



Application Note

MCF5272 Interrupt Service Routine for the Physical Layer Interface Controller

Jean Louis Dolmeta
Networking and Computing Systems Group

Part I Summary and Scope

1.1 Overview

The physical layer interface controller (PLIC) is a peripheral module of the ColdFire[®] MCF5272 intended to support ISDN applications such as CODECs, ISDN transceivers, and other peripherals. The PLIC supports two modes of operation: IDL and GCI physical layer protocols. It also has four dedicated TDM ports for connecting to external devices.

This document consists of four main parts:

1. A brief description of the inter digital link (IDL) mode of operation
2. The general circuit interface (GCI) explanation
3. A description of the interrupt service routine used to handle the data transfer for both modes of operation
4. Some examples of ColdFire microprocessor assembly code to perform quick evaluation of the MCF5272. See Part VIII, "Appendix A."

The reader is strongly recommended to read the *MCF5272 User's Manual* at www.mot.com/ColdFire before going through this document. The register and bit explanations therein help the reader to better understand the device's internal architecture.

1.2 Contents

| Paragraph Number | Title | Page |
|------------------|---|------|
| II | Inter Digital Link Mode of Operation | 1 |
| 2.1 | Introduction | 1 |
| III | General Circuit Interface Mode of Operation | 2 |
| 3.1 | GCI History | 2 |
| 3.2 | Monitor Channel Operation | 3 |
| 3.3 | Command Indicate Operation | 4 |
| IV | GCI/IDL to the MCF5272 | 4 |
| 4.1 | Data Registers | 5 |
| 4.2 | Monitor Channel Registers | 5 |
| 4.2.1 | Monitor Channel Receive | 5 |
| 4.2.2 | Monitor Channel Transmit | 5 |
| 4.3 | Command Indicate Registers | 6 |
| 4.3.1 | Command Indicate Receive | 6 |
| 4.3.2 | Command Indicate Transmit | 6 |
| V | Periodic Interrupt Process | 7 |
| 5.1 | Bubbles Definitions | 7 |
| 5.2 | One-Port Processing | 8 |
| 5.3 | Multi-Ports Processing | 11 |
| VI | Aperiodic Interrupt Process | 12 |
| 6.1 | Aperiodic One-Port Sequence | 12 |
| 6.2 | Multi-Ports Case | 13 |
| 6.3 | Brief Register Explanation | 14 |
| 6.4 | Monitor Channel Sequence | 15 |
| 6.4.1 | Transmit Sequence | 15 |
| 6.4.2 | Receive Sequence | 16 |
| 6.5 | Transmit Abort Condition | 17 |
| 6.6 | Command Indicate Channel | 17 |
| 6.6.1 | Transmit Sequence | 17 |
| 6.6.2 | Receive Sequence | 18 |

| | | |
|-------|--|----|
| VII | Assembly Code | 19 |
| 7.1 | Interrupt Controller | 19 |
| 7.2 | Interrupt Vector Generation | 20 |
| 7.3 | Prioritization Level: ICR2 Register | 21 |
| 7.4 | Programmable Interrupt Vector Register: PIVR | 22 |
| 7.5 | MBAR Configuration | 22 |
| 7.6 | Hardware Configuration | 22 |
| 7.7 | Software Configuration | 23 |
| 7.7.1 | Customer Premises Equipment Software | 23 |
| 7.7.2 | ColdFire Port Configuration | 23 |
| 7.7.3 | Debugger Configuration | 25 |
| VIII | Appendix A | 26 |

FIGURES and TABLES

| Item | Title | Page |
|-----------|--|------|
| Figure 1 | IDL 10-Bit Mode | 4 |
| Figure 2 | IDL 8-Bit Mode | 5 |
| Figure 3 | GCI Frame | 6 |
| Figure 4 | Monitor Channel Protocol | 6 |
| Figure 5 | GCI Monitor Channel Receive Register | 7 |
| Figure 6 | GCI Monitor Channel Transmit Register | 8 |
| Figure 7 | $PnGCIR$ Register | 9 |
| Figure 8 | $PnGCIT$ Register | 10 |
| Figure | One-Port Processing Interrupt Service Routine Flow Diagram | 13 |
| Figure 10 | Multi-Port Processing Interrupt Service Routine Flow Diagram | 14 |
| Figure 11 | One-Port Processing Aperiodic Interrupt Service Routine Flow Diagram | 15 |
| Figure 12 | Multi-Processing Aperiodic Interrupt Service Routine Flow Diagram | 16 |
| Figure 13 | Monitor Channel Transmit Sequence Flow Diagram | 17 |
| Figure 14 | Monitor Channel Receive Sequence Flow Diagram | 18 |
| Figure 15 | Transmit Abort Condition Flow Diagram | 19 |
| Figure 16 | CI Transmit Sequence Flow Diagram | 20 |
| Figure 17 | CI Receive Sequence Flow Diagram | 21 |
| Figure 18 | Interrupt Control Register 2 | 23 |

Interchip Digital Link Mode of Operation Introduction

| | | |
|-----------|--|----|
| Figure 19 | Programmable Interrupt Vector Register | 24 |
| Figure 20 | Hardware Configuration | 25 |
| Table 1 | PnGMR Register Field Descriptions | 8 |
| Table 2 | PnGMT Register Field Descriptions | 8 |
| Table 3 | PnGCIR Register Field Descriptions | 9 |
| Table 4 | PnGCIT Register Field Descriptions | 10 |
| Table 5 | PIV Register Field Descriptions | 24 |
| Table 6 | Port Pins Assignment | 25 |
| Table 7 | Port Control Register Values | 26 |

Part II Interchip Digital Link Mode of Operation

2.1 Introduction

The IDL mode of operation is a four-wire interface used for full-duplex communication between ICs at the board level. This interface consists of a transmit path, a receive path, an associated clock, and a synchronization signal. These signals are known as Dout, Din, DCL, and FSC. The clock determines the rate of exchange of data in both transmit and receive directions and the sync controls when this exchange is to take place. Three channels of data are exchanged every 125 microseconds. These channels consist of two 64-kbps B channels and one 16-kbps D channel used for full-duplex communication. The waveform diagrams are shown in Figure 1 and Figure 2.

