt can be determined by reading status bits in a register, which can then be cleared.

For PWM0 and PWM2 to act as outputs the DENC must operate in master mode. To set the Denc to master mode, set register DEN_CFG0=xx11 0xxx.

28.1 External interface

Name	In/out	Function
PWM0, PWM1, PWM2	out	PWM outputs
Capture_In0,Capture_In1, Capture_In2	in	Capture trigger inputs
Comp_Out0, Comp_Out1	out	Compare output

Table 113 PWM and counter pins

28.2 PWM outputs

There are four PWM outputs which share a common counter. The relative width (in counts) of the output pulse on pin PWM*n* is set between 1 and 256 by loading a value from 0 to 255 into the register PWM_nVal. The width cannot be less than 1, and if it is 256 the pin is continuously high. Pulses occur every 256 counts.

The counter is clocked by the 27 MHz clock ClockIn divided by a prescaler. The prescaling factor, and therefore the period represented by one count, is determined by the value of register PWM_CONTROLFIELD.PWMClkValue. The factor can be from 1 to 16.

The counter (in register PWM_Count) is enabled by setting the register PWM_CONTROLFIELD.PWMEnable to 1. When it is disabled (PWMEnable is 0), the PWM output is forced low. Register PWM_COUNT can be written to at any time, but can have a synchronization latency.

When the PWM counter overflows, an interrupt is generated if register bit PWM_INTENABLE.IntEn is set to 1. Register bit PWM_INTSTATUS.Int becomes 1, and can be reset by writing 1 to register bit PWMINTACK.IntAck.

28.3 Capture inputs

There are four capture inputs which share a common counter with four compare facilities.

What constitutes an event on input CaptureIn*N* is defined by the code in register PWM_nCaptureEdge. Possible events are rising edge, falling edge, both or neither (in other words, disabled).

When an input event occurs on input PWM_nCAPTUREEDGE, the value of the counter (in register PWM_nCAPTURECOUNT) is captured in register PWM_nCAPTUREVAL. The value can be 0x00000000 to 0xFFFFFFFF.

When an input event occurs, an interrupt is generated, provided that the register bit PWM_INTENABLE.IntEn is set to 1. Register bit PWM_INTSTATUS.IntN becomes 1, and can be reset by writing 1 to register bit PWM_INTACK.Ack*N*.

The counter is not stopped nor reset by any of these events. See Capture/compare counter, prescaling and clocking on page 250 for details.

28.4 Compare (programmable timer) facilities

There are four programmable timer facilities which share a common counter with four capture inputs.Each of four compare registers PWM_nCompareVal in the module can be set to a value 0x00000000 to 0xFFFFFFFF.

When the counter in register PWM_CaptureCount reaches the value of register PWM_nCOMPAREVAL, two things happen:

- An interrupt is generated if register bit PWM_INTENABLE.IntEn is set to 1. Register bit PWM_INTSTATUS.Int*N* becomes 1, and can be reset by writing 1 to register bit PWM_INTACK.Ack*N*.
- Pin PWM_nCompareOut takes on the value set in register PWM_nCOMPAREOUTVAL.

The counter is niether stopped nor reset by any of these events. See Capture/compare counter, prescaling and clocking on page 250 below for details of the counter.

28.5 Capture/compare counter, prescaling and clocking

The capture/compare counter is clocked by the prescaled system clock, and is common to all capture and compare functions. The prescaling factor, and therefore the period represented by one count, is determined by the value of register bit PWM_CONTROL.CaptureClkValue. The factor can be from 1 to 32.

The counter (in register PWM_CAPTURECOUNT) is enabled by setting register bit PWM_CONTROL.CaptureEnable to 1. When it is disabled (PWM_CONTROL.CaptureEnable=0), none of the capture or compare functions work. PWM_CaptureCount, like PWM_COUNT, can be read or written to at any time.

When the capture/compare counter reaches its maximum count of 0xFFFFFFFF, it wraps round to count up from zero again.

29 Smartcard interface

The SmartCard interface supports asynchronous protocol SmartCards as defined in the ISO7816-3 standard. Limited support for synchronous SmartCards can be provided in software by using the PIO bits to provide the clock, reset, and I/O functions on the interface to the card. Two SmartCard interfaces are supported on the STi5518.

The UART function of the SmartCard interface is provided by a UART (ASC). UART ASC0 can be used by SmartCard0 and ASC2 can be used by SmartCard1.

Each ASC used by a SmartCard interface must be configured as eight data bits plus parity, 0.5 or 1.5 stop bits, with SmartCard mode enabled. A 16-bit counter, the SmartCard clock generator, divides down either the CPU clock, or an external clock connected to a pin shared with a PIO bit, to provide the clock to the SmartCard. PIO bits in conjunction with software are used to provide the rest of the functions required to interface to the SmartCard. The inverse signalling convention, as defined in ISO7816-3, is handled in software, inverted data and most significant bit first. See Asynchronous serial controller on page 253 for details of the ASC and Parallel input/output port on page 272 for details of the PIO ports.

29.1 External interface

The SmartCard pin functions are described in the table below

Pin	In/Out	Function
SCn_CLK	Out, open drain for 5V cards	Clock for SmartCard
SC External Clock	In	External clock input to SmartCard clock divider
SCn_DATA	Out, open drain driver	Serial data output. Open drain drive
SCn_DATA	In	Serial data input
SCn_RST	Out, open drain	Reset to card
SCn_CMD_VCC	Out	Supply voltage enable/disable
SCn_DETECT	In	SmartCard detection
SCn_DATA_DIR	Out	Indicates if the SmartCard is operating in Serial data output (open drain drive) mode or Serial data input mode.

Table 114 SmartCard interface pins

The SCn_RST, SCn_CMD_VCC, and SCn_DETECT signals are provided by alternate functions of the PIO pins. The UARTn_TXD data signal is connected to the SCn_DATA pin with the correct driver type and the clock generator is connected to the SCn_CLK pin.

The ISO standard defines the bit times for the asynchronous protocol in ETUs, which are related to the clock frequency received by the card. One bit time = one ETU.

The ASC transmitter output and receiver input must be connected together externally. For the transmission of data from the STi5518 to the SmartCard, the ASC must be set up in SmartCard mode.

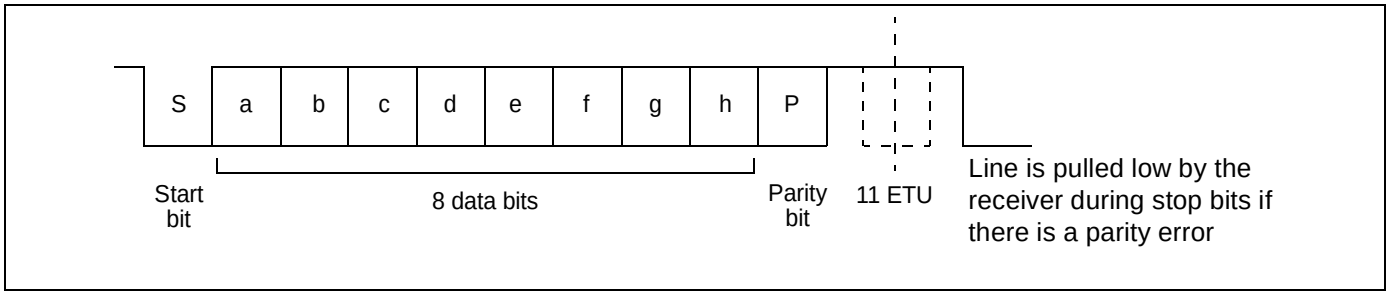


Figure 179 ISO 7816-3 asynchronous protocol

29.2 SmartCard clock generator

The SmartCard clock generator provides a clock signal to the SmartCard. The SmartCard uses this clock to derive the baud-rate clock for the serial I/O between the SmartCard and another UART. The clock is also used for the CPU in the card, if there is one present.

Operation of the SmartCard interface requires that the clock rate to the card is adjusted while the CPU in the card is running code, so that the baud rate can be changed or the performance of the card can be increased. The protocols that govern the negotiation of these clock rates and the altering of the clock rate are detailed in the *ISO7816-3* standard. The clock is used as the CPU clock for the SmartCard, so updates to the clock rate must be synchronized with the clock to the SmartCard. This means the clock high or low pulse widths must not be shorter than either the old or new programmed value.

The clock generator clock source can be set to the system clock or an external pin. Two following two registers control the period of the clock and the running of the clock.

- The SCI_n_CLKVAL determines the SmartCard clock frequency. The value given in the register is multiplied by 2 to give the division factor of the input clock frequency. The divider is updated with the new value for the divider ratio on the next rising or falling edge of the output clock.

  The desired, non zero, value must be programmed into register SCI_n_CLKVAL before the clocks are enabled (by setting the enable bit in register SCI_n_CLKCON.)

- The SCI_n_CLKCON controls the source of the clock and determines whether the SmartCard clock output is enabled. The programmable divider and the output are reset when the enable bit is set to 0.

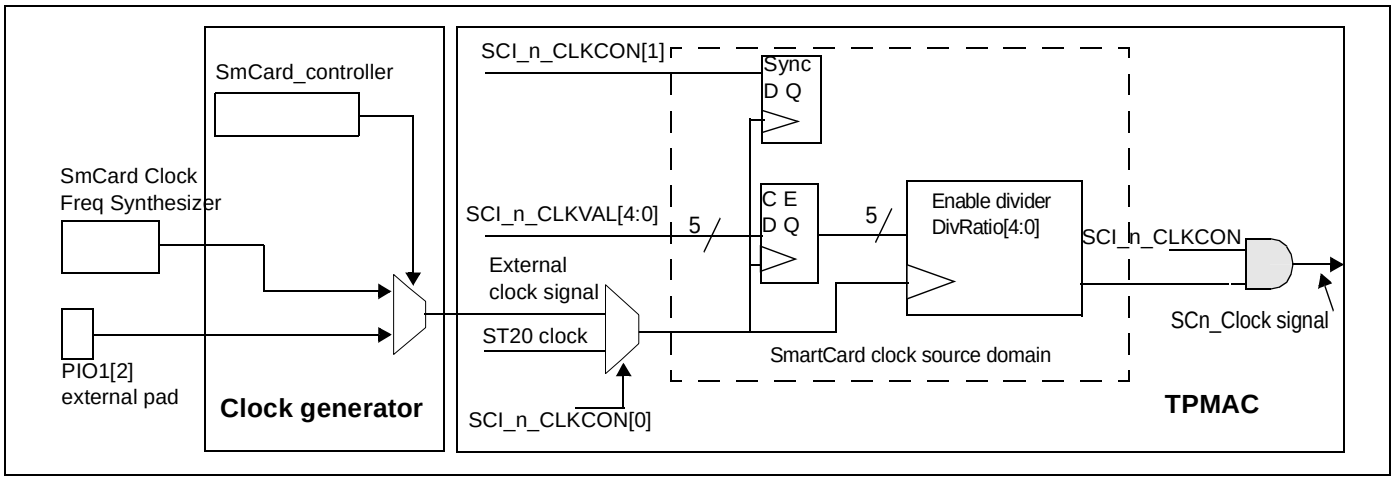


Figure 180 SmartCard clock generation schematic

30 Asynchronous serial controller

The Asynchronous Serial Controller (ASC), also referred to as the UART interface, provides serial communication between the STi5518 and other microcontrollers, microprocessors or external peripherals. The STi5518 provides four ASCs, two of which are generally used by the SmartCard controllers.

Eight or nine bit data transfer, parity generation, and the number of stop bits are programmable. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data can simply be double-buffered, or 16-deep FIFOs may be used. Handshaking is supported on both transmission and reception. For multiprocessor communication, a mechanism to distinguish the address from the data bytes is included. Testing is supported by a loop-back option. A dual mode 16-bit baud rate generator provides the ASC with a separate serial clock signal.

Two ASCs support full-duplex and 2 half-duplex asynchronous communication, where both the transmitter and the receiver use the same data frame format and the same baud rate. For the full-duplex ASCs, data is transmitted on the transmit data output pin TxD and received on the receive data input pin RxD.

Each ASC can be set to operate in SmartCard mode for use when interfacing to a SmartCard.

The registers for each ASC are grouped in a 4 Kbyte block, with the base of the block for ASC number *n* at the address *ASCnBaseAddress*. The value of each *ASCnBaseAddress* is given in the *STi5518 Register Manual*.

30.1 Control

The ASC_n_CONTROL register controls the operating mode of the ASC. It contains control and enable bits, error check selection bits, and status flags for error identification.

Serial data transmission or reception is only possible when the baud rate generator run bit (Run) is set to 1. When the Run bit is set to 0, TxD will be 1. Setting the Run bit to 0 will immediately freeze the state of the transmitter and receiver and should only be done when the ASC is idle.

Note: Programming the mode control field (Mode) to one of the reserved combinations may result in unpredictable behavior.

The ASC can be set to use either double-buffering or a 16-deep FIFO on transmission and reception.

30.1.1 Resetting the FIFOs

The ‘registers’ ASC_n_TXRESET and ASC_n_RXRESET have no actual storage associated with them. A write of any value to one of these registers resets the corresponding FIFO.

30.1.2 Transmission and reception

Serial data transmission or reception is only possible when the baud rate generator run bit (Run) is set to 1. A handshaking protocol is supported on both transmission and reception, using CTS and RTS signals.

A transmission is started by writing to the transmit buffer register ASC_n_TXBUFFER. Because data transmission is double-buffered or uses a FIFO (selectable in the ASC_n_CONTROL register), a new character may be written to the transmit buffer register before the transmission of the previous character is complete. This allows characters to be sent back-to-back without gaps.

Data reception is enabled by the receiver enable bit (RxEnable) in the control register. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the parity error bit, can be read from the receive buffer register ASC_n_RxBuffer.

Reception of a second character may begin before the received character has been read out of the receive buffer register. The overrun error status flag (OverrunError) in the status register ASC_n_STATUS will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received

character in the receive buffer is overwritten, and the ASC_n_STATUS register is updated to reflect the reception of the new character.

The loop-back option (selected by the LoopBack bit) internally connects the output of the transmitter shift register to the input of the receiver shift register. This may be used to test serial communication routines at an early stage without having to provide an external network.

30.2 Data frames

Data frames may be 8-bit or 9-bit, with or without parity and with or without a wake-up bit. The data frame type is selected by the setting of the Mode bit field in the control register.

The transmitted data frame consists of three basic elements:

- The start bit;
- The data field (8 or 9 bits, least significant bit (LSB) first, including a parity bit or wake-up bit, if selected);
- The stop bits (0.5, 1, 1.5 or 2 stop bits).

30.2.1 8-bit data frames

Figure 181 illustrates a 8-bit transmitted data frame. 8-bit frames may use of one of the following formats:

- Eight data bits D0-7 (Mode set to 001);
- Seven data bits D0-6 plus an automatically generated parity bit (Mode set to 011).

Parity may be odd or even, depending on the ParityOdd bit in the ASC_n_CONTROL register. If the modulo 2 sum of the seven data bits is 1, then the even parity bit will be set and the odd parity bit will be cleared.

In receive mode the parity error flag (ParityError) will be set if a wrong parity bit is received. The parity error flag is stored in the 8th bit (D7) of the ASC_n_RXBUFFER register.The parity error bit is set high if there is a parity error.

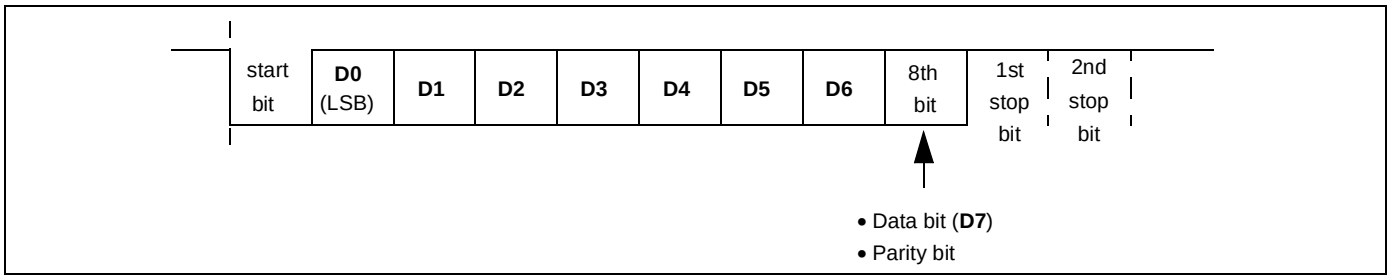


Figure 181 8-bit Tx data frame format

30.2.2 9-bit data frames

Figure 182 illustrates a 9-bit transmitted data frame. 9-bit data frames use of one of the following formats:

- Nine data bits **D0-8** (**Mode** set to 100);
- Eight data bits **D0-7** plus an automatically generated parity bit (**Mode** set to 111);
- Eight data bits **D0-7** plus a wake-up bit (**Mode** set to 101)

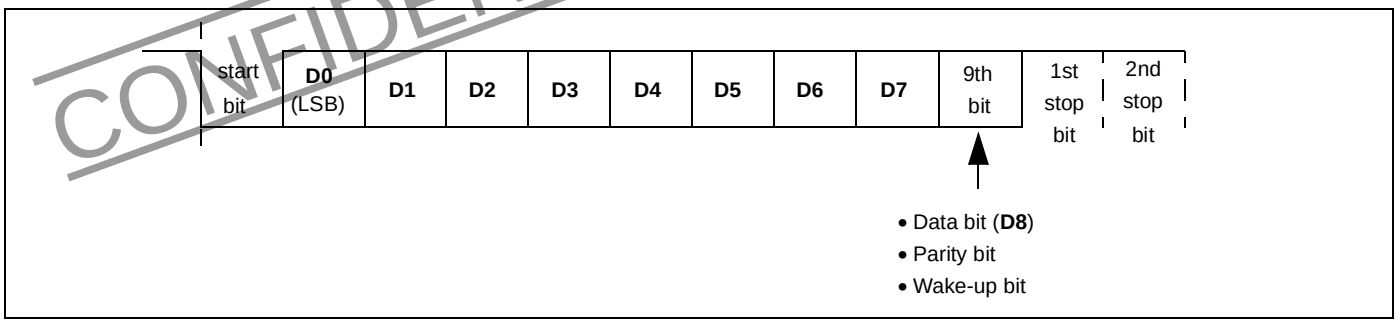


Figure 182 9-bit Tx data frame format

Parity may be odd or even, depending on the ParityOdd bit in the ASC_n_CONTROL register. If the modulo 2 sum of the eight data bits is 1, then the even parity bit will be set and the odd parity bit will be cleared. The parity error flag (ParityError) will be set if a wrong parity bit is received. The parity error flag is stored in the 9th bit (D8) of the ASC_n_RXBUFFER register. The parity error bit is set high if there is a parity error.

