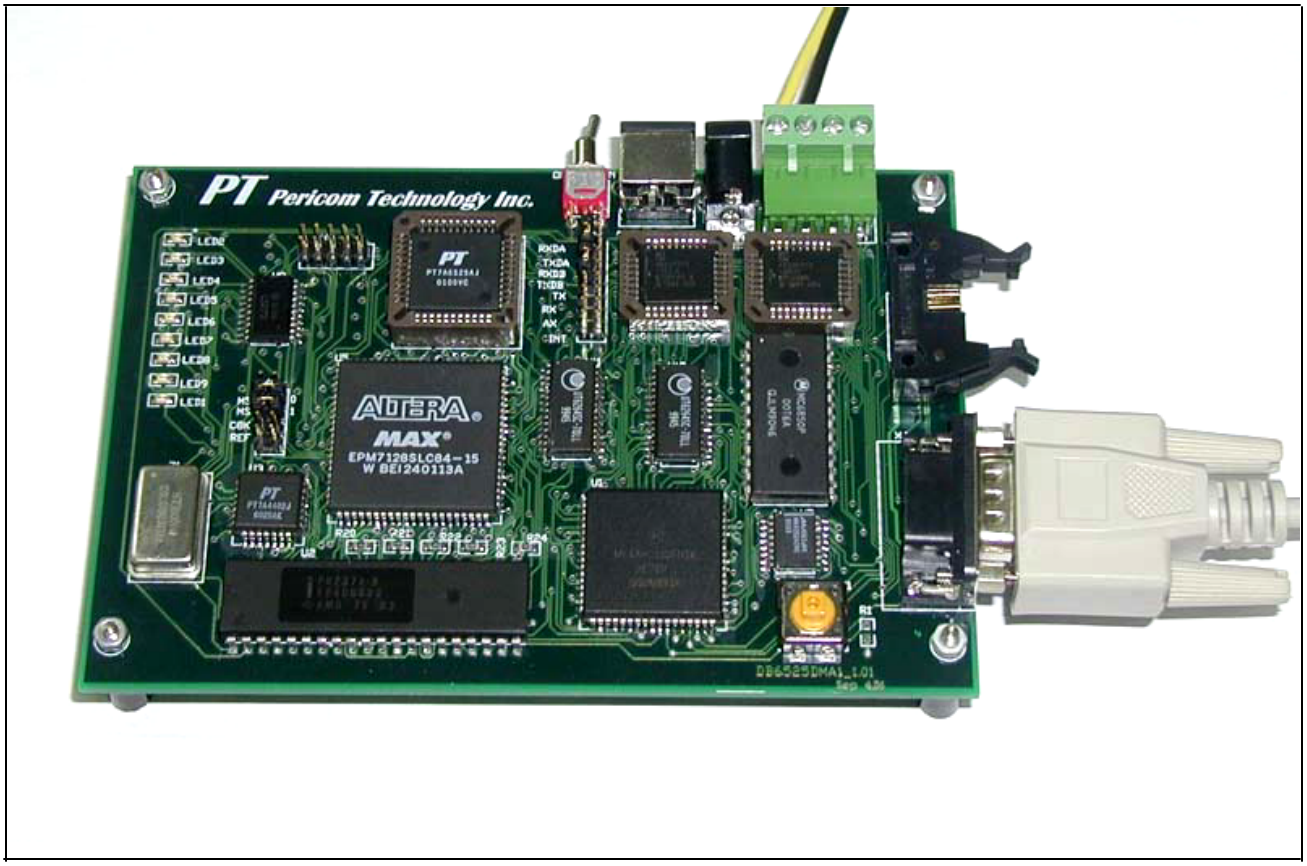
The Demo Board of PT7A6525 in DMA Mode



Brief

This Demo Board shows a simplified application of PT7A6525 (or PT7A6526) in DMA transfer mode. It demonstrates a typical circuit design of the interface between PT7A6525 and system bus, CPU, DMA controller, system memory etc. in DMA transmission mode. This board adopts a short loop in transferring rate 4.096M bps between TxD and RxD of Channel B of PT7A6525. This communication works as a point-point connection. Because PT7A6526 has a same channel as Channel B of PT7A6525, it can work in this demo board normally, too.

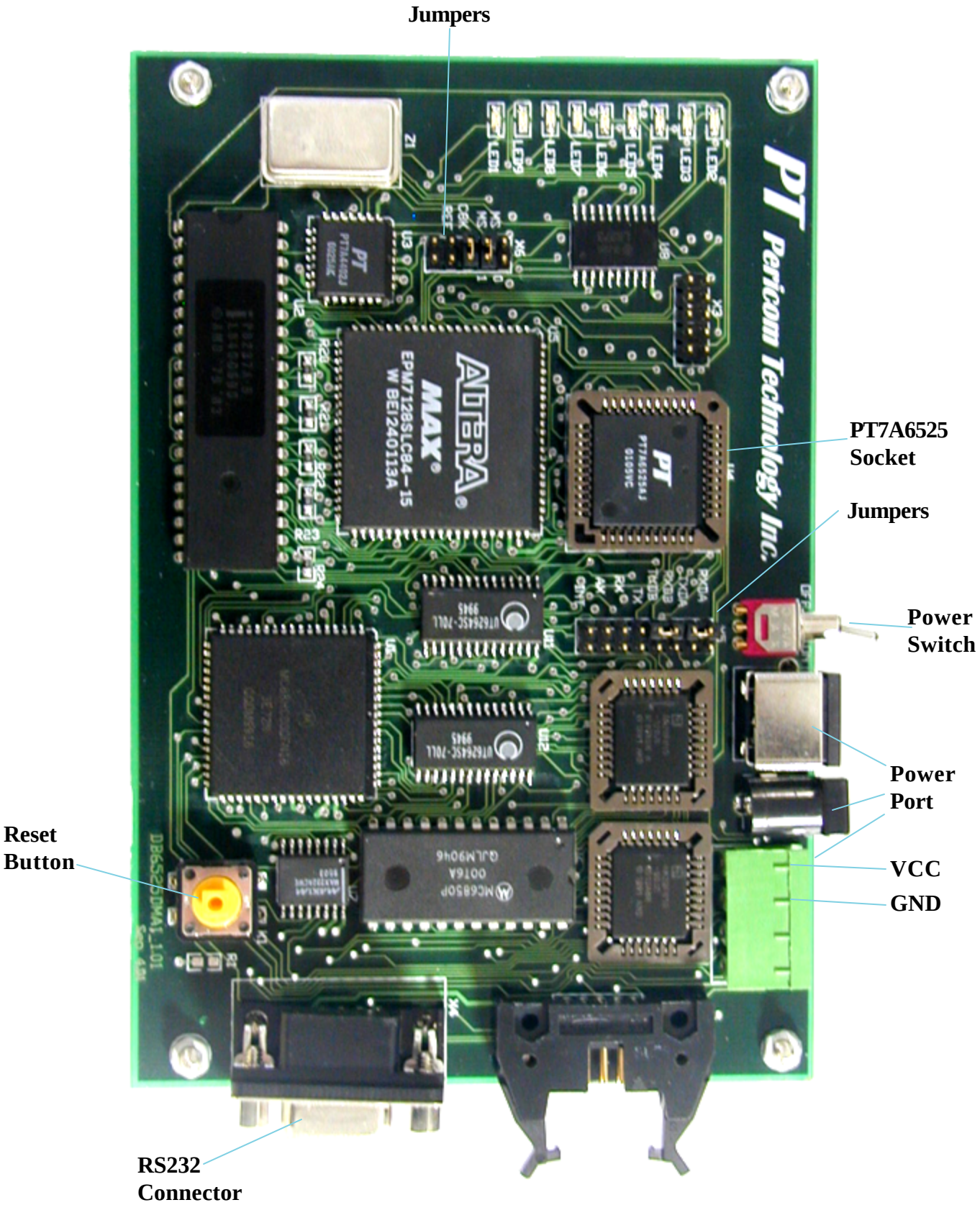
The demo board uses MC68HC000 as CPU, and 8237 as DMA controller. PT7A4402 provides clock signals to the transmitter and receiver of PT7A6525. An EPLD is used to generate control bus signals.

This application note introduces the principle of this DMA tranfer application and the operation of the demo program on PC. This program can be used to read or write the register contents of PT7A6525, and transfer or read the data in system memory.

Table of Contents

Content **Page**

Figure 1. The Photo of the Demo Board of PT7A6525 in DMA Mode



Introduction

This application note provides a simplified application of PT7A6525 in DMA mode. DMA mode provides higher data block transfer rate between PT7A6525 HDLC channel and system memory, and needs less CPU control in data block transferring than in Interrupt Request mode.

In DMA mode, the circuit designment is more complex. This demo board shows a typical circuit which can implement the fundamental functions of DMA interface in an application system.

This application mainly describes the function block of the demo board and the principle of PT7A6525 DMA interface, which includes the DMA transferring process of PT7A6525, the programming of DMA controller and PT7A6525, and the program flow of CPU to implement the transmitting or receiving process of a HDLC frame.

This application note provides the User Manual of the demo program, too.

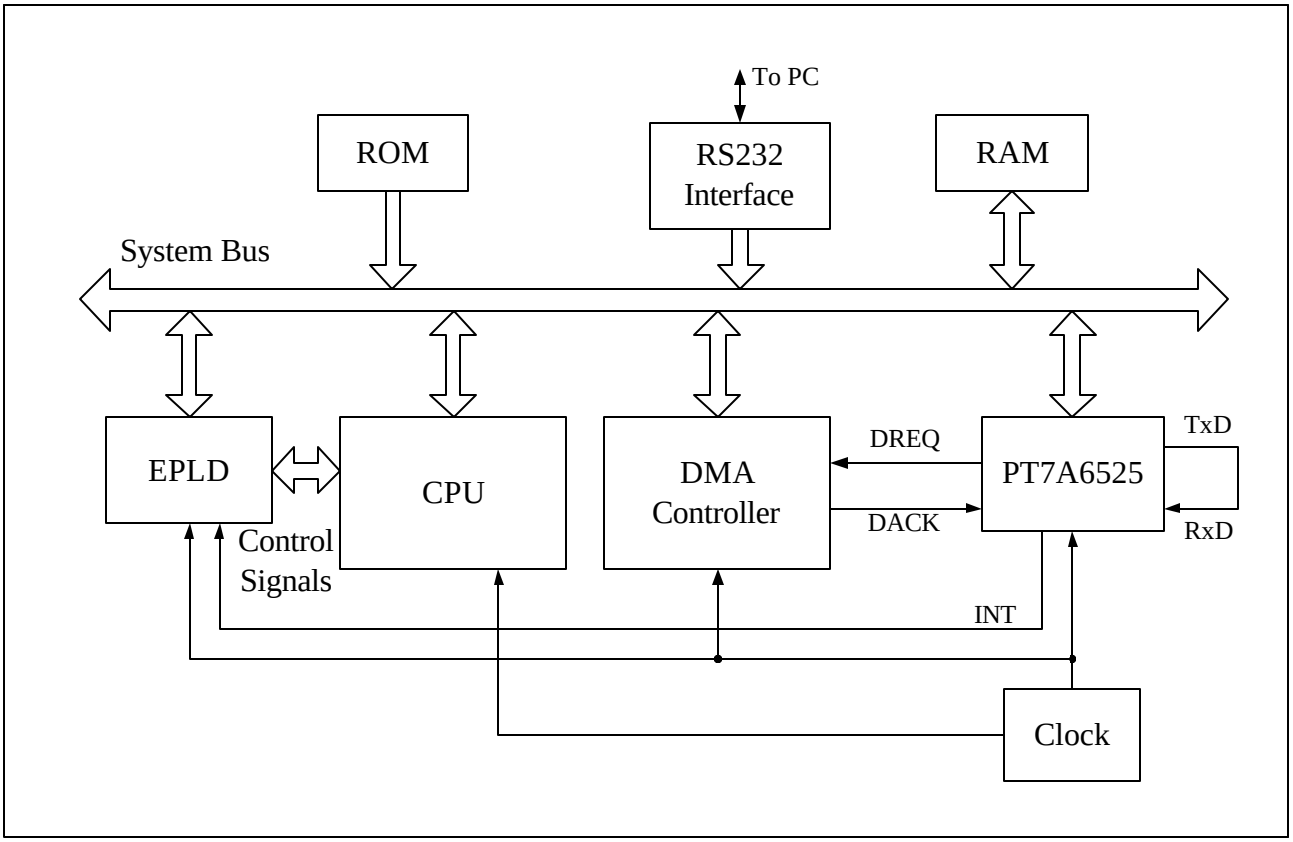
Function Description

Figure 2 shows the function block diagram of the demo board of PT7A6525 in DMA mode. The figure shows the main function block and some important signals between them.

According to Figure 2, the function blocks connected to system bus respectively. There are some other signals to implement control function. Because the CPU is MC68HC000, which is a 16-bit microprocessor, the width of the databus in system bus is 16-bit. The control bus is in Intel Bus Mode, so the control signals of MC68HC000 can not connect to system bus directly. The EPLD adapts these signals.

The principles of the function blocks will be described in the following parts.

Figure 2. Function Block Diagram



CPU of the Demo Board

The demo board uses MC68HC000 as its CPU. This microprocessor is of 16-bit data bus and 20-bit address bus. These two bus are not multiplexed. The low-order byte (D0~D7) of the data bus has a odd address, one count higher than the address of the high-order byte (D8~D15).
The bus of MC68HC000 is Motorola Bus Mode. But the system bus is in Intel Bus Mode because the ROM, RAM, and 8237 all work in Intel Bus Mode. So the control signals as BR, BG, BGACK, IPL0~2 cannot connect to DMA controller directly. All these signals need be adapted by EPLD. They are related to HLDA, HRQ signals of DMA controller 8237.
The circuit of MC68HC000 is shown in Appendix A, Schematics 1. This schematic also shows the reset circuit. The detail of MC68HC000 can be seen in its User's Manual and Programmer's Reference Manual of MC68HC000.
The supervisor program of the MC68HC000 is stored in ROM, two AM29F010. This program is a simple communication routine between the demo board and the demo program on PC. It only received the read/write commands from PC via RS232 serial interface and read/write the data from/to the address defined in the commands.
To simplify the designment of the peripheral circuit of MC68HC000 and the supervisor program of microprocessor, the interrupt is not processed by MC68HC000 and the IPL0~2 are not influenced by PT7A6525 INT output. The status of PT7A6525 is checked by demo program from ISTA register of PT7A6525.