Two modes are available:

- The 10-bit mode:

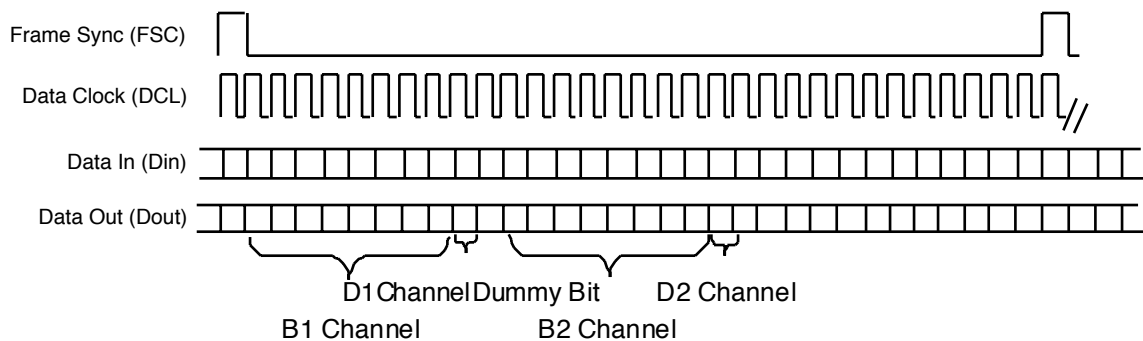


Figure 1. IDL 10-bit Mode

- The 8-bit mode:

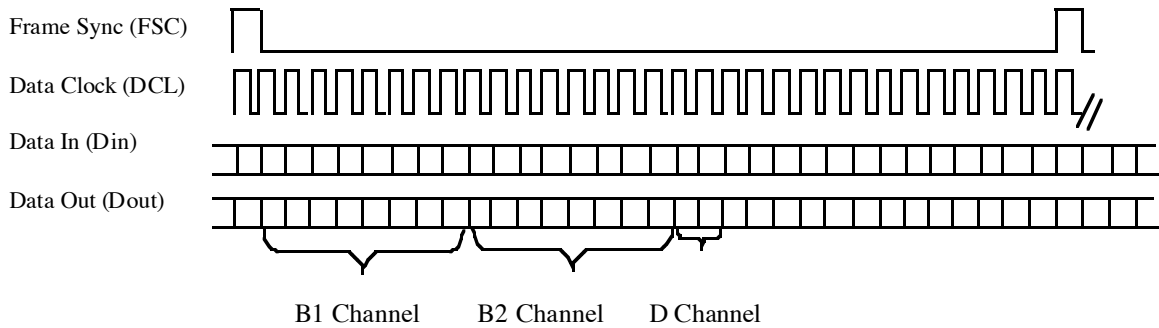


Figure 2. IDL 8-bit Mode

For more information about the different configurations of IDL, please refer to any ISDN product user’s manual, such as the MC145574/572 or MC145576 at <http://www.motorola.com/semiconductors>.

Part III General Circuit Interface Mode of Operation

3.1 GCI History

The GCI mode was defined by European companies (Italtel, Siemens, Alcatel, and GPT). GCI is a time division multiplex (TDM) bus that combines the ISDN 2B+D data, control, and status information onto four signal pins. Some benefits of the GCI include the following:

- Operation and maintenance features
- Activation and deactivation facilities (via CI channel)
- Well-defined transmission protocols to ensure correct information transfer between GCI-compatible devices
- Point-to-point and multi-point communication links
- Multiplexed mode of operation where up to eight GCI channels can be combined to form a single data stream

Those four signals consist of the following:

- FSC, frame synchronization: 8-kHz frame pulse
- DCL, data, clock signal: two clocks per data bit
- Din, data in: the data in
- Dout, data out: This pin is an open-drain output and must be pulled to Vdd through a 1.2-kΩ resistor.

The GCI frame structure has the following format: two B channels, a monitor channel, the ISDN D channel, the command/indicate channel, and the A and E bits. The frame is shown in Figure 3.

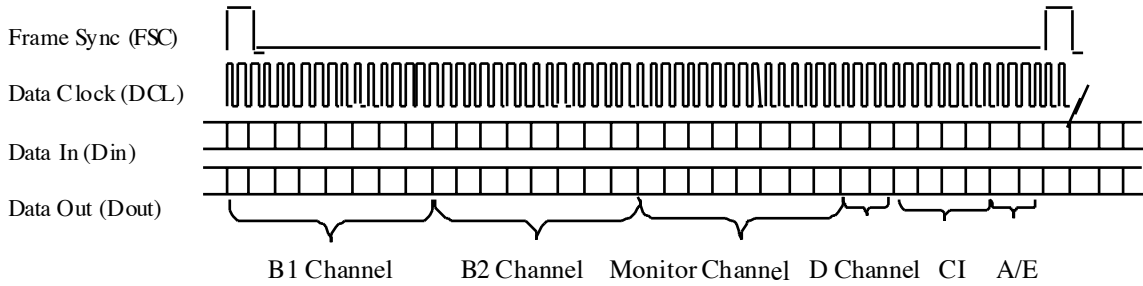


Figure 3. GCI Frame

3.2 Monitor Channel Operation

This feature is available only in GCI mode. The monitor channel is used to access the internal registers of any GCI device in order to support maintenance channel operations. All monitor channel messages are two bytes in length. Each byte is sent twice to permit the receiving GCI device to verify data integrity. In ISDN applications, the monitor channel is used for access to the maintenance messages. The A and E bits in the GCI channel are used to control and acknowledge monitor channel transfers between the MCF5272 and another GCI device.

Figure 4 shows the monitor channel protocol used. When the monitor channel is inactive, the A and E bits are both high. The A and E bits are active when they are driven low during their respective bit times. (Note that pull-up resistors are required on Din and Dout.) The E bit represents the transmission of a new monitor channel byte. The A bit from the opposite direction is used to acknowledge the monitor channel byte transfer. An idle monitor channel is indicated by both A and E bits being inactive for two GCI frames. The monitor channel data is 0xFF when inactive. The originating GCI device transmits a byte onto the monitor channel after receiving the A and E bits equal to 1 for at least two consecutive GCI frames. The originating GCI device also clears its outgoing E bit in the same GCI frame as the byte that is transmitted. The transmitted byte is repeated for at least two GCI frames, or is repeated in subsequent GCI frames, until the MCF5272 acknowledges receiving two consecutive GCI frames containing the same monitor byte.

Once the MCF5272 acknowledges the first byte, the sending device sets E high and transmits the first frame of the second byte. Then, the second byte is repeated with the E bit low until it is acknowledged. The destination GCI device verifies that it has received the first byte by clearing the A bit towards the originating GCI device for at least two GCI frames. Successive bytes are acknowledged by the receiving device setting A high on the first instance of the next byte, followed by A being cleared when the second instance of the byte is received. If the receiving GCI device does not receive the same monitor channel byte in two consecutive GCI frames, it indicates this by leaving A = 0 until two consecutive identical bytes are received. The last byte of the sequence is indicated by the originating GCI device when it sets its E bit for two successive GCI frames. The procedure is summarized in Figure 4.