In wake-up mode, received frames are only transferred to the receive buffer register if the ninth bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in multi-processor systems. When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional ninth bit is a 1 for an address byte and a 0 for a data byte, so no slave will be interrupted by a data byte. An address byte will interrupt all slaves (operating in *8-bit data plus wake-up bit* mode), so each slave can examine the 8 least significant bits (LSBs) of the received character, which is the address. The addressed slave will switch to 9-bit data mode, which enables it to receive the data bytes that will be coming (with the wake-up bit cleared). The slaves that are not being addressed remain in *8-bit data plus wake-up bit* mode, ignoring the data bytes which follow.

30.3 Transmission

Transmission begins at the next baud rate clock tick, provided that the Run bit is set and data has been loaded into the ASC_n_TXBUFFER. If the CTSEnable bit is set in the ASC_n_CONTROL register then transmission only occurs when CTS is high.

The transmitter empty flag (TxEmpty) indicates whether the output shift register is empty. It will be set at the beginning of the last data frame bit that is transmitted, i.e. during the first system clock cycle of the first stop bit shifted out of the transmit shift register.

The loop-back option (selected by the LoopBack bit of the ASC_n_CONTROL register) internally connects the output of the transmitter shift register to the input of the receiver shift register. This may be used to test serial communication routines at an early stage without having to provide an external network.

30.3.1 Transmission with FIFOs enabled

The FIFOs are enabled by setting the FifoEnable bit of the ASC_n_CONTROL register. The output FIFO is implemented as a 16-deep array of 9-bit vectors. Values to be transmitted are written to the output FIFO by writing to ASC_n_TXBUFFER.

The TxFull bit of the ASC_n_STATUS register is set when the transmit FIFO is considered full, i.e. when it contains 16 characters. Further writes to ASC_n_TXBUFFER will fail to overwrite the most recent entry in the output FIFO. The TxHalfEmpty bit of the ASC_n_STATUS register is set when the output FIFO contains 8 or fewer characters.

Values are shifted out of the bottom of the output FIFO into a 9-bit output shift register in order to be transmitted. If the transmitter is idle (i.e. the output shift register is empty) and something is written to the ASC_n_TXBUFFER so that the

output FIFO becomes non-empty, the output shift register is immediately loaded from the output FIFO and transmission of the data in the output shift register begins at the next baud rate tick.

When the transmitter is just about to transmit the stop bits, and if the output FIFO is non-empty, the output shift register will be immediately loaded from the output FIFO, and the transmission of this new data will begin as soon as the current stop bit period is over (i.e. the next start bit will be transmitted immediately following the current stop bit period). If the output FIFO is empty at this point, the output shift register will become empty. Thus back-to-back transmission of data can take place. If the output FIFO is empty at this point, the output shift register will become empty. Writing anything to ASC_n_TXRESET empties the output FIFO.

After changing the **FifoEnable** bit, it is important to reset the FIFO to empty (by writing to the ASC_n_TXRESET register), or garbage may be transmitted.

30.3.2 Double-buffered transmission

Double buffering is enabled and the FIFOs disabled by writing 0 to the FifoEnable bit of the ASC_n_CONTROL register. When the transmitter is idle, the transmit data written into the transmit buffer ASC_n_TXBUFFER is immediately moved to the transmit shift register, thus freeing the transmit buffer for the next data to be sent. This is indicated by the transmit buffer empty flag (TxHalfEmpty) being set. The transmit buffer can be loaded with the next data while transmission of the previous data is still going on.

When the FIFOs are disabled, the TxFull bit is set when the buffer contains 1 character, and a write to ASC_n_TXBUFFER in this situation will overwrite the contents. The TxHalfEmpty bit of the ASC_n_STATUS register is set when the output buffer is empty.

30.4 Reception

Reception is initiated by a falling edge on the data input pin RxD, provided that the Run and RxEnable bits of the ASC_n_CONTROL register are set.

Controlled data transfer can be achieved using the RTS handshaking signal provided by the UART. The sender checks the RTS to ensure the UART is ready to receive data. In double-buffered reception RTS goes high when ASC_n_RXBUFFER is empty, in FIFO-controlled operation it goes high when RxHalfFull is zero.

The RxD pin is sampled at 16 times the rate of the selected baud rate. A majority decision of the first, second and third samples of the start bit determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value of the first bit of a frame is not a 0, then the receive circuit is reset and waits for the next falling edge transition at the RxD pin. If the start bit is valid, i.e. is 0, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register. For subsequent data and parity bits, the majority decision of the seventh, eighth and ninth samples in each bit time is used to determine the effective bit value. The effective values received on RxD are shifted into a 10-bit input shift register.

For 0.5 stop bits, the majority decision of the third, fourth, and fifth samples during the stop bit is used to determine the effective stop bit value. For 1 and 2 stop bits, the majority decision of the seventh, eighth, and ninth samples during the stop bits is used to determine the effective stop bit values. For 1.5 stop bits, the majority decision of the fifteenth, sixteenth, and seventeenth samples during the stop bits is used to determine the effective stop bit value.

Reception is stopped by clearing the RxEnable bit of ASC_n_CONTROL. Any currently received frame is completed including the generation of the receive status flags. Start bits that follow this frame will not be recognized.

30.4.1 Hardware error detection

To improve the safety of serial data exchange, the ASC provides three error status flags in the ASC_n_STATUS register which indicate if an error has been detected during reception of the last data frame and associated stop bits.

- The parity error bit (ParityError) in the ASC_n_STATUS register is set when the parity check on the received data is incorrect. In FIFO operation parity errors on the buffers are OR-ed to yield a single parity error bit.

- The framing error bit (FrameError) in the ASC_n_STATUS register is set when the RxD pin is not a 1 during the programmed number of stop bit times (see section 30.4). In FIFO operation the bit remains set while at least one of the entries has a frame error.
- The overrun error bit (OverrunError) in the ASC_n_STATUS register is set when the input buffer is full and a character has not been read out of the ASC_n_RXBUFFER register before reception of a new frame is complete.

These flags are updated simultaneously with the transfer of data to the receive input buffer.

30.4.1.1 Frame and parity errors

The most significant bit (bit 9 of 0-9) of each input entry records whether or not there was a frame error when that entry was received (i.e. one of the effective stop bit values was '0'). The FrameError bit of the ASC_n_STATUS register is set when the input buffer (double-buffered operation), or at least one of the valid entries in the input buffering (FIFO-controlled operation), has its most significant bit set.

If the mode is one where a parity bit is expected, then the next bit (bit 8 of 0-9) records whether there was a parity error when that entry was received. It does not contain the parity bit that was received. For 7-bit+parity data frames the parity error bit is set in both the eighth (bit 7 of 0-9) and the ninth (bit 8 of 0-9) bits. The ParityError bit of ASC_n_STATUS is set when the input buffer (double-buffered operation), or at least one of the valid entries in the input buffering (FIFO-controlled operation), has bit 8 set.

When receiving 8-bit data frames without parity (see section 30.2.1), the ninth bit of each input entry (bit 8 of 0-9) is undefined.

30.4.2 Input buffering modes

30.4.2.1 FIFO enabled reception

The FIFOs are enabled by setting the FifoEnable bit of the ASC_n_Control register. The input FIFO is implemented as a 16-deep array of 10-bit vectors (each 9 down to 0). If the input FIFO is empty i.e. no entries are present, the RxBufFull bit of the ASC_n_STATUS register is set to '0'. If one or more FIFO entries are present, the RxBufFull bit of the ASC_n_STATUS register is set to 1. If the input FIFO is not empty, a read from ASC_n_RXBUFFER will get the oldest entry in the input FIFO.

The RxHalfFull bit of the ASC_n_STATUS register is set when the input FIFO contains more than 8 characters. Writing anything to ASC_n_RXRESET empties the input FIFO. As soon as the effective value of the last stop bit has been determined, the content of the input shift register is transferred to the input FIFO (except during wake-up mode, in which case this happens only if the wake-up bit, bit 8, is a '1'). The receive circuit then waits for the next falling edge transition at the RxD pin.

The OverrunError bit of the ASC_n_STATUS register is set when the input FIFO is full and a character is loaded from the input shift register into the input FIFO. It is cleared when the ASC_n_RXBUFFER register is read.

After changing the FifoEnable bit, it is important to reset the FIFO to empty by writing to the ASC_n_RXRESET register; otherwise the state of the FIFO pointers may be garbage.

30.4.2.2 Double buffered reception

Double buffered operation is enabled and the FIFOs disabled by writing 0 to the FifoEnable bit of the ASC_n_CONTROL register. This mode can be seen as equivalent to a FIFO-controlled operation with a FIFO of length 1 (the first FIFO vector is in fact used as the buffer). When the last stop bit has been received (at the end of the last programmed stop bit period) the content of the receive shift register is transferred to the receive data buffer register (ASC_n_RXBUFFER). The receive buffer full flag (RxBufFull) is set, and the parity (ParityError) and framing error (FrameError) flags are updated at the same time, after the last stop bit has been received, i.e. at the end of the last stop bit programmed period. The flags are updated even if no valid stop bits have been received. The receive circuit then waits for the next falling edge transition at the RxD pin.

30.4.3 Time-out mechanism

The ASC contains an 8-bit time-out counter. This reloads from ASC_n_TIMEOUT whenever one or more of the following is true:

- ASC_n_RXBUFFER is read;
- the ASC is in the middle of receiving a character;
- ASC_n_TIMEOUT is written to.

If none of these conditions hold the counter decrements towards 0 at every baud rate tick.

The TimeoutNotEmpty bit of the ASC_n_STATUS register is ‘1’ when the input FIFO is not empty and the time-out counter is zero.

The TimeoutIdle bit of the ASC_n_STATUS register is ‘1’ when the input FIFO is empty and the time-out counter is zero.

The effect of this is that whenever the input FIFO has got something in it, the time-out counter will decrement until something happens to the input FIFO. If nothing happens, and the time-out counter reaches zero, the TimeoutNotEmpty bit of the ASC_n_STATUS register will be set.

When the software has emptied the input FIFO, the time-out counter will reset and start decrementing. If no more characters arrive, when the counter reaches zero the TimeoutIdle bit of the ASC_n_STATUS register will be set.

30.5 Baud rate generation

Each ASC has its own dedicated 16-bit baud rate generator with 16-bit reload capability. The baud rate generator has two possible modes of operation.

The ASC_n_BAUDRATE register is the dual-function baud rate generator and reload value register. A read from this register returns the content of the counter or accumulator (depending on the mode of operation); writing to it updates the reload register.

If the Run bit of the control register is 1, then any value written in the ASC_n_BAUDRATE register is immediately copied to the counter/accumulator. However, if the Run bit is 0 when the register is written, then the counter/ accumulator will not be reloaded until the first CPU clock cycle after the Run bit is 1.

The baud rate generator supports two modes of operation, offering a wide range of possible values. The mode is set via the BaudMode bit in the ASC_n_CONTROL register. Mode 0 is a simple counter driven by the CPU clock whereas Mode 1 uses a loop-back accumulator. Mode 0 is recommended for low baud rates (below 19.2K baud), where its error deviation is low, and Mode 1 is recommended for baud rates above 19.2 K.

30.5.1 Baud rates

The baud rate generator provides an internal oversampling clock at 16 times the external baud rate. This clock only ticks if the Run bit of the ASC_n_CONTROL register is set to 1. Setting this bit to 0 will immediately freeze the state of the ASCs transmitter and receiver.

30.5.1.1 Mode 0

When the **BaudMode** bit in the ASC_n_CONTROL register is set to 0, the baud rate and the required reload value for a given baud rate can be determined by the following formulae:

$$BaudRate = \frac{f_{CPU}}{16 \times ASCBaudRate}$$

$$ASCBaudRate = \frac{f_{CPU}}{16 \times BaudRate}$$

where: *ASCBaudRate* represents the content of the ASC_n_BAUDRATE reload value register, taken as an unsigned 16-bit integer and f_{CPU} is the frequency of the CPU.

The baud rate counter is clocked by the CPU clock. It counts downwards and can be started or stopped by the **Run** bit in the ASC_n_CONTROL register. Each underflow of the timer provides one oversampling baud rate clock pulse. The counter is reloaded with the value stored in its 16-bit reload register each time it underflows.

Writes to the ASC_n_BAUDRATE register update the reload register value. Reads from the ASC_n_BAUDRATE register return the current value of the counter.

30.5.1.2 Mode 1

When the BaudMode bit in the ASC_n_CONTROL register is set to 1, the baud rate is controlled by the following circuit.

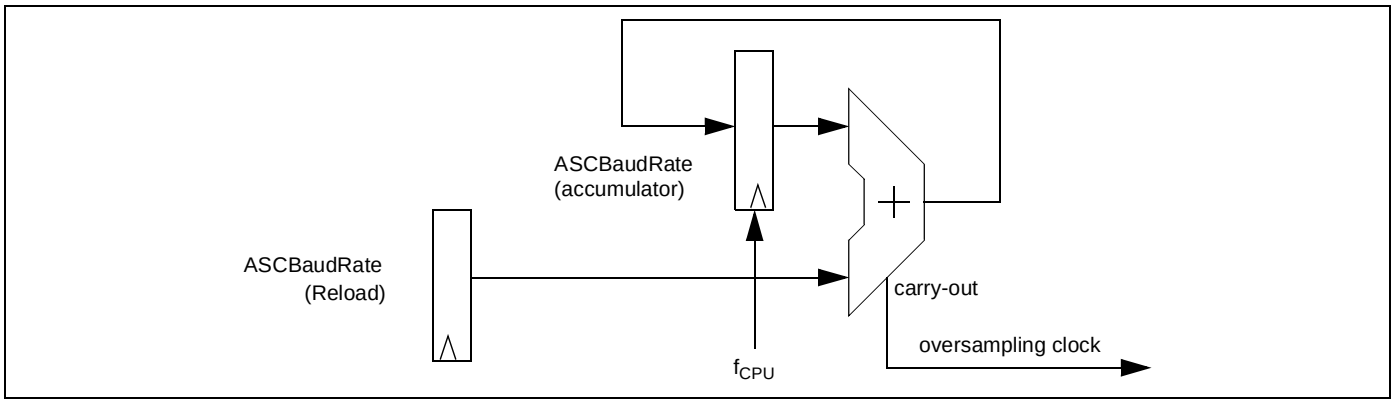


Figure 183 Mode1

Writes to ASC_n_BAUDRATE go to the reload register. Reads from ASC_n_BAUDRATE return the value in the accumulator register. Both registers are 16 bit wide and are clocked by the CPU clock.

If the system clock frequency is f_{CPU}, writing a value of *ASCBaudRate* to the ASC_n_BAUDRATE register results in an average oversampling clock frequency of:

$$\frac{ASCBaudRate \times f_{CPU}}{2^{16}}$$

so the baud rate is given by:

$$BaudRate = \frac{ASCBaudRate \times f_{CPU}}{16 \times 2^{16}}$$

This gives good granularity, and hence low baud rate deviation errors, at high baud rate frequencies.

30.6 Interrupt control

Each ASC contains two registers that are used to control interrupts, the status register (ASC_n_STATUS) and the interrupt enable register (ASC_n_INTENABLE). The status bits in the ASC_n_STATUS register show the cause of any interrupt. The interrupt enable register allows certain interrupt causes to be masked. Interrupts will occur when a status bit is 1 (high) and the corresponding bit in the ASC_n_INTENABLE register is 1.

The ASC interrupt signal is generated from the OR of all interrupt status bits after they have been ANDed with the corresponding enable bits in the ASC_n_INTENABLE register, as shown in Figure 184.

The status bits cannot be reset by software because the ASC_n_STATUS register cannot be written to directly. Status bits are reset by operations performed by the interrupt handler:

- Transmitter interrupt status bits (TxEmpty, TxHalfEmpty) are reset when a character is written to the transmitter buffer.
- Receiver interrupt status bit (RxBufFull) is reset when a character is read from the receive buffer.
- ParityError and FrameError status bits are reset when all characters containing errors have been read from the receive input buffer.
- The OverrunError status bit is reset when a character is read from ASC_n_RXBUFFER.

30.6.1 Using the ASC interrupts when FIFOs are disabled (double-buffered operation)

The transmitter generates two interrupts; this provides advantages for the servicing software. For normal operation (i.e. other than the error interrupt) when FIFOs are disabled the ASC provides three interrupt requests to control data exchange via the serial channel:

- **TxHalfEmpty** is activated when data is moved from ASC_n_TXBUFFER to the transmit shift register;
- **TxEmpty** is activated before the last bit of a frame is transmitted;

- **RxBufFull** is activated when the received frame is moved to ASC_n_RXBUFFER.