ROM

The two ROM chips are all 8-bit data bus. To connect them to 16-bit data bus of MC68HC000, the two chips works in bit-extended mode. The A1 of address bus is connected to A0 of ROM chip, and A2 to A1 of ROM chip, etc. The D0~D7 pins of these two chips are connected to D0~D7 and D8~D15 respectively. The chip-selected signal is derived from the decoder in EPLD. The circuit is shown in Appendix A, Schematics 2.

RS232 Serial Interface

The demo board connects to PC via a RS232 Serial Interface. The connector is a 9-PIN RS232 female connector. The pin definition is as follows:

Pin 2	TxD
Pin 3	RxD
Pin 5	GND

This definition is the same as a DCE.
MC6850 is used to implement parallel-serial transform. It is a UART interface for MC68HC000. The clock of serial transmitter and receiver is 16-divided from the 1.544MHz output of PT7A4402.
MAX232A is used as the RS232 level driver and receiver.
The circuit of the RS232 serial interface is shown in Appendix A, Schematics 3.
Schematics 3 shows some other peripheral circuits. The LEDs are used here only as working flags.

RAM

There are two UT6264 chips used in this demo board as system RAM. These memory chips are of 8-bit data bus. They are connected to 16-bit data bus in bit-extended mode. The RAM circuit is shown in Appendix A, Schematics 2.
The address bus connections are the same as ROM. One chip connects to low byte (D0~D7) of the data bus, the other to high byte. These two chips have different chip-select signals. Signal RAML is for the chip connected to D0~D7 and RAMH for the chip connected to D8~D15. These two chip-select signal are derived from EPLD and they are controlled by LDS and UDS from MC68HC000.

Clock

The clock circuit is shown in Appendix A, Schematics 4. The 20MHz clock and PT7A4402 (or PT7A4401) provide clock signals for MC68HC000, PT7A6525 and 8237. The MC68HC000 master clock is 10MHz. And PT7A4402 provided 8KHz frame pulse, 4.096MHz clock and 1.544MHz clock.

PT7A6525 (or PT7A6526)

PT7A6525 is designed to implement high-speed communication links using HDLC protocols. It has two completely independent full-duplex HDLC channels (Channel A and Channel B), while PT7A6526 supports only one (Channel B). Associated with each serial channel there are a set of independent command and status registers and 64-byte FIFOs. Data blocks from/to system memory can be transferred by either Interrupt Request or Direct Memory Access (DMA).
For each channel of PT7A6525/PT7A6526, there are two DMA interface. So PT7A6525 contains a 4-channel DMA interface, while PT7A6526 contains a 2-channnel DMA interface.
For each channel, a separate DMA Request output for the Transmit (DRQT) and Receive direction (DRQR), and a DMA Acknowledgment (DACK) for both directions are provided.
As long as data transfers from/to the specific FIFO are needed, PT7A6525 activates the DRQ lines. It deactivates the DRQ lines immediately after the last read/write cycle of the data transfer has started. If the DMA controller is in Level-Triggered Demand Transfer Mode, it executes the correct number of bus cycles by watching the DRQ. Read cycles will be executed if DMA controller is requested by DRQR and write cycles will be executed if DMA controller is requested by DRQT. If a DMA acknowledgment signal provided by the DMA controller is connected to the $\overline{\text{DACK}}$ pin of PT7A6525, PT7A6525 can be accessed as an external I/O device. The address and chip-select signal ($\overline{\text{CS}}$) are not needed, and the top byte of the FIFO is read/written in each bus cycle. If $\overline{\text{DACK}}$ is not provided, memory-memory transfer must be performed, and chip-select and address are needed.
In DMA mode, the bit D7 of XBCH register must be set. The length of the next frame to be transmitted must be written into XBCH and XBCL before transmitting. And the length of the last received frame can be read from RBCH and RBCL. The detail can be seen from PT7A6525 datasheet.
The circuit of PT7A6525 is shown in Appendix A, Schematics 3.

DMA Controller

The D8237AC-5 is used in this demo board as the DMA controller. It is compatible with Intel 8237A-5. It provide four independent programmable DMA channels. The circuit of 8237 is shown in Appendix A, Schematics 3. In this demo board, channel 0 serves for the Receiver of Channel B of PT7A6525 and channel 1 for the Transmitter of Channel B of PT7A6525.
When 8237 works in active cycles, it controls the system bus. 8237 has 8-bit data lines and A0-A7 address bus, and the A8-A15 are multiplexed on the data lines. If A8-A15 is used, an external 8-bit address latch is needed to generate 16-bit address bus. In order to simplify the circuit design, only A0-A7 is used in this demo board. So the active address range is 0 to FFh in DMA transferring process. Because in this demo board transmitting and receiving buffer share the same RAM area in the UT6264 connected to D0~D7, the length of a frame transmitted/received must be less than 128.
There are four independent DMA channel in 8237. Each channel has a set of registers including Current Address Register, Current Word Count Register, Base Address Register and Base Word Count Register. Before a DMA process, the Base Address Register and Base Word Count Register should be set. The values of the registers in 8237 will be discussed in the following parts.
8237 has some registers such as Mode Register, Command Register, Mask Register, etc. The details can be seen from its Datasheet. Some of these register are initialized in Demo Program.

EPLD

A EPLD (Altera EPM7128S) is used in this demo board to generate control signals. These signals include chip-select signals, DMA control signals, control signals of MC68HC000, and frequency-divided clock signals. The schematics of the EPLD internal logic functions is shown in Appendix B.

Power

There are three power ports on the demo board. +5V VCC power should be provided through one of these ports. When the power switch is on the "on" direction, the power on the demo board is connected to the input of the power port.

Jumpers

There are two groups of jumpers on the demo board. Normally, these jumpers should be connected as shown in Figure 1.

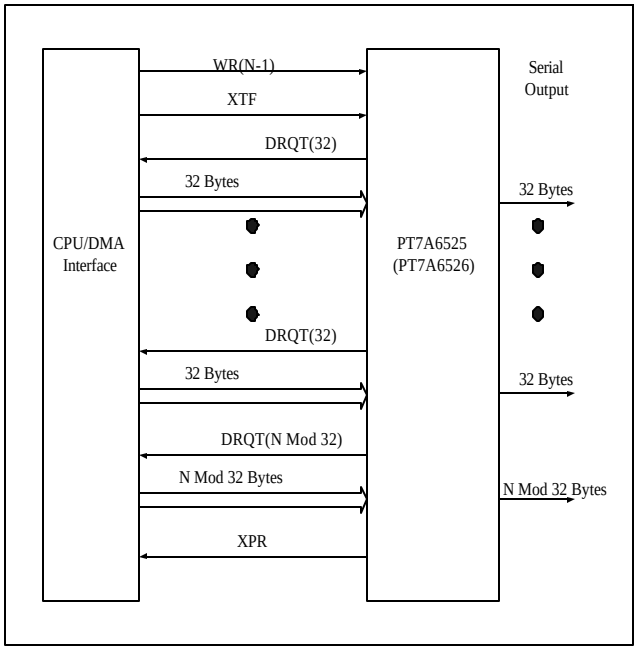
The Principle of PT7A6525 DMA Interface

This part includes the DMA transferring process of PT7A6525 and the program flow of CPU to implement the transmitting or receiving process of a HDLC frame.
The PT7A6525 works in DMA mode when the bit D7 of the register XBCH is set to "1".

The DMA Transferring Process of PT7A6525

When PT7A6525 works in DMA mode, the process of transmitting/receiving a frame is simpler than in interrupt request mode.
Before transmitting a frame, the data of the frame must be stored in the system memory, and DMA controller must be programmed. Because the XFIFO in PT7A6525 is 32 bytes, a frame whose length is longer than 32 bytes will be divided into several parts in transferring process. Figure 3 shows the process of Frame Transmission in DMA Mode when the length of frame is more than 32 bytes.

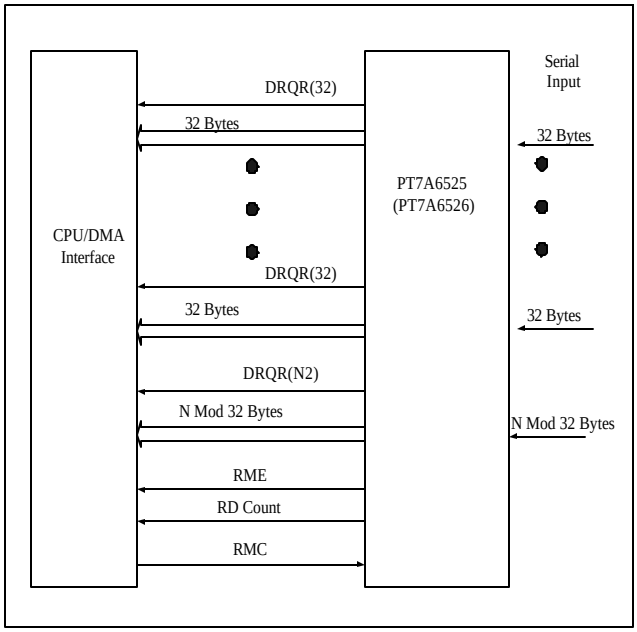
Figure 3. Frame Transmission in DMA Mode



During the transmission of a frame of N bytes, PT7A6525 will request the DMA controller to transmit for N1 times by setting DRQT high, and N = (N1-1)*32 + (N Mod 32). During the transmission of a block, DRQT remains high and a byte of data is transferred from system memory to PT7A6525 in every bus cycle when $\overline{\text{DACK}}$ is active. When a block reading is finished, the DRQT goes low until the next request of PT7A6525. After the frame is sent, PT7A6525 generates an interrupt to CPU and sets the XPR bit in ISTA register. In this demo board, CPU doesn't acknowledge the interrupt. The Demo Program inquires the ISTA register and decides the next operation.
In the Demo Program, the DMA channels of 8237 are programmed to work in Demand mode. Thus 8237 checks the current status of DRQT and transfers the correct byte count of data in each block transferring process. When DRQT is inactive, 8237 discharges the control of system bus and CPU is allowed to operated.

When PT7A6525 is receiving a frame, it requests the DMA controller to transfer the data from RFIFO to system memory after a 32-byte data block is received or the frame is finished. Only after the reception of a frame is finished, PT7A6525 generates a RME interrupt to CPU. Before this interrupt, the receiving process need not any control of CPU.
If the length of the frame is not definite, the maximum of the frame length should be wrote to the Base Word Count Register of the corresponding channel in 8237. Thus it is assured that the total frame can be transferred to a successive area in system memory. As a reference, the maximum of the frame which PT7A6525 can transmit/receive in DMA mode is 4096. (In this demo board, the length is less than 128.) After a frame is received, CPU processes the data, and then sets PT7A6525 and DMA controller to wait for the reception of the next frame.
Figure 4 shows the process of the Frame Reception in DMA Mode when the length of frame is more than 32 bytes.

Figure 4. Frame Reception in DMA Mode



During the reception of a frame of N bytes, PT7A6525 will request the DMA controller for N1 times by setting DRQR high, and N = (N1-1)*32 + (N Mod 32). Here, the last DRQR remains high during the transferring of N2 bytes. Table 1 shows N2 (the count of request bus cycles) and N Mod 32 (the count of the bytes in RFIFO at the last DRQR).

Table 1. The bus cycles of the last DRQR

N Mod 32	N2
1 ~ 3	4
4 ~ 7	8
8 ~ 15	16
16 ~ 32	32

The DMA channels of 8237 are also programmed to work in Demand mode.
After a reception, CPU may check the RME interrupt of PT7A6525, read the count of the data, and send RMC command to PT7A6525. CPU can also set the DMA controller for the next reception.

The Programming of DMA Controller and PT7A6525

8237 and PT7A6525 can be programmed by writing their registers. In this demo board, all the content of registers are written by Demo Program via the CPU on the demo board. The details of the definitions of these registers , such as address and bit definition, can be found in 8237 datasheet or PT7A6525 datasheet.

Initialization

To initialize 8237 in the correct status, Command Register, Mode Register, Mask Register and some other registers needs to be written by Demo Program. Demo Program writes the contents of some registers of 8237 before or after each DMA transferring process, too. PT7A6525 is initiated when Demo Program begins, too.
Table 2 shows the initialization process of 8237 in Demo Program. It shows the content of the register.

Table 2. Initialization of 8237

Step	Register	Content
1	Command	04h
2	Master Clear	
3	Base Address of Channel 0 (Low byte)	00h
4	Base Address of Channel 0 (High byte)	00h
5	Base Word Count of Channel 0 (Low byte)	7Fh
6	Base Word Count of Channel 0 (High byte)	00h
7	Base Address of Channel 1 (Low byte)	80h
8	Base Address of Channel 1 (High byte)	00h
9	Mode	14h
10	Mode	19h
11	Mode	16h
12	Mode	1Bh
13	Command	10h
14	Mask	00h
15	Mask	01h

After the initialization, 8237 is enabled. Channel 0 (for Receiver of Channel B of PT7A6525) is in Read Transfer Mode. Channel 1 (for Transmitter of Channel B of PT7A6525) is in Write Transfer Mode. Channel 2 and Channel 3 are masked. The limit of the length of data is 128 (7Fh+1). And the reading buffer and the writing buffer in system memory are independent. The DMA priority is in Rotating mode. All the four channels are Autoinitialization Enable. Thus they needn't to be restarted after a transfer. Because the $\overline{EOP}$ is not provided in this demo board by hardware, the Demo Program must initiate the Base Address register and Base Word Count register of Channel 0 again after each DMA transfer. After a frame transmission, the word count decreases to FFFFh, so a internal $\overline{EOP}$ in 8237 is generated. This $\overline{EOP}$ restores the correct Base Address (so the re-initialization of this register is not needed), but the Base Word Count must be written to fit to the length of the next frame before the next transmission.

The initialization of PT7A6525 setups the contents of some registers. This initialization process will be shown in the User Manual of the Demo Program. The register contents are shown in Figure 10. After Initialization, the PT7A6525 works in Transparent Mode, Clock Mode 0. Channel B is in DMA Transmission Mode, while Channel A is in Interrupt Transmission Mode. The details of the registers can be seen from PT7A6525 datasheet.
The registers can be set as other values if PT7A6525 is needed to work in other modes.

