Technical Data

MCF 5272HDLCUG
Rev. 0, 2/2002

MCF5272 Soft HDLC
User's Guide

MOTOROLA
intelligence everywhere™

digital dna™

High Level Data Link Control (HDLC) is a bit-oriented Open Systems Interconnection (OSI) Layer 2 protocol commonly used in data communications systems. Many other common layer 2 protocols, for example, ISDN LAP-B, ISDN LAP-D, and Ethernet, are heavily based on HDLC.

Motorola has developed a soft HDLC framer/deframer function which is designed to run on an MCF5272 processor. The peripheral independent main software block is capable of running on any ColdFire® Version 2 (V2) based processor; however, the software object module, as delivered, assumes the presence of a number of tables in ROM, currently only present on the MCF5272 device.

This document provides customers with information on how to use this software, and how to integrate it into an MCF5272-based system. It describes the functionality and the interface provided to the customer. In addition, this guide outlines the profiling tests performed and the resulting performance analysis information.This document assumes familiarity with HDLC and the architecture of the MCF5272 processor.

This document contains the following topics:

Table 1 shows the acronyms, abbreviations and their meanings used through out this document.

Table 1. Acronyms and Abbreviations

Acronym	Meaning
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
GCI	General Circuit Interface
HDLC	High Level Data Link Control
ISDN	Integrated Services Digital Network
MIEL	Motorola India Electronics Ltd.
OSI	Open Systems Interconnection
Rx	Receive
Tx	Transmit

Table 2 lists the documents referenced in this report.

Table 2. References

Title	Order Number
MCF5272 User’s Manual	MCF5272UM/D
QMC Supplement to MPC68360 and MPC860 User’s Manual	QMCSUPPLEMENT/D

Part I Interface Description

The functional description provides a detailed overview of the MCF5272 SoftHDLC. Summarized are the software functionality, including the standard implemented HDLC Tx and HDLC Rx features, the delivery format, and tools used. The HDLC Tx framing and HDLC Rx deframing drivers are broken down to the parameter level, detailing the inputs and other parameters. Part I concludes with calling sequence demonstrations for initializing a channel, a transmit call, and a receive.

1.1 Software Functionality

This section highlights the transmit and receive features of the SoftHDLC for the MCF5272 interface control. Information regarding the delivery format is included. The tools used to compile the C code and test the SoftHDLC are also identified.

1.1.1 Standard Implemented

The soft HDLC function, as supplied by Motorola, implements an HDLC framer/deframer function. Following are lists of the HDLC transmit and receive features:

- HDLC Tx Features

  - Operates at 56 or 64 Kbps
  - CRC calculation option
  - Aborts transmission at any frame boundary
  - Fills output buffer with ones or flags as needed
- HDLC Rx Features
  - Operates at 56 or 64 Kbps
  - Enables or disables CRC checking
  - Reports number of CRC errors and aborts to calling function
  - Address recognition of up to three independent addresses per channel: two regular independent addresses, one independent address associated with a mask, and the broadcast address
  - Recognizes 0-, 8-, or 16-bit addresses
  - Restarts reception on any frame boundary

1.1.2 Delivery Format

The software is delivered to the customer in object format library (tixxx\libhdlc.a) ready to be linked together with the customer's own software. It consists of two main functions: HDLC_Tx_Driver and HDLC_Rx_Driver.

1.1.3 Tools Used

A DIAB C compiler (V4.3 Rev D) assembled the C code. Testing was performed on an HSEVB MCF5272 platform, using an SDS debugger (7.4).

1.2 HDLC Tx Driver

This function performs the HDLC framing. It reads input from a location in memory, completes the HDLC framing, and writes the data to another location in memory. The software has the capability to operate at two different bit rates (64 Kbps and 56 Kbps). The CRC calculation is optional at run-time.

Function prototype:

```
VOID HDLC_Tx_Driver (enumHdlcMessage eHdlcMessage,
            pstructHdlcTxChannelInfo psChannelInfo);
```

1.2.1 Parameters

This section identifies HDLC Tx driver parameters. Included are summaries of the main inputs and wModeControlWord, pstructHdlcStatus psHdlcStatus, and wStatusReturnWord.

1.2.1.1 Input

The two main inputs to this function are:

- eHDLCMessage—Indicates to the Tx driver the basic function to be performed
- psChannelInfo—A structure containing all the additional information required by the function

NOTE

All pointers and sizes passed to the function are assumed to be valid. No validation is done within the function.

1.2.1.1.1 eHDLCMessage

The possible values which can be passed to the function in this message field, and their meaning, are listed below:

- HDLC_INIT_CHANNEL—Initialize Tx HdlcContext parameters
- HDLC_SEND_PACKET—Prepare a bit stuffed HDLC packet
- HDLC_FILL_WITH_FLAGS—Fill the output buffer completely with flags
- HDLC_FILL_WITH_ONES—Fill the output buffer completely with ones

NOTE

The HDLC_Tx_Driver function must be called with HDLC_INIT_CHANNEL before a channel is opened on the Tx side. It should also be called when the operating bit rate is to be changed — it is not possible to change the bit rate without re-initializing the channels.

1.2.1.1.2 psChannelInfo

The pointer, psChannelInfo, points to the structure containing all the additional information required by the HDLC Tx driver function. The format of the structure is shown below.

```
typedef struct
  {
     unsigned char           *pbInputBuffer;
     unsigned short int      wMaxInputbufferSize;
     unsigned char           *pbOutputBuffer;
     unsigned short int      wOutputBufferSize;
     unsigned short int      wModeControlWord;
     void                    *psHdlcContext;
     pstructHdlcStatus psHdlcStatus;
     unsigned short int      wCrcErrorCount;
     unsigned short int      wAbortErrorCount;
  } structHdlcTxChannelInfo, *pstructHdlcTxChannelInfo;
```

Table 3 defines the structure items for the HDLC Tx driver.

Table 3. psChannelInfo Terminology Defined

Terminology	Definition
unsigned char *pbInputBuffer	Pointer to the input data which needs to be set up by calling function. Not modified by HDLC_Tx_Driver function. If this pointer is NULL then output buffer is filled with ones or flags, or returns without action (as defined in 1.2.1.1.3, “Tx wModeControlWord”).
unsigned short int wMaxInputbufferSize	Maximum input data size. Value not altered by Tx function, number of octets consumed is reflected in the status return structure.
unsigned char *pbOutputBuffer	Pointer to output data buffer. Updated by HDLC_Tx_Driver on return from function.