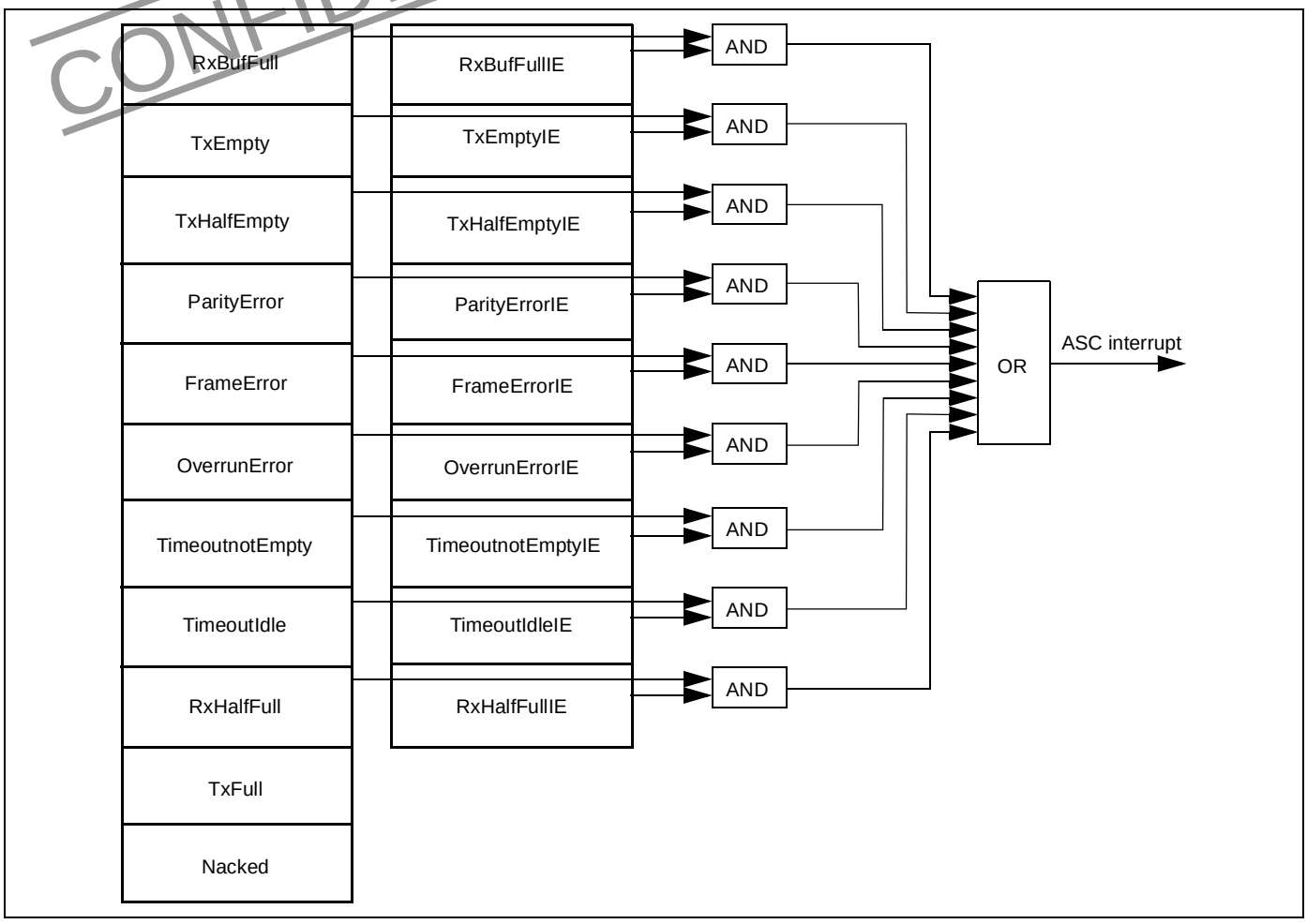


Figure 184 ASC status and interrupt registers

As shown in Figure 185,TxHalfEmpty is an early trigger for the reload routine, while TxEmpty indicates the completed transmission of the data field of the frame. Therefore, software using handshake should rely on TxEmpty at the end of a data block to make sure that all data has really been transmitted.

For single transfers it is sufficient to use the transmitter interrupt (TxEmpty), which indicates that the previously loaded data has been transmitted, except for the last bit of a frame.

For multiple back-to-back transfers it is necessary to load the next data before the last bit of the previous frame has been transmitted. The use of TxEmpty alone would leave just one stop bit time for the handler to respond to the interrupt and initiate another transmission. Using the output buffer interrupt (TxHalfEmpty) to signal for more data allows the service routine to load a complete frame, as ASC_n_TXBUFFER may be reloaded while the previous data is still being transmitted.

30.6.2 Using the ASC interrupts when FIFOs are enabled

To transmit a large number of characters back to back, the driver routine would initially write 16 characters to ASC_n_TXBUFFER. Then every time a TxHalfEmpty interrupt fired, it would write 8 more. When there is nothing more to send, a TxEmpty interrupt would tell the driver that everything has been transmitted.

When receiving, the driver could use RxBufFull to interrupt every time a character arrived. Alternatively, if data is coming in back-to-back, it could use RxHalfFull to interrupt it when there was more than 8 characters in the input FIFO to read. It would have as long as it takes to receive 8 characters to respond to this interrupt before data could overrun.

If less than 8 characters streamed in, and no more were received for at least a time-out period, the driver could be woken up by one of the two time-out interrupts, TimeoutNotEmpty or TimeoutIdle.

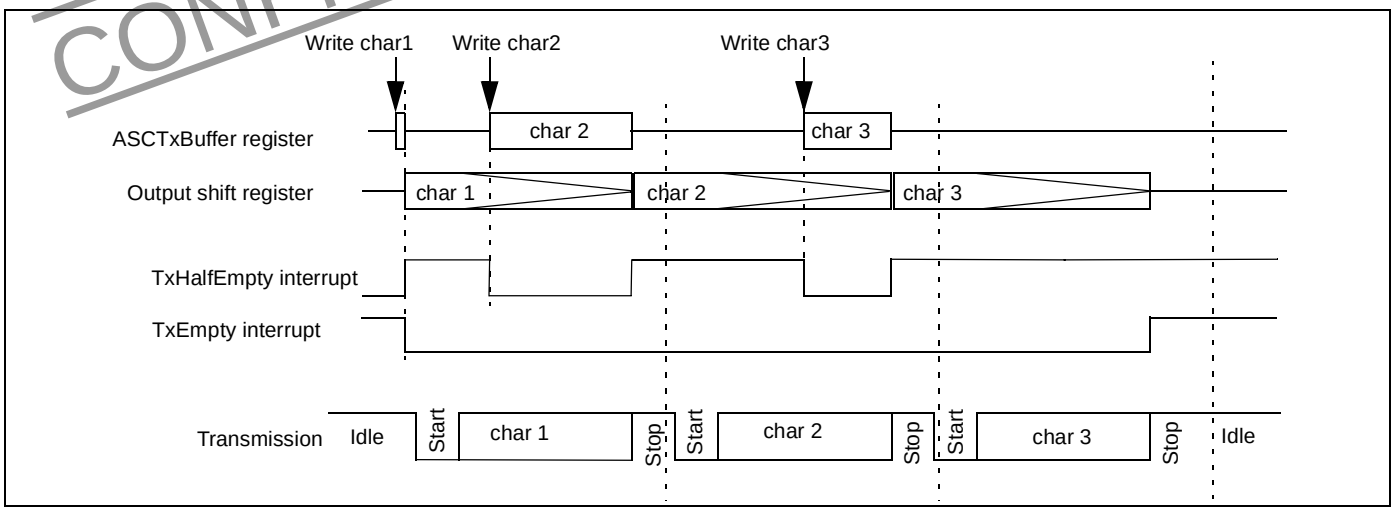


Figure 185 ASC transmission

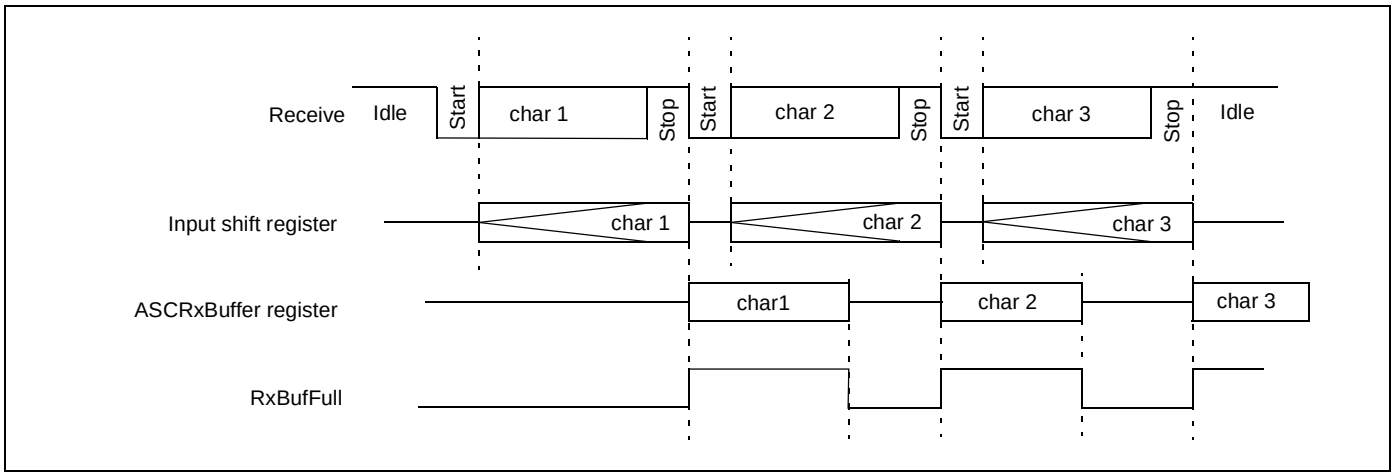


Figure 186 ASC reception

30.7 SmartCard operation

SmartCard mode is selected by setting the SCEnable bit in the ASC_n_CONTROL register to 1. In SmartCard mode the RxD and TxD ports of the UART are both connected externally via a single bidirectional line to a smart card IO port. Characters are transferred to and from the smart card as 8-bit data frames with parity (see Section 30.2). Handshaking between the UART and the SmartCard ensures secure data transfer.

When the SCEnable bit in the ASC_n_CONTROL register is set to 0, normal UART operation occurs.

SmartCard operation complies win the ISO SmartCard specification except where noted (see Section 30.7.4).

30.7.1 Control registers

ASC_n_GUARDTIME

A programmable 8-bit register ASC_n_GUARDTIME controls the time between transmitting the parity bit of a character and the start bit of any further bytes, or transmitting a ‘nack’ (‘no acknowledge’ signal, see Section 30.7.2.1). During the guardtime period the UART receiver is insensitive to possible start bits and the smart card is free to send ‘nacks’.

Guardtime should always be set to at least 2.

ASC_n_RETRIES

A programmable 8-bit register ASC_n_RETRIES defines the number of times the UART will automatically try to send a ‘nacked’ character before giving up.

30.7.2 Transmission

In SmartCard mode FIFOs can be either enabled or disabled. If FIFOs are disabled, the UART transmission behaves according to NDC requirements.

30.7.2.1 Handshaking

When the UART is transmitting data to the smart card, the smart card can ‘nack’ (‘not acknowledge’) the transmission by pulling the line low 0.5 baud clock period into the Guardtime period and holding it low for at least 1 baud clock period. The UART should also be programmed in 1.5 stop bit mode, and since it receives what it transmits, nacks will be detected as receive framing errors.

30.7.2.2 Behavior with FIFOs enabled

At about 1 baud clock period into the Guardtime period, the UART knows whether or not the transmitted character has been ‘nacked’. If no nack has been received and the Tx FIFO is not empty, the next character is transmitted after the guardtime period.

If a transmitted character is nacked by the receiving UART, the character is retransmitted as soon as the Guardtime period expires (or if Guardtime is 2, an extra baud clock period later), and retransmission is attempted up to the number of retries set in the ASC_n_RETRIES register. If the last retry is also ‘nacked’ the Tx FIFO is emptied, putting the transmitter into an idle state, and the Nacked bit is set in the ASC_n_STATUS register.

Emptying of the FIFO causes an interrupt, which can be handled by software. The Nacked bit in the ASC_n_STATUS register can be reset by writing to the ASC_n_TXRESET register.

All ‘un-nacked’ (successfully transmitted) data is looped-back into the receive FIFO. This FIFO can be read by software to determine the status of the data transmission.

30.7.2.3 Behavior with FIFOs disabled

When the SmartCard mode bit is set to 1, the following operation occurs.

- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register will start shifting on the next baud clock edge. In SmartCard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- If a parity error is detected during reception of a frame programmed with a 1/2 stop bit period, the transmit line is pulled low for a baud clock period after the completion of the receive frame, i.e. at the end of the 1/2 stop bit period. This is to indicate to the SmartCard that the data transmitted to the UART has not been correctly received.
- The assertion of the TxEmpty interrupt can be delayed by programming the ASC_n_GUARDTIME register. In normal operation, TxEmpty is asserted when the transmit shift register is empty and no further transmit requests are outstanding.
- The receiver enable bit in the control register is automatically reset after a character has been transmitted. This avoids the receiver detecting a ‘nack’ from the SmartCard as a start bit.

In SmartCard mode an empty transmit shift register triggers the guardtime counter to count up to the programmed value in the ASC_n_GUARDTIME register. TxEmpty is forced low during this time. When the guard time counter reaches the programmed value TxEmpty is asserted high.

The de-assertion of TxEmpty is unaffected by SmartCard mode.

30.7.3 Reception

Reception can be done with FIFOs either enabled or disabled. The behavior is the same as in normal (non-smartcard) mode except that if a parity error occurs, then providing the transmitter is idle, the UART will transmit a nack on the TxD for 1 ETU from the end of the received stop bit. RxD is masked when transmitting a nack, since TxD is tied to RxD and a nack must not be seen as a start bit.

30.7.4 Divergence from ISO SmartCard specification

This UART does not support guardtimes of 0 or 1, and does not have any special behavior for a guardtime of 255.

31 Synchronous serial controller

31.1 Introduction

The high-speed Synchronous Serial Controller (SSC) interfaces to a wide variety of serial memories, remote control receivers, and other microcontrollers. Several interface standards can be used including the I^2C bus in the set-top box application. The figure below shows how the SSC is interfaced to an I^2C bus as the bus master. Software or hardware handles the I^2C bus protocol such as byte acknowledgment, see Section 31.9: *I2C hardware configuration* on page 271.

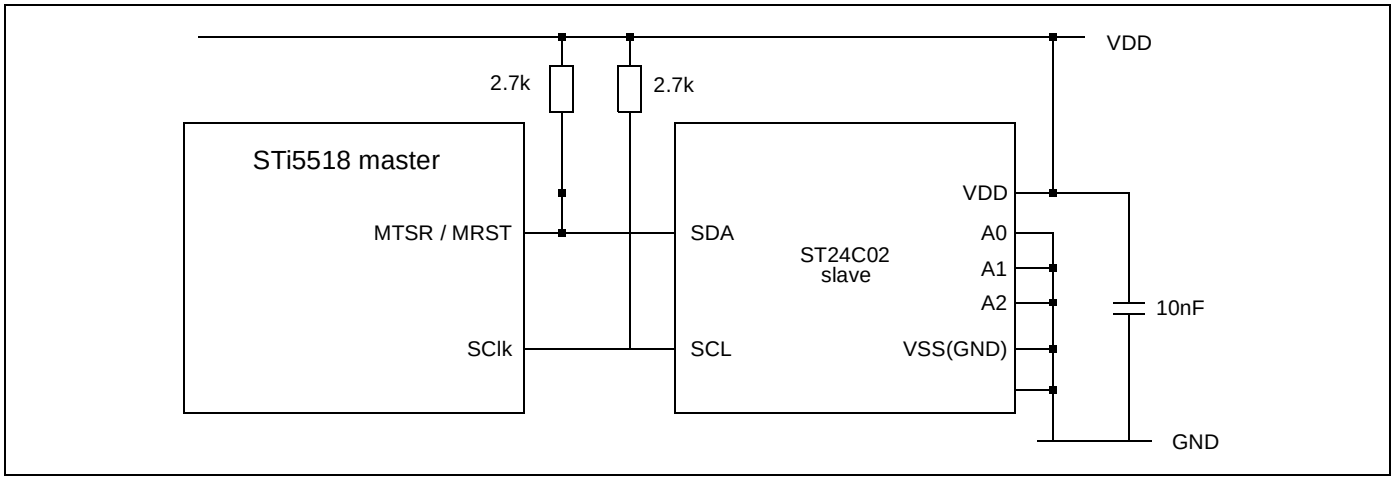


Figure 187 SSC interface to I^2C bus

The SSC provides flexible high-speed serial communication between the STi5518 and other microprocessors or external peripherals, using the I^2C bus protocol, as a master or slave.

The SSC supports half-duplex synchronous communication. The serial clock signal can be generated by the SSC itself in master mode, and data width is programmable. Transmission and reception of data is double-buffered. A 16-bit baud rate generator provides the SSC with a separate serial clock signal.

The high-speed synchronous serial controller can be used to communicate with shift registers (I/O expansion), peripherals (e.g. EEPROMs) or other controllers (networking). The SSC supports half-duplex communication.