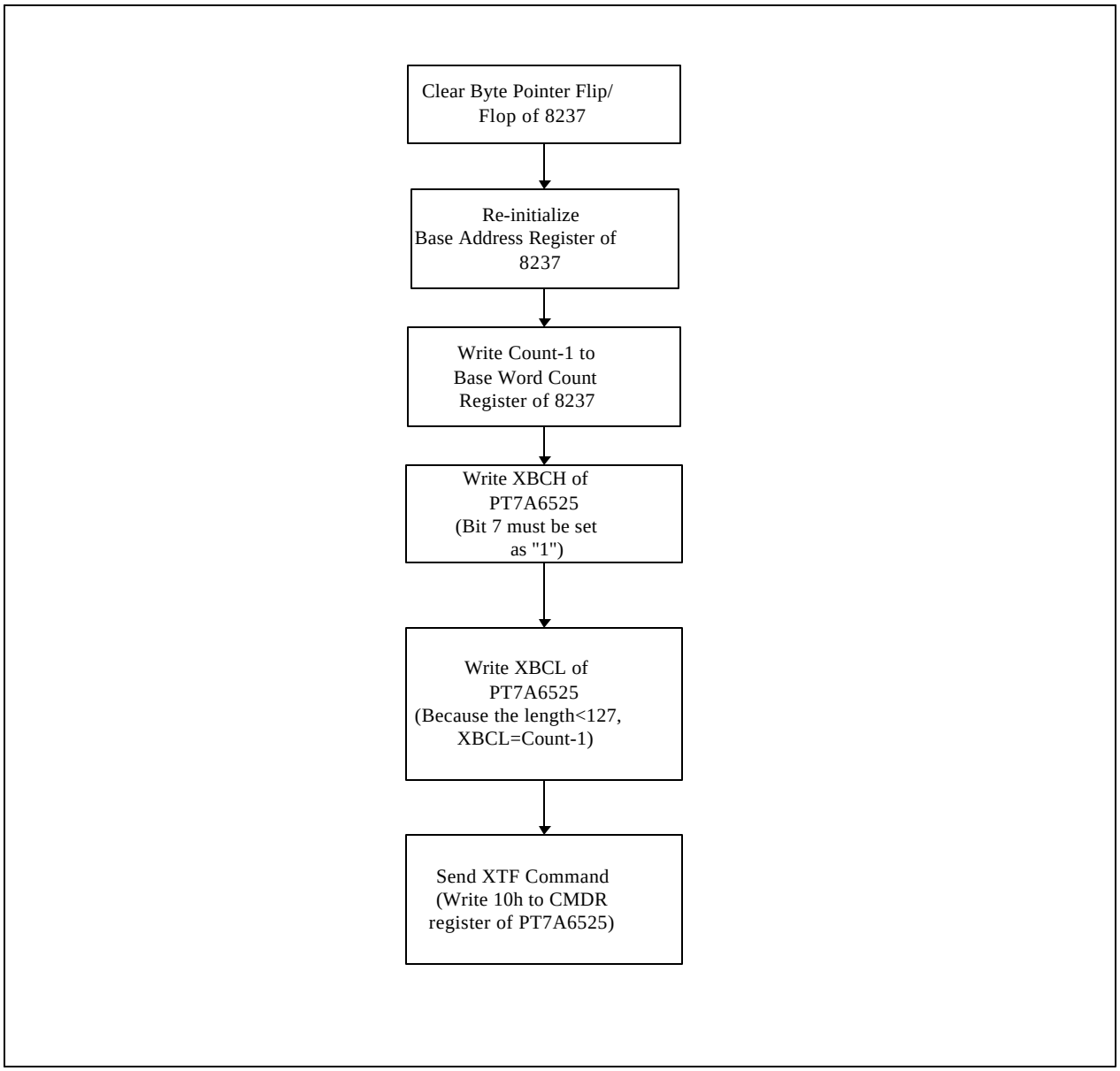
The Process of Transmission/Reception

The operations of transferring or receiving a frame is shown in the User Manual of the Demo Program. Here shows the process of the transmission/reception in the demo board. Some parts of the process are invisible to user but by the Demo Program in back ground. All the steps in the process are via CPU and the flowcharts are shown on the CPU interface.

Figure 5 shows the process of the transmission of a frame in Demo Program.

The DMA transfer from system memory to PT7A6525 shown as Figure 3 will perform after this process.

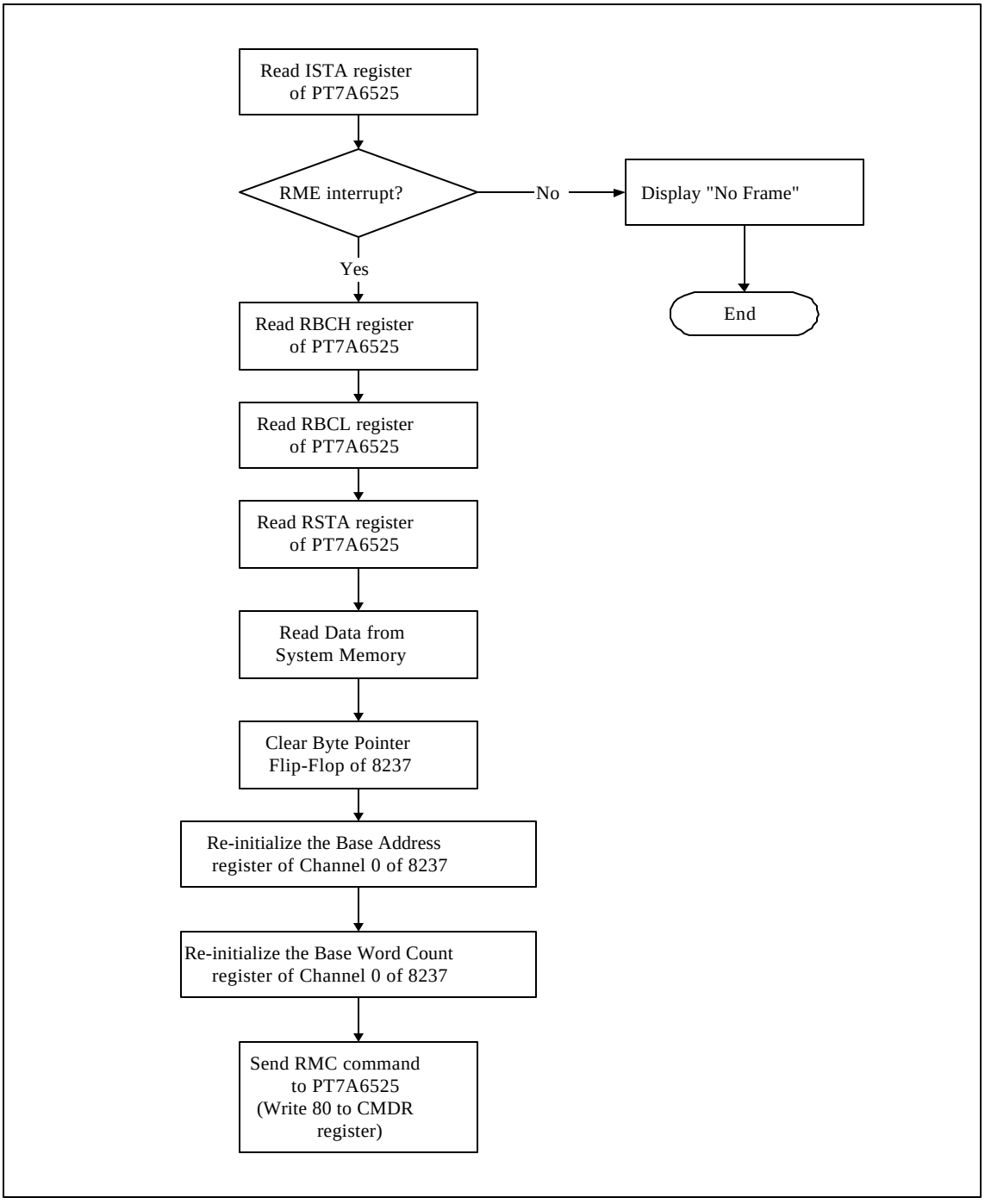
Figure 5. The Process of Transmission of a Frame



Note: The definition of the high 4 bit of XBCH can be found in PT7A6525 datasheet and be changed by user in Demo Program.

Figure 6 shows the process of reception of a frame in the Demo Program. The DMA transfer which is shown in Figure 4 has finished before this process.

Figure 6. The Process of Reception of a Frame



Note: In re-initialization, the Word Count register of 8237 is set as 7Fh to receive the frame of maximum length.

The Hardware Configuration of the Demo Board

Component on the demo board

Shown in Figure 1, there are some jumpers, connector and button which are configured by user.
To connect the circuit correctly, there are 4 pins in jumper set X5 should be connected. The pin “RxDA” and the pin “TxDA” is connected to each other by jumper, and so does the pin “RxDB” and the pin “TxDB”.
The button K1 is for reset. Pressing this button will reset all the chips on the demo board. After power-on, a reset is needed before the demo program can run normally.
X4 is a RS232 9-pin female connector to PC. The pin definition is shown in “Function Description - RS232 Serial Interface”. This connector is connected to the COM1 or COM2 port of PC. After the demo program starts to run, it will check the port which the demo board is connected to.
There is a power port on the board. One pin of the port is for VCC and another for GND. VCC is a +5V power. The range of VCC is 4.5V to 5.5V. A supply voltage beyond this range may cause damage on the demo board. The pin definition is shown in Figure 1.

Connect the Hardware

Before the demo program on PC is run, the demo board should be connected to power source.
The X4 should be connected to the COM1 or COM2 port of PC by RS232 cable with correct pin-connections. The baud rate of COM1 is 9600 bps which is set by the Demo Program. The serial port is of no parity check, eight data bit, and one stop bit.
After the power-on of the demo board, press button K1 to generate the reset signal.

The User Manual of the Demo Program

The Demo Program runs on the PC connected to the demo board. The user can write/read the registers in PT7A6525 and transmit/receive a data frame with this program.

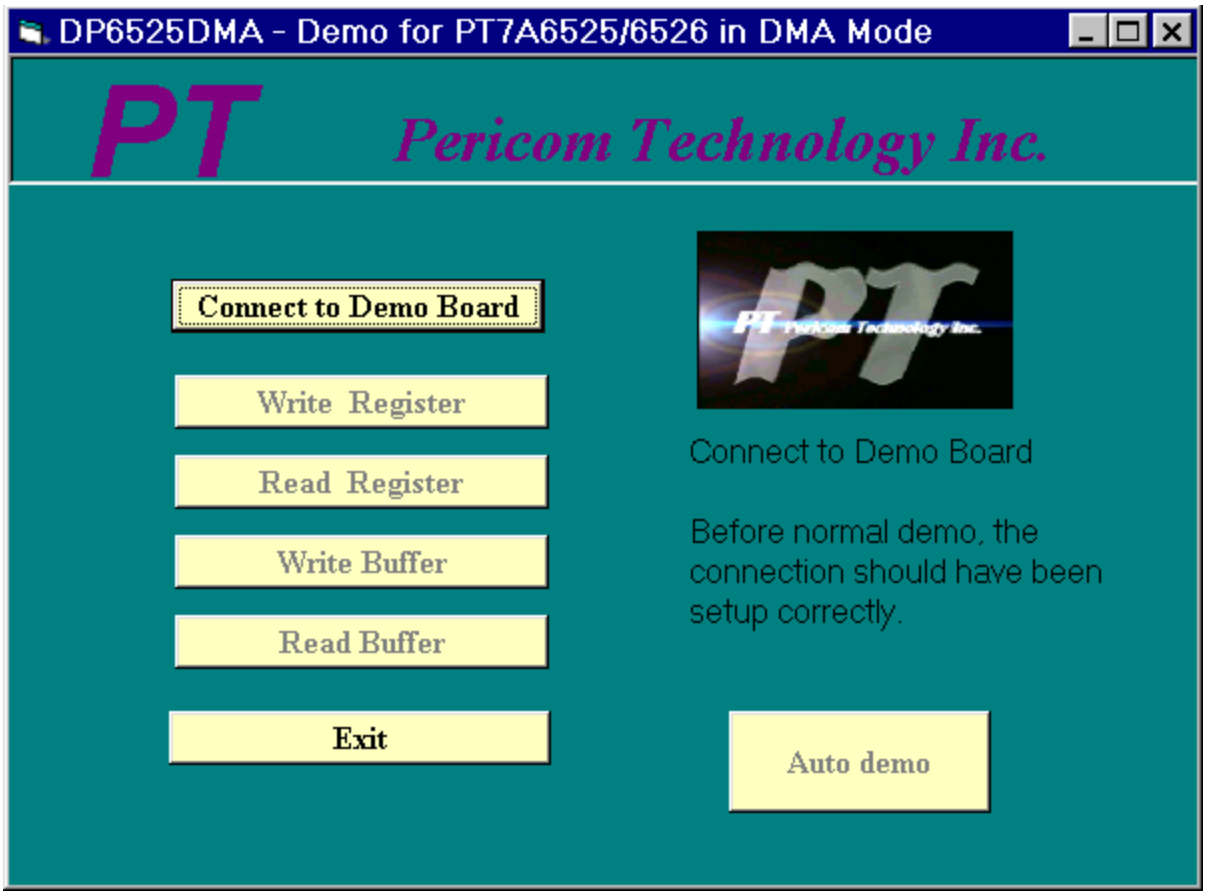
Install and Run the Demo Program

This demo program is provided to user as a setup program on disk. Running "setup.exe" in Windows 98 will install the demo program in the PC.
To start this demo program, the user can run it in Windows 98 explorer, command line or the start menu of Windows 98.

Enter the Main Menu Window

After the demo program starts running, it will show the window as an introduce. After a few seconds, the Main Menu Window will be shown as Figure 7.
Pressing the button "Exit" will exit this demo program immediately.
Press the button "Connect to Demo Board" to establish the connection between the demo program and the supervisor program in the demo board.
There are four other buttons to implement the read register, write register, read buffer, and write buffer functions. When the demo program enter the Main Menu Window at the first time after running, these four button are not enabled. To run these functions, establishing the connection at first is needed.
The button "Auto demo" will start the auto-demo process, which includes the four functions described in the previous paragraph.

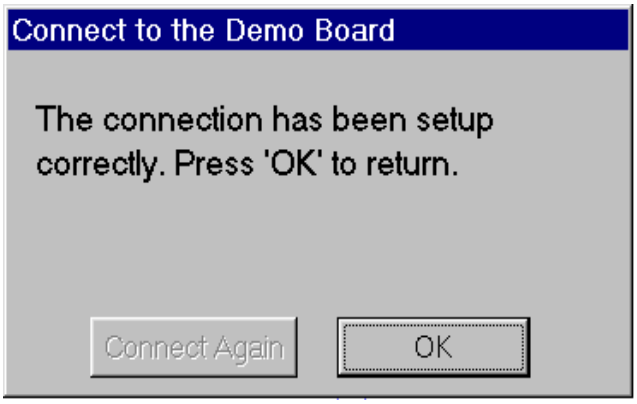
Figure 7. The Main Menu Window



Connect to Demo Board

When the demo program starts to run or after a error has happened on the connection between the demo program on PC and the supervisor program on demo board, the demo program cannot communicates with the demo board correctly. To establish a connection is needed. It can be completed by pressing the button “Connect to Demo Board” in the Main Menu Window.
This window sends a command to the supervisor program on the demo board and waits for the acknowledgment of the supervisor program in a limited time span. If the acknowledgment is received correctly, the connection is established.
Before the connection has been established, all the function will be disabled. The user must return to the Main Menu if in other windows at this time. If a communication error has happened in the other windows, a error message box will be displayed.
After the button “Connect to Demo Board” is pressed, a window will be shown as Figure 8. If the connection is established correctly, the button “OK” is enabled as shown in Figure 8 and the window will prompt the user to return to Main Menu.