Table 3. psChannelInfo Terminology Defined (continued)

unsigned short int wOutputBufferSize	Output buffer capacity. Treated as a countdown counter and updated by the Tx function on return.
unsigned short wModeControlWord	16-bit control word. Conveys modes of operation. See 1.2.1.1.3, “Tx wModeControlWord.”
void pointer (32 bytes) *psHdlcContext	32-byte-pointer points to context structure which will be used by the function to maintain context between calls.
pstructHdlcStatus psHdlcStatus	Pointer to a structure containing status. See 1.2.1.1.4, “Tx pstructHdlcStatus psHdlcStatus.”

1.2.1.1.3 Tx wModeControlWord

The HDLC Tx driver wModeControlWord instruction format is defined in Figure 1.

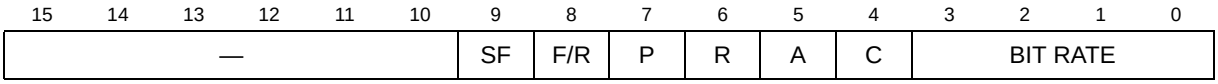

Figure 1. Tx wModeControlWord Instruction Format

Table 4 provides the Tx wModeControlWord field definitions.

Table 4. Tx wModeControlWord Field Definitions

Bits	Field	Description
15–10	—	Reserved. Cleared to zero.
9	SF	Share Frame. 0 Flag not shared between frames 1 Flag could be shared between frames
8	F/R	Fill Return. 0 Return on completion of input 1 Fill wth pattern (1s or flags)
7	P	Pointer. 0 Fill with flags 1 Fill with ones

Table 4. Tx wModeControlWord Field Definitions (continued)

Bits	Field	Description
6	R	Restart. 0 Normal Operation 1 Restart Transmission
5	A	Abort. 0 Normal Operation 1 Abort Frame
4	C	CRC. 0 No CRC 1 CRC On
3–0	BIT RATE	Bit Rate. Controls the bit rate of the Soft HDLC transmitter **Bit Rate** **Kbps** 0000–0110 Invalid 0111 56 1000 64 1001–1111 Reserved

1.2.1.1.4 Tx pstructHdlcStatus psHdlcStatus

The structure of the pstructHdlcStatus psHdlcStatus is defined below.

```
typedef struct
  {
     unsigned short int      wStatusReturnWord;
     unsigned short int      wBufferUtilisation;
  } structHdlcStatus, *pstructHdlcStatus;
```

Table 5 shows definitions of the related terminology.

Table 5. Tx pstructHdlcStatus psHdlcStatus Terminology Defined

Terminology	Definition
unsigned short int wBufferUtilisation	Number of octets processed by the core. Must be reset to zero when a new input buffer is provided.
unsigned short int wStatusReturnWord	16-bit word containing the return status. See 1.2.1.1.5, “Tx wStatusReturnWord.”

1.2.1.1.5 Tx wStatusReturnWord

Figure 2 defines the Tx wStatusReturnWord instruction format.

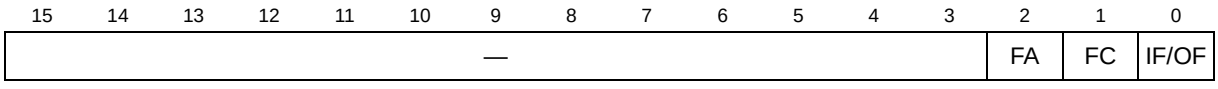

Figure 2. Tx wStatusReturnWord Instruction Format

Tx wStatusReturnWord fields are defined in Table 6.

Table 6. Tx wStatusReturnWord Field Definitions

Bits	Field	Description
15–3	—	Reserved.
2	FA	Frame Abort. 0 Normal Operation 1 Frame Aborted
1	FC	Frame Complete 0 Frame not completed 1 Frame completed (i.e. input buffer & CRC complete, closing flag may or may not yet have been added)
0	IF/OF	In Frame/Out Frame. 0 Begin a new frame 1 Midway through frame

1.3 HDLC Rx Driver

This driver performs the HDLC deframing. It reads input from a location in memory, completes the HDLC deframing, and writes the data to another location in memory. The software is capable of operating at several different bit rates.

Function prototype :

```
VOID HDLC_Rx_Driver (enumHdlcMessage eHdlcMessage,
                pstructHdlcRxChannelInfo psChannelInfo);
```

1.3.1 Parameters

This section identifies HDLC Rx driver parameters. Included are summaries of the main inputs and Rx wModeControlWord, pstructHdlcStatus psHdlcStatus, and Rx wStatusReturnWord.

1.3.1.1 Input

This function’s two main inputs include:

- eHDLCMessage—Tells the HDLC_Rx_Driver which basic function to perform
- psChannelInfo—Contains all the additional information the function requires to execute

1.3.1.1.1 eHDLC Message

Below are the possible values which can be passed to the function in this message field, and their meaning.

- HDLC_INIT_CHANNEL—Initialize Rx HdlcContext parameters.
- HDLC_SEND_PACKET—Prepare a bit stuffed HDLC packet.

NOTE

The HDLC_Rx_Driver function must be called with HDLC_INIT_CHANNEL before a channel is opened on the Rx side. It should also be called when the operating bit rate is to be changed — it is not possible to change the bit rate without re-initializing the channels.

1.3.1.1.2 psChannelInfo

The psChannelInfo pointer indicates a structure containing all the additional information required by the HDLC Rx Driver function. The format of the structure is shown below.

```
typedef struct
  {
     unsigned char           *pbInputBuffer;
     unsigned short int w    InputbufferSize;
     unsigned char           *pbOutputBuffer;
     unsigned short int       wMaxOutputBufferSize;
     unsigned short int       wModeControlWord;
     unsigned short int       awAddress [4];
     void*psHdlcContext;
     pstructHdlcStatus       psHdlcStatus;
     unsigned short int      wCrcErrorCount;
     unsigned short int      wAbortErrorCount;
  } structHdlcRxChannelInfo, *pstructHdlcRxChannelInfo;
```

Table 7 describes the psChannelInfo structure items for the HDLC Rx Driver.