31.2 Synchronous serial channel operation

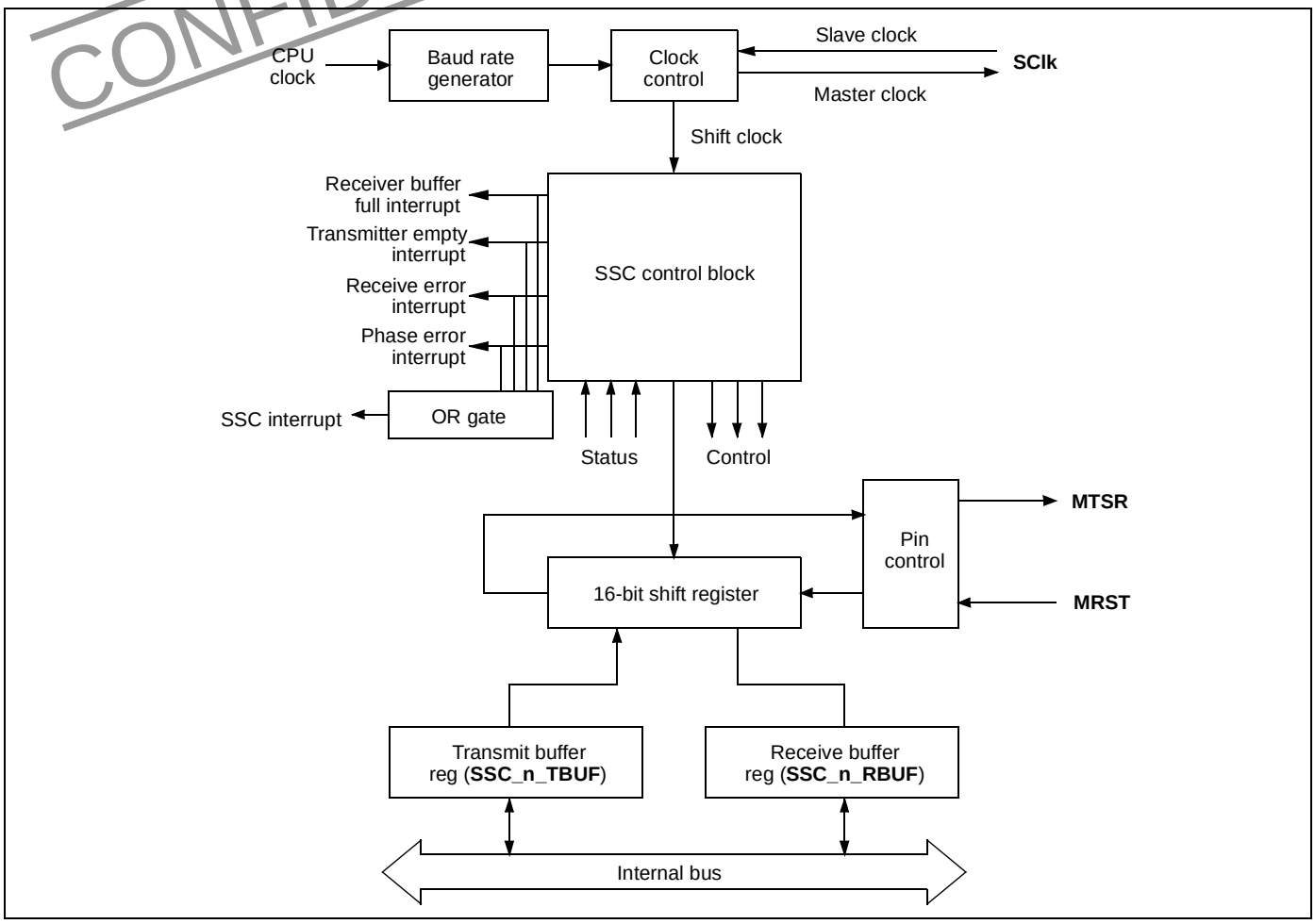


Figure 188 Synchronous serial channel block diagram

The SSC shift register is connected to both the transmit pin and the receive pin via the pin control logic. This is illustrated in the block diagram above. Transmission and reception of serial data is synchronized and the same number of bits are transmitted as received. Transmit data is written into the Transmit Buffer (SSC_n_TXBUF) register, and moved to the shift register as soon as the shift register is empty. Then it is transmitted via the SSC. When the data has transferred to the shift register, the transmit buffer empty (TxBufEmpty) flag is set to indicate that the transmit buffer may be reloaded. When the programmed number of bits (from 2 to 16) has been transferred, the contents of the shift register are moved to the Receive Buffer (SSC_n_RBUF) register and the receive buffer full (RxBufFull) flag is set. If no further transfer is to take place, i.e. the transmit buffer is empty, the SSC reverts back to an idle state, waiting for a load of the transmit register.

Note Only one SSC can be master at a given time.

The serial data bits can be transferred with data width from 2 to 16 bits (set by register bit SSC_n_CON.BM) and for a wide range of baud rates (set by register SSC_n_BRG).

Unused bits of registers SSC_n_TBUF and SSC_n_RBUF must be ignored.

31.3 SSC clocking

When SSC_n_CON register bits ClkPhase=0 and ClkPolarity=0, then the clock and data relationship are I^2C compatible. The data is stable during the high level of the clock, and I^2C setup and hold times are met. This is illustrated in the figure below.

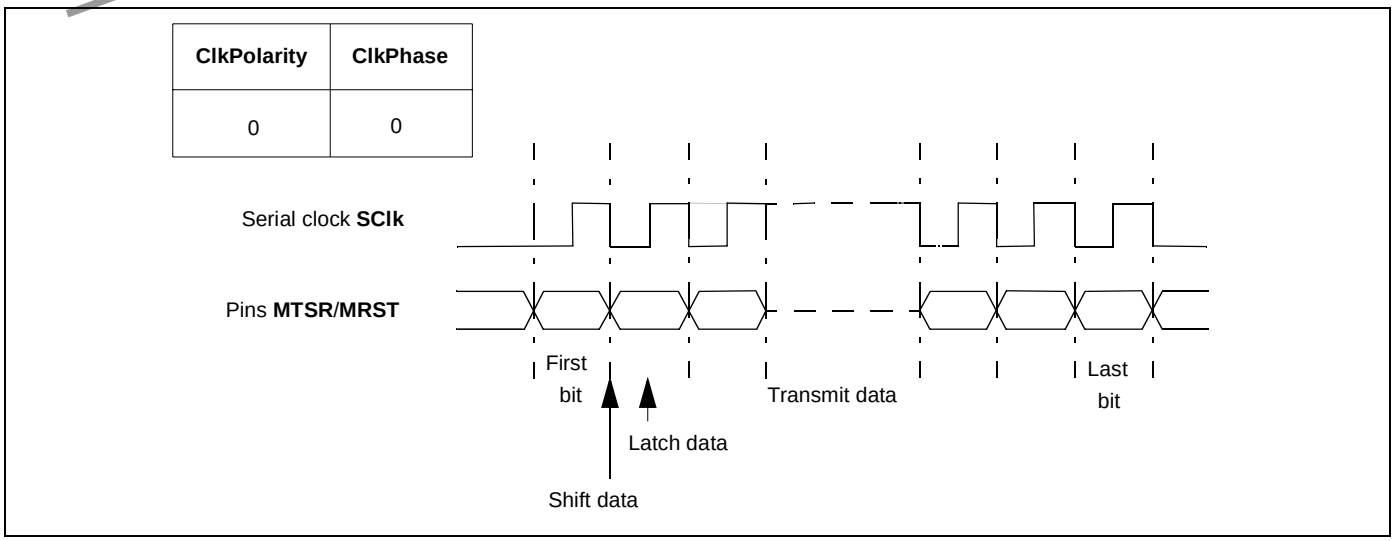


Figure 189 Clock and data relationships

31.4 Half-duplex operation

In a half duplex configuration, only one data line is necessary for both the reception *and* transmission of data. The data exchange line is connected to both pins MTSR and MRST of each device, the clock line is connected to the SClk pin.

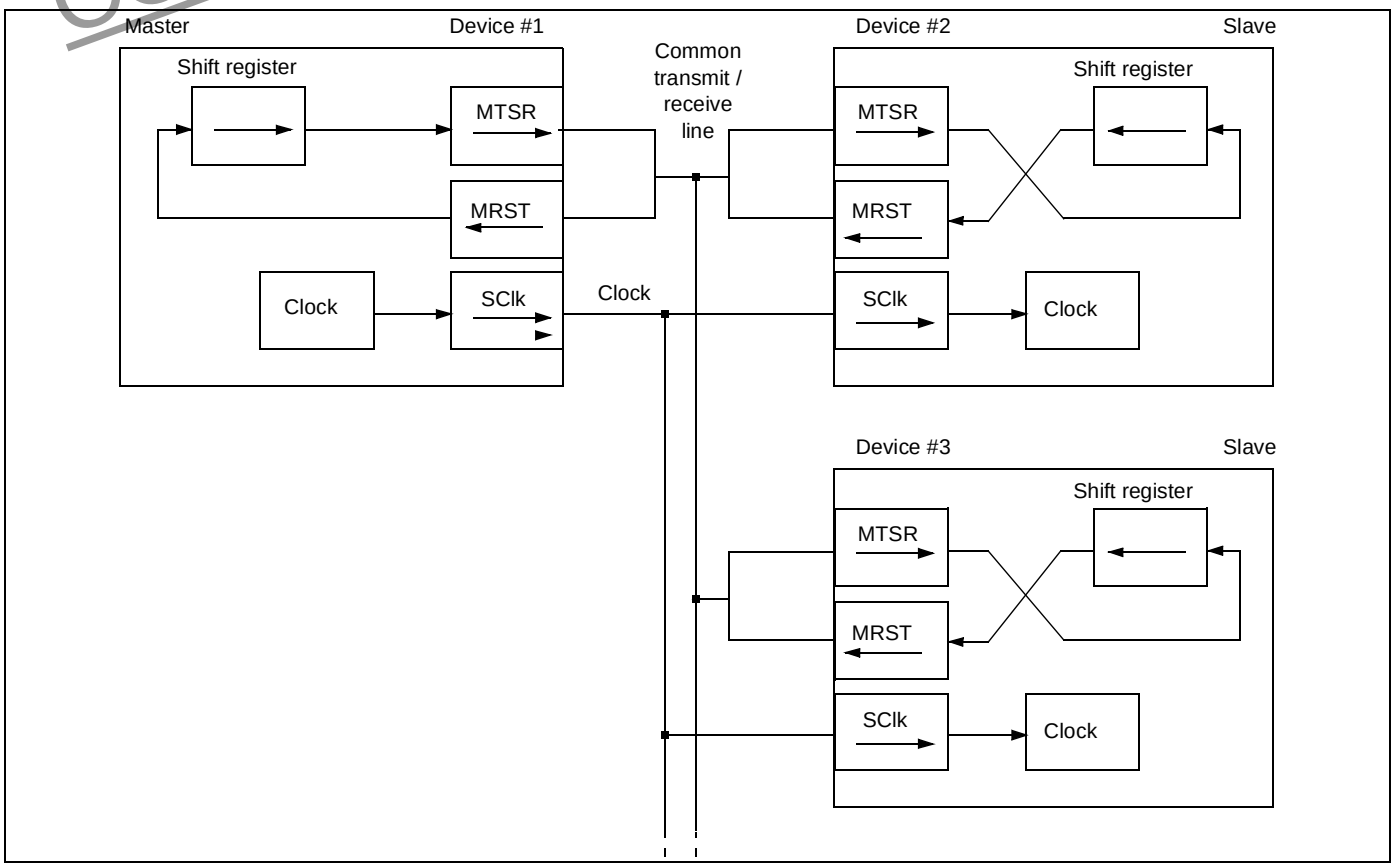


Figure 190 Half-duplex configuration

The master device controls data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

Similar to full duplex mode, there are two ways to avoid collisions on the data exchange line:

- only the transmitting device may enable its transmit pin driver
- the non-transmitting devices use open drain output and only send ones.

Since the data inputs and outputs are connected together, a transmitting device clocks its own data at the input pin (MRST for a master device). This allows detection of any corruptions on the common data exchange line, where the received data is not equal to the transmitted data.

31.5 Continuous transfers

When the register bit SSC_n_STAT.TIR=1, the transmit buffer SSC_n_TBUF is empty and ready to be loaded with the next transmit data. If SSC_n_TBUF has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission starts without any delay. On the data line there is no gap between the two successive frames. For example, two byte transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with, or require more than, 16 data bits per transfer. Software determines how long a total data frame length can be. This option can also be used to interface to

byte-wide and word-wide devices on the same serial bus. Note that this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

31.6 Baud rates

The SSC has its own dedicated 16-bit baud rate generator with 16-bit reload capability. The resultant baud rate for transmission and reception is half the value in the SSC_n_BRG register.

The formulae below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baud rate:

$$\text{Baudrate} = \frac{f_{CPU}}{2 \times \langle \text{SSCBaudRate} \rangle} \qquad \langle \text{SSCBaudRate} \rangle = \left(\frac{f_{CPU}}{2 \times \text{Baudrate}}\right)$$

Where, <SSCBaudRate> represents the content of the reload register as an unsigned 16-bit integer, and f_{CPU} represents the CPU clock frequency.

The maximum baud rate that can be achieved with a CPU clock of 40 MHz is 5 MBaud. The table below lists some possible baud rates, together with the required reload values and the resulting bit times, assuming a CPU clock of 40 MHz.

Baud rate	Bit time	Reload value
Reserved. Use a reload value > 0	-	#0000
5 MBaud	200 ns	#0004
3.3 MBaud	300 ns	#0006
2.5 MBaud	400 ns	#0008
2.0 MBaud	500 ns	#000A
1.0 MBaud	1 µs	#0014
100 KBaud	10 µs	#00C8
10 KBaud	100 µs	#07D0
1.0 KBaud	1 ms	#4E20

Table 115 Baud rates and bit times for different SSC_n_BRG reload values

31.7 Hardware error detection capabilities

The SSC can detect two different error conditions.

- Receive Error
- Phase Error

When an error is detected, the respective error flag is set in the SCC_n_Status register. The error interrupt handler can then check the error flags to determine the cause of the error interrupt.

- A Receive Error is detected, when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSC_n_RBUF. This condition sets the error (RxError) flag and, when enabled via RxErrorIE, the error interrupt request flag (ErrorInterrupt). The old data in the receive buffer SSC_n_RBUF will be overwritten with the new value and is irretrievably lost.

- A Phase Error is detected, when the incoming data on the MRST pin, sampled at the same frequency as the CPU clock, changes between one sample before and two samples after the latching edge of the clock signal. This condition sets the error flag PhaseError and, when enabled via PhaseErrorIE, the error interrupt request flag (ErrorInterrupt).

31.8 Interrupt control

The SSC has two registers to control interrupts, a status (SSC_n_Status) register and an interrupt enable (SSC_n_IEn) register. The status bits in the SSC_n_Status register determine the cause of the interrupt. Interrupts occur when a status bit =1 and the corresponding bit in the SSC_n_IEn register=1.

The error interrupt signal (ErrorInterrupt) is generated by the SSC from the OR of the receive error and phase error status bits after they have been ANDed with the corresponding enable bits in the SSC_n_IEn register.

An overall interrupt request signal (SSCinterrupt) is generated from the OR of the receive interrupt request (RxBufFull), transmit interrupt request (TxBufEmpty) and error interrupt request (ErrorInterrupt) signals.

The status register cannot be written to directly by software. The set and reset mechanism for the status register is described below.

- The receiver interrupt status bit (RxBufFull) is set when a character is loaded from the shift register into the receive buffer (SSCRxBuffer). The RxBufFull bit is reset when a character is read from the receive buffer (SSCRxBuffer).
- The transmitter interrupt status bit (TxBufEmpty) is set when a character is loaded from the transmitter buffer (SSCTxBuffer) into the shift register. The TxBufEmpty bit is reset when a character is written into the transmitter buffer (SSCTxBuffer).
- The status bits (RxError, PhaseError) are reset when a character is read from the receive buffer (SSCRxBuffer).

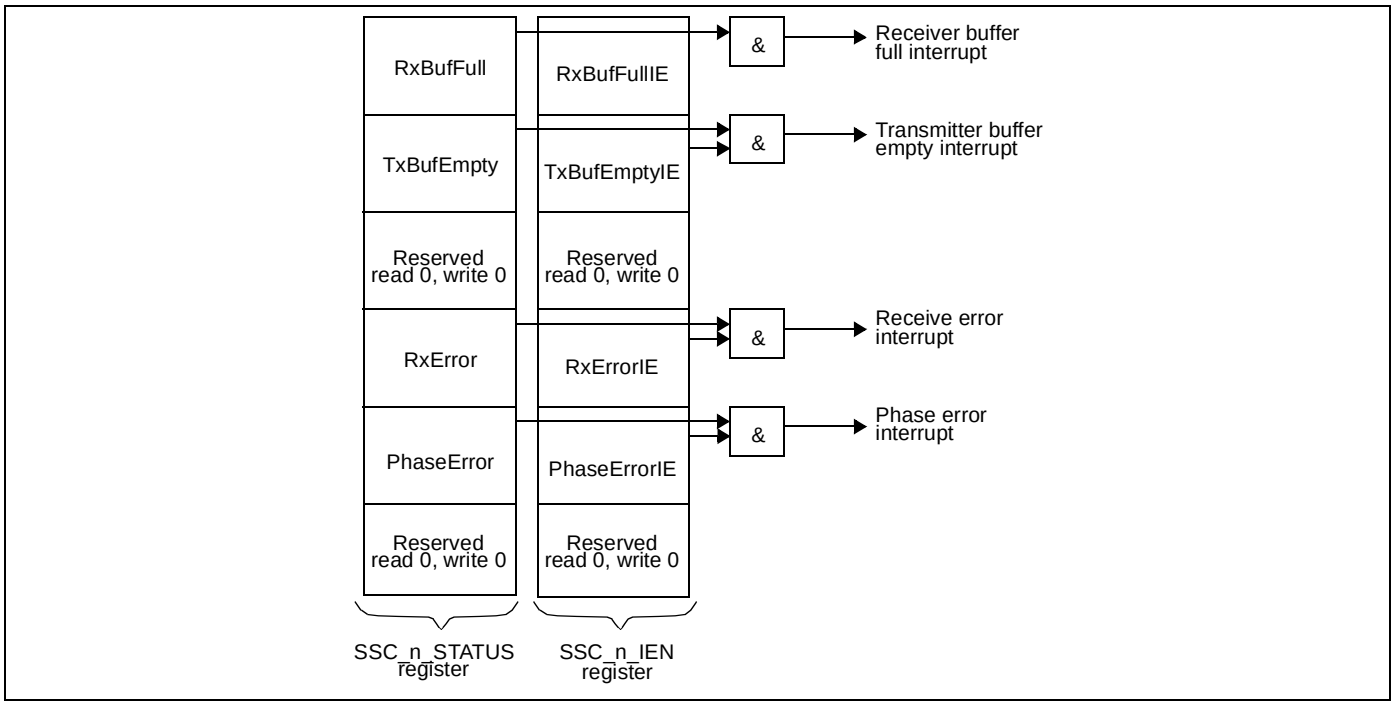


Figure 191 SSC status and interrupt registers

An interrupt handler for the SSC must read the SCC_n_STATUS register *before* writing the SCC_n_TBUF *or* reading the SCC_n_RBUF, as there might have been an error. The error flags are cleared by these read or write operations.