Figure 8. The Window of Connection (When correctly)



If the connection is still incorrect, the button “Connect Again” and the button “Cancel” is enabled. Check the RS232 cable, and make it sure that the power of the demo board is on and the board has been reset, then press “Connect Again”. If there is still error message shown by the window, some damage on demo board, RS232 connectors and cable, or PC may have happened. Pressing “Cancel” can return to Main Menu.
Figure 9 shows an example of connection error. When an error happens in the Write Register Window, this error message box will be shown. Press the “OK” button to return to the Write Register Window.

Figure 9. An Example of Error Message Box

Initialize the Demo Board

The Initialization of the demo board is implemented by writing registers in Write Register Window when the check box “Initiate” is selected. This initialization process writes the registers of PT7A6525 and 8237. The contents of the registers is shown in Table 2 and Figure 10.
The register contents of PT7A6525 can be changed by user. The detail is shown in the following page.
8237 is initialized only by this process. The initialization of 8237 is in the background and invisible. An initialization process is necessary before the functions of the demo program can work normally.

Write PT7A6525 Registers

Press the button "Write PT7A6525 Registers" in Main Menu to enter the Write Register Window shown as Figure 10.

All the writable registers are shown in this window. The text box under a specific register name shows the content which will be written to this register. This content can be changed by user. If the check box in the left of text box is selected, this specific register will be written. The contents and the selection of the check box shows the last writing operation.

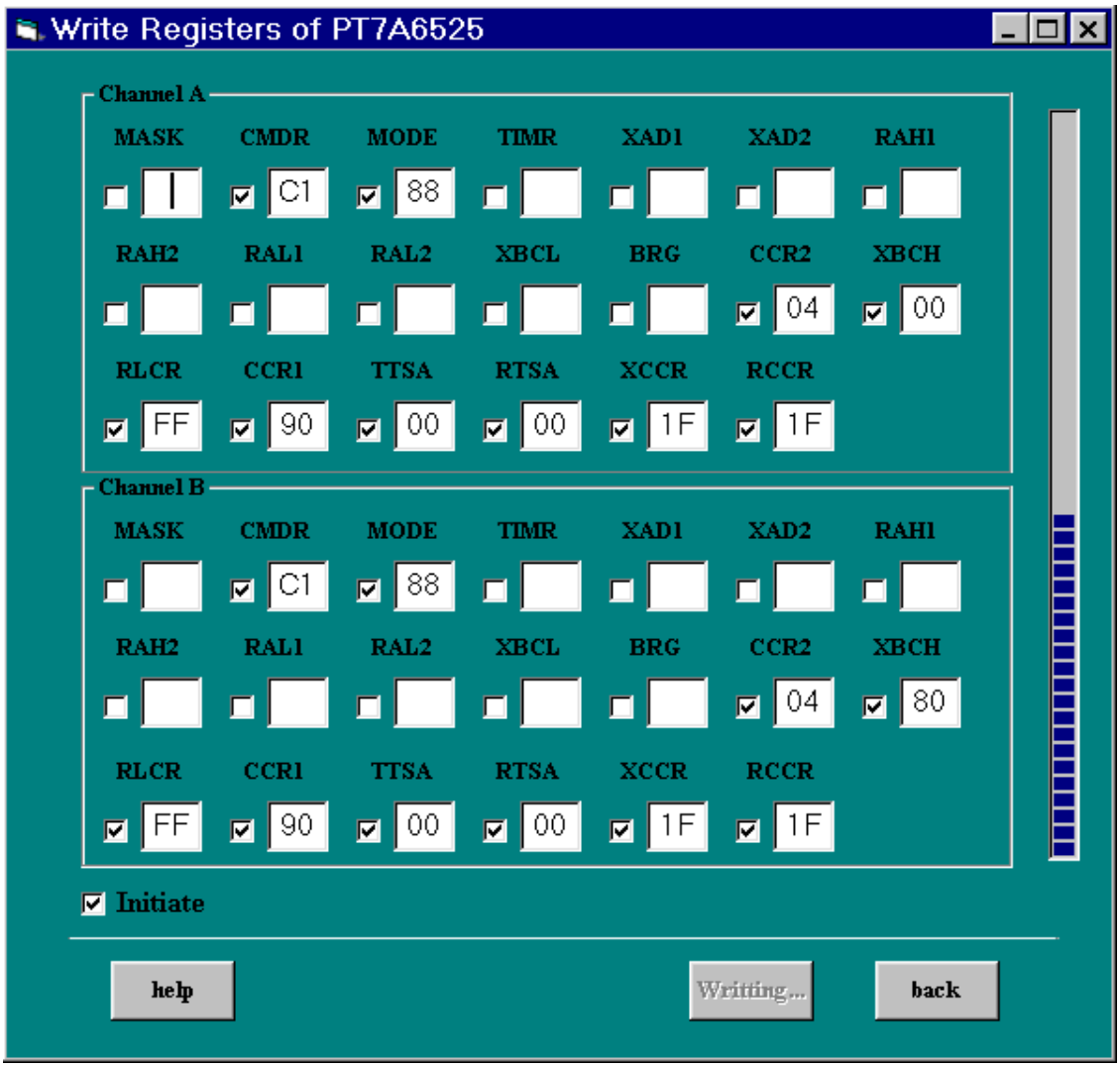
After all the contents are filled in and the check boxes are selected, press the button "Write" to start to write all the selected registers of PT7A6525. During the process of writing, the progress bar in the window will show the current progress.

If the check box "Initiate" in the left-down of the window is selected, all the registers which need to be initiated will be set as the default content in initialization and be selected. When an initialization hasn't been processed after the running of program, the check box "Initiate" will be selected when this window is shown at the first time.

After the writing process is finished correctly, the button "Cancel" will be shown as "OK". Press button "OK" to return to Main Menu. If an error occurs, an error message box will be shown and press button "Cancel" to return.

Note: All the contents of registers in all the windows of this demo program are shown as hexadecimal format.

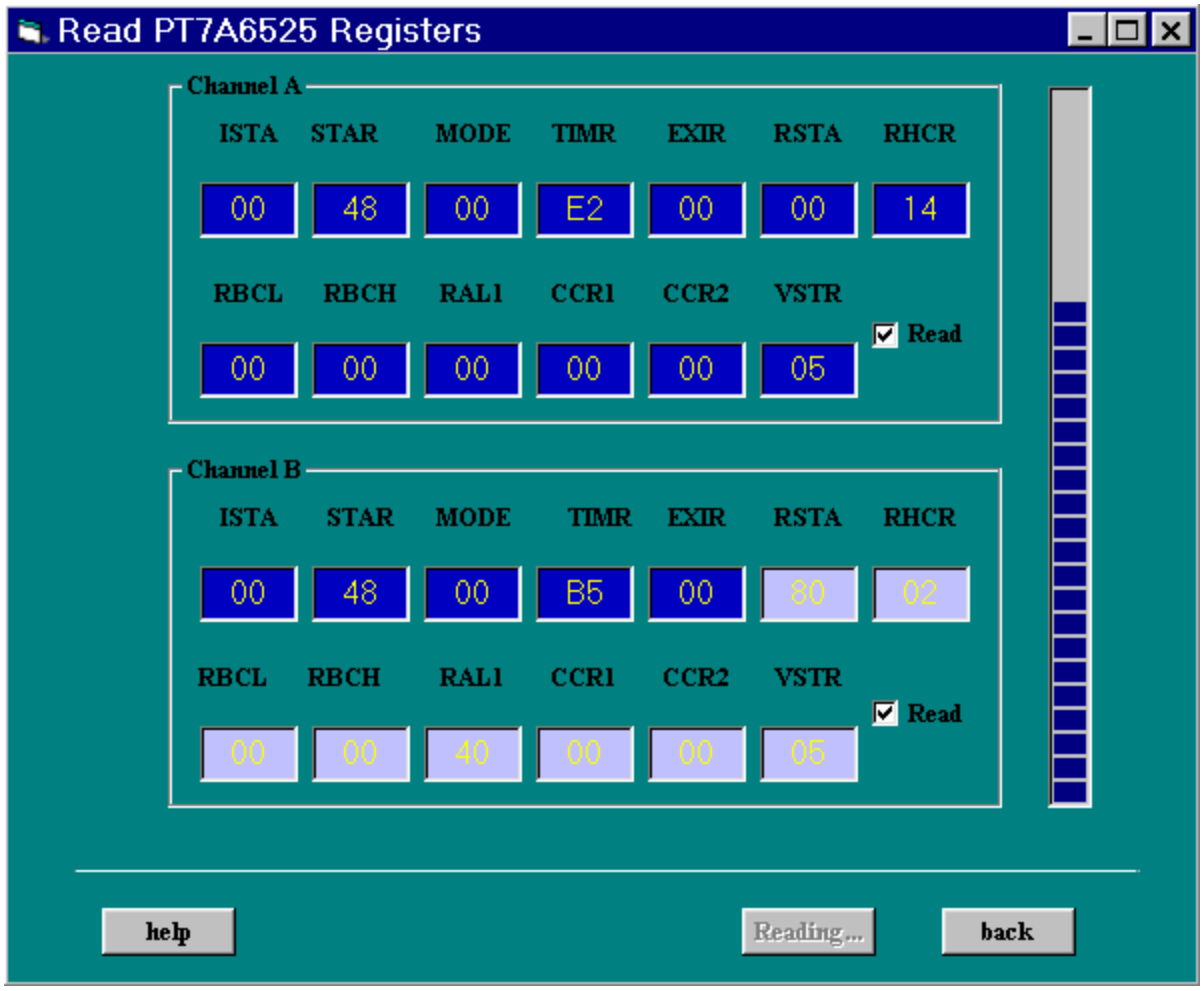
Figure 10. The Write Register Window



Read PT7A6525 Registers

Press the button “Read PT7A6525 Registers” in Main Menu to enter the Read Register Window shown as Figure 11.
There are two check boxes in the windows. If a check box is selected, all the registers of the corresponding channel will be read.
Pressing the button “Read” will start the reading process. The contents are shown in the text boxes under the register names. Before reading process, the text boxes show the last contents read by the demo program.
After the reading process is finished correctly, press the button “OK” to return to Main Menu.
If error happens during the reading process, a error message box will be shown. It is needed that pressing button “Cancel” to return to Main Menu and to establish the connection again.

Figure 11. Read Register Window

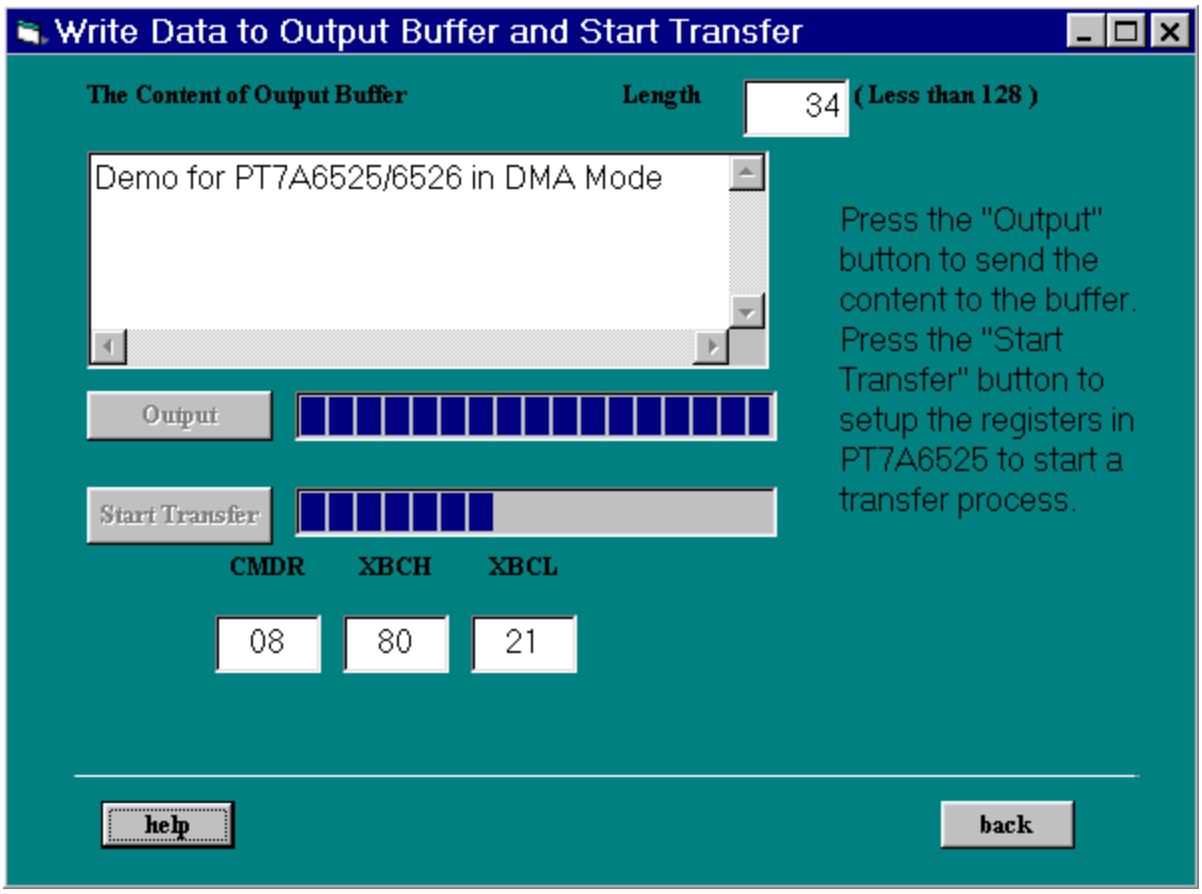


Write the Output Buffer and Transmit a Frame

Press the button “Write Buffer” in Main Menu to enter the Write Buffer Window shown as Figure 12.
The data of the frame is written in the text box “the Content of Output Buffer”. The length of the data is shown in the text box “Length”. It is noted that the length can not be more than 127.
After the input of the data, pressing the button “Output” will start the transfer of the data from PC to the output buffer of system memory.
After the data has been saved in the output buffer, pressing the button “Start Transfer” will start the transmission of a frame. The contents of CMDR, XBCH and XBCL are shown in the window. The XBCL content is length-1. These three contents will be written into the registers before the transmission. The user can change these content if needed.