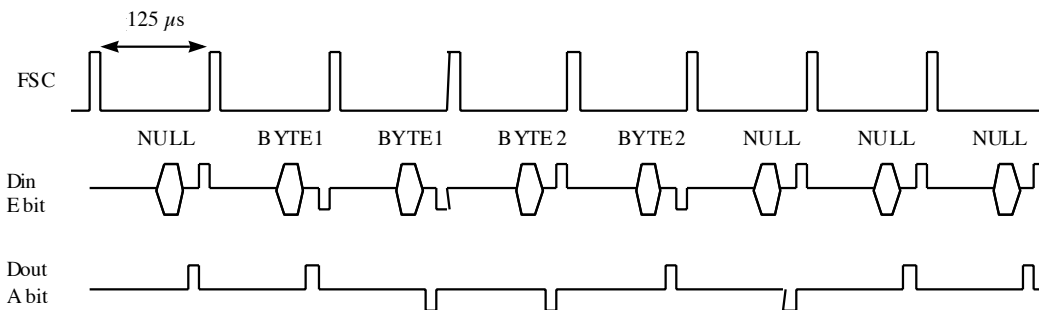


Figure 4. Monitor Channel Protocol

3.3 Command Indicate Operation

The command/indicate, or C/I channel, is used to activate and deactivate any GCI devices. Some control functions (such as loopbacks) are also supported over the C/I channel. C/I codes are four bits in length and must be received for two consecutive GCI frames before they are acted on. C/I channel bits are numbered bit 3 through 0, with bit 3 being the most significant. The C/I channel bits are transmitted starting with bit 3.

Part IV GCI/IDL and the MCF5272

This section gives a brief description of the internal registers in the PLIC.

4.1 Data Registers

For both GCI and IDL modes of operation, the maximum data rate transmitted for each digital port is 144 kbps (two 64-kbps B channels and one 16-kbps D channel). Frames of B1, B2, and D channels are packed together in PnRBx/PnTBx registers (with x = 1, 2, 3, 4) for receive and transmit direction respectively. Since the reception and transmission of information on the GCI/IDL interface is deterministic, a common interrupt is generated at 2 kHz. It is expected that a common interrupt service routine will be programmed to service the transmit and receive registers. After reset, the B and D channel shift registers and shadow registers are initialized to all 1's. For more information about the data registers, please refer to the *MCF5272 User's Manual*.

4.2 Monitor Channel Registers

This section describes receive and transmit channels.

4.2.1 Monitor Channel Receive

The PnGMR registers are 16-bit registers containing the received monitor channel bits for each of the four receive ports on the MCF5272. A byte of monitor channel data received on a certain port is put into an associated register using the format shown in Figure 5 and described in Table 1. A maskable interrupt is generated when a byte is written into any of the four available MCF5272 ports.

| | | | | | | | |
|-------|---|----|-----|----|----|---|---|
| | 15 | 11 | 10 | 9 | 8 | 7 | 0 |
| Field | — | | EOM | AB | MC | M | |
| Reset | 0000_0000_1111_1111 | | | | | | |
| R/W | Read Only | | | | | | |
| Addr | MBAR + 0x360 (P0GMR); 0x362 (P1GMR); 0x364 (P2GMR); 0x366 (P3GMR) | | | | | | |

Figure 5. GCI Monitor Channel Receive Register (PnGMR)

Table 1. PnGMR Register Field Descriptions

| Bits | Name | Description |
|-------|------|--|
| 15–11 | — | Reserved, should be cleared. |
| 10 | EOM | End of message. 0 Default at reset. 1 Indicates to the CPU that an end-of-message condition has been recognized on the E bit. EOM is automatically cleared when the PnGMR register has been read by the CPU. |
| 9 | AB | Abort. 0 Default at reset. 1 Indicates that the GCI controller has recognized an abort condition and is acknowledging the abort. It is automatically cleared by the CPU when the PnGMR register has been read. |
| 8 | MC | Monitor change. 0 Default at reset. 1 Indicates to the CPU that the monitor channel data byte written to the respective PnGMR register has changed and that the data is available for processing. This bit is automatically cleared by the CPU when the PnGMR register has been read. Clearing this bit also clears the aperiodic GMR interrupt. |
| 7–0 | M | Monitor channel data byte. These bits are written by the monitor channel controller when valid monitor channel bytes are received. |

4.2.2 Monitor Channel Transmit

The PnGMT registers are 16-bit registers containing the control and monitor channel bits to be transmitted for each of the four ports on the MCF5272. A byte of monitor channel data to be transmitted on a certain port is put into an associated register using the format shown in Figure 6 and described in Table 2. A maskable interrupt is generated when this byte of data has been successfully transmitted.

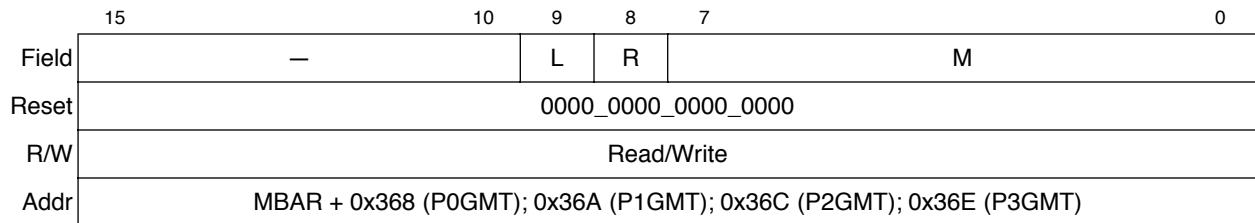


Figure 6. GCI Monitor Channel Transmit Register (PnGMT)

Table 2. PnGMT Register Field Descriptions

| Bits | Name | Description |
|-------|------|---|
| 15–10 | — | Reserved, should be cleared. |
| 9 | L | Last. 0 Default reset value 1 Set by the CPU. Indicates to the monitor channel controller to transmit the end of message signal on the E bit. Both PnGMT[L] and PnGMT[R] must be set for the monitor channel controller to send the end of message signal. The L bit is automatically cleared by the GCI controller. |

Table 2. PnGMT Register Field Descriptions

| Bits | Name | Description |
|------|------|---|
| 8 | R | Ready. 0 Default reset value. 1 Set by the CPU. Indicate to the monitor channel controller that a byte of data is ready for transmission. Automatically cleared by the GCI controller when it generates a transmit acknowledge (ACK bit in PnGMTS register) or when the L bit is reset. |
| 7–0 | M | Monitor channel data byte. Written by the CPU when a byte is ready for transmission. |

4.3 Command Indicate Registers

This section describes receive and transmit command registers.

4.3.1 Command Indicate Receive

The PnGCIR register is an 8-bit register containing the received C/I bits for one of each of the four ports on the MCF5272. The register is shown in Figure 7 and described in Table 3.