31.9 I²C hardware configuration

In order to reduce the load on the CPU, the hardware configuration of the I²C interface can be used. This is selected by setting register bit SSC_n_I2C.I2CM=1. In this configuration, start, stop and acknowledge are handled automatically by the hardware. When bit I2CM=1, register SSC_n_I2C is used in the following way:

- To generate a start condition set bit STRTG=1 and then load register SSC_n_TBUF with the data to be transmitted; transmission begins automatically when this register is loaded. You must reset STRTG after the device address is sent
- To generate a stop condition in order to terminate the transmission, set bit STOPG=1. Reset STOPG after the stop condition is sent.
- To generate an acknowledge set ACKG=1.

32 Parallel input/output port

44 bits of parallel I/O are configured in 6 ports, and each bit is programmable as output or input. The output can be configured as a totem-pole or open-drain driver. The input compare logic can generate an interrupt on any change of any input bit. Many parallel IO have alternate functions and can be connected to an internal peripheral signal such as a UART or SSC.

The PIO ports can be controlled by registers, mapped into the device address space. The registers for each port are grouped in a 4 Kbyte block, with the base of the block for port *n* at the address *PIOnBaseAddress*. During reset all of the registers are reset to zero.

Each eight-bit PIO port has a set of eight-bit registers. Each of the eight bits of each register refers to the corresponding pin in the corresponding port. These registers hold:

- The output data for the port (PIO_PnOut).
- The input data read from the pin (PIO_PnIn).
- PIO bit configuration registers (PIO_PnC0-2).
- The two input compare function registers (PIO_PnComp and PIO_PnMask).

Each of the registers, except PIO_PnIn, is mapped onto two additional addresses so that bits can be set or cleared individually.

- PIO_Set_ registers set bits individually; writing a ‘1’ in these registers sets a corresponding bit in an associated register, a ‘0’ leaves the bit unchanged.
- PIO_Clear_ registers clear bits individually; writing a ‘1’ in these registers resets a corresponding bit in an associated register, a ‘0’ leaves the bit unchanged.

The PIO5[3] input is inversed for the PIO and UHF input functions, but not inverted for the SDAV functions.

33 Modem analog front-end interface

33.1 Overview

Modem Analog Front-end Interface (MAFEIF) is an integrated interface to a Modem Analog Front End (AFE) such as the STLC7550.

In this chapter, the term “sample” is a 16-bit data-object that is transferred to ***or*** from the modem through the MAFEIF, and the term “sample period” is the time from the start of one sample to the start of the next.

The MAFEIF simultaneously transmits samples into and out of the AFE. It typically operates at a rate of 9600 samples/second, giving a typical sample period of 100µs. That is, every 100µs, one sample is transmitted and another received through the MAFEIF.

The MAFEIF receives its system clock signal (SClk) from the AFE. The SClk frequency is typically 256 ticks/sample period, or 2.56 MHz. The first 16 ticks of the 256 tick sample period are used to exchange a 16-bit sample pair (1 bit per tick).

The MAFEIF uses one DMA to transfer samples from a transmit memory buffer to the AFE, and simultaneously uses a second DMA to receive samples from the AFE and write them into the receive memory buffer. The software driver is “woken-up” every time a simultaneous transfer is completed - that is, every time a transmit memory buffer has been emptied and a receive memory buffer has been filled. For example, if each memory buffer contains 100 samples, the software is “woken up” every (100 x 100µs) 10ms. This is more stringent for handshake signals where the buffer-size could be as low as a few samples, e.g. 4.

The software modem has two pairs of pointers (i.e. four pointers) that point to two pairs of transmit/receive buffers. The modem and the MAFEIF alternately switch between the two pairs of pointers. While the MAFEIF transmits and receives using one pair of buffers, the software modem processes the information in the other pair. Using the above example for a buffer containing 100 samples, the software has 10ms to wake-up and then process one pair of transmit/receive buffers before they are required again by the MAFEIF.

33.2 Using the MAFEIF to connect to a modem

The following table lists the pins that are by the MAFEIF to connect a modem:

Name	Pin #	Type	MAFEIF function name (alt)	MAFEIF function description
PIO2[1]	205	O	MAFEIF_DOUT/PARA_REQ	Line for serially transmitting samples to the AFE.
PIO2[2]	206	O	MAFEIF_HC1	Indicates to the AFE that a control/status exchange will take place.
PIO3[0]	6	I	MAFEIF_SCLK/ PARA_DATA{0]	Modem system clock. The frequency should be less than half of the device system clock
PIO3[1]	7	I	MAFEIF_DIN/PARA_DATA[1]	Line for serially receiving samples from the AFE.
PIO3[2]	8	I	MAFEIF_FSI/PARA_DATA[2]	Signal from the AFE indicating the start of a sampling period. This is latched on falling edges of Sclk. For normal operation it should not remain high for more than 16 Sclk cycles, and there should be at least 20 Sclk ticks between consecutive rising edges of Fs.

Table 116 MAFEIF pins

33.3 Software

The MAFEIF software manages the data exchange between the software modem and MAFEIF, and handles the control/status exchange.

33.3.1 Data exchange

When the MAFEIF exchanges data, the software:

1 disables all interrupts;

2 sets the buffer size, e.g. 100 samples;
(For handshake response times, the buffer-size could be as low as a few samples, e.g. 4.)

3 sets-up both pairs of memory pointers in the MAFEIF (this will probably not be changed again);

4 enables status (complete) interrupt;

5 sets the control (run) bit;

6 deschedules.

The MAFEIF then processes a buffer-load of samples (that is, it transmits 100 samples and receives 100 samples). When this is complete, the MAFEIF sets the status (complete) bit, causing the software to be “woken-up”. The software then continues as follows:

7 processes the receive memory buffer and fills the next transmit memory;

8 confirms that there has been no overflow (i.e. failure to finish the software processing of a buffer before that buffer has started to be overwritten again);

9 confirms that there have been no memory latency problems during the exchange of the previous buffer, by reading the status(missed) bit;

10 if everything is OK, SW writes to the acknowledge register and deschedules.

33.3.2 Control/status exchange

For a control/status exchange, the software writes to the MOD_CONTROL register to enable the status interrupt (ctrl_empty), and then deschedules.

When the software “wakes-up”, it reads the modem status and disables the status interrupt (ctrl_empty) again.

34 Infrared transmitter/receiver

34.1 Introduction

The IR transmitter/receiver is an ST20 peripheral. For each symbol transmitted, the SW driver determines the symbol period and the symbol on-time of the IR pulse, and transfers these parameters into a 4-word deep FIFO. The IR transmitter/receiver then generates coded symbols using an internally generated subcarrier clock.

The parameters **symbol period** and **symbol on-time** are illustrated in the figure below.

The incoming signal must be detected, and the subcarrier must be suppressed, externally. Only the symbol envelope can be used by the IR and UHF processors. It is sampled at 10 MHz and the sample values are transferred into the input buffer in microseconds.

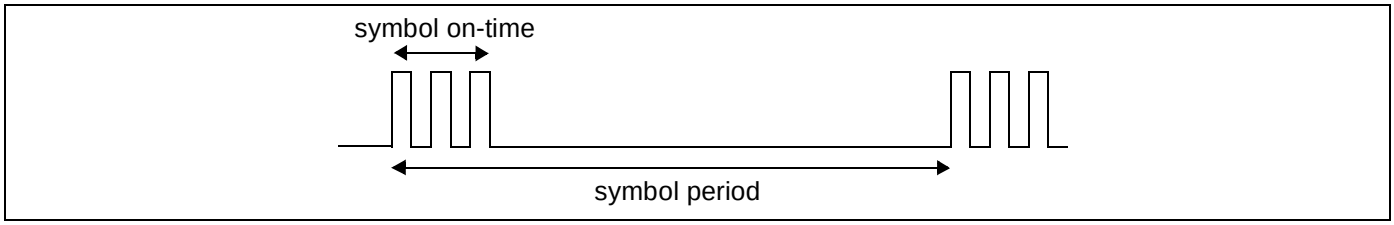


Figure 192 IR transmitter/receiver symbol

34.2 Functional description

Overview

The IR transmitter/receiver transmits infrared(IR)-data and receives both IR- and UHF-data. The IR and UHF receivers are independent and identical, except that the IR receiver does not use the noise filter. Both receivers are simultaneously active. The IR transmitter/receiver supports RC (remote control) codes only.

Figure 193 shows the IR transmitter/receiver block diagram in a typical circuit configuration with input demodulating and output buffering (open drain).

In the transmitter there are two programmable dividers to generate the prescaled clock and the subcarrier clock. The subcarrier clock sets the resolution for the transmitted data. Both receivers contain a sampling-rate clock, which samples the incoming data, and is programmed to 10 MHz.

FIFOs buffer both the transmitter output and the receivers' inputs to avoid timing problems with the CPU. Interrupts can be set on the FIFOs' levels to prevent input data overrun and output data under-run.

The two receivers each have one input pin, and the transmitter has two output pins (one driven directly and the other inverted as open drain).

There are two 4-word FIFOs in the RC transmitter and two in each RC receiver. The fourth element in each 4-word FIFOs is used internally and is not accessible to the ST-20 bus. Therefore, the 4-word FIFO is empty when there are three empty words and full when it contains three words. At all times, the fullness level of the 4-word FIFO is given in the corresponding status register, as described later.

The FIFO pair, "symbol period" and "symbol on-time", in each sub-module must be treated as a set and must be consecutively accessed for read or for write. They share a common pointer which is incremented only when they have

been accessed correctly. Repeated reads on one FIFO will always give the same data, and repeated writes will always over-write the previous data.

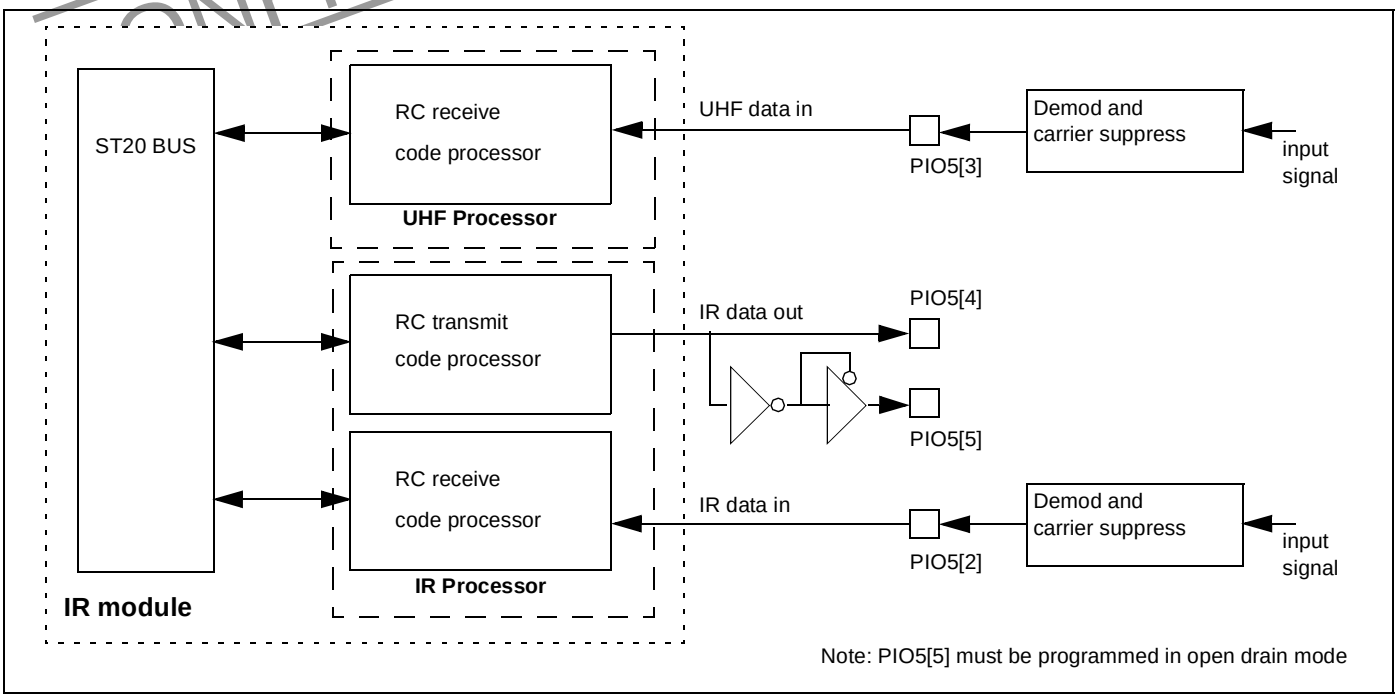


Figure 193 IR transmitter/receiver block diagram and implementation

RC transmit code processor

RC codes are generated by programming the transmit frequency and writing the symbol information into a FIFO. The FIFO is then read internally and the data processed to provide a serial PWM data stream. The transmit interrupt is set on a pre-selected FIFO level. An interrupt and a flag in the status register indicate an under-run condition (i.e. an empty FIFO). RC data transmission is disabled by setting bit 0 of register 'IRB_TX_EN_IR to "0".

The transmit interrupt is set by register IRB_TX_INT_EN_IR, on one of three FIFO levels:

- when three words are empty (buffer is empty);
- when two or more words are empty (buffer is half full);
- when at least one word is empty.

The transmit interrupt is cleared automatically when new data is written to the registers IRB_TX_SYM_PERIOD_IR and IRB_TX_ON_TIME_IR. Register bits IRB_TX_INT_STATUS_IR[5:4] give the FIFO's fullness status.

The frequency of the sub-carrier is set by programming the registers 'IRB_TX_PRE_SCALER_IR and 'IRB_TX_SUB_CARRIER_IR'.

The symbol period, in sub-carrier cycles, is programmed in the register IRB_TX_SYM_PERIOD_IR and the on-time of the IR pulse is written to the register 'IRB_TX_ON_TIME_IR. These two registers are four-word FIFOs. They must be programmed sequentially as a pair to increment the write-pointer and be ready for the next data. Transmission is enabled by setting register 'IRB_TX_EN_IR bit 0 to "1". If new data is not written before the last symbol in the buffer is transmitted, no RC codes are generated. The output is driven to logic "0" and the register IRB_TX_INT_STATUS_IR bit 1 is set.

Before data can be transmitted, the under-run condition must be cleared as follows:

- Disable the transmission by writing "0" to register IRB_TX_EN_IR.

- Load at least one block of data into IRB_TX_SYM_PERIOD_IR and IRB_TX_ON_TIME_IR.
- Clear the “Tx_UnderRun” status bit by writing “1” to register IRB_TX_CLR_UNDERRUN_IR

Transmission is resumed by writing “1” to register IRB_TX_EN_IR.

RC receive code processor

This section describes the UHF-data and the IR-data receivers. They are independent and identical except that the noise suppression filter is programmable in the UHF receiver, and is not used in the IR receiver. The 10 MHz sampling clock is common to both receivers and is set by register IRB_RX_SAMPLING_RATE_COMMON. This register is programmed with the value 5 for a 50 MHz IRB system clock, or with the value 6 for a 60 MHz clock.

Each receiver processes the incoming RC code symbol envelope and stores the values “symbol period” and “symbol on-time” (in microseconds) in a four-word FIFO buffer, until the data can be read by the microcontroller.

The receive interrupt is set by register IRB_RX_INT_EN to one of the following three FIFO levels:

- at least one word is available to be read;
- two or more words or more are available to be read (FIFO half full);
- three words are available to be read (FIFO full).

The interrupt is cleared automatically when the registers IRB_RX_SYM_PERIOD and IRB_RX_ON_TIME have been read. They must be read consecutively, as a pair, to increment the FIFO read pointer. The register IRB_RX_INT_STATUS bits 4 and 5 give the fullness level of the FIFO.

If the FIFO is full and has not been read before the arrival of new data, then this data is lost and a receive overrun flag is set in the status register IRB_RX_INT_STATUS. No new data is written to the FIFO while this condition exists. To reset the overrun flag the following operations must be performed:

- Read at least one word from each of the receive FIFO registers, IRB_RX_SYM_PERIOD and IRB_RX_ON_TIME.
- Clear the RxOverRunStatus bit by writing 0x01 to register IRB_RX_CLR_OVERRUN.

The last symbol is detected using a time-out condition whose value is stored in microseconds in register IRB_RX_MAX_SYM_PERIOD. If no pulse has been received during this time then the last word in the FIFO IRB_RX_SYM_PERIOD has a value 0xFFFF. If the value of register IRB_RX_INT_EN bit 1 (LastSymbolIrqEnable bit), is “1”, then an interrupt is triggered and the status register IRB_RX_INT_STATUS bit 1 is set. The interrupt and its status bit are cleared automatically when the last value in the FIFO has been read.

When register IRB_RX_INT_EN bit 0 is set to “0” then both the FIFO level interrupt and the last symbol interrupt are inhibited.

RC data reception can be disabled by setting register IRB_RX_EN bit 0 to “0”. However, both receivers are normally always enabled.

Noise suppression filter

This filter is turned off in the IR receiver and is programmable in the UHF receiver using register IRB_RX_NOISE_SUPPRESS_WIDTH_UHF. Any pulses, either high or low, having a value in microseconds of less than the programmed width, are assumed to be noise and, therefore, suppressed.

The noise suppression filter can be disabled by writing “0x00” to register IRB_RX_NOISE_SUPPRESS_WIDTH_UHF.

35 Electrical specifications

35.1 Absolute maximum ratings

Maximum limits indicate where permanent device damage occurs. Continuous operation at these limits is not intended and should be limited to those conditions specified in *DC electrical characteristics*.

Symbol	Parameter	Min.	Max.	Units
VDD3_3	Power Supply (pads)	-0.5	4	V
VDD2_5	Power Supply (core)	-0.5	3	V
VDD_RGB, VDD_YCC, VDD_PLL, VDD_PCM	Power Supply	-0.5	4	V
VI, VO	Voltage on input and output pins	-0.5	4	V
T_{stg}	Storage Temperature	-65	+150	ºC
T_{oper}	Ambient Operating Temperature	0	+70	ºC

Table 117 Absolute maximum ratings

35.2 DC electrical characteristics

35.2.1 Static

Operating conditions: VDD3_3 = 3.3V ±0.3V, VDD2_5 = 2.5V ±0.25V, T_{amb} = 0 to 70ºC unless otherwise specified.