The transmission process of the demo program is shown in Figure 5. After this process, the PT7A6525 will request the DMA controller to get the data from system memory and will transmit the data in a frame. This process is shown in Figure 3.
If the data and registers have both been written to the demo board correctly, press the button “OK” to return to Main Menu. Otherwise, an error message box will be shown.
Note: The length is shown in decimal format.

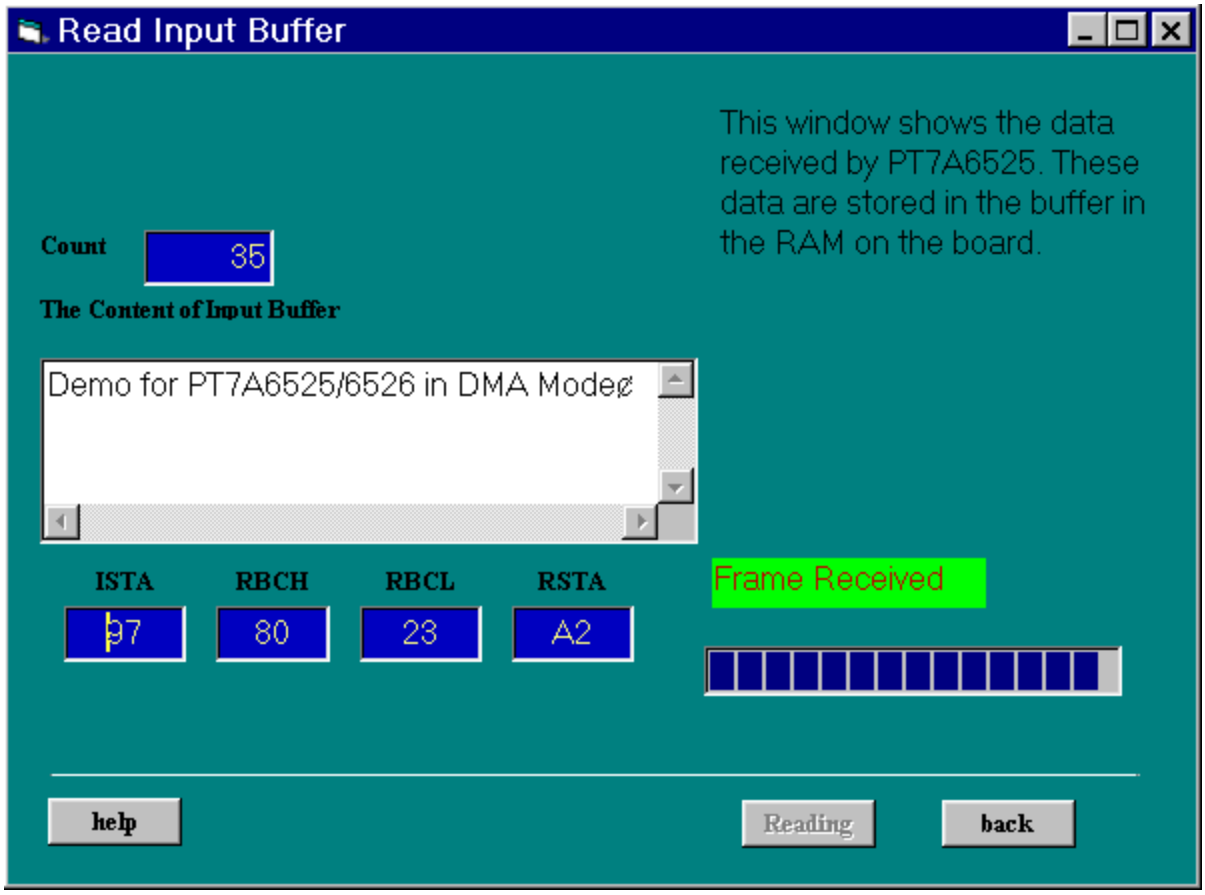
Figure 12. Write Buffer Window



Read the Data in Input Buffer

Press the button “Read Buffer” in Main Menu to enter the Read Buffer Window shown as Figure 13.
Pressing the button “Read” will start to read the data from the input buffer in the system memory.
The reading process of the demo program is shown in Figure 6. If a frame reception shown as Figure 4 has been finished before reading input buffer, the demo program will get a RME interrupt.
If a RME interrupt has happened, this window will show the content of ISTA, RBCH, RBCL and RSTA. And the count of the data byte is shown in a text box. Then the demo program reads the data in the input buffer and displays them in the text box “The content of input buffer”.
After the reading process, the demo program will re-initialize channel 0 of 8237 and send a RMC command to PT7A6525.
The length of the frame received is one byte more than the length transmitted. This byte is the RSTA content and it is added to the end of the data. This byte can be seen in Figure 13.
If the reading process is finished correctly, press the button “OK” to return to Main Menu. Otherwise, an error message box will be shown.
Note: The count of data is shown in decimal format.

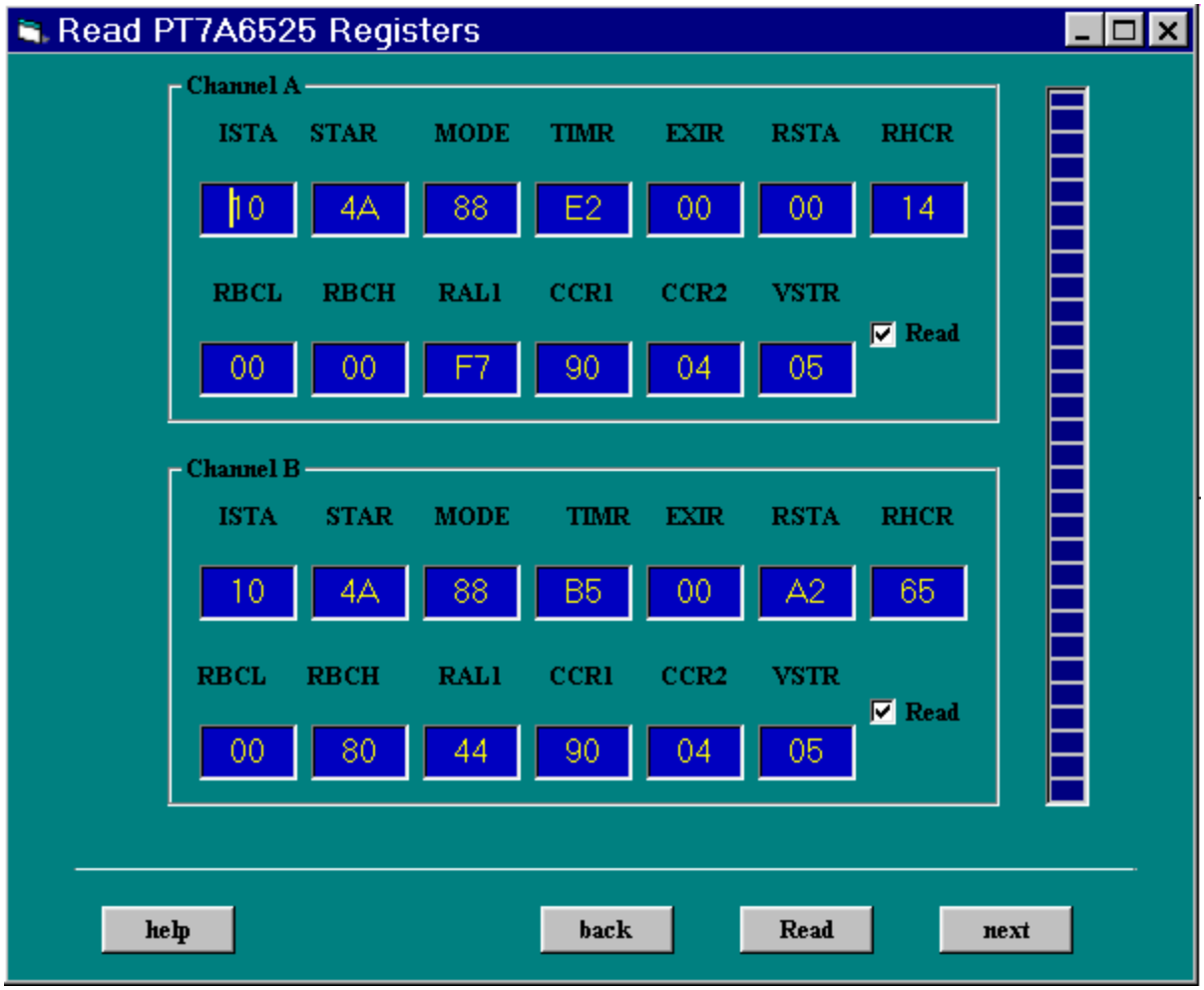
Figure 13. Read Buffer Window



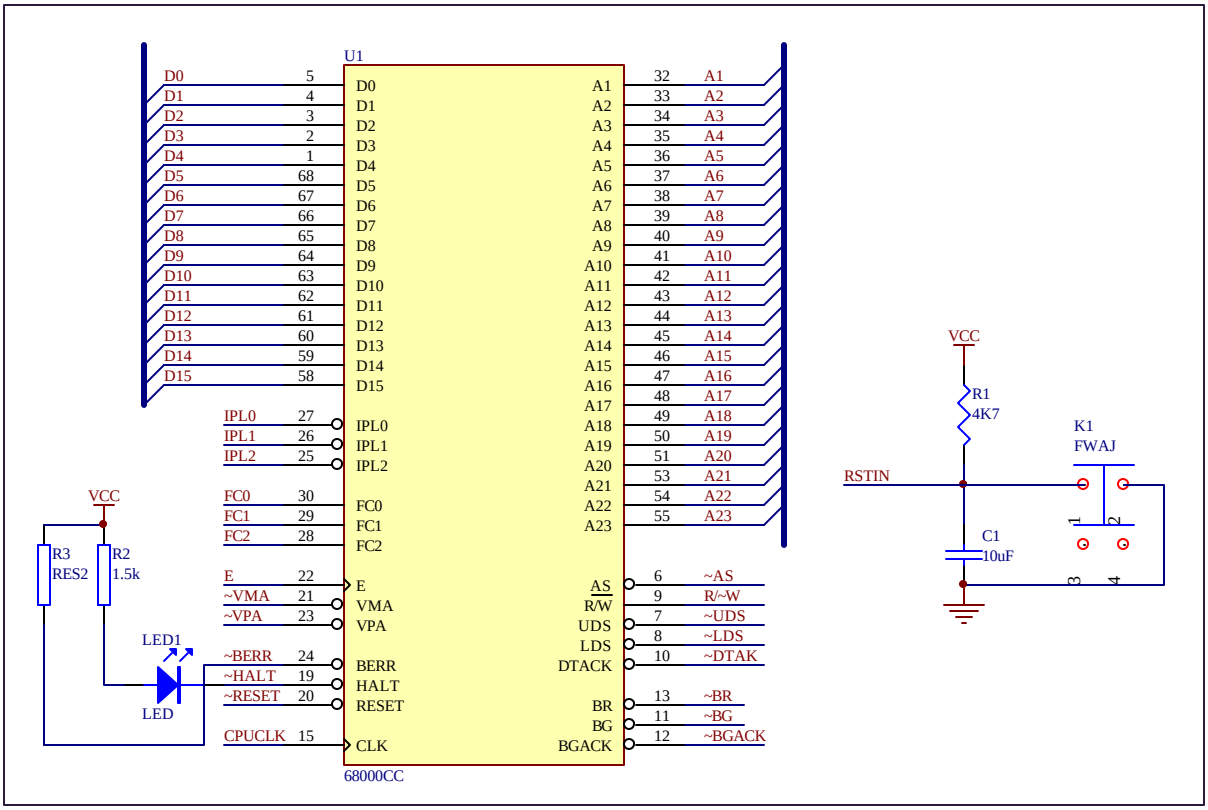
Auto Demo Flow

Press the button “Auto Demo” in Main Menu to enter the Auto Demo Flow.
This flow includes the four functions described previously: Write Register, Read Register, Write Buffer and Read Buffer. The windows of these four functions will be shown sequentially.
There are some little difference in the window comparing to the windows described in the previous pages. Figure 14 shows the Read Register Window in Auto Demo Flow. Comparing to Figure 11, a “next” button is added. When this button is pressed, the auto demo flow will enter the next function. As an example, after pressing the “next” button in the window shown as Figure 14, the program will enter the Write Buffer window.