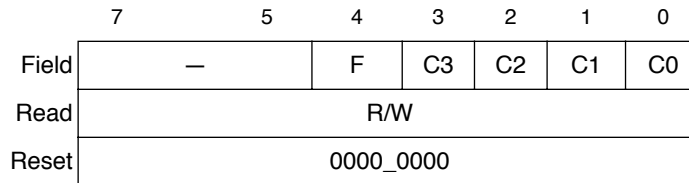


Figure 7. PnGCIR Register

Table 3. PnGCIR Register Field Descriptions

| Bits | Name | Description |
|------|-------|---|
| 7–5 | — | Reserved, should be cleared. |
| 4 | F | Full. This bit is set by the C/I channel controller to indicate to the CPU that new C/I channel data has been received and is available for processing. It is automatically cleared by a CPU read. The clearing of this bit by reading this register also clears the aperiodic GCR interrupt. |
| 3–0 | C3–C0 | C/I bits. These four bits are received on the GCI or SCIT channel 0. When a change in the C/I data value is received in two successive frames, it is interpreted as being valid and is passed on to the CPU via this register. A maskable interrupt is generated when data is written into any of the four available positions. |

4.3.2 Command Indicate Transmit

The PnGCIT registers are 8-bit registers containing the monitor channel bits to be transmitted for each of the four ports on the MCF5272. The register is shown in Figure 8 and described in Table 4.

Freescale Semiconductor, Inc.

Periodic Interrupt Process ISR Bubble Definitions

| | | | | | | | |
|-------|-----------|---|----|----|----|----|---|
| | 7 | 5 | 4 | 3 | 2 | 1 | 0 |
| Field | — | F | C3 | C2 | C1 | C0 | |
| Read | R/W | | | | | | |
| Reset | 0000_0000 | | | | | | |

Figure 8. PnGCIT Register

Table 4. PnGCIT Register Field Descriptions

| Bits | Name | Description |
|------|-------|--|
| 7–5 | — | Reserved, should be cleared. |
| 4 | R | Ready. This bit is set by the CPU to indicate to the C/I channel controller that data is ready for transmission. Setting this bit starts the C/I state machine, which responds with the transmit acknowledge (ACK bit in the PnGCITS register) once transmission of two successive C/I words is complete. This bit is automatically cleared by the GCI controller when it generates an ACK. The clearing of this bit by reading this register also clears the aperiodic GCT interrupt. |
| 3–0 | C3–C0 | C/I bits. The CPU writes C/I data to be transmitted, on the GCI or SCIT channel 0, into these positions. The CPU must ensure that this data is not overwritten before it has been transmitted the required minimum number of times, that is, before a change is detected and confirmed by a receiver. A maskable interrupt is generated when this data has been successfully transmitted. |

Part V Periodic Interrupt Process

This document does not intend to define all the meanings of the PnPSR register. For more information, the user should read the *MCF5272 User's Manual*. This PnPSR register is involved in the data processing. All PnRBx and PnTBx registers, either in IDL or GCI modes of operation, will use these registers. There is no difference between those two modes as far as they are concerned. PnPSR register is updated every 500 μ s. As long as the interrupt enable (IE) bits are set to invoke the interrupt service routine, the BxRDF bits will be set every 500 μ s, and a register access will be achieved to clear this interrupt.

This section is explained in two parts:

- One port only is enabled: description of how to handle all the bits involved in PnPSR
- Multiple ports are enabled: description of how to access the ports that created this interrupt

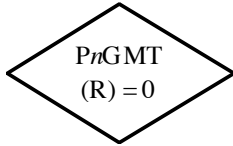
5.1 ISR Bubble Definitions

To assure that the following flow charts are well understood, this section defines the bubble shapes used in the illustrations:

- Start or End of the process



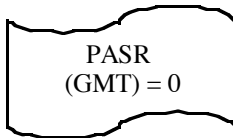
- Test Condition



- Interrupt creation



- Return from Interrupt



- Action taken



5.2 One-Port Processing

For this PLIC interrupt service routine, the channel priorities are fixed. In the periodic status registers, the B1RDF, B1TDE, and DRDF have top priority, then the B2RDF, B2TDE, and DTDE follow. The xTUE and xROE should be given lowest priority because they should be cleared. This interrupt service routine implementation gives some flexibility by dynamically jumping between the action to take and the verification of the bit that creates the interrupt. Furthermore, the purpose of the code was to first evaluate the product after first silicon. The code and the ISR have been written with this objective in mind. In conclusion, all the actions are taken inside the ISR instead of raising a flag in order to perform it once the ISR has completed. The flow chart is shown in Figure 9.