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Units	Notes
VDD3_3	Operating voltage		3.0	3.3	3.6	V	
VDD2_5	Operating voltage		2.25	2.5	2.75	V	
V_{IL}	Input Logic Low Voltage		-0.3		+0.8	V	
V_{IH}	Input Logic High Voltage		2.0		3.6	V	
I_I I_{OZ}	Input leakage Current Inputs Outputs		 -10 -10		 +10 +10	 µA µA	
V_{OL}	Output Logic Low Voltage				0.4	V	
V_{OH}	Output Logic High Voltage		2.4			V	
C_{IN}	Input Capacitance				10	pF	
I_{DDA}	Analog Current Consumption R_{I_REF} = 16.9KΩ, R_L = 200Ω		20		50	mA	
10-bits D/A converter							
R_{I_REF}	Resistance for reference Current Source for 3 D/A Converters	$I_{\_REF} = V_{I\_REF}/R_{I\_REF}$		16.9		kΩ	
V_O	Output Voltage Dyn	R_{I_REF} = 16.9 KΩ, R_L = 274Ω, VDD2_5 = 2.5V	1.21	1.31	1.41	V_{PP}	
	DAC to DAC V_O max code (tri-DAC only)	R_{I_REF} = 16.9 KΩ, R_L = 274Ω, VDD2_5 = 2.5V	-5		+5	%	
Iout	DAC output current				5.0	mA	
Vout	DAC output voltage				1.45	V	

Table 118 DC electrical characteristics

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Units	Notes
ILE	LF Integral Non-linearity	R_{I_REF} = 16.9 KΩ, R_L = 274Ω, VDD2_5 = 2.5V	-2		+2	LSBs	
DLE	LF Differential Non-linearity	R_{I_REF} = 16.9 KΩ, R_L = 274Ω, VDD2_5 = 2.5V	-1		+1	LSBs	

Table 118 DC electrical characteristics

35.2.2 ST20 running at 60.75 MHz

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Units	Notes
VDD3_3	Operating voltage		3.0	3.3	3.6	V	
VDD2_5	Operating voltage		2.25	2.5	2.75	V	
IDD3_3	Average power supply current	ST20 operating frequency 60.75 MHz		60	150	mA	
IDD2_5	Average power supply current	ST20 operating frequency 60.75 MHz		650	750	mA	1

Table 119 Current consumption with ST20 running at 60.75 MHz

1. This figure includes the analogue current consumption.

35.2.3 ST20 running at 81.0 MHz

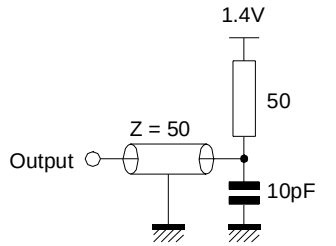
Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Units	Notes
VDD3_3	Operating voltage		3.15	3.3	3.45	V	
VDD2_5	Operating voltage		2.35	2.5	2.65	V	
IDD3_3	Average power supply current	ST20 operating frequency 81.0 MHz, VDD3_3 = 3.3V ±0.15V, VDD2_5 = 2.5V ±0.15V		60	150	mA	
IDD2_5	Average power supply current	ST20 operating frequency 81.0 MHz, VDD3_3 = 3.3V ±0.15V, VDD2_5 = 2.5V ±0.15V		710	800	mA	1

Table 120 Current consumption with ST20 running at 81.0 MHz

1. This figure includes the analogue current consumption.

35.3 AC test conditions

Test Conditions: $V_{DD}3\_3$ = 3.3V ± 0.3V, T_{amb} = 0 to 70ºC, unless otherwise specified.



AC Test Conditions (50Ω adapted)

Figure 194 AC test conditions

35.4 Operating conditions

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
C_{LD}	Load Capacitance per SMI pin (address, data and control)		15		pF	
C_{LA}	Load Capacitance per EMI pin (address, data and control)		30		pF	
C_{LP}	Load Capacitance per PIO Pin		30		pF	

Table 121 Operating conditions

35.5 Timing diagrams for IO interfaces

Timings, other than rise and fall times, are specified with respect to a threshold of 1.5V.

35.5.1 Input clock

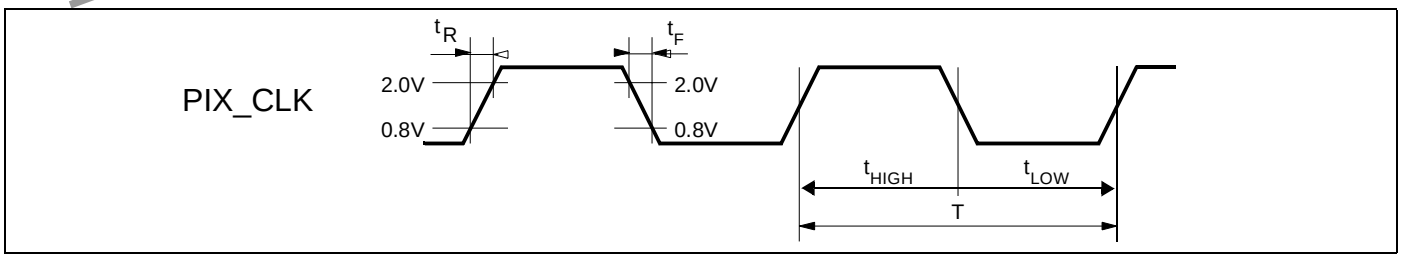


Figure 195 Input clock timing definitions

Symbol	Parameter	Min.	Typ.	Max.	Unit	Notes
T	PIX_CLK clock period (typically 27 MHz)		37.0		ns	
T_{HIGH}	Clock high time	16.5	18.5	20.5	ns	
T_{LOW}	Clock low time	16.5	18.5	20.5	ns	
T_R/T_F	Clock rise/fall time		1.0	5.0	ns	

Table 122 Input clock timing values

35.5.2 SMI interface

When the SMI interface is reading, the reference clock is SMI_CLKIN (rising edge), and when it is writing, the reference is SMI_CLKOUT (falling edge).

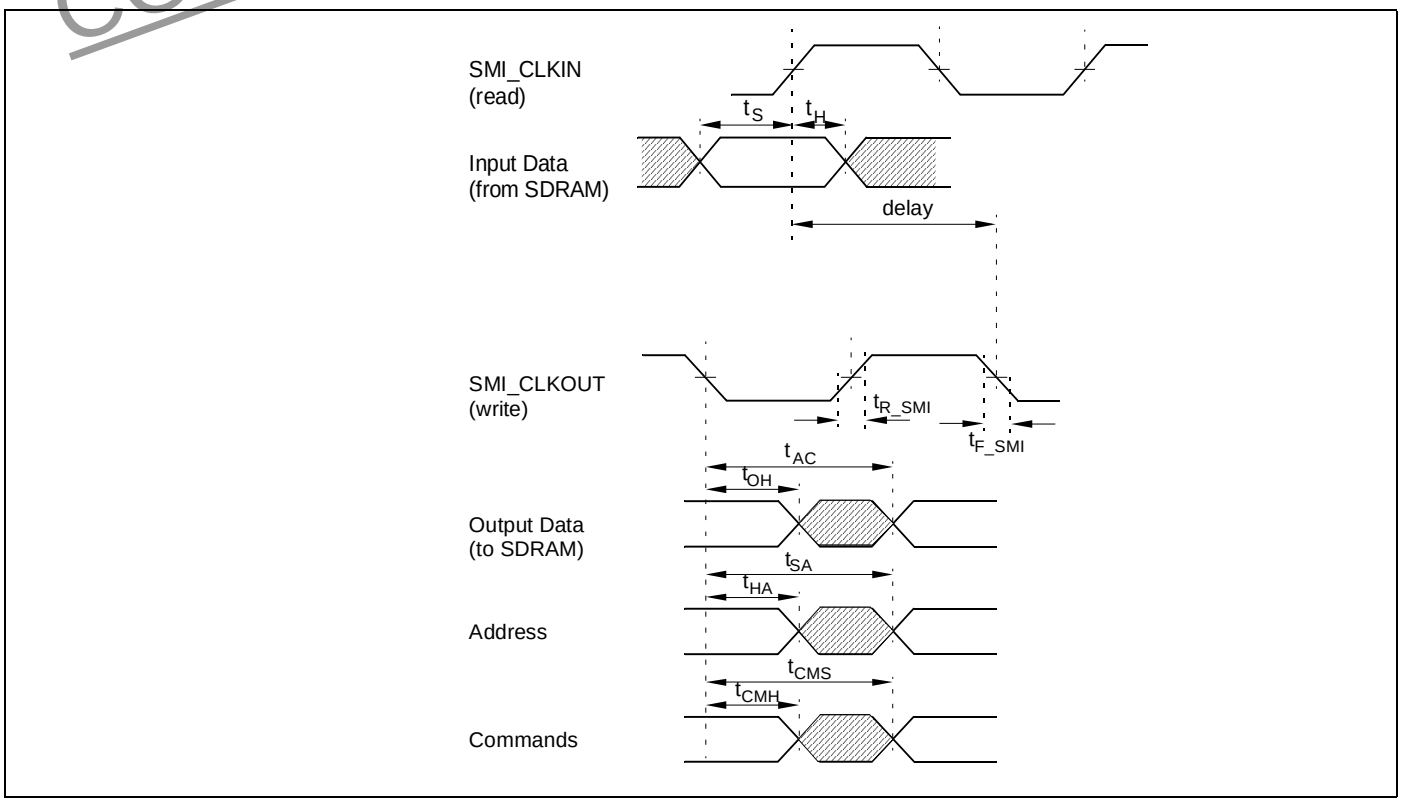


Figure 196 AC parameters of read & write (synchronous DRAM) timing definitions

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{R_SMI}	SMI_CLKOUT rise time			2	ns	1
t_{F_SMI}	SMI_CLKOUT fall time			2	ns	1
t_{CK}	Clock cycle time	7			ns	
t_S	Data input setup time	2			ns	
t_H	Data input hold time	2			ns	
t_{AC}	Output data access time			2.1	ns	2
t_{OH}	Output data hold time	-3.1			ns	2
t_{SA}	Address output delay time			2.1	ns	2
t_{HA}	Address output hold time	-3.1			ns	2
t_{CMS}	Command ($\overline{CS}$, $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$, DQM) delay time			2.1	ns	2
t_{CMH}	Command ($\overline{CS}$, $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$, DQM) hold time	-3.1			ns	2
t_{RCD}	Delay time ACTIVE to READ/WRITE command	4			T	
t_{RC}	REF to REF / ACTIVE Command Period	8			T	

Table 123 SMI interface timing values

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{RP}	ACTIVE to PRE Command Period	3			T	
t_{RRD}	ACTIVE(A) to ACTIVE(B) command period	4			T	
t_{DAL}	Data out to ACTIVE command period	5			T	
t_{DPL}	Data out to PRECHARGE command period	2			T	
t_{RAS}	ACTIVE to PRECHARGE command period	9			T	

Table 123 SMI interface timing values

1. Test conditions: C_{load} = 10pF, 50Ω adapted mode at 1.4V, edge measured at 20%-80%.
2. Negative values indicate that the timing is "before" the falling edge of SMI_CLKOUT.

The output parameter definitions in Figure 196, are relative to the falling edge of SMI_CLKOUT. Care must, therefore, be taken when interpreting the SDRAM data sheet which might have timings relative to the clock rising edge.

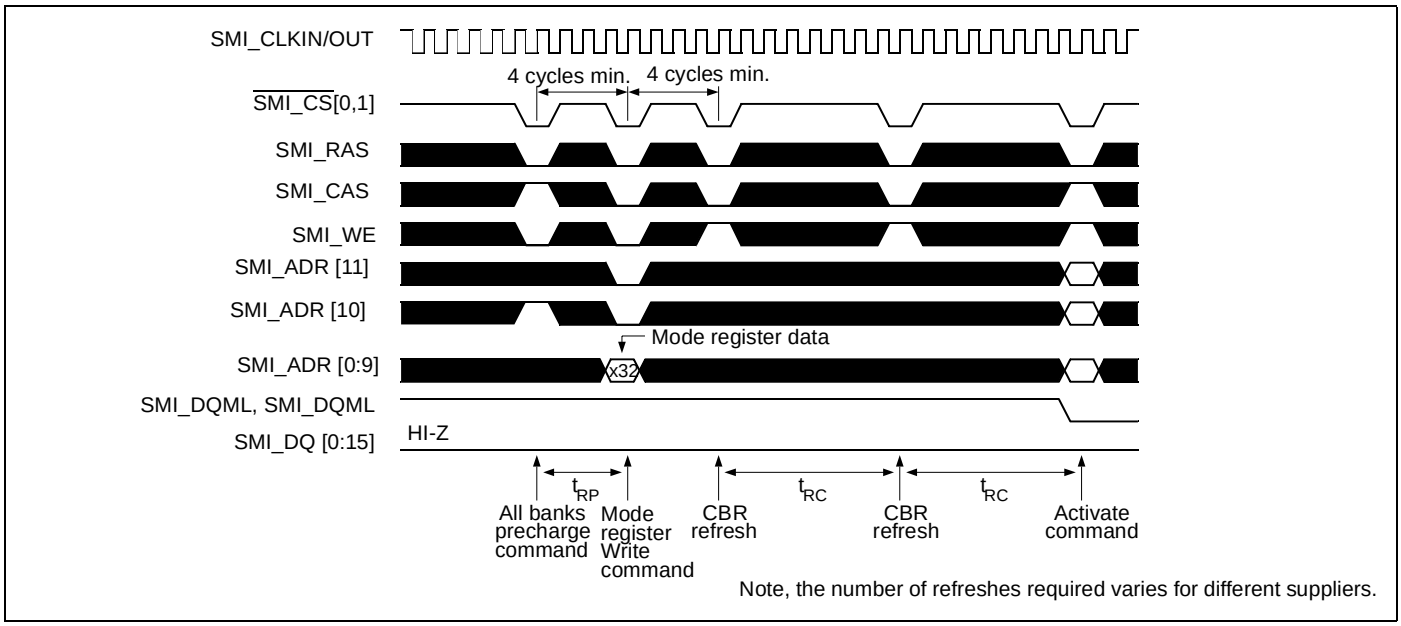


Figure 197 Synchronous DRAM power-on sequence timing definitions

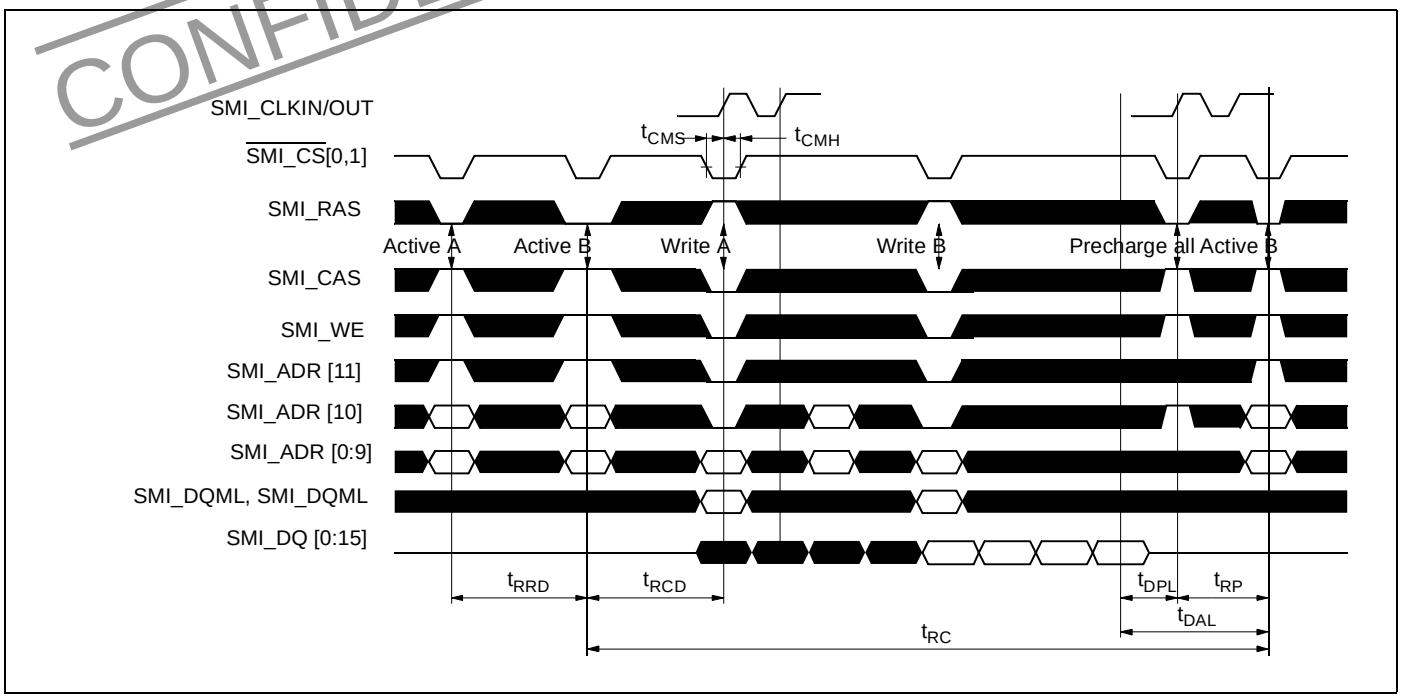


Figure 198 Synchronous DRAM write burst (Burst Length = 4 CAS Latency = 3) timing definitions

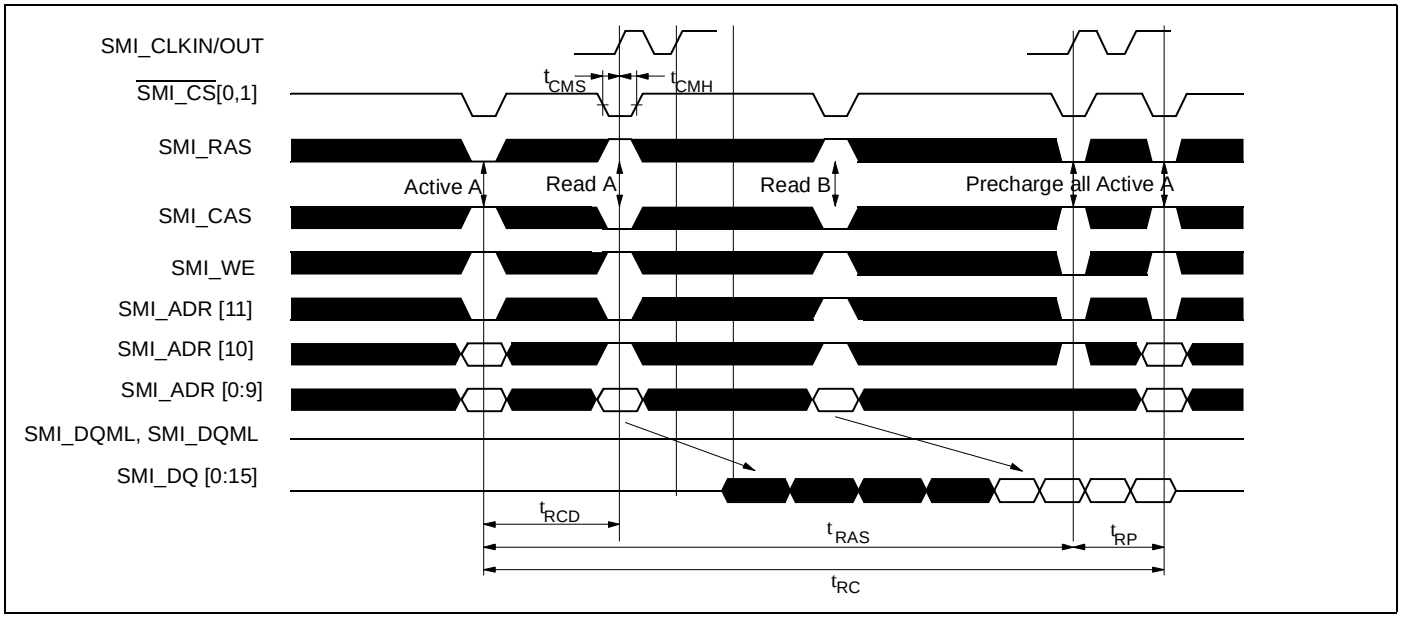


Figure 199 Synchronous DRAM read (burst length = 4 CAS latency = 3) timing definitions

35.5.3 Video interface

The video timings given in the table below are referenced to the rising edge (unless otherwise indicated) of the external clock PIX_CLK.