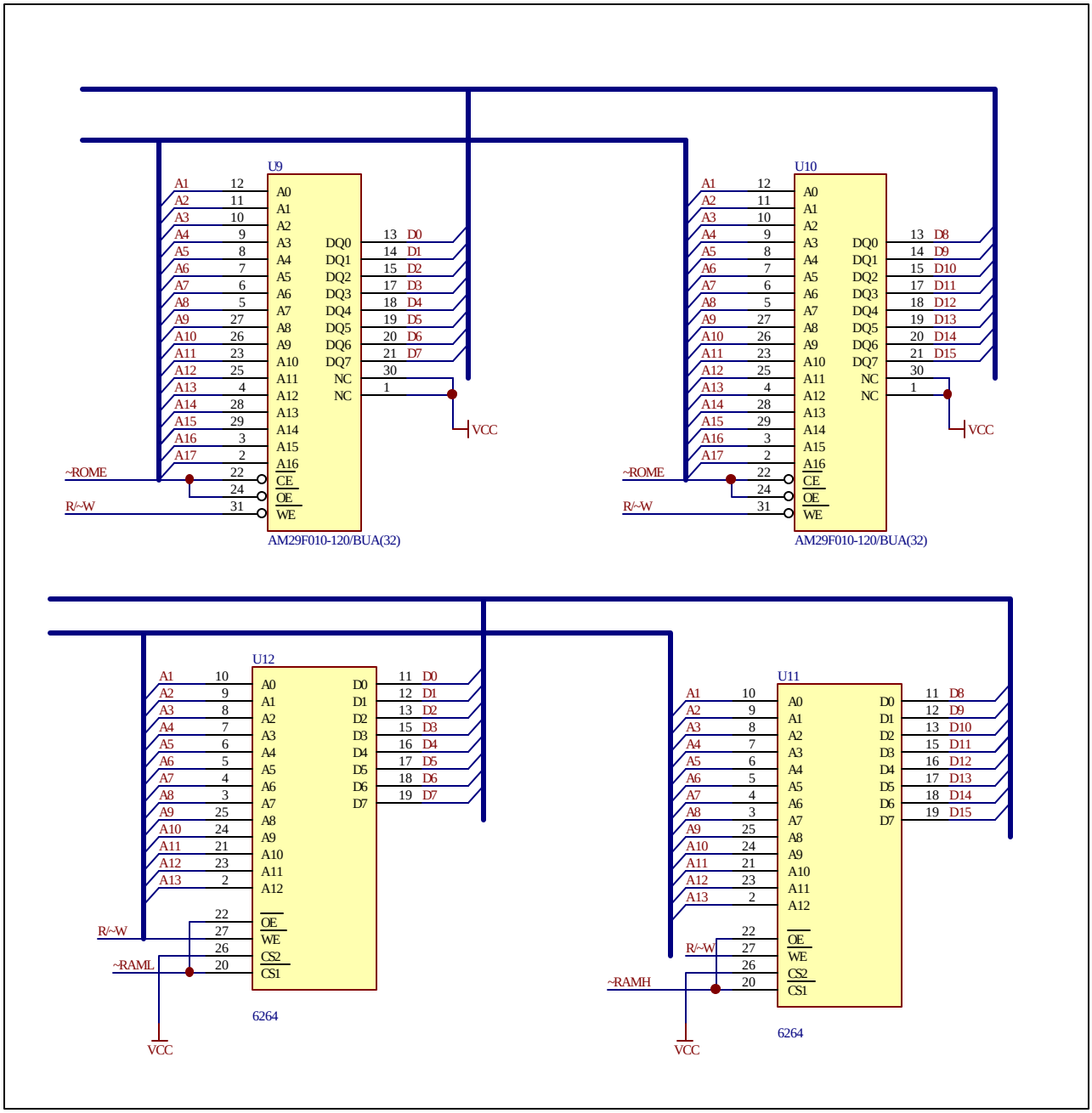
Figure 14. Read Register Window in Auto Demo Flow



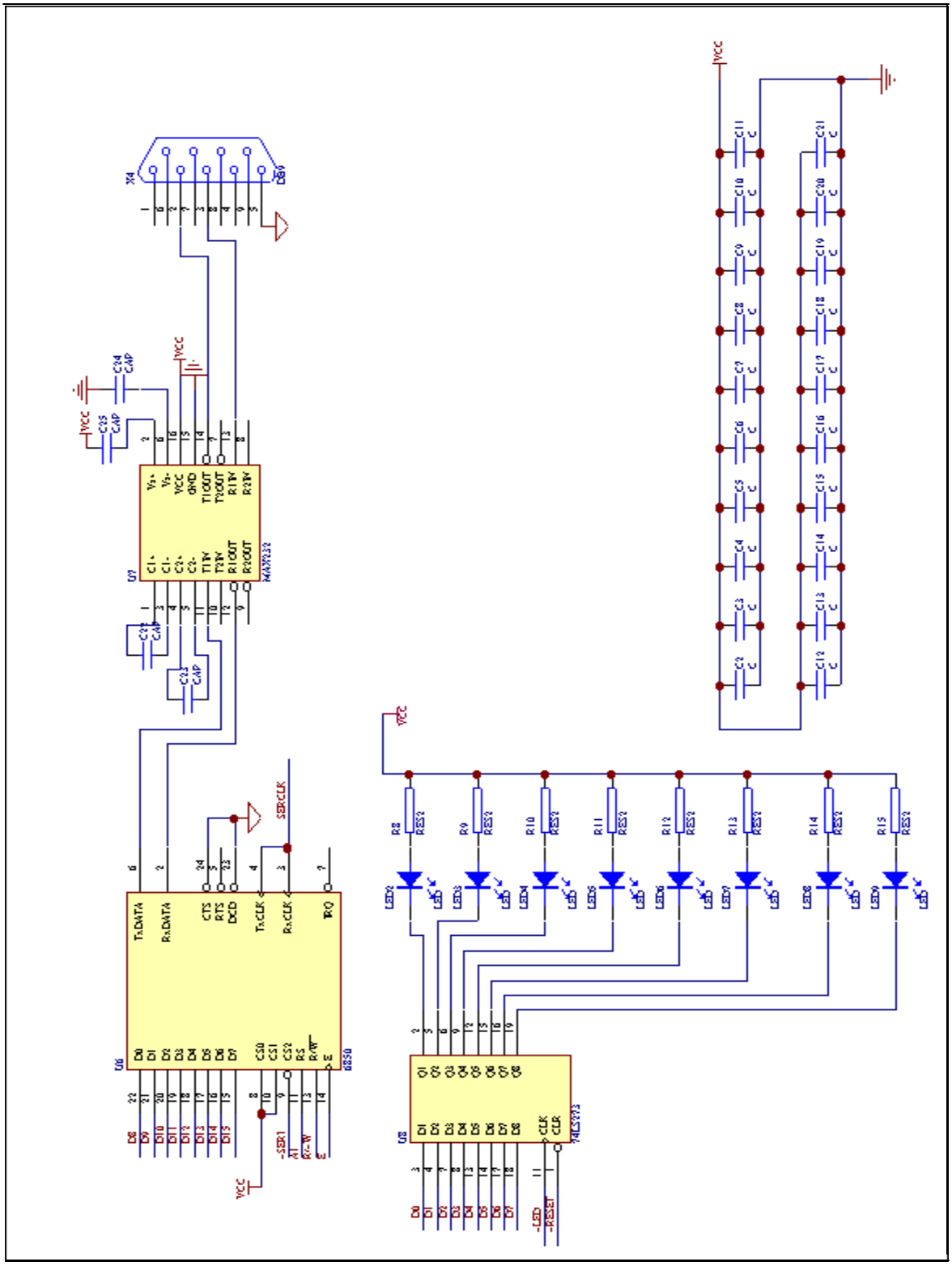
Appendix A. The Circuit Schematic of Demo Board



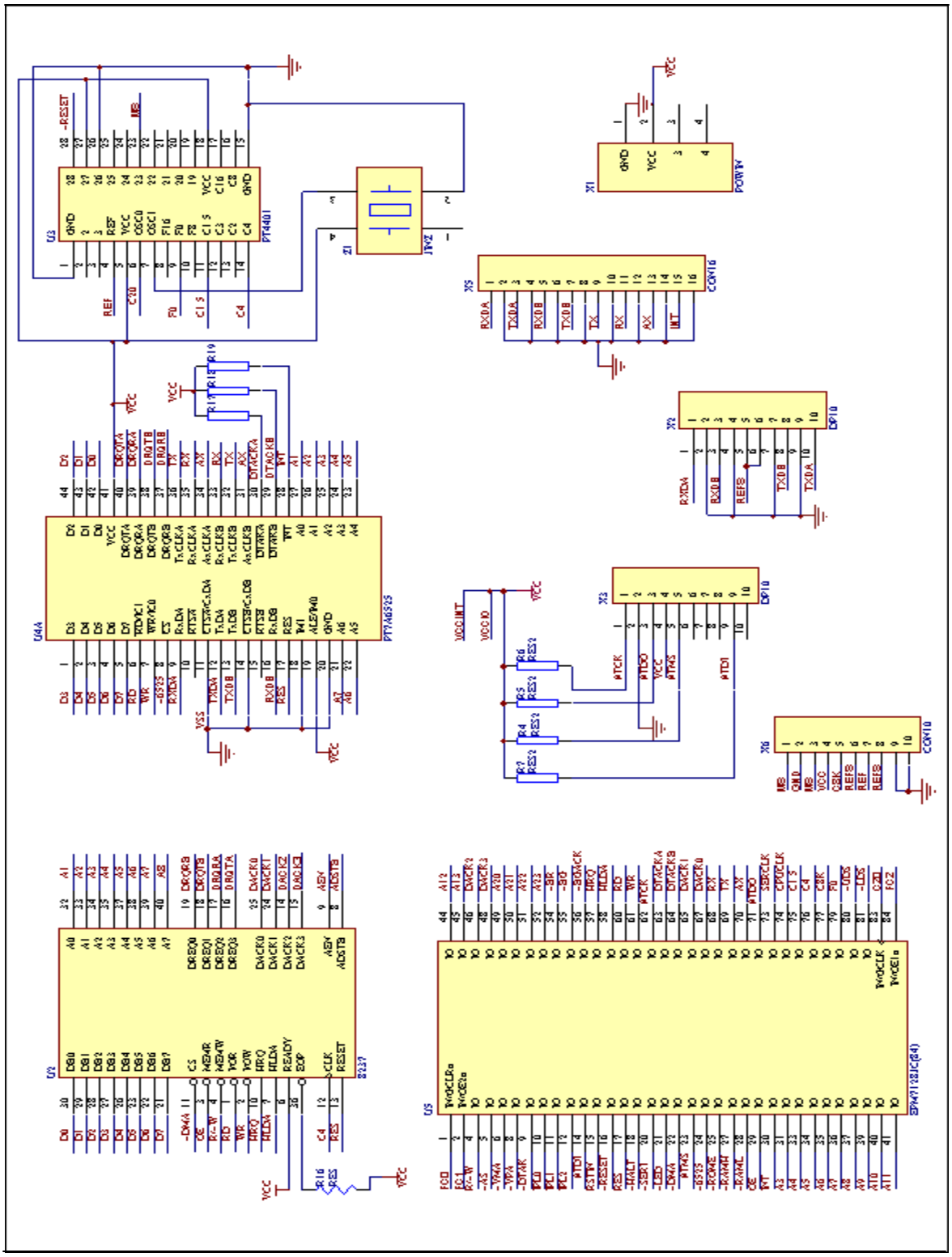
Schematics 1. The circuit of Microprocessor