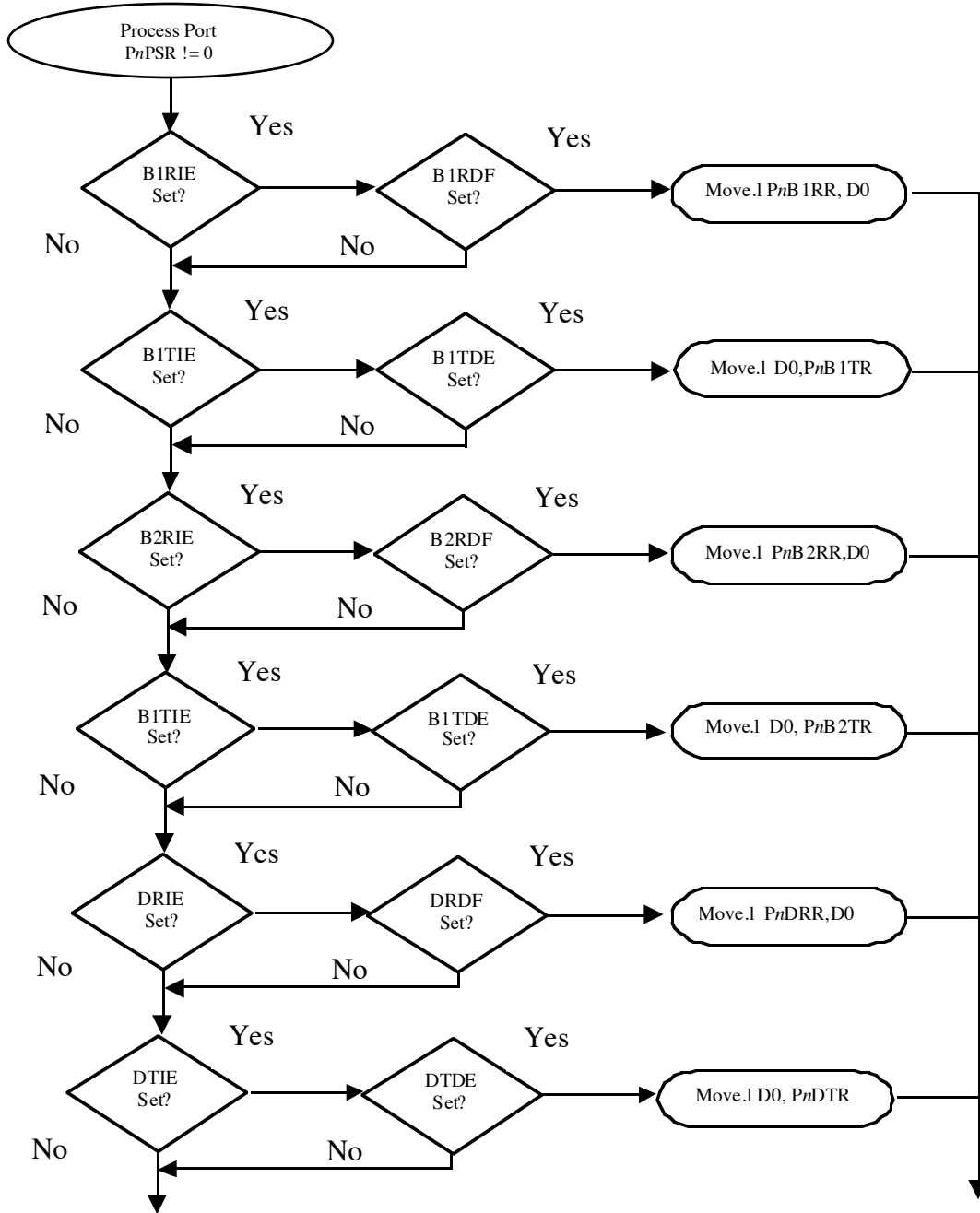


Figure 9. One-Port Processing Interrupt Service Routine Flow Diagram

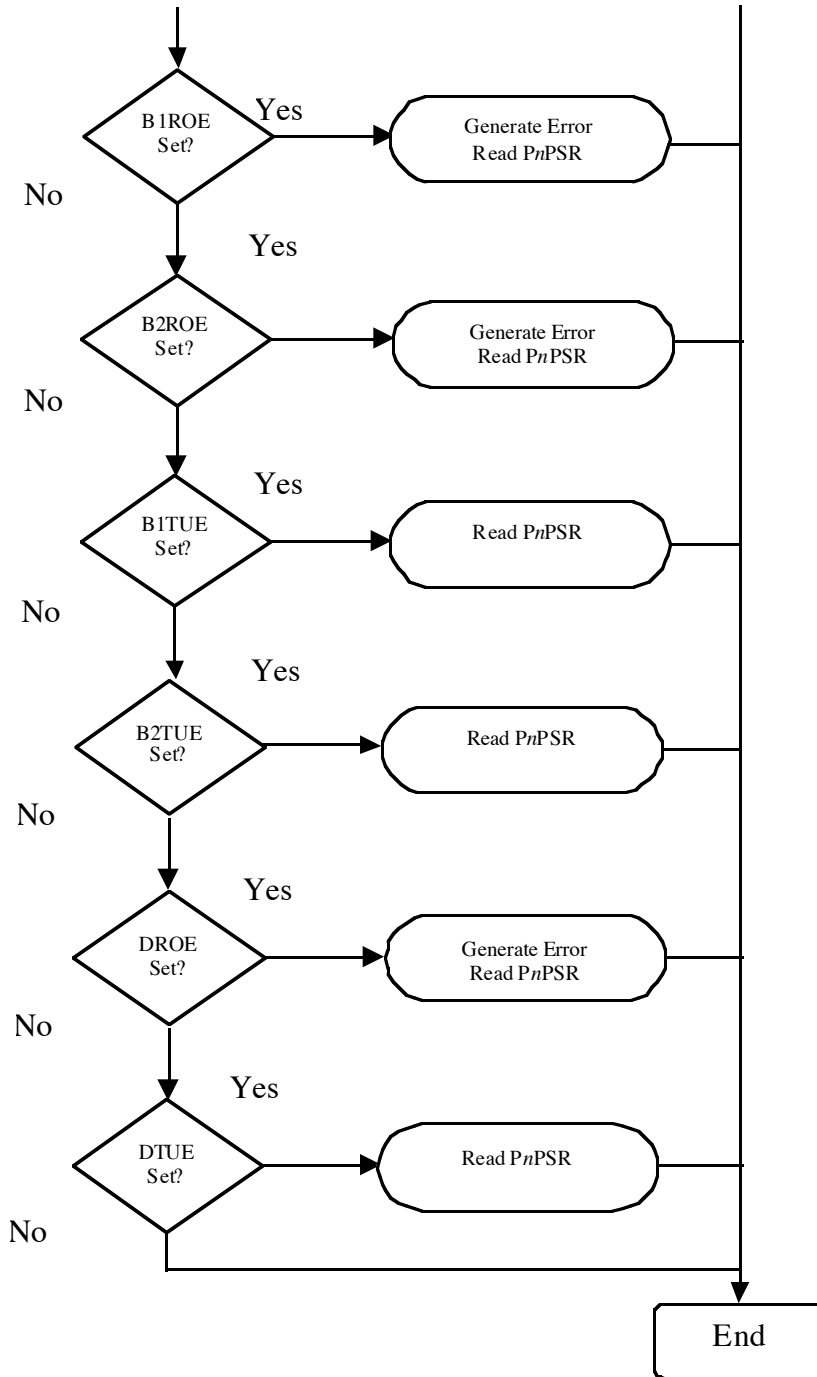


Figure 9. One-Port Processing Interrupt Service Routine Flow Diagram (Continued)

5.3 Multi-Ports Processing

Most of the time, more than one port will be enabled, and in order to make the ISR subroutine more efficient, several steps should be taken. When the periodic interrupt occurs, the user does not know yet which port has caused it. To find out, check the following registers:

Aperiodic Interrupt Process Aperiodic One-Port Sequence

1. PnICR: PLIC Interrupt Configuration register
2. PnPSR: PLIC Periodic Status Register

In fact, even if the PLPSR has been updated, the corresponding bit in the PnICR (IE) may not have been programmed; thus the ISR process cannot be called by this port. The flow chart is shown in Figure 10.