Table 7. psChannelInfo Terminology Defined

Terminology	Definition
unsigned char *pbInputBuffer	Pointer to the input data, must be set up by calling function. Updated by Rx function on return.
unsigned short int wInputbufferSize	Input data size. Decremented by the amount of data bytes processed by the HDLC driver.
unsigned char *pbOutputBuffer	Pointer to output data buffer. Unchanged by the HDLC Rx Driver function.
unsigned short int wMaxOutputBufferSize	Maximun output buffer capacity. This variable must be at least one byte more than the maximum expected frame size.
unsigned short wModeControlWord	16-bit control word. Conveys modes of operation. See 1.3.1.1.3, “Rx wModeControlWord.”
unsigned short int awAddress[4]	An array of bytes containing addresses 1, 2, 3 and mask for address 3.
void pointer (32 bytes) *psHdlcContext	Pointer to context structure which will be used by the function to maintain context between calls.
pstructHdlcStatus psHdlcStatus	Pointer to a structure containing status. Described in 1.3.1.1.4, “Rx pstructHdlcStatus psHdlcStatus.”
unsigned short int wCrcErrorCount	Number of CRC errors detected. User can set this counter to zero at any time. Permits user to monitor line quality.
unsigned short int wAbortErrorCount	Number of aborts received. User can set this counter to zero at any time. Permits user to monitor line quality.

1.3.1.1.3 Rx wModeControlWord

The HDLC Rx driver wModeControlWord instruction format is defined in Figure 3.

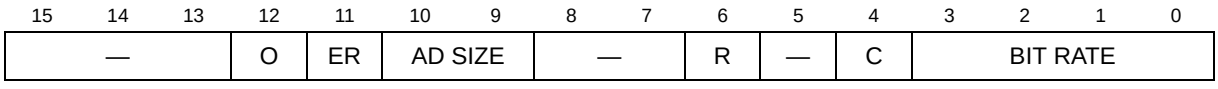

Figure 3. Rx wModeControlWord Instruction Format

Table 8 provides the Rx wModeControlWord field definitions.

Table 8. Rx wModeControlWord Field Definitions

Bits	Field	Description
15–13	—	Reserved. Cleared to 0.
12	O	Overflow. 0 Do not return 1 Return on overflow
11	ER	Error. 0 Do not return 1 Return on CRC error or Abort
10–9	AD SIZE	Address Size. Specifies size of address field on each frame. 00 0 01 1 10 2 11 Reserved
8–7	—	Reserved. Cleared to 0.
6	R	Restart. 0 Normal Operation 1 Restart
5	—	Reserved. Cleared to 0.
4	C	CRC. 0 No CRC 1 CRC On
3–0	BIT RATE	Bit Rate. Controls the bit rate of the Soft HDLC transmitter **Bit Rate** **Kbps** 0000–0110 Invalid 0111 56 1000 64 1001–1111 Reserved

1.3.1.1.4 Rx pstructHdlcStatus psHdlcStatus

The structure of the pstructHdlcStatus psHdlcStatus is defined below.

```
typedef struct
  {
     unsigned short int wStatusReturnWord;
     unsigned short int wBufferUtilisation;
  } structHdlcStatus, *pstructHdlcStatus;
```

Provided in Table 9 are definitions of the related terminology.

Table 9. Rx pstructHdlcStatus psHdlcStatus Terminology Defined

Terminology	Definition
unsigned short int wBufferUtilisation	Number of output octets written to the output buffer, updated by the Rx function. Should be reset to zero, by calling function, when a new output buffer pointer is given.
unsigned short int wStatusReturnWord	16-bit word containing the return status. See 1.3.1.1.5, “Rx wStatusReturnWord.”

1.3.1.1.5 Rx wStatusReturnWord

Figure 4 defines the Rx wStatusReturnWord instruction format.

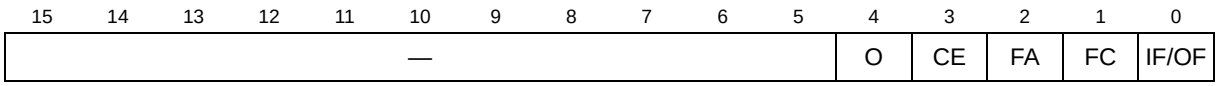

Figure 4. Rx wStatusReturnWord Instruction Format

The Rx wStatusReturnWord fields are defined in Table 10

Table 10. Rx wStatusReturnWord Field Definitions

Bits	Field	Description
15–5	—	Reserved.
4	O	Overflow. 0 Normal Operation 1 Output buffer overflow
3	CE	CRC Error. 0 Normal Operation 1 CRC Error detected
2	FA	Frame Abort. 0 Normal Operation 1 Frame Aborted
1	FC	Frame Complete. 0 Frame not completed 1 Frame completed
0	IF/OF	In frame/Out of Frame. 0 Out of frame 1 In frame

1.4 Calling Sequence

The following section demonstrates how the modules should be called. The calling sequence is composed of three steps: initializing a channel, transmitting, and receiving.

1.4.1 Initializing a Channel

Before calling the HDLC Tx and Rx functions with the initialization flag, it is necessary to set up *psHdlcContext, structTxHdlcChannelInfo, and structRxHdlcChannelInfo and set the bit rate in the wModeControlWord. Below is a sample calling sequence which performs each of these tasks:

```
WORD  wBitRate = 56;
structTxHdlcChannelInfo sTxChannelInfo;
structRxHdlcChannelInfo sRxChannelInfo;
void *psTxHdlcContext;
void *psRxHdlcContext;
psTxHdlcContext =  malloc(40);
psRxHdlcContext =  malloc(40);
sTxChannelInfo.psHdlcContext = psTxHdlcContext;
sRxChannelInfo.psHdlcContext = psRxHdlcContext;
sTxChannellnfo.wModeControlWord = wBitRate >> 3;
sRxChannelInfo.wModeControlWord = wBitRate >> 3;
HDLC_Tx_Driver (HDLC_INIT_CHANNEL,&sTxChannelInfo);
HDLC_Rx_Driver (HDLC_INIT_CHANNEL,&sRxChannelInfo);
```