Schematics 2. The ROM and RAM



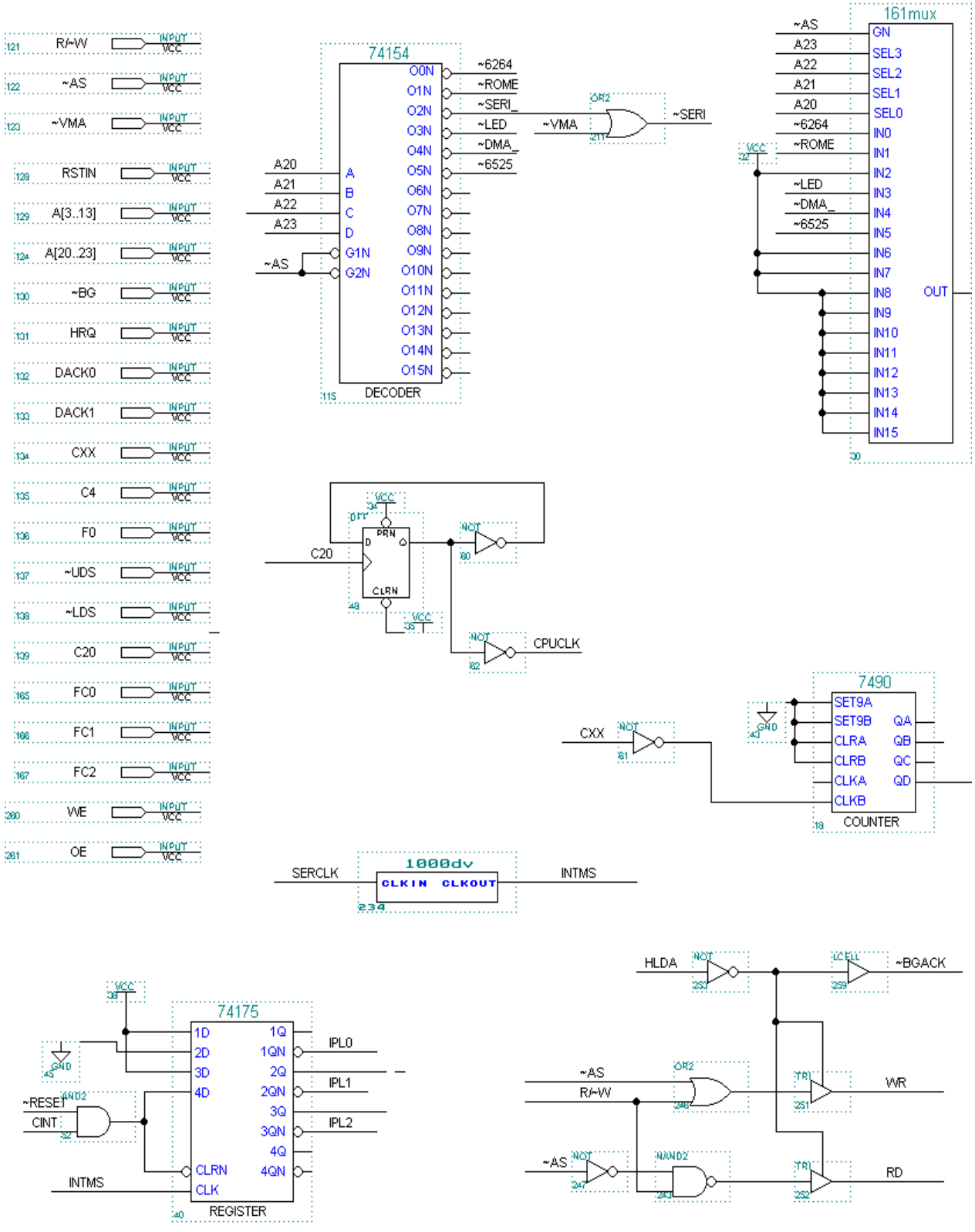
Schematics 3. The Peripheral Interface



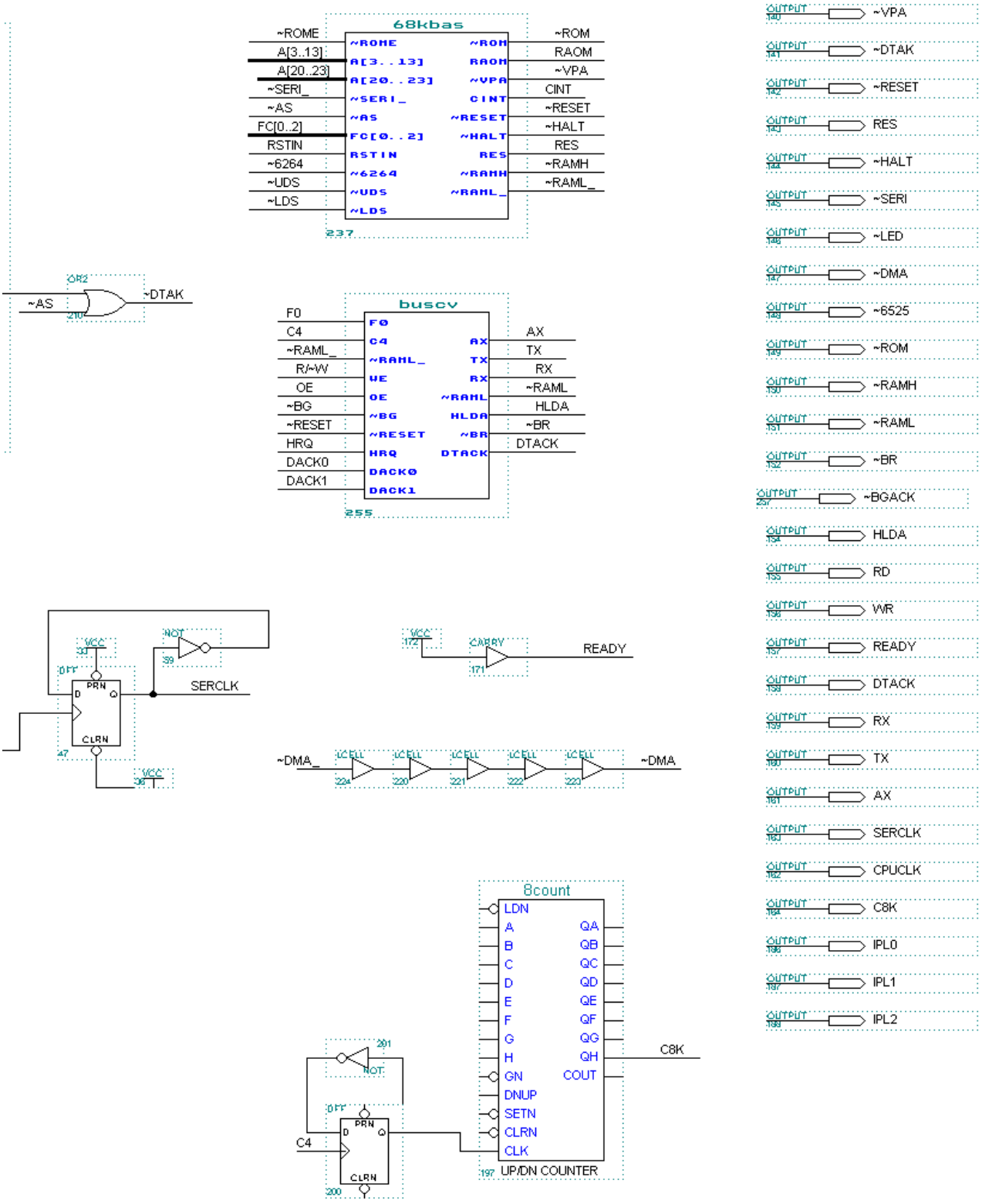
Schematics 4. PT7A6525, Clock, DMA Controller and EPLD circuit

Appendix B. The Logic Schematic of EPLD

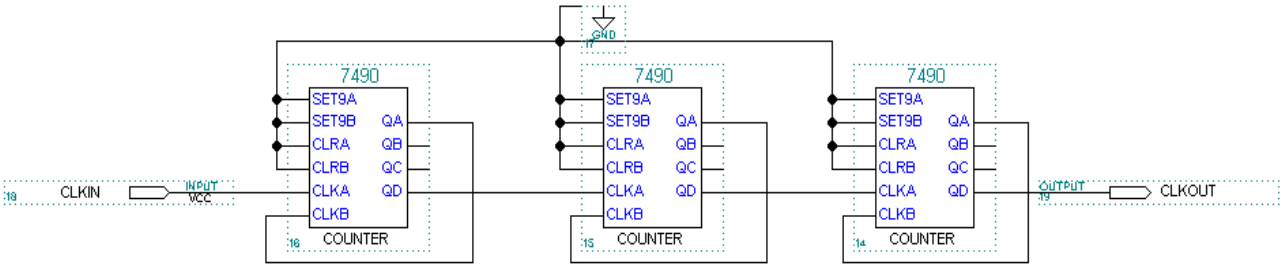
Schematics 1. The Top Hierarchy (left part)



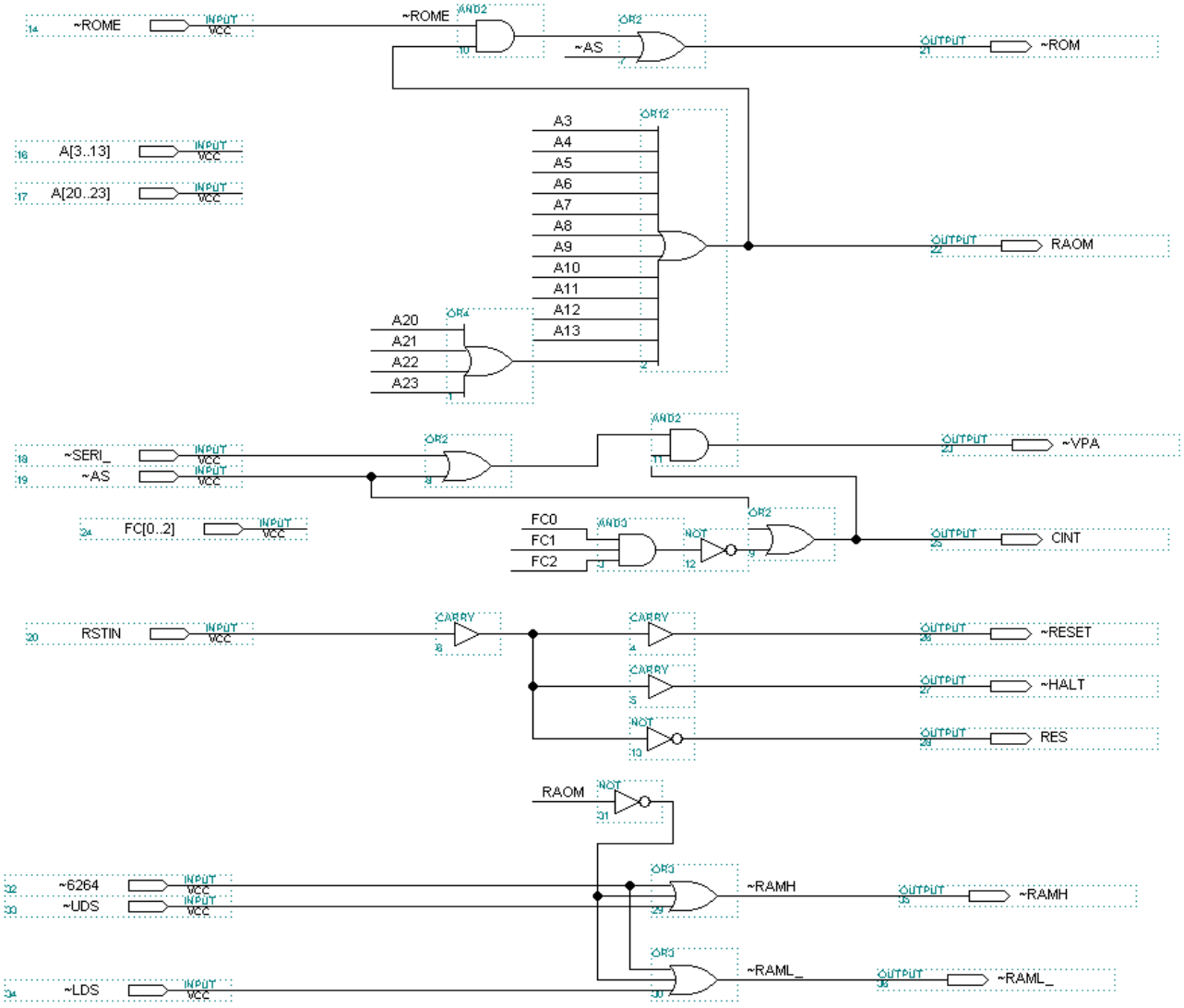
Schematics 1. The Top Hierarchy (right part)