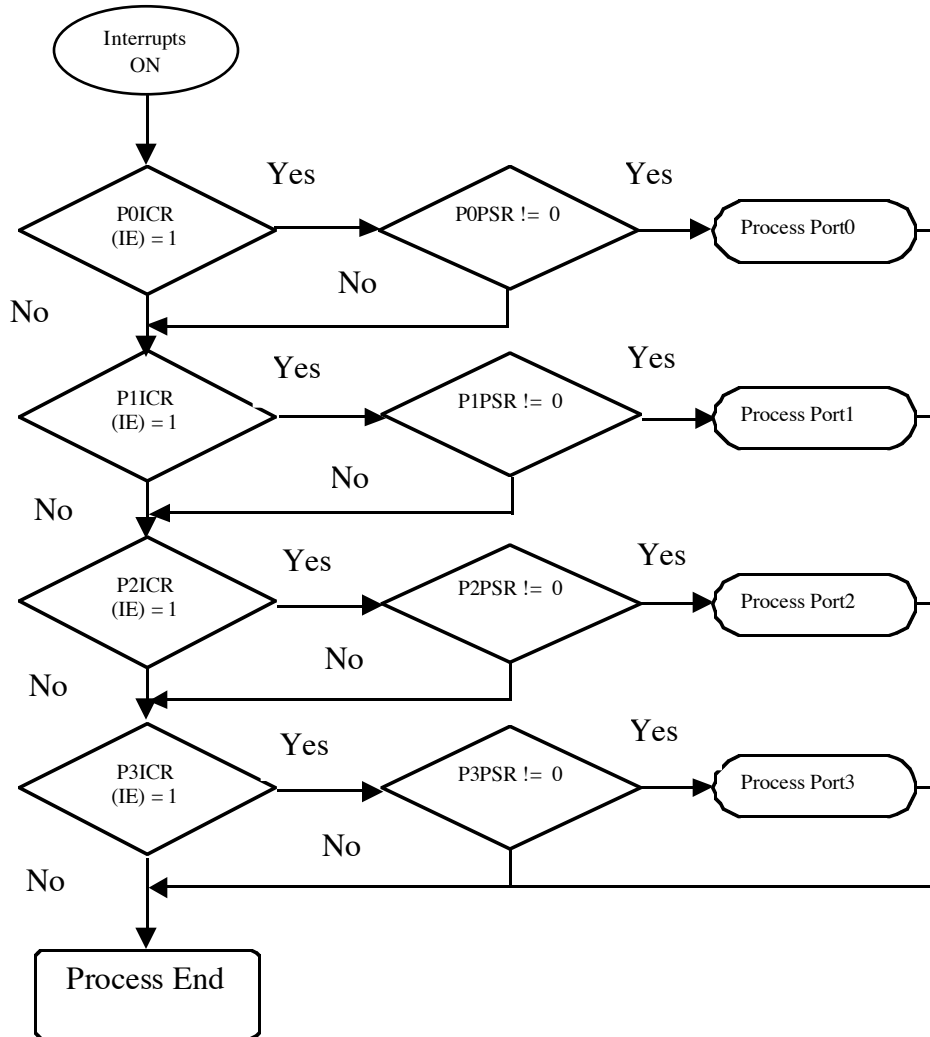


Figure 10. Multi-Port Processing Interrupt Service Routine Flow Diagram

Part VI Aperiodic Interrupt Process

This part describes port sequence, multi-port case, registers, channel sequence, abort condition, and the command indicate channel.

6.1 Aperiodic One-Port Sequence

This section explains how the aperiodic interrupt is used with one port enabled. The next section describes a multi-channel approach. When one port is enabled, a level of priority must be set up. For obvious reasons, the CI channel will be top priority because of its ability to activate and deactivate. As far the

monitor channel is concerned, the same philosophy will be applied, and the received path takes higher priority. The flow chart is shown in Figure 11.

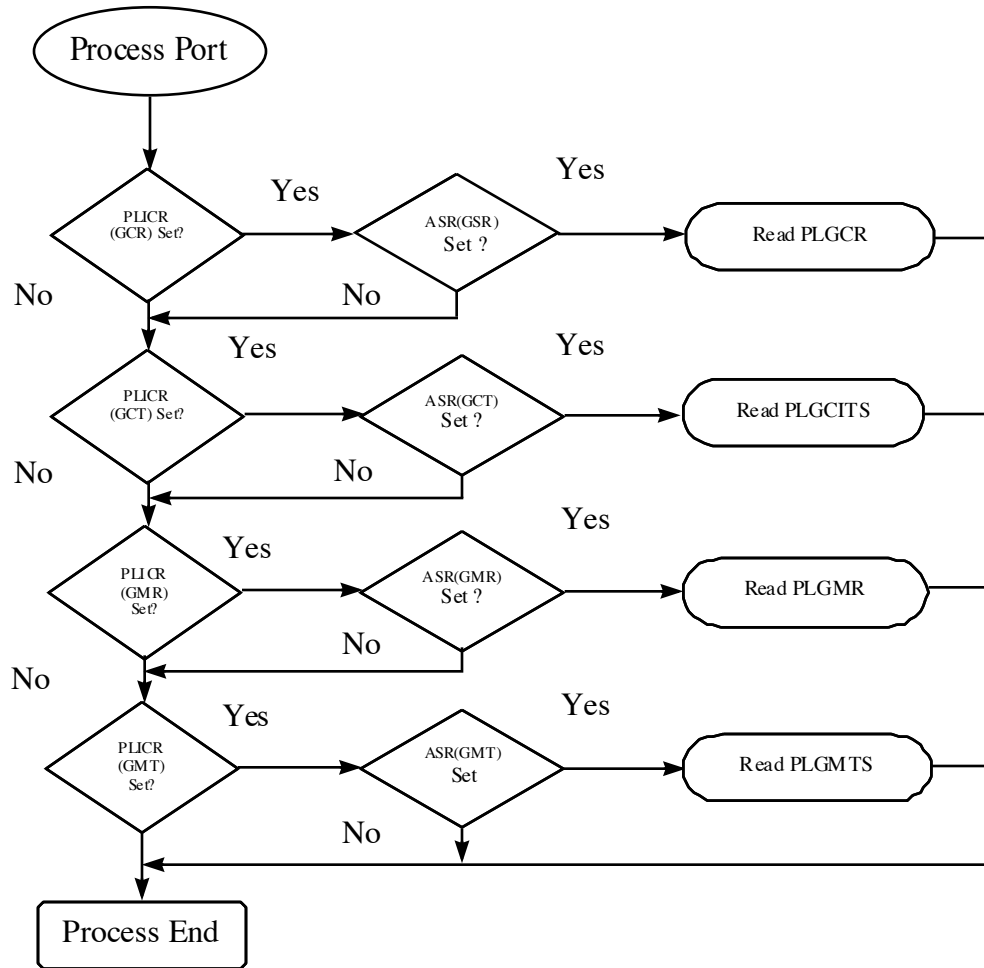


Figure 11. One-Port Processing Aperiodic Interrupt Service Routine Flow Diagram

6.2 Multi-Port Case

Once the PLIC has many GCI ports enabled, a generic aperiodic interrupt is required to handle all of the possibilities. As previously done in the periodic interrupt, the PnICR will be read so it can save some MIPS in case the port is not enabled. The flow chart is shown in Figure 12.