1.4.2 Illustration of a Transmit Call

The HDLC_Tx_Driver needs to be initialized, as shown in 1.4.1, “Initializing a Channel,” before making a call to bit stuff data.

```
WORD  wBitRate = 56;
structTxHdlcChannelInfo sTxChannelInfo;
void *psTxHdlcContext;
structHdlcStatus   sTxHdlcStatus;
char*pbInData, *pbOutData;
pbInData = malloc(32);
pbOutData = malloc(64);
sTxChannelInfo.psHdlcContext = psTxHdlcContext;
sTxChannellnfo.wModeControlWord = wBitRate >> 3;
sTxChannelInfo.psHdlcStatus = &sTxHdlcStatus;
sTxChannelInfo.wModeControlWord |=
     (TX_CONTROL_INSERT_CRC |    TX_CONTROL_FILL_WITH_FLAGS);
sTxChannelInfo.pbInputBuffer = pbInData;
sTxChannelInfo.pbOutputBuffer = pbOutData;
sTxChannelInfo.wMaxInputBufferSize = 32;
sTxChannlInfo.wOutputBufferSize = 64;
     HDLC_Tx_Driver (HDLC_SEND_PACKET,&sTxChannelInfo);
```

In this call scenario, the status on return would be:

```
sTxChannelInfo.psHdlcStatus->wBufferUtilisation = 32;
sTxChannelInfo.psHdlcStatus->wStatusReturnWord = (STATUS_FRAME_COMPLETE);
sTxChannelInfo.wOutBufferSize = ZERO;
```

1.4.3 Illustration of a Receive

HDLC_Rx_Driver needs to be initialized before making a call to de-stuff data.

```
WORD  wBitRate = 56;
structRxHdlcChannelInfo sRxChannelInfo;
```

```
void *psRxHdlcContext;
structHdlcStatus   sRxHdlcStatus;
char*pbInData, *pbOutData;
pbInData     = malloc(64);
pbOutData    = malloc(34);
sRxChannelInfo.psHdlcContext = psRxHdlcContext;
sTxChannellnfo.wModeControlWord = wBitRate >>3;
sRxChannelInfo.psHdlcStatus = &sRxHdlcStatus;
sRxChannelInfo.wModeControlWord |= (RX_CONTROL_CHECK_CRC |
 RX_CONTROL_ADDRESS_SIZE_ZERO |
 RX_CONTROL_RETURN_ON_OVERFLOW);
sRxChannelInfo.pbInputBuffer = pbInData;
sRxChannelInfo.pbOutputBuffer = pbOutData;
sRxChannelInfo.wInputBufferSize = 64;
sRxChannlInfo.wMaxOutputBufferSize = 34;
        HDLC_Rx_Driver (HDLC_SEND_PACKET,
     &sRxChannelInfo);
```

In this call scenario, the status on return would be:

```
sTxChannelInfo.psHdlcStatus->wBufferUtilisation  =  34;  (including  2  CRC
Octets)
sTxChannelInfo.psHdlcStatus->wStatusReturnWord = (STATUS_FRAME_COMPLETE);
sTxChannelInfo.wInputBufferSize = 24;
```

Part II Functional Tests

This section provides a summary of all testing completed on the SoftHDLC module for the MCF5272. In addition to detailing the SoftHDLC for MCF5272 functionality tests, it documents the profiling, or performance, testing conducted on the HDLC software developed by Motorola for the MCF5272 device.

The functionality test objective is to test all the implemented features of the SoftHDLC routine, with various data patterns and buffer sizes, and also to test the routine with a hardware implementation of the HDLC protocol, to verify conformity. Three distinct types of tests were performed, including:

- ColdFire unit testing (see Section 2.1)
- Conformance testing (see Section 2.2)
- MCF5272 testing (see Section 2.3)

The ColdFire unit testing was completed prior to the MCF5272 silicon release to verify the performance with respect to the specification. The conformance testing ensures that the SoftHDLC implementation is operable with a ‘standard’ HDLC device. The MCF5272 testing verifies the operation of the code on the MCF5272 device, including verification of the ROM look-up tables.

The profiling test objective (see Section Part III) was to determine CPU cycle consumption for both standard and non-standard scenarios. In addition to identifying routine performance standards, dependent on bit rates of either 16, 56, and 64 Kbps and transmit output buffer sizes of either 16, 32, and 64 bytes, the tests also evaluated performance in the following scenarios:

- Small buffer size
- Different memory map
- Different frame size

2.1 ColdFire Unit Testing

The SoftHDLC routine and the associated test routines were cross-compiled using the DIAB compiler. The generated executable was loaded and executed on the target platform (MCF5206E Evaluation Board, Rev. A 1.3) using the SDS single step on-chip tool. The ColdFire unit testing was performed with the SoftHDLC routine operating in loopback mode — that is, rerouting the transmit routine output to the receive function. A set of eight test cases were designed to test the various features.

2.1.1 Test Case 01

Test case 01 tests to confirm CRC insertion and verification, filling the remaining output buffer with flags, and a zero address size.

2.1.1.1 Features Tested

- Tx side—CRC insertion, fill with flags
- Rx side—CRC verification, address size zero

2.1.1.2 Description

This test case uses an input sequence of 1024 bytes, either set to all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit routine's input and output buffer sizes are fixed at equal size. If the previous data chunk is fully exhausted, a new block of data is provided to the transmit function. The transmit routine is configured to fill the remaining output buffer with HDLC flags if the input buffer is fully exhausted. The transmit routine is configured to insert CRC for every frame. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine configuration checks CRC for every frame, sets the address size to zero, and returns on overflow. A "while" loop calls the receive routine until the buffer received from transmit is fully exhausted.

When the receive routine receives a complete frame, the received frame is compared with the transmitted frame.

2.1.1.3 Test Pass Criterion

The received frame data bytes must be identical to those in the transmitted frame for every frame in the test sequence.

2.1.2 Test Case 02

Test case 02 tests to confirm CRC insertion and verification, broadcast address insertion, one 8-bit address size, and return on completion of the input buffer.