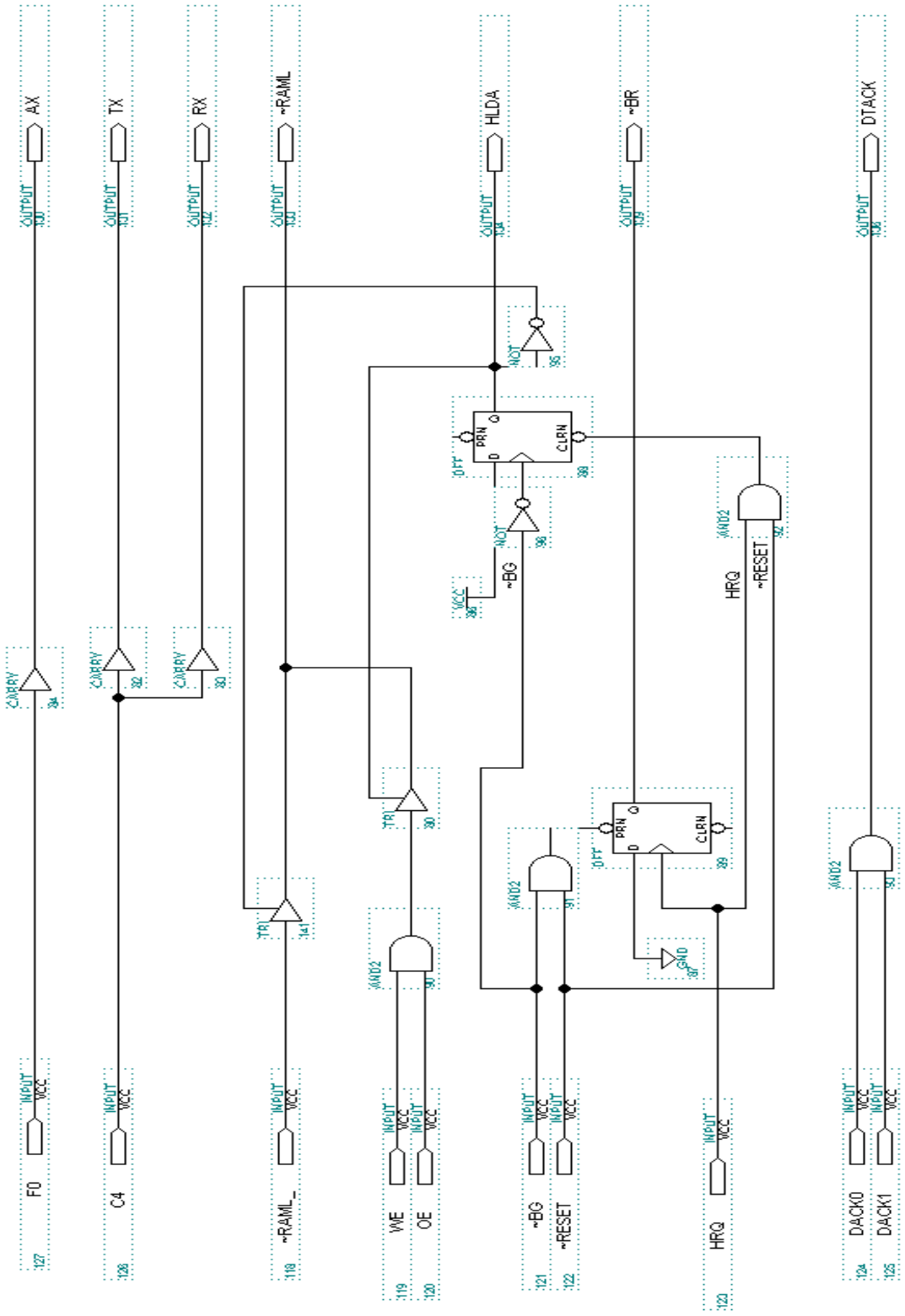
Schematics 2. Logic of 1000 Div



Schematics 3. Logic of 68kbas



Schematics 4. Logic of buscv



Appendix C The Source File of 89C51 Assemble Supervision Program

```
                org     $100000
*****************************
m6850cs         equ     $200000
m6850dr         equ     $200002
led             equ     $300000
rom             equ     $100000

sub_ec equ      $7e0
sp_buf equ      $800
adr_ld  equ     $810
clk_ld  equ     $818
second equ      $81a
minu    equ     $81c
ledzt   equ     $81e
trans   equ     $820
regist  equ     $822
*******************************
        dc.l    $0444
        dc.l    $100008
        bra     start
vector  dc.l    $100008,$100008,$100008,$100008,$100008,$100008,$100008,$100008
        dc.l    $100008,$100008,$100008,$100008,$100008,$100008,$100008,$100008
        dc.l    $100008,$100008,$100008,$100008,$100008,$100008,$100008,$100008
        dc.l    $100008,$100008,$100008,$100008,$100008,$100008,$100008,$100008
        dc.l    $100008,$100008,$100008,$100008,$100008,$100008,$100008,$100008

messag0 dc.b    $0d,$0a,'The PT7A6525 demo board Program booting...V1.1',$0d,$0a,0
start   move    #$2700,sr
        move    #$00fe,led
        nop
        lea.l   $786,a7
        move.l  a7,$444
        lea.l   vector(pc),a0
        lea.l   $000008,a1
        move.b  #39,d3
mvect   nop
        move.l  (a0)+,(a1)+
        add.b   #1,d0
        sub.b   #1,d3
        bne     mvect
        lea.l   time(pc),a0
        move.l  a0,$74
        lea.l   ecec(pc),a0
        move.l  a0,$08
        move.l  a0,$0c
```

```
        move.l  a0,$10
        move.b  #$43,m6850cs
        move.b  #$19,m6850cs                ; if equate #$19, that's even parity
        lea.l   messag0(pc),a0
        move    #$00fd,led
        bsr     disp
        lea     zfc1(pc),a0
        move    #$00fc,led
        bsr     disp
        move    #$00fb,led
        clr.l   d0
        clr.l   d1
        clr.l   d2
        clr.l   d3
        clr.l   d4
        clr.l   d5
        clr.l   d6
        clr.l   d7
        clr     sp_buf
        clr     second
        clr     ledzt
        clr     minu
        clr     trans
        move    #$00f8,led
        nop
        move    #$2400,sr
        move    #$00f5,led
        bra     read0
*****************************************
time    nop
        movem.l d0-d7/a0-a6,-(a7)
time00  sub.b   #1,clk_ld
*       cmp.b   #$aa,trans
*       beq     retun
        add.b   #1,second
        cmp.b   #10,second
        bhi     time1
        bra     retun
time1   add.b   #1,minu
        clr.b   second
        bra     times
        bra     times
****************************
times   nop
        cmp.b   #$0,ledzt
        beq     time2
```

```
        move.b  ledzt,d0
        lsl     #1,d0
        move.b  d0,ledzt
        bra     ledx
time2   move.b  #1,ledzt
        move.b  ledzt,d0
ledx    not.b   d0
        move.b  d0,led
        bra     retun
retun   movem.l (a7)+,a0-a6/d0-d7
        rte
ecec    move.l  -(a7),sub_ec
        move.l  -(a7),sub_ec+4
        move.l  -(a7),sub_ec+8
        move.l  (a7),sub_ec+$c
        bra     start
************************************************
read0   nop
read    nop
        clr.l   d0
        clr.l   d1
        clr.l   d2
        clr.l   d3
        clr.l   d4
        clr.l   d5
        clr.l   d6
        clr.l   d7
        btst.b  #0,m6850cs
        beq     read
        move.b  m6850dr,d6
        move.b  d6,m6850dr
        cmp.b   #$0d,d6
        beq     exit00
*       cmp.b   #'T',d6
*       beq     test
        cmp.b   #'\',d6
        beq     seea
        cmp.b   #'/',d6
        beq     seeb
        lea.l   sp_buf,a0
        add.b   #1,(a0)
        clr.l   d0
        move.b  (a0),d0
        move.b  d6,(a0,d0)
        cmp.b   #7,(a0)
        beq     www
```

```
        cmp.b   #8,(a0)
        bhi     exit
        bra     read
************************
seea    add.l   #$10,adr_ld
        move.l  #2,d2
        lea.l   adr_ld+1,a0
see4a   move.b  (a0)+,d0
        move.b  d0,d1
        and.b   #%11110000,d0
        lsr     #4,d0
        bsr     hexasc
see41   btst    #1,m6850cs
        beq     see41
        move.b  d0,m6850dr
        move.b  d1,d0
        and.b   #%00001111,d0
        bsr     hexasc
see42   btst    #1,m6850cs
        beq     see42
        move.b  d0,m6850dr
        dbf     d2,see4a
        move.l  adr_ld,a1
        bra     seeab
seeb    sub.l   #$10,adr_ld
        move.l  #2,d2
        lea.l   adr_ld+1,a0
see4b   move.b  (a0)+,d0
        move.b  d0,d1
        and.b   #%11110000,d0
        lsr     #4,d0
        bsr     hexasc
see43   btst    #1,m6850cs
        beq     see43
        move.b  d0,m6850dr
        move.b  d1,d0
        and.b   #%00001111,d0
        bsr     hexasc
see44   btst    #1,m6850cs
        beq     see44
        move.b  d0,m6850dr
        dbf     d2,see4b
        move.l  adr_ld,a1
        bra     seeab

****************************
```

```
exit    bsr     dela
        lea     sp_buf,a0
        clr.b   (a0)
        lea.l   zfc0(pc),a0
        bsr     disp
        bra     read
www     clr.l   d0
        lea     sp_buf,a0
        cmp.b   #7,(a0)
        bne     exit
        add     #1,a0
        move.b  (a0),d0
        cmp.b   #'W',d0             WMSDXRLUV
        beq     see
        cmp.b   #'M',d0
        beq     modi
        cmp.b   #'D',d0
        beq     seew
        cmp.b   #'X',d0
        beq     modiw
        cmp.b   #'R',d0
        beq     run
        cmp.b   #'O',d0
        beq     output
        cmp.b   #'I',d0
        beq     input
        cmp.b   #'L',d0
        beq     load
        cmp.b   #'l',d0
        beq     load
        bra     exit
run     bsr     word
        btst.b  #0,d5
        bne     exit
        jmp.l   (a1)
modi    bsr     dela
*       move.b  #$20,m6850dr
        bsr     word
        move.b  (a1),d0
loop60  btst    #1,m6850cs
        beq     loop60
        move.b  #$20,m6850dr
        move.b  d0,d1
        and.b   #%11110000,d0
        lsr     #4,d0
        bsr     hexasc
```

```
loop40  btst    #1,m6850cs
        beq     loop40
        move.b  d0,m6850dr
        move.b  d1,d0
        and.b   #%00001111,d0
        bsr     hexasc
loop50  btst    #1,m6850cs
        beq     loop50
        move.b  d0,m6850dr
        bsr     byte0
        move.b  d3,(a1)
        bra     exit00
**********************
input   nop
*       bsr     dela
*       move    #$20,m6850dr
        bsr     word
*       move.b  (a1),d0
loop160 btst    #1,m6850cs
        beq     loop160
*       move    #$20,m6850dr
        bsr     byte0
        move.b  d3,(a1)
        bra     read
**********************
output  nop
*       bsr     dela
*       move    #$20,m6850dr
        bsr     word
        move.b  (a1),d0
loop260 btst    #1,m6850cs
        beq     loop260
*       move    #$20,m6850dr
        move.b  d0,d1
        and.b   #%11110000,d0
        lsr     #4,d0
        bsr     hexasc
loop240 btst    #1,m6850cs
        beq     loop240
        move.b  d0,m6850dr
        move.b  d1,d0
        and.b   #%00001111,d0
        bsr     hexasc
loop250 btst    #1,m6850cs
        beq     loop250
        move.b  d0,m6850dr
```

```
        bra     read
**********************
modiw   bsr     dela
        move.b  #$20,m6850dr
        bsr     word
*       move.b  (a1),d0
*loop61  btst    #1,m6850cs
*       beq     loop61
*       move    #$20,m6850dr
*       move.b  d0,d1
*       and.b   #%11110000,d0
*       lsr     #4,d0
*       bsr     hexasc
*loop41  btst    #1,m6850cs
*       beq     loop41
*       move.b  d0,m6850dr
*       move.b  d1,d0
*       and.b   #%00001111,d0
*       bsr     hexasc
*loop51  btst    #1,m6850cs
*       beq     loop51
*       move.b  d0,m6850dr
        bsr     byte0
        move.b  d3,regist
        bsr     byte0
        move.bd3,regist+1
        move.w          regist,(a1)
        bsr     dela
        lea     sp_buf,a0
        clr.b   (a0)
        lea.l   zfc1(pc),a0
        bsr     disp
        bra     read
**********************
see     bsr     word
        btst.b  #0,d5
        bne     exit
        move.l  a1,adr_ld
seeab   move.b  #1,d2
jjjj    move.b #8,d7
loop6   btst    #1,m6850cs
        beq     loop6
        move.b  #$20,m6850dr
        move.b  (a1)+,d3
        move.b  d3,d0
        and.b   #%11110000,d0
```

```
        lsr     #4,d0
        bsr     hexasc
loop4   btst    #1,m6850cs
        beq     loop4
        move.b  d0,m6850dr
        move.b  d3,d0
        and.b   #%00001111,d0
        bsr     hexasc
loop5   btst    #1,m6850cs
        beq     loop5
        move.b  d0,m6850dr
        sub.b   #1,d7
        bne     loop6
        cmpi.b  #1,d2
        bne     loop7
loop9   btst    #1,m6850cs
        beq     loop9
        move.b  #'-',m6850dr
        move.b  #0,d2
        bra     jjjj
loop7   bra     exit00
*******************************
seew    bsr     word
loop6w  btst    #1,m6850cs
        beq     loop6w
        move.b    #$20,m6850dr
        move.w  (a1),d3
        move    d3,regist
        move.b  regist,d0
        and.b   #%11110000,d0
        lsr     #4,d0
        bsr     hexasc
loop4w  btst    #1,m6850cs
        beq     loop4w
        move.b  d0,m6850dr
        move.b  regist,d0
        and.b   #%00001111,d0
        bsr     hexasc
loop5w  btst    #1,m6850cs
        beq     loop5w
        move.b  d0,m6850dr
        clr.l   d0
        move.b  d3,d0
        and.b   #%11110000,d0
        lsr     #4,d0
        bsr     hexasc
```

```
loop8w  btst    #1,m6850cs
        beq     loop8w
        move.b  d0,m6850dr
        move.b  d3,d0
        and.b   #%00001111,d0
        bsr     hexasc
loop7w  btst    #1,m6850cs
        beq     loop7w
        move.b  d0,m6850dr
        bra     exit00
*****************************
load    bsr     dela
        move.b    #$20,m6850dr
        bsr     word
        move.l  a1,adr_ld
reada0  btst.b  #0,m6850cs
        beq     reada0
        move.b  m6850dr,d6
        move.b  d6,m6850dr
        bsr     bytea
        lsl.l   #4,d6
        move.b  d6,d3
reada1  btst.b  #0,m6850cs
        beq     reada1
        move.b  m6850dr,d6
        move.b  d6,m6850dr
        bsr     bytea
        or.b    d3,d6
        move.b  d6,d2
        cmp.b   #0,d6
        beq     ldpro
        bra     ldtn11
ldtn11  move.b  #100,clk_ld
reada2  cmp.b   #0,clk_ld
        beq     exit0a
        btst.b  #0,m6850cs
        beq     reada2
        move.b  m6850dr,d6
        move.b  #10,clk_ld
        bra     reada2
ldpro   clr.l   d0
        move.b  #1,d0
        move.l  adr_ld,a0
ldpro0  move.b  #100,clk_ld
reada3  cmp.b   #0,clk_ld
        beq     exit0a
```

```
        btst.b   #0,m6850cs
        beq      reada3
        move.b   #100,clk_ld
        move.b   m6850dr,d6
        move.b   d6,(a0)
        add      d0,a0
        bra      reada3
exit0a  bra      exit00
*******************************
exit00  bsr      dela
        lea      sp_buf,a0
        clr.b    (a0)
        lea.l    zfc1(pc),a0
        bsr      disp
        bra      read
*********************************
word    clr.l    d6
        clr.l    d5
        move.b   #0,d5
        lea      sp_buf,a0
        add      #2,a0
        bsr      byte
        lsl.l    #8,d6
        or.l     d3,d6
        move.l   d6,d5
        bsr      byte
        clr.l    d6
        move.b   d3,d6
        bsr      byte
        lsl.l    #8,d6
        or.l     d3,d6
        lsl.l    #8,d5
        lsl.l    #8,d5
        or.l     d6,d5
        movea.l  d5,a1
word0   rts
*******************************
byte    move.b   (a0),d3
        cmpi.b   #'F',d3
        bgt      exit0
        add      #1,a0
        cmpi.b   #'A',d3
        blt      rcmpe
        sub.b    #'7',d3
        bra      byte33
rcmpe   cmpi.b   #'9',d3
```

```
        bgt     exit0
        cmpi.b  #'0',d3
        blt     exit0
        sub.b   #'0',d3
byte33  move.b  (a0),d4
        cmpi.b  #'F',d4
        bgt     exit0
        add     #1,a0
        cmpi.b  #'A',d4
        blt     rcmp0
        sub.b   #'7',d4
        bra     byte44
rcmp0   cmpi.b  #'9',d4
        bgt     exit0
        cmpi.b  #'0',d4
        blt     exit0
        sub.b   #'0',d4
byte44  lsl     #4,d3
        or.b    d4,d3
        rts
exit0   move.b  #1,d5
        rts
**********************************
byte0   lea.l   m6850cs,a0
        btst.b  #0,(a0)
        beq     byte0
        move.b  2(a0),d3
        move.b  d3,2(a0)
        cmpi.b  #'F',d3
        bgt     err000
        cmpi.b  #'A',d3
        blt     rcmpe0
        sub.b   #'7',d3
        bra     byte330
rcmpe0  cmpi.b  #'9',d3
        bgt     err000
        cmpi.b  #'0',d3
        blt     err000
        sub.b   #'0',d3
byte330 btst.b  #0,(a0)
        beq     byte330
        move.b  2(a0),d4
        move.b  d4,2(a0)
        cmpi.b  #'F',d4
        bgt     err000
        cmpi.b  #'A',d4
```

```
        blt     rcmp00
        sub.b   #'7',d4
byte440 lsl     #4,d3
        or.b    d4,d3
        rts
rcmp00  cmpi.b  #'9',d4
        bgt     err000
        cmpi.b  #'0',d4
        blt     err000
        sub.b   #'0',d4
        bra     byte440
err000  clr.l   d3
        rts
**************************
hexasc  cmpi.b  #9,d0
        ble     ascz
        addi.b  #$07,d0
ascz    addi.b  #'0',d0
        rts
*************************************************************
byteb   dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          0
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          8
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          10
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          18
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          20
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          28
        dc.b    $00,$01,$02,$03,$04,$05,$06,$07         30
        dc.b    $08,$09,$ff,$ff,$ff,$ff,$ff,$ff          38
        dc.b    $ff,$0a,$0b,$0c,$0d,$0e,$0f,$ff          40
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          48
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          50
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          58
        dc.b    $ff,$0a,$0b,$0c,$0d,$0e,$0f,$ff          60
        dc.b    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff          68
*************************************************************
zfc0    dc.b    $0d,$0a,'command bad!',$0d,$0a,'>',0
zfc1    dc.b    $0d,$0a,'>',0
zf_ok   dc.b    'OK',0
zf_no   dc.b    'NO',0
disp            move #$0001,led
                movem.l         d0-d7/a0-a6,-(a7)
                nop
                move #$0002,led
disp1           cmp.b   #0,(a0)
                beq     disp2
                move.b  (a0)+,m6850dr
```

```
                move #$0003,led
                move    #400,d1
dela1           rol.l   #4,d2
                ror.l   #4,d2
                dbf     d1,dela1
                move #$04,led
                bra     disp1
disp2           movem.l  (a7)+,a0-a6/d0-d7
                move   #$05,led
                rts
************************************
dela            move.l  d1,-(a7)        ;1ms delay.
                move.l   d2,-(a7)
                move    #400,d1
dela2           rol.l   #4,d2
                ror.l   #4,d2
                dbf     d1,dela2
                move.l   (a7)+,d2
                move.l   (a7)+,d1
                rts
delay           move.l  d1,-(a7)
                move.l  #8000,d1        ;40ms
dela3           nop
                nop
                nop
                nop
                dbf     d1,dela3
                move.l   (a7)+,d1
                rts
************************************
bytea   and     #$00ff,d6
        lea.l   byteb(pc),a1
        move.b  (a1,d6),d6
        rts
******************************
        end
```

Notes

Pericom Technology Inc.

Email: support@pti.com.cn

China: No. 20 Building, 3/F, 481 Guiping Road, Shanghai, 200233, China
Tel: (86)-21-6485 0576 Fax: (86)-21-6485 2181

Asia Pacific: Unit 1517, 15/F, Chevalier Commercial Centre, 8 Wang Hoi Rd, Kowloon Bay, Hongkong
Tel: (852)-2243 3660 Fax: (852)- 2243 3667

U.S.A.: 2380 Bering Drive, San Jose, California 95131, USA
Tel: (1)-408-435 0800 Fax: (1)-408-435 1100