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
$t_{SYCKenot0}$	ODDEVEN setup time	4.0			ns	
$t_{SYCKsync}$	HSYNC setup time	4.0			ns	
$t_{CKSYenot0}$	ODDEVEN hold time	4.0			ns	
$t_{CKSYsync}$	HSYNC hold time	4.0			ns	
t_{CKPV}	YC7-YC0 output delay time			15	ns	

Table 124 Video interface timing values

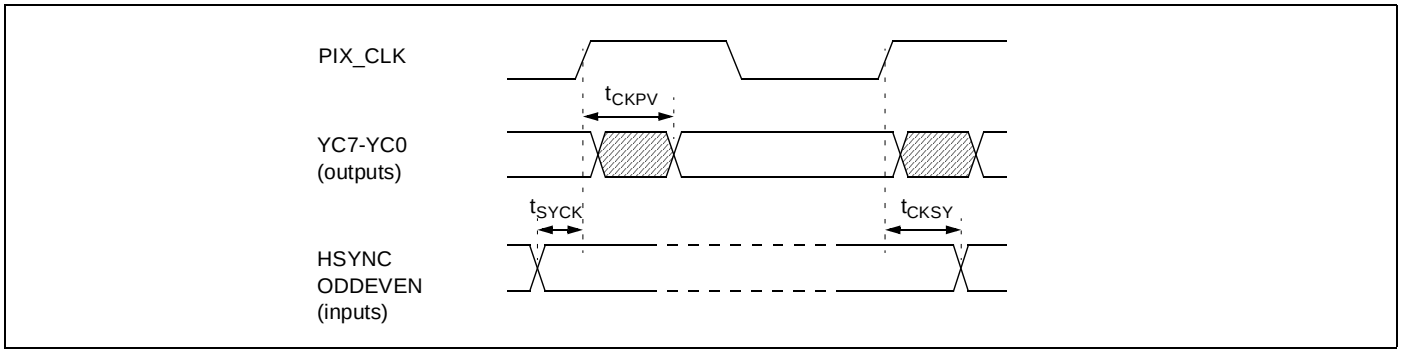


Figure 200 Video interface timing definitions

35.5.4 EMI interface

There are 2 EMI modes:

- mode *no SDRAM* (register bit EMI_CONFIGPADLOGIC[11]=0):

  the reference is clock CPU_PROCLK rising edge;

- mode *SDRAM* (register bit EMI_CONFIGPADLOGIC[11]=1):

  outputs (CPU_ADR, CPU_DATA, CPU_RW and commands) are driven by clock CPU_PROCLK falling edge and inputs (memory read data) are latched with clock CPU_PROCLK rising edge;

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{CHAV}	CPU_ADDR access time	-4.0		4.0	ns	
t_{CLSV}	Strobe output delay time (from CPU_PROCLK falling)	-4.0		4.0	ns	
t_{CHSV}	Strobe output delay time	-4.0		4.0	ns	
t_{RDVCH}	Read CPU_DATA setup time	5.0			ns	
t_{CHRDX}	Read CPU_DATA hold time	0			ns	
t_{CHWDV}	Write CPU_DATA output delay time	0.0		4.0	ns	
t_{CHRSV}	"Remaining strobes" output delay time	-1.0		4.0	ns	
t_{WVCH}	CPU_WAIT setup time	5.0			ns	
t_{CHWX}	CPU_WAIT hold time	0.0			ns	

Table 125 EMI interface timing values

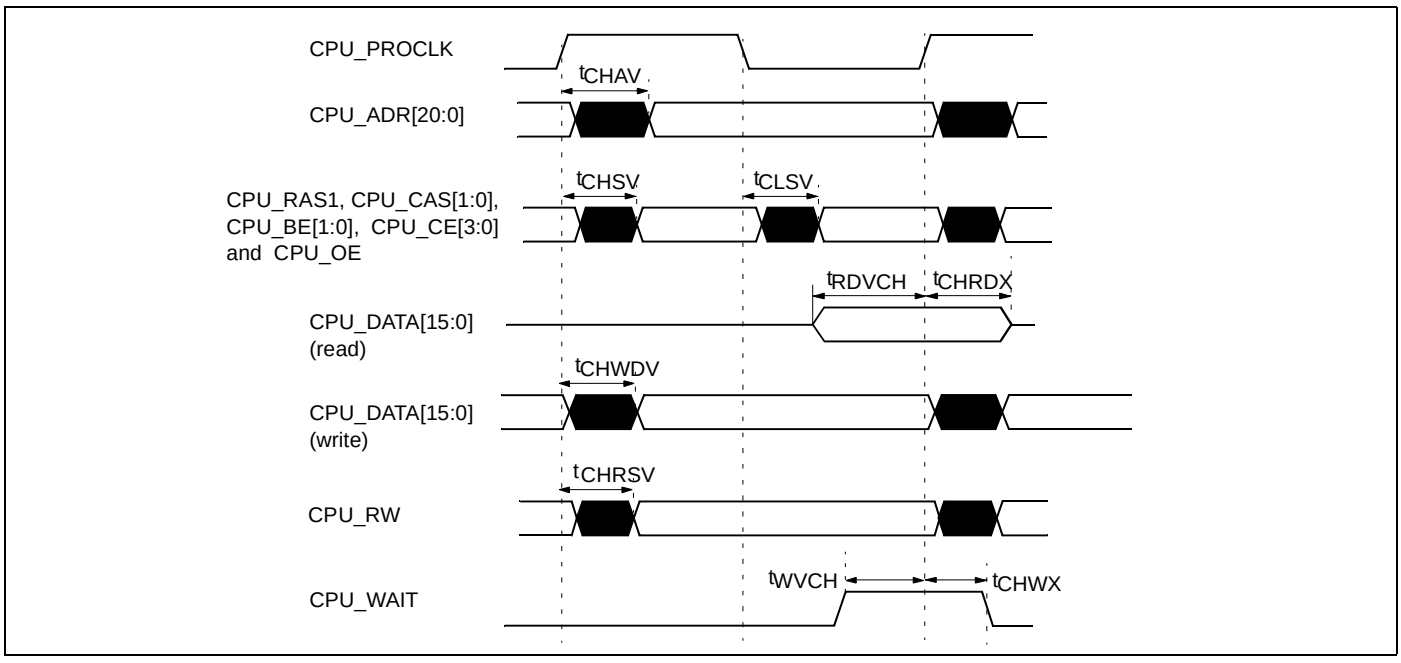


Figure 201 EMI interface timing definitions for mode *no SDRAM*

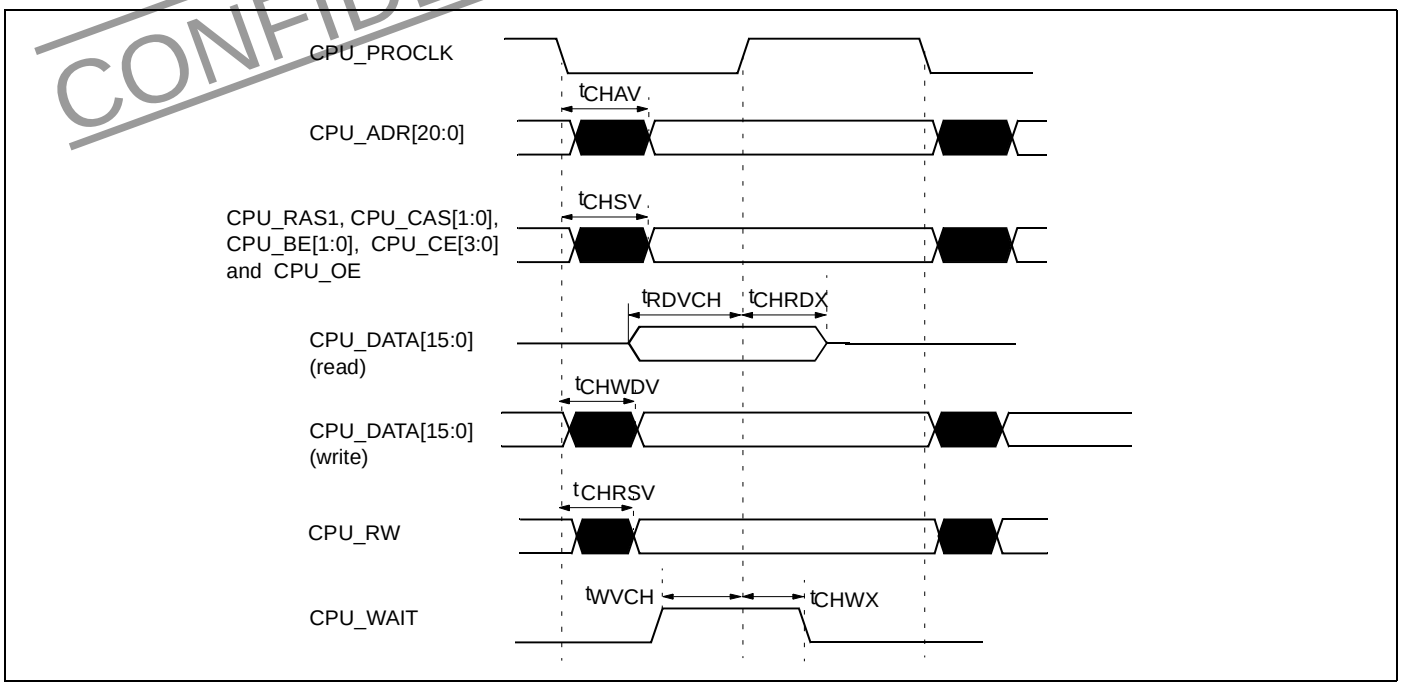


Figure 202 EMI interface timing definitions for mode *SDRAM*

35.5.5 TAP interface

Symbol	**Parameter** (default reference is TCK rising edge)	**Min.**	**Typ.**	**Max.**	**Units**	**Notes**
t_{TIVTCH}	Input setup time	4			ns	
t_{TCHTIX}	Input hold time	4			ns	
t_{TCHTOV}	Output delay time (reference TCK falling edge)			15	ns	

Table 126 Tap timing values

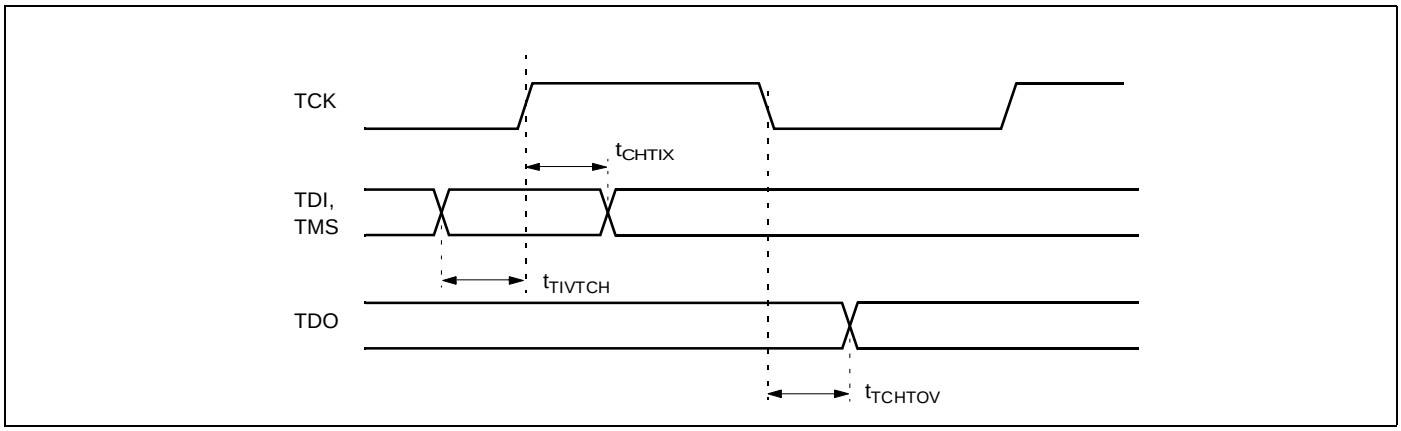


Figure 203 TAP timing definitions

35.5.6 Link interface

Symbol	Parameter (default reference is B_BCLK falling edge)	Min.	Typ.	Max.	Units	Notes
t_{LDVLCH}	Input setup time	2			ns	
t_{LCHLDX}	Input hold time	2			ns	

Table 127 Link interface timing values

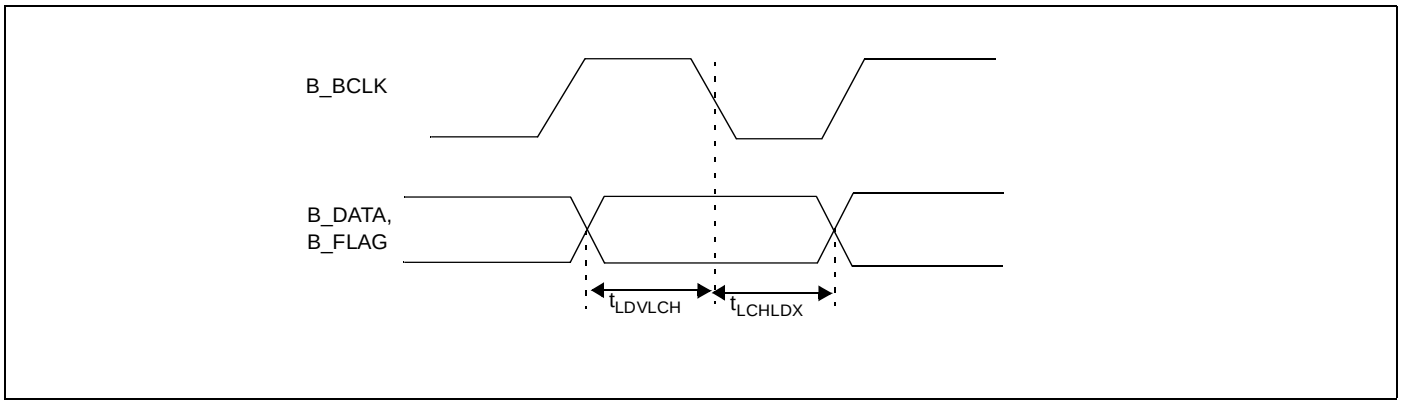


Figure 204 Link interface timing definitions

35.5.7 I^2S interface

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
$t_{I2SSETUP}$	Input setup time	4.5			ns	
$t_{I2SHOLD}$	Input hold time	1			ns	

Table 128 I^2S interface timing values

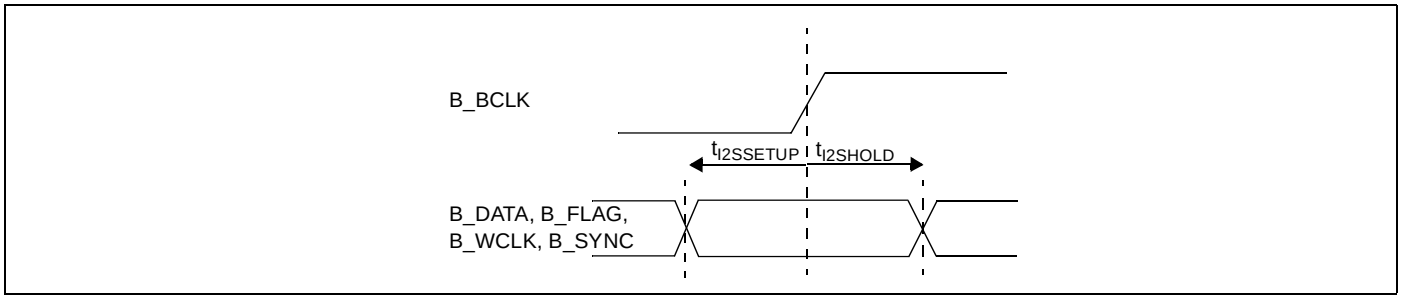


Figure 205 I^2S interface timing definitions

35.5.8 Parallel interface

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
Tsetup	Input data setup time	3			ns	
Thold	Input data hold time	8			ns	
Treq_to_str	Request to rising input strobe time	0			ns	
Tstr_to_req	Rising input strobe to request inactive time	5		35	ns	