2.1.2.1 Features Tested

- Tx Side—CRC insertion, broadcast address insertion, return on completion of input buffer
- Rx Side—CRC verification, address size equals one (8-bit)

2.1.2.2 Description

This test case uses an input sequence of 1024 bytes, either set to all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit routine's input and output buffer sizes are fixed at equal size. If the previous data chunk is fully exhausted, a new block of data is provided to the transmit function. The transmit routine is configured to return on completion of the input buffer. A one-byte broadcast address is inserted in every frame. The transmit routine inserts CRC for every frame. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine configuration checks CRC for every frame, sets the address size to eight bits, and returns on overflow. Broadcast address comparison is performed. A "while" loop calls the receive routine until the buffer received from transmit is fully exhausted.

Whenever receive routine receives a complete frame, the frame received is compared with the frame transmitted to ensure data integrity.

2.1.2.3 Test Pass Criterion

The data bytes prior to transmission and bytes received after reception must match for every frame.

2.1.3 Test Case 03

Test case 03 tests to confirm CRC insertion and verification, broadcast address insertion, and a 16-bit address size.

2.1.3.1 Features Tested

- Tx Side—CRC insertion, broadcast address insertion, fill with ones
- Rx Side—CRC verification, address size two (16-bit)

2.1.3.2 Description

This test case uses an input sequence of 1024 bytes, either set to all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit routine's input and output buffer sizes are a fixed equal size. If the previous data chunk is fully exhausted, a new block of data is provided to the transmit function. When the input has been exhausted, the remaining output buffer space is filled with ones. In this test case, every input chunk requires two Tx calls to be exhausted fully. A two-byte broadcast address is inserted in every frame. The transmit routine is also configured to insert CRC in every frame. Additional configuration options include inserting share flags between frames or an opening and closing flag for every frame. (The pre-processor switch, SHARED_FLAG, when defined, configures Tx in shared flag mode.)

The receive routine configuration checks CRC for every frame, sets the address size to sixteen bits, and the returns on overflow. Broadcast address comparison is performed. A "while" loop calls the receive routine until the buffer received from transmit is fully exhausted.

When receive routine receives a complete frame, the frame received is compared with the frame transmitted to ensure data integrity.

2.1.3.3 Test Pass Criterion

The data bytes prior to transmission and bytes received after reception must match for every frame.

2.1.4 Test Case 04

Test case 04 confirms that a frame drop occurs when there is a CRC error on the Rx side.

2.1.4.1 Features Tested

Frame drop on CRC error on Rx side.

2.1.4.2 Description

This test case uses an input sequence of 1024 bytes, set to either all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit routine's input and output buffer sizes are a fixed equal size. If the previous data chunk is fully exhausted, a new block of data is provided to the transmit function. The Tx routine is configured to fill the output buffer with ones, when the input buffer is exhausted but the output buffer is not yet filled. The transmit routine configuration does not insert CRC for every frame. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine configuration checks CRC for every frame, and returns on either overflow or error. A "while" loop calls the receive routine until the buffer received from transmit is fully exhausted. In this test case, the Rx routine assumes the last two data bytes as CRC, since CRC is set to 'off' in Tx side and 'on' in Rx side, and CRC check fails for every frame.

When the complete input sequence is exhausted, the CRC error counts are compared for the Tx and Rx side.

2.1.4.3 Test Pass Criterion

The error counts on the Tx and Rx sides must match, ensuring that each Tx frame is fully received and dropped.

Note: In this testcase, we have fixed the Tx input buffer at 32, for ZERO_INPUT. If the buffer size is two (16 bits), the Rx side still receives a valid frame, even though CRC was not inserted on the Tx side.

2.1.5 Test Case 05

Test case 05 confirms that a frame drop occurs when there is an abort on the Tx side.

2.1.5.1 Features Tested

Frame drop on Rx side due to abort on Tx side.

2.1.5.2 Description

This test case uses an input sequence of 1024 bytes, set to either all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit routine's input and output buffer sizes are a fixed equal size. If the previous data chunk is fully exhausted, a new block of data is provided to the transmit function. In this test case, every input chunk requires two Tx calls to be

exhausted fully. Before the second call, the Tx routine is configured to abort this particular frame and resets on completion of a successful abort. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine is configured to return on either overflow or error. A “while” loop calls the receive routine until the buffer received from transmit is fully exhausted. In this test case the Rx routine receives no valid frame as each frame is aborted mid-way.

When the complete input sequence is exhausted, we compare the abort error counts on Tx and Rx side.

2.1.5.3 Test Pass Criterion

The abort error counts on Tx and Rx side need to match, ensuring that each Tx frame is successfully aborted and dropped.

2.1.6 Test Case 06

Test case 06 confirms transmission restart..

2.1.6.1 Features Tested

Restart transmission in Tx side

2.1.6.2 Description

This test case uses an input sequence of 1024 bytes, set to either all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The output buffer size is selected to ensure that the entire input buffer is exhausted in one Tx call. To simulate a frame loss situation, this buffer is not given to Rx. The next time the Tx routine is called with the same input pointer, a transmission restart is requested. The transmit routine is configured to insert CRC for every frame. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine is configured to check CRC for every frame, and the routine is configured to return on either overflow or error. A “while” loop calls the receive routine until the buffer received from transmit is fully exhausted.

Whenever the Rx routine receives a complete frame, the frame received is compared with the frame transmitted to ensure data integrity.

2.1.6.3 Test Pass Criterion

The data bytes prior to transmission and bytes received after reception must match for every frame.

2.1.7 Test Case 07

Test case 07 tests CRC insertion and verification, address insertion and comparison, Tx flag fill, and address size two.

2.1.7.1 Features Tested

- Tx Side—CRC insertion, address insertion, fill with flags
- RX Side—Address comparison, CRC verification, address size two

2.1.7.2 Description

This test case uses an input sequence of 1024 bytes, either set to all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit routine's input and output buffer sizes are fixed at equal size. If the previous data chunk is fully exhausted, a new block of data is provided to the transmit function. The Tx routine is configured to fill the remaining output buffer with HDLC flags if the input buffer is fully exhausted. Specific address bytes are added to each frame prior to transmission.The transmit routine is configured to insert CRC for every frame. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine is configured to check CRC for every frame, and is configured to return on overflow. The Rx is configured to receive frames with only those addresses inserted on the Tx side. The address size is fixed at 2. A "while" loop calls the receive routine until the buffer received from transmit is fully exhausted.