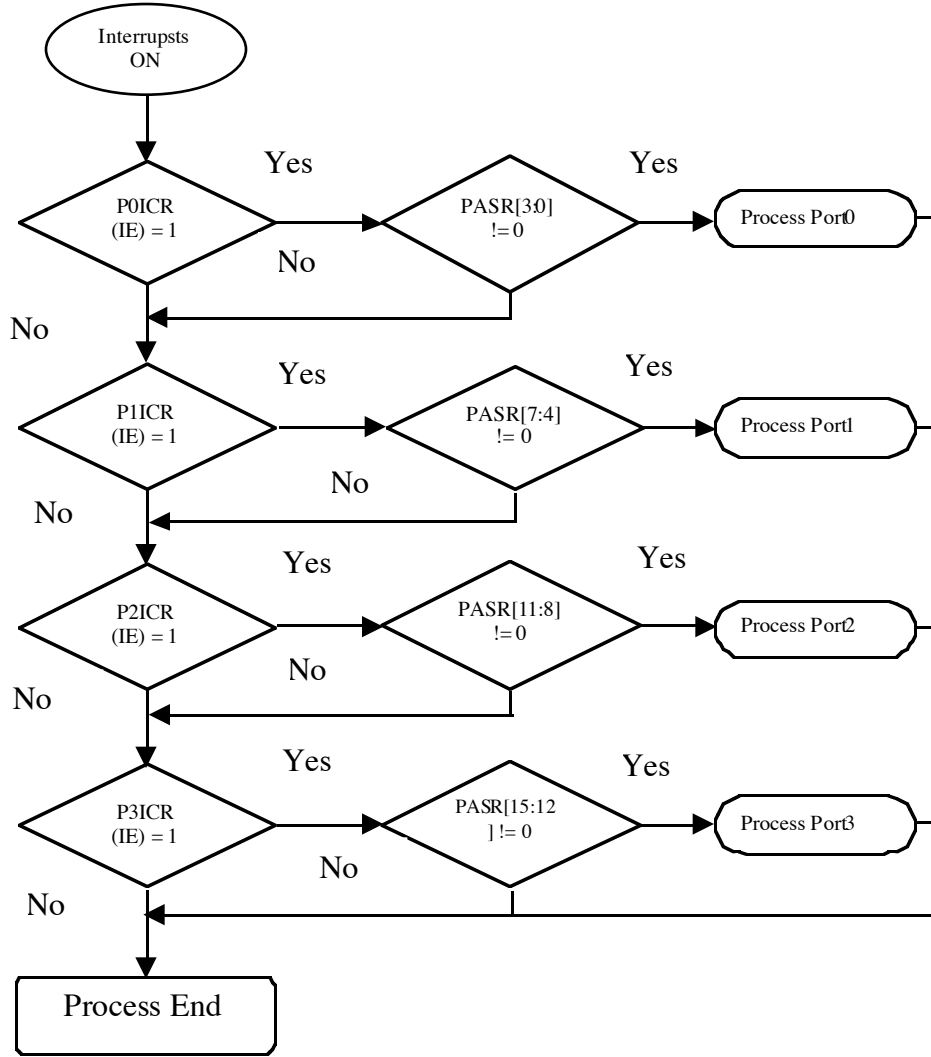


Figure 12. Multi-Processing Aperiodic Interrupt Service Routine Flow Diagram

6.3 Brief Register Explanation

This section describes the registers involved in the aperiodic interrupts service routine. The GCI oriented registers are $PnGMR$, $PnGMT$, $PnGCIR$, and $PnGCIT$. These are the only register that affect the PASR register. For more information about the definition of all those registers, please refer to the PLIC section in the *MCF5272 User's Manual* for a more detailed description.

6.4 Monitor Channel Sequence

This section details the manner in which GCI monitor channel data is handled. Based on the low-level protocol specification, the following information will help the user better understand the GCI monitor channel operation.

6.4.1 Transmit Sequence

Here is the transmit sequence of the GCI protocol using the GCI monitor channel transmit registers. The flow chart is shown in Figure 13.

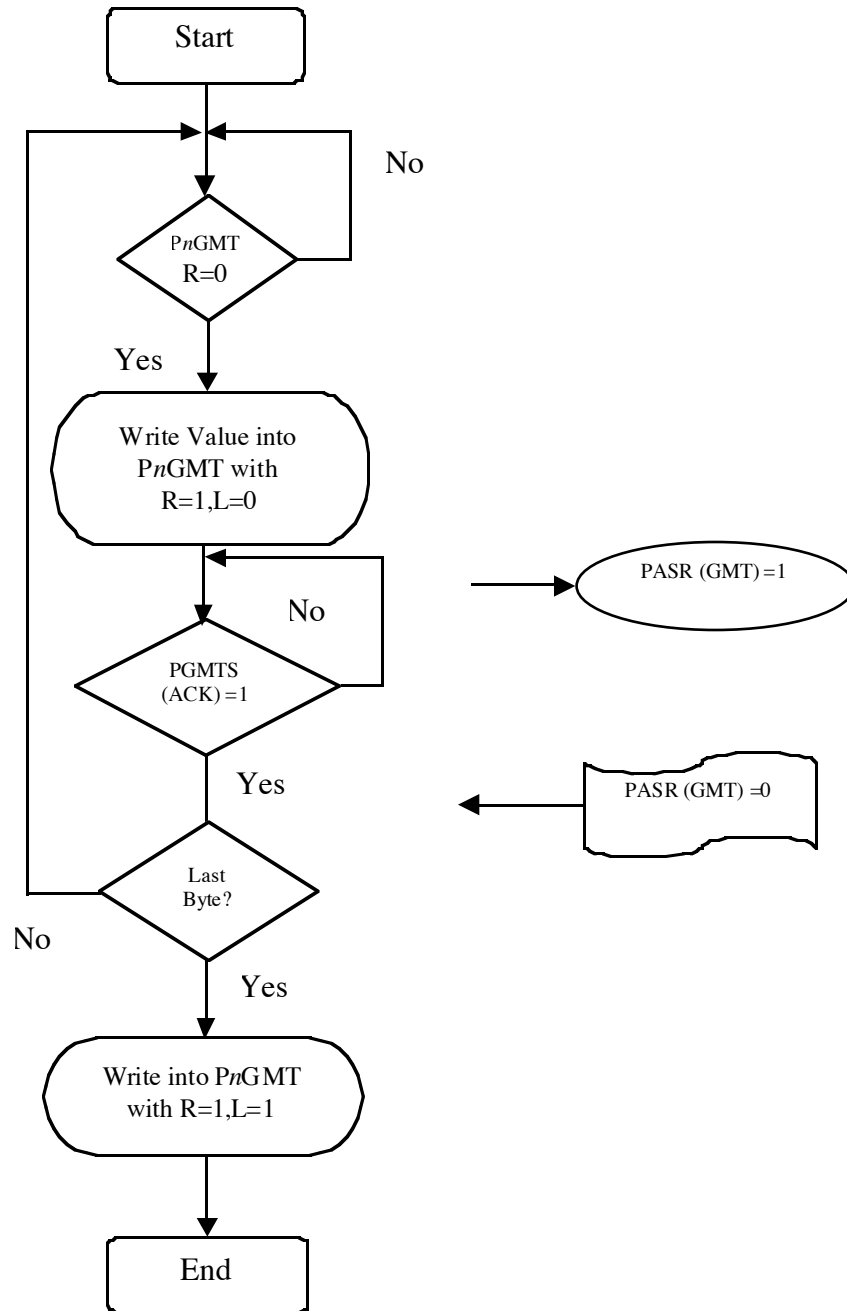


Figure 13. Monitor Channel Transmit Sequence Flow Diagram

6.4.2 Receive Sequence

This section describes with the receive sequence of the GCI protocol. The flow chart is shown in Figure 14.