Table 129 Parallel interface timing values

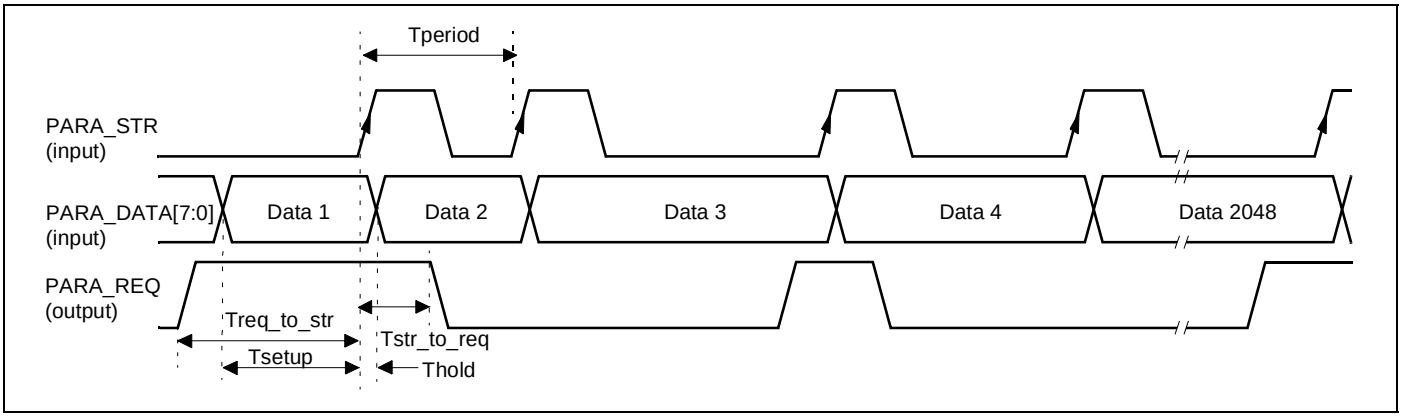


Figure 206 Parallel interface timing definitions

35.5.9 Audio interface

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{SCLPD}	SCLK falling edge to data valid			10	ns	
t_{SCLLR}	SCLK falling edge to LRCLK hold time			10	ns	

Table 130 Audio timing values

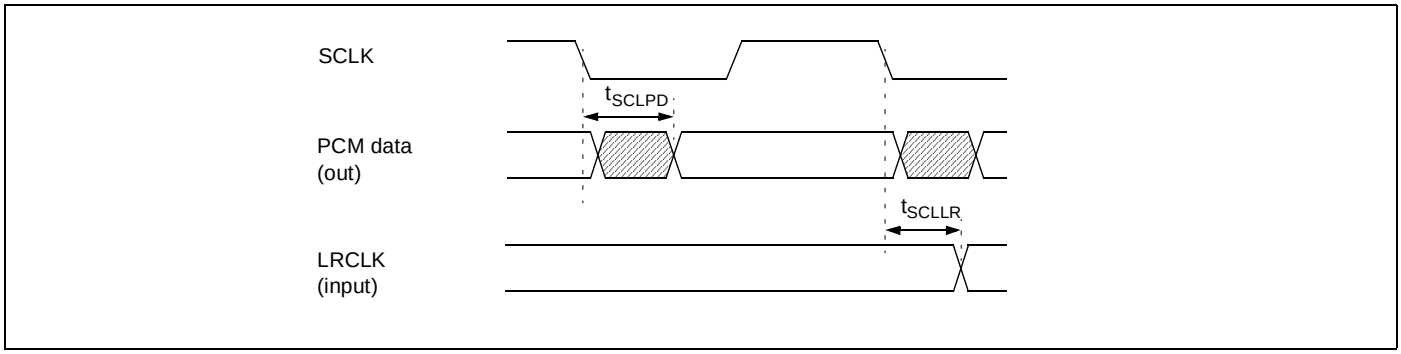


Figure 207 Audio timing definitions

35.5.10 ATAPI interface

Symbol	**Parameter** (default reference is DIO rising edge)	Min.	Typ.	Max.	Units	Notes
$t_{AD\_TO\_DIOW}$	Address setup time (reference is DIOW falling edge)	30		240	ns	
$t_{AD\_TO\_DIOR}$	Address setup time (reference is DIOR falling edge)	30		240	ns	
$t_{DIO\_TO\_AD}$	Address hold time	10		220	ns	
t_{D_SETUP}	Data in setup time	15			ns	
t_{D_HOLD}	Data in hold time	5			ns	
t_{D_VALID}	Data output delay time	15			ns	

Table 131 ATAPI interface timing values

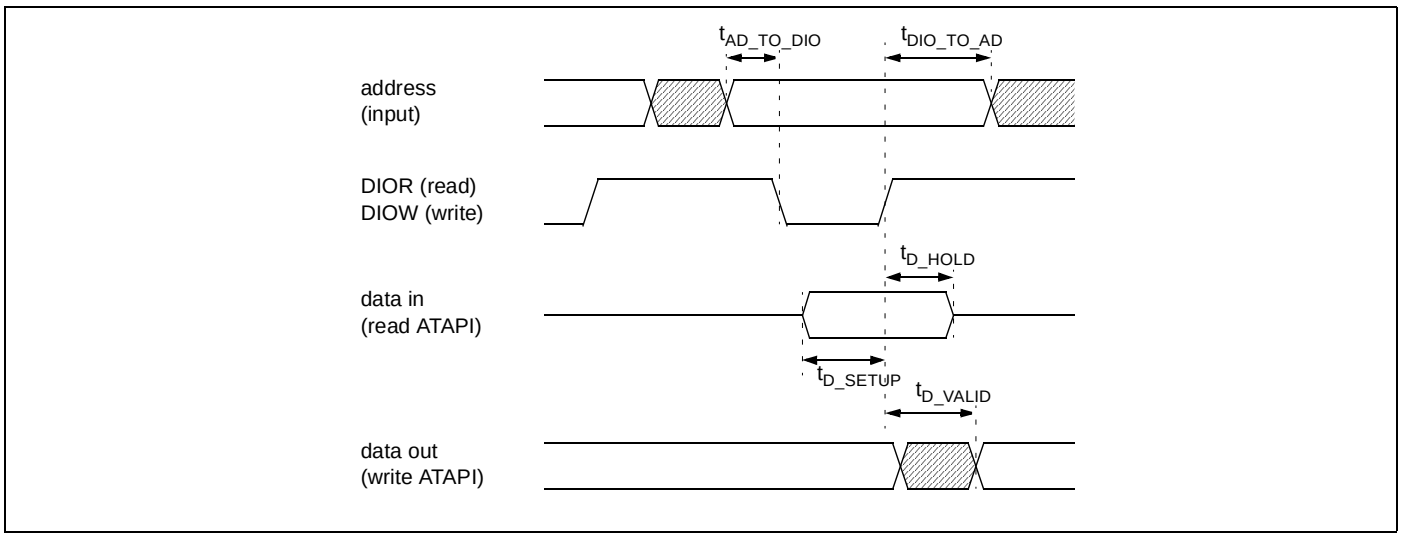


Figure 208 ATAPI interface timing definitions

36 Package mechanical data

TheSTi5518 is packaged in a 208-pin Plastic Quad Flat Pack

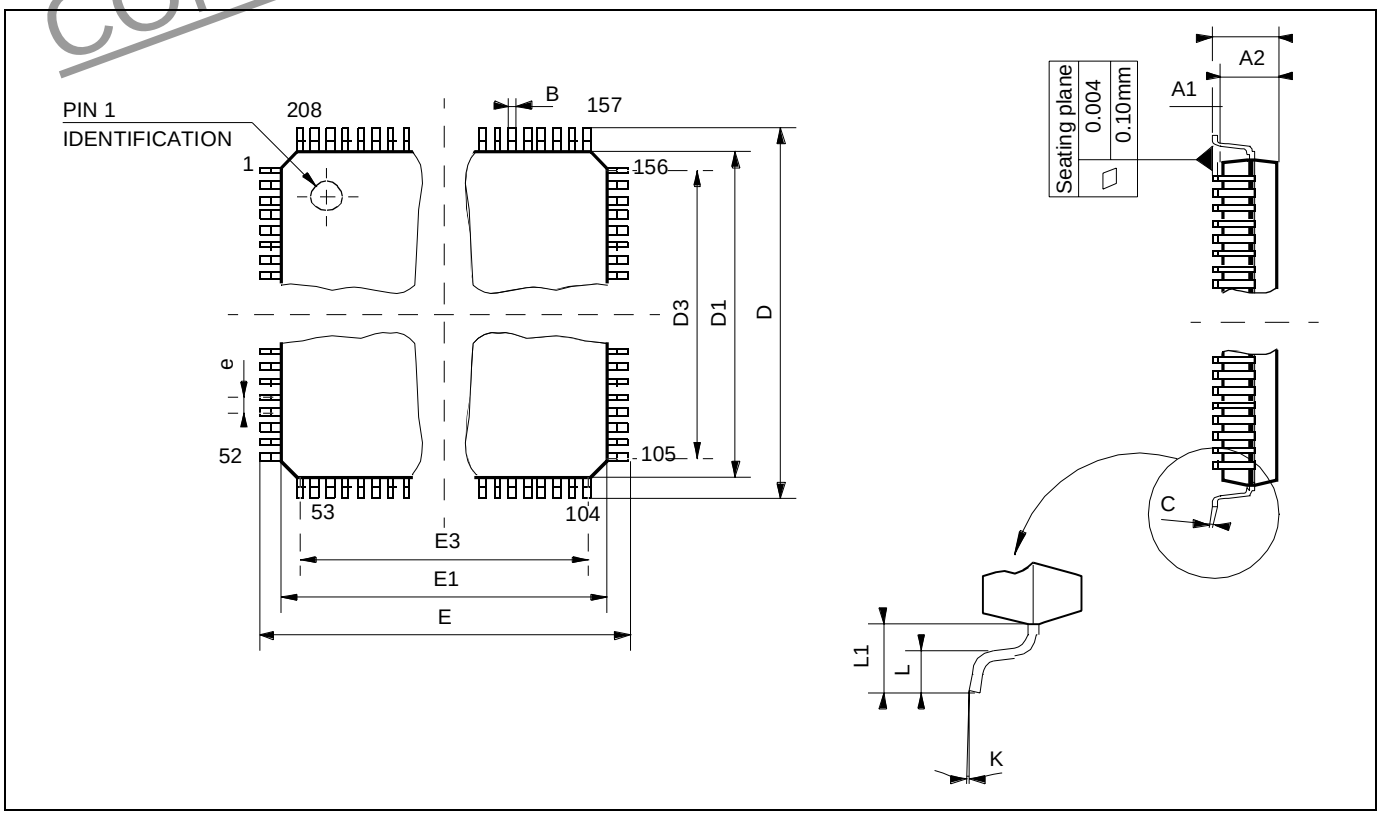


Figure 209 PQFP208 schematic

Dimensions	Millimeters			Inches		
	minimum	typical	maximum	minimum	typical	maximum
A			4.10			0.0161
A1	0.25			0.010		
A2	3.20	3.40	3.60	0.126	0.134	0.142
B	0.17		0.27	0.007		0.011
C	0.09		0.20	0.004		0.008
D		30.60			1.205	
D1		28.00			1.102	
D3		25.50			1.004	
e		0.50			0.020	
E		30.60			1.205	
E1		28.00			1.102	
E3		25.50			1.004	
L	0.45	0.60	0.75	0.018	0.024	0.030
L1		1.30			0.051	
K	0° (Min.), 7° (Max.)					

Table 132 PQFP208 dimensions

37 Revision history

37.1 Changes for rev D

The corrections made to STi5518 datasheet rev B to create rev D are given in the table below. Rev C was an internal revision control, and no changes were made to the document

Change	Description
Chapter 2: *Pin data* on page 15	Pin (118) name CPU_RAM_CLK is replaced by CPU_PROCLK (previously, both these names were used for pin 118).
Section 35.5.4: *EMI interface* on page 286	
Table 37: *List of strobes used for all EMI configurations on page 68*	Clarified peripheral/DRAM combinations in bank configuration column.
Chapter 11: *Test access port* on page 88	Added register for silicon cut identification.
Section 13.3: *DVB-CI mode (optional)* on page 94	New section (optional) for set-top box applications.
Section 13.5: *ATAPI interface* on page 97	Clarification of ATAPI drive speed.
Chapter 20: *Digital encoder* on page 187	Removed references containing "non-interlaced" (no longer required) and deleted register bit name nintrl.
Chapter 22: *Double triple video DAC* on page 216	Updated sentence (in para 3): Current-sources provide ....
Chapter 31: *Synchronous serial controller* on page 265	SSC can be master or slave; Added Section 31.9: *I2C hardware configuration* on page 271.
Chapter 35: *Electrical specifications* on page 278	New characterizations.

Table 133 Rev B to rev D changes

37.2 Changes for rev C

Internal revision control, no changes to document.

37.3 Changes for rev B

The corrections made to STi5518 datasheet rev A to create rev B are given in the table below.

Change	Description
Cover page	Revised text and modified features: CPU frequency 80 MHz, Macrovision (optional).
Section 1.4: *Audio decoder* on page 11	Dolby Digital, MPEG-1: added layer III to description.
Table 1: *Pins sorted by function on page 16* and Table 2: *Pins sorted by number on page 20*	Pin numbers 114 and 116 changed to I/O type. Added main and alternate functions to pins 1, 2, 3, 187, 188.
Table 1: *Pins sorted by function on page 16* and Table 2: *Pins sorted by number on page 20*	Alternate functions YC[7:0] are outputs.
Table 1: *Pins sorted by function on page 16* and Table 2: *Pins sorted by number on page 20*	Alternate functions for pins 16 - 19 updated.
Section 3.5: *Timers* on page 30	Erroneous name ProcClockOut replaced by CPU_PROCLK
Chapter 5: *Interrupt system* on page 45	corrected interrupt numbers in Table 34: *STi5518 Interrupt assignments on page 50*
Table 35: *STi5518 memory map on page 52*	Removed unused variable MPEGDMA2BaseAddress
Section 7.1: *External memory* on page 56	Updated shared SDRAM memory size.

Table 134 Rev A to rev B changes

Change	Description
Section 7.3: *Caching* on page 56	Updated complete section.
Sub-Section 8.3.2: *SDRAM* on page 75	Erroneous name ProcClockOut replaced by CPU_PROCLK
Table 41: *Default configuration on page 80*	EMI default device type changed to ’000’ (peripheral).
Chapter 10: *Diagnostic controller* on page 83	Erroneous names TriggerIn replaced by TRIGGER_IN, and TriggerOut by TRIGGER_OUT.
Chapter 11: *Test access port* on page 88	Removed text concerning "... for cut A0, the value is 0x20 ...".
Section 13.2: *Serial interface* on page 93	Updated signal names referring to serial bus.
Section 13.4: *Parallel interface* on page 95	Modified text and Figure 38 and Figure 39 on page 96.
Section 13.5: *ATAPI interface* on page 97	Added 5th. paragraph: ...accessed through bank 1 of CPU ....
Section 14.4: *Detailed description* on page 109	sub-Section 14.4.1: *Input interface*: serial bus names updated.
Section 14.4: *Detailed description* on page 109	sub-Section 14.4.2: *NRSS interface*: serial bus names updated.
Section 14.4, sub-section Not equal filtering on page 119	Added first paragraph: ...used only in DVB, not in DSS.
Section 14.6: *Hard disk drive buffer control* on page 132	Added this new section.
Chapter 19: *SDRAM block move* on page 186	Removed register USD_BSK from Table 96: *SDRAM block move registers on page 186*.
Section 20.4: *Master mode* on page 192	Added section.
Section 20.7: *Subcarrier generation* on page 199	Modified section.
Figure 154: *Double triple video DAC schematic* on page 216	Updated signal and block names.
Section 22.5: *Output-stage adaptation and amplification* on page 218	Modified section.
Figure 157: *Audio decoder block-diagram* on page 221	Added MP3 block to diagram.
Section 23.7, sub-section MP3 decoding mode on page 228	Removed references to 8 kHz and 11.025 kHz.
Figure 165: *MP3 decoding flow* on page 228	New diagram for MP3 flow.
Section 23.8: *PCM output* on page 229	Modified text in sub-section Output configurations on page 229concerning configuration 2 & 3 and associated diagram in Figure 166.
Section 23.9: *SPDIF output* on page 233: Overview	Added text regarding register AUD_PCMCONF.
Chapter 25: *Clock generator* on page 242	Deleted section "Audio clock frequency synthesizer".
Section 25.3: *PCM clock* on page 244	Updated text and table values.
Section 29.2: *SmartCard clock generator* on page 252	Clarified programming of register SCI_n_CLKVAL.
Figure 189: *Clock and data relationships* on page 267	Changed value of ClkPhase to 0 in figure and in text.
Chapter 35: *Electrical specifications* on page 278	New characterizations.
Table 118: *DC electrical characteristics on page 278*	Re-formatted (tables 117 & 118)and updated current consumption and V_{IH} max. value.
Table 120: *Current consumption with ST20 running at 81.0 MHz on page 279*	Added table.
Figure 200: *EMI interface timing definitions* on page 284	Updated signal names.
Figure 206: *Parallel interface timing definitions* on page 289	Corrected definition of T_{STR_REQ} (see also Figure 38).

Table 134 Rev A to rev B changes