Whenever the Rx routine receives a complete frame, the frame received is compared with the frame transmitted to ensure data integrity.

2.1.7.3 Test Pass Criterion

The data bytes prior to transmission and bytes received after reception must match, for every frame.

2.1.8 Test Case 08

Test case 08 tests the operation of the "continue on overflow" Rx feature.

2.1.8.1 Features Tested

"Continue on overflow" Rx feature.

2.1.8.2 Description

This test case uses an input sequence of 1024 bytes, either set to all 0xFFs, 0x00s or any random data patterns. The input sequence data is sent in small blocks to the transmit routine. The transmit output buffer is selected to ensure that the entire input buffer is exhausted by one Tx call. The transmit routine is configured to insert CRC for every frame. The routine can either be configured with share flags between frames or opening and closing flags inserted for every frame.

The receive routine is configured to continue on overflow. The receive output buffer size is selected to ensure that an overflow occurs for every Rx call. A "while" loop calls the receive routine until the buffer received from transmit is fully exhausted.

As a result of output overflow, the Rx receives no frames, and a counter increments whenever Rx sets the overflow error in the status return word.

2.1.8.3 Test Pass Criterion

The number of transmitted frames must match the overflow count.

2.2 Conformance Testing

To ensure conformance to the HDLC standard, the C code for this implementation was tested with an HDLC capable device. The MPC860MH, a mature device used in many applications, was selected because it has a hardware HDLC implementation as part of its communication core. This conformance test verified the bit stuffing and un-stuffing functionality and the 16-bit CRC features. Address comparison functionality could not be tested for conformance in this test environment.

To make use of existing hardware, the C code for this implementation was compiled for a DSP56300, and this DSP communicated with an MPC860MH using HDLC protocol. This test sequence is shown in Figure 5., “Conformance Test Sequence.

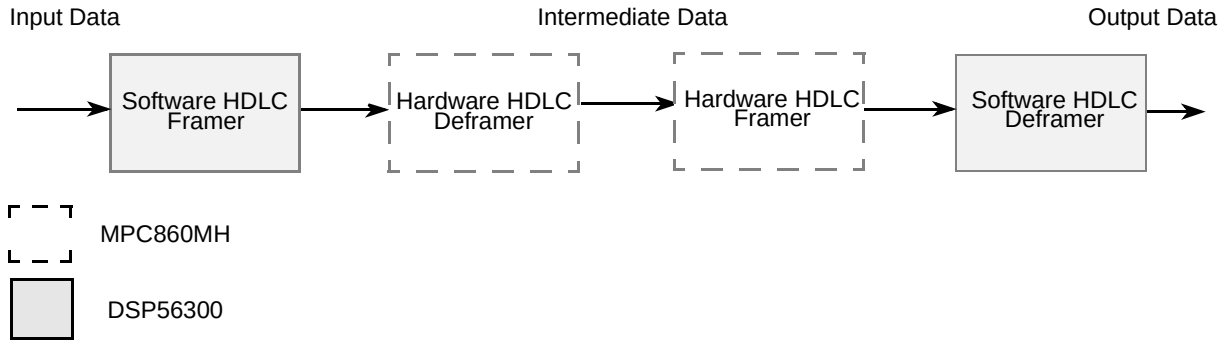

Figure 5. Conformance Test Sequence

For each test sequence, the input data, intermediate data, and output data were logged and compared. The recorded data was identical at each of these three test points.

2.3 MCF5272 Testing

All of the above tests have been verified on the MCF5272.

Part III Profiling Test

This section defines the test setup for the performance tests conducted on the HDLC software written by MIEL. The features tested are identified, the test case and test pass criterion described, and test results and comments are listed. In addition to determining standard performance counts, based on buffer size, bit rate and the number of calls per second, counts were determined for small buffer performance, different memory map performance, and different frame size performance. Also provided is information regarding modifying parameters in the profiling program and the conclusion reached by the testers.

3.1 Test Setup

The SoftHDLC routine and the associated test routines were cross-compiled using the DIAB tools (version 4.3 release d). The generated executable was loaded and executed on the MCF5272 silicon evaluation board using the single-step debugger (SDS), version 7.4. The routine was profiled using modified versions of test case 07 (hdlctest07.c) and the profiling program (hdlcprof.c, hdlctst.h). The HDLC Profiling Test, written in C, was used with the parameters listed below.

Default assumptions:

- Cache on
- On-chip ROM look-up tables

Unless otherwise stated, the following parameters also apply:

- Bit rate is 64 Kbps
- Address size = 2
- Transmitted data = $00 for no bit stuffing (set by default in the profiling program)
- Memory configuration per the following:
  - — Context data—internal SRAM (Read/Write: 1 CPU cycle)
  - — Framed data—internal SRAM (Read/Write: 1 CPU cycle)
  - — Unframed data—SDRAM (Write: (7 + 1 + 1 + 1) - Read: (9 + 1 + 1 + 1) ÷ maximum wait state – for example, case of page miss)

3.2 HDLC Profiling Test Description

Test case 07, originally designed for the functionality test phase, was used for the HDLC profiling test. See 2.1.7.1, “Features Tested” for a list of the features tested. Refer to 2.1.7.2, “Description” for a detailed test case description and to 2.1.7.3, “Test Pass Criterion” for the pass criterion used.

3.3 Standard Performances

This test’s goal is to determine the CPU cycle consumption for standard scenarios of the HDLC routine for different bit rates and different Tx output buffer sizes. Table 11 lists the standard performance results.

Table 11. Standard Performance Results

Buffer Size (Bytes) [1]	Bit Rate (Kbps) [2]	Number of Calls per Second [3]	Tx Count per Call (Cycles) [4]	Rx Count per Call (Cycles) [5]	Tx Count @ R Kbps (MCycles) [6]	Rx Count @ R Kbps (MCycles)	Total Count (Tx + Rx) @ R Kbps (MCycles) [7]
16	**56**	437.50	2837	2750	1.24	1.20	**2.44**
	64	500.00	3153	3118	1.58	1.56	**3.14**
	16	125.00			0.39	0.39	**0.78**
32	**56**	218.75	3770	4307	0.82	0.94	**1.77**
	64	250.00	4270	4638	1.07	1.16	**2.23**

Table 11. Standard Performance Results

Buffer Size (Bytes) [1]	Bit Rate (Kbps) [2]	Number of Calls per Second [3]	Tx Count per Call (Cycles) [4]	Rx Count per Call (Cycles) [5]	Tx Count @ R Kbps (MCycles) [6]	Rx Count @ R Kbps (MCycles)	Total Count (Tx + Rx) @ R Kbps (MCycles) [7]
	16	62.50			0.27	0.29	**0.56**
64	**56**	109.38	4316	4514	0.47	0.49	**0.97**
	64	125.00	4270	4638	0.53	0.58	**1.11**
	16	31.25			0.13	0.14	**0.28**

1 Size (in bytes) of the output Tx driver
2 Bit rate of the HDLC channel
3 Calculated as follows: (bit rate) ÷ (8 × buffer size)
4 Maximum CPU cycle for one call to the Tx driver
5 Maximum CPU cycle for one call to the Rx driver
6 Calculated as follows: (number of calls per second) × (Tx count per call), it represents the CPU cycle consumption for the Tx driver, at R Kbps (R = 16, 56, or 64)
7 CPU cycle consumption for both Tx and Rx drivers, at R Kbps (R = 16, 56, or 64)

3.3.1 Comments

Note that the CPU cycle consumption for bit rate equal to 16 Kbps is the CPU cycle consumption for bit rate equal to 64 Kbps divided by 4.

For a 2B + D data link, with buffers size equal to 32 byte and bit rate equal to 56 for B channels, the CPU cycle consumption is: $1.77 \times 2 + 0.56 = 4.1$ megacycles.

For a 2B + D data link, with buffers size equal to 32 byte and bit rate equal to 64 for B channels, the CPU cycle consumption is: $2.23 \times 2 + 0.56 = 5.02$ megacycles.

For both, it represents less than 10% for a 66 MHz clocked silicon.

3.4 Performances with Small Buffers

This test's goal is to evaluate the influence of a very small Tx output buffer on the CPU cycle consumption. Table 12 details the small buffer test results.

Table 12. Performances with Small Buffers Results

Buffer (Frame) Size	Bit Rate (Kbps)	Number of Calls per Second	Tx Count per Call (Cycles)	Rx Count per Call (Cycles)	Tx Count @ R Kbps (MCycle)	Rx Count @ R Kbps (MCycle)	Total Count (Tx + Rx) @ R Kbps (MCycle)
6	**56**	1166.67	1933	1329	2.26	1.55	**3.81**
Address	**64**	1333.33	2004	1594	2.67	2.13	**4.80**
Size = 2	**16**	333.33			0.67	0.53	**1.20**
5	**56**	1400.00	1892	1315	2.65	1.84	**4.49**
Address	**64**	1600.00	1959	1511	3.13	2.42	**5.55**
Size = 1	**16**	400.00			0.78	0.60	**1.39**

3.4.1 Comments

Worst case conditions are simulated by setting the buffers as small as possible because the Tx and Rx drivers are output driven.

In addition, to force all operations to be performed (flag detection, address comparison, bit (un)stuffing, CRC calculation) in the call to the routine, the frame size is set equal to the buffer size.

As the worst case scenario, for a 2B + D link, with B at 64 Kbps, then the global CPU cycle consumption is: $5.55 \times 2 + 1.39$ equal to 12.49 megacycles, which represents 19% of the CPU resources.

A test using both a buffer size and frame size of four can also be performed, but it is not considered a valid scenario since the CRC check is removed because the CRC calculation has the largest impact on the CPU clock usage.

3.5 Performances with Different Memory Map

In this test, the Tx and Rx drivers routine are profiled for different memory scenarios, as shown in Figure 6.

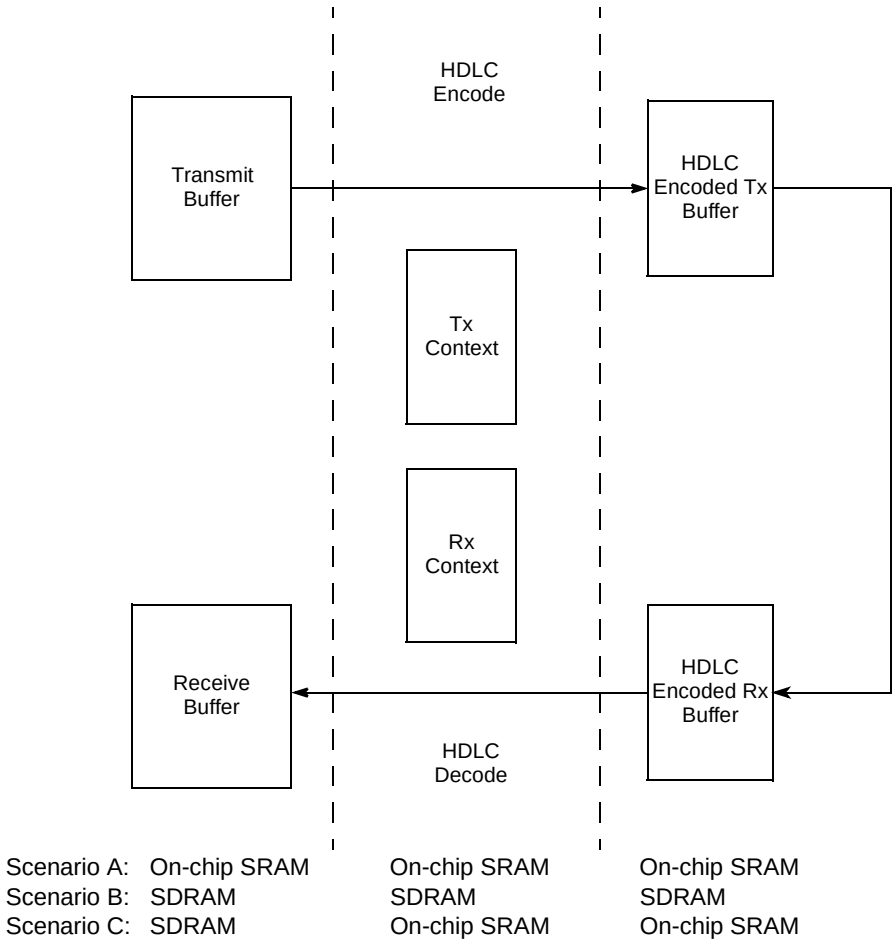

Figure 6. Memory Map Scenarios

The scenarios were executed with the following the configuration:

```
Tx output buffer size = 32
“Data field” = 27
= 32 bytes of buffer – 5 bytes of structure (1 for the flag, 2 for the
address, 2 for the CRC).
```

Table 13 details the results achieved.

Table 13. Performances with Different Memory Map Results

Operation	Scenario			Memory Utilization
	A	B	C	
Unframed Data	int SRAM	SDRAM	SDRAM	2 × 512 = 1024 bytes
Context Data	int SRAM	SDRAM	int SRAM	2 × 28 = 56 bytes
Framed Data	int SRAM	SDRAM	int SRAM	2 × 32 bytes [1]
Tx MAX	3466	4338	3942	
Tx CPU count	0.87	1.08	0.99	
Rx CPU count	0.95	1.21	1.01	
Rx + Tx CPU count	1.81	2.29	1.99	

[1] Framed data occupies only 1 × 32 bytes in the HDLC test case 07 because of the loopback.

3.5.1 Comments

The memory utilization given here concerns the profiling program only and is not linked to the routines themselves.

Scenario C is suggested because it offers the best compromise between low occupation internal memory utilization and low CPU cycle consumption.

3.6 Performance with Different Frame Size

This test evaluates the influence of a variable frame size and a fixed buffer size on the CPU cycle consumption. A frame is composed of a flag, an address field, a data field, and a CRC field.

With a buffer size equal to 32 bytes, that is, equal to or greater than 250 packets, and frames ranging from 6 to 128 bytes, the matrix shown in Table 14 results.

Table 14. Performance with Different Frame Size Results

Operation	Frame Size						
	6	16	24	32	64	96	128
Tx count per call	2268	2937	3412	3824	3663	3637	3639

Table 14. Performance with Different Frame Size Results (continued)

Operation	Frame Size						
	6	16	24	32	64	96	128
Rx count per call	1777	2675	3302	4021	4007	4014	4002
Tx CPU cycles count	0.57	0.73	0.85	0.96	0.92	0.91	0.91
Rx CPU cycles count	0.44	0.67	0.83	1.01	1.00	1.00	1.00
Rx + Tx CPU count	1.01	1.40	1.68	1.96	1.92	1.91	1.91

3.6.1 Comments

When the frame size is smaller than the buffer size, all the operations are done in the call routine. In this situation, the Tx output buffer contains the frame and the balance of the buffer is filled with flags or ones.

When the frame size is greater than the buffer size, the CRC calculation is greater. And as the CRC calculation dominates the CPU cycles count, for a frame size equal to either 64, 96, or 128, the CPU cycle consumption remains effectively constant as shown in Table 14.

Thus the worst case occurs when the frame size is exactly equal to the buffer size because of the following two phenomena.

1. Since D channel packets tend to be short, CPU cycle consumption could be modeled by the short frames scenario (6 or 16 bytes). As D channel packets are infrequent, Rx buffer processing is dominated by buffers containing all ones.
2. Since B channel packets tend to be longer, CPU cycle consumption could be modeled by the long frames scenario (64, 96, or 128 bytes).

3.7 Modifying Parameters in the Profiling Program

Detailed here are directions for modifying the various buffer sizes, the data field size, bit rate, address size, and the memory configuration.

3.7.1 Modifying the Buffer Size

In the profiling program, three buffers are used: input data buffer, loopback data buffer, and output data buffer. Referring to Figure 6 on page 21, for this test, the HDLC Encoded Tx Buffer equals the HDLC Encoded Rx Buffer and acts as the loopback buffer. The data goes from the input buffer through the loopback buffer to the output buffer. Only the size of the loopback buffer has an impact on the CPU cycle consumption.

The size of these three buffers can be modified at the beginning of the file in “/hdlc/code/hdlctest/include/hdlctst.h”.

```
#define     LOOPBACK_BUFFER_SIZE 32
#define     INPUT_BUFFER_SIZE   1024
#define     OUTPUT_BUFFER_SIZE  1024
```

3.7.2 Modifying the Size of the Data Field

Also called CHUNK_SIZE in the code, the data field value can be modified at the beginning of the file in "/hdlc/code/hdlctest/include/hdlctst.h."

The parameter INPUT_CHUNK_SIZE will not affect the behavior of the profiling program since it is only used in the functional test program to set the maximum data field in HDLC frames.

```
#define     CHUNK_SIZE          24
```

3.7.3 Modifying the Bit Rate and Address Size

These two parameters can be modified in the file "/hdlc/code/hdlctest/src/hdlcprof.c."

```
WORD wBitRate = 56;
WORD wAddressSize = 2;
```

3.7.4 Modifying the Memory Configuration

To force a variable, or place a buffer to a defined memory map location (internal SRAM or external SDRAM), add the following line before the C code declaration at the beginning of the "/hdlc/code/hdlctest/src/hdlcprof.c" file.

```
#pragma section SECTION_NAME far-absolute RW address=0x[Address in hexa]
```

By default, all variables and buffers are located in the external SDRAM. For more information, please contact SDS-DIAB support at http://www/windriver/com/.

3.8 Conclusion

The profiling test results met Motorola's expectations. CPU cycle consumption is small enough to allow for other software functions, even under worst case conditions.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu Minato-ku
Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre, 2 Dai King Street
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

DOCUMENT COMMENTS:

FAX (512) 933-2625
Attn: RISC Applications Engineering

Information in this document is provided solely to enable system and software implementers to use Motorola products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

Motorola and the Stylized M Logo are registered in the U.S. Patent and Trademark Office. digital dna is a trademark of Motorola, Inc. All other product or service names are the property of their respective owners. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

© Motorola, Inc. 2002

MCF5272HDLCUG
Rev. 0, 2/2002