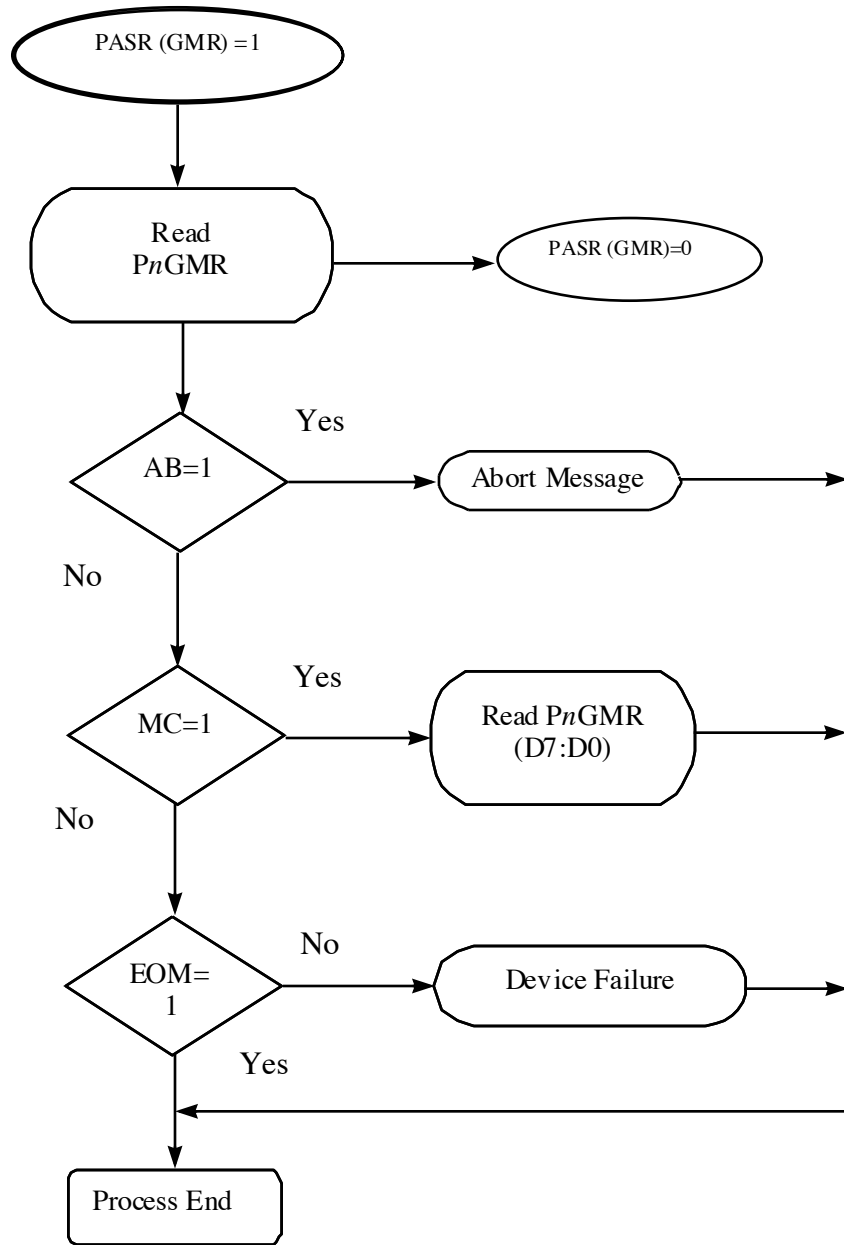


Figure 14. Monitor Channel Receive Sequence Flow Diagram

6.5 Transmit Abort Condition

One register is used to indicate an abort condition. The flow chart is shown in Figure 15.

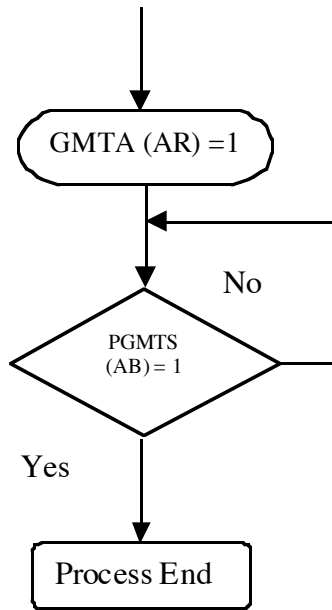


Figure 15. Transmit Abort Condition Flow Diagram

6.6 Command Indicate Channel

This section details the channel for command indication.

6.6.1 Transmit Sequence

Figure 16 explains CI channel operation.

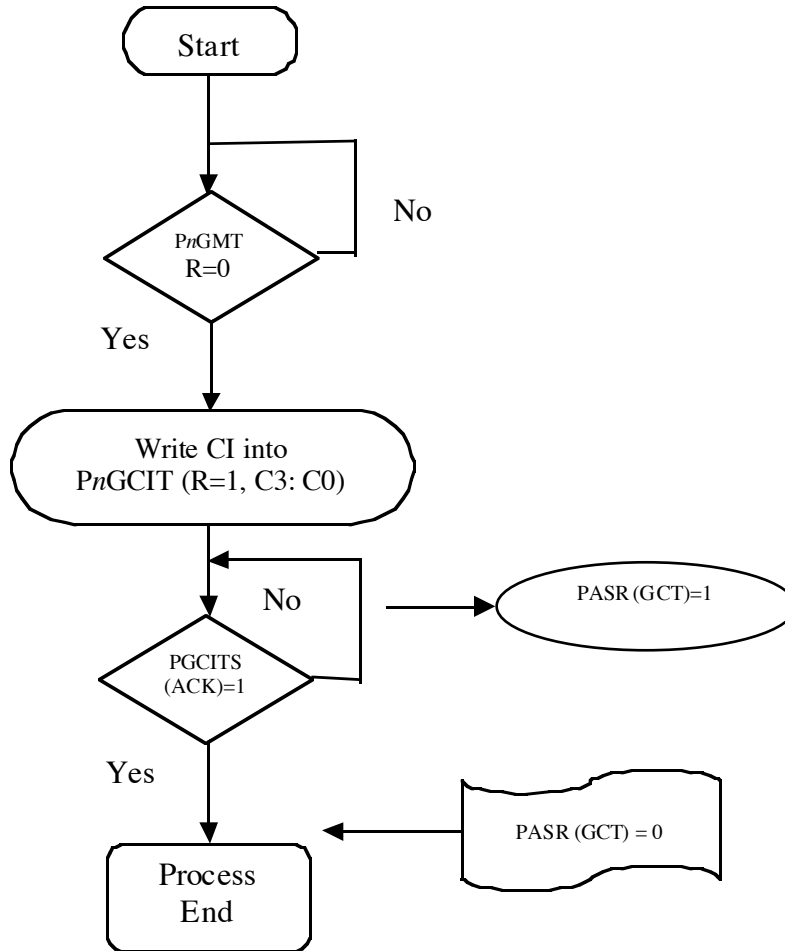


Figure 16. CI Transmit Sequence Flow Diagram

6.6.2 Receive Sequence

The flow chart for the receive sequence is shown in Figure 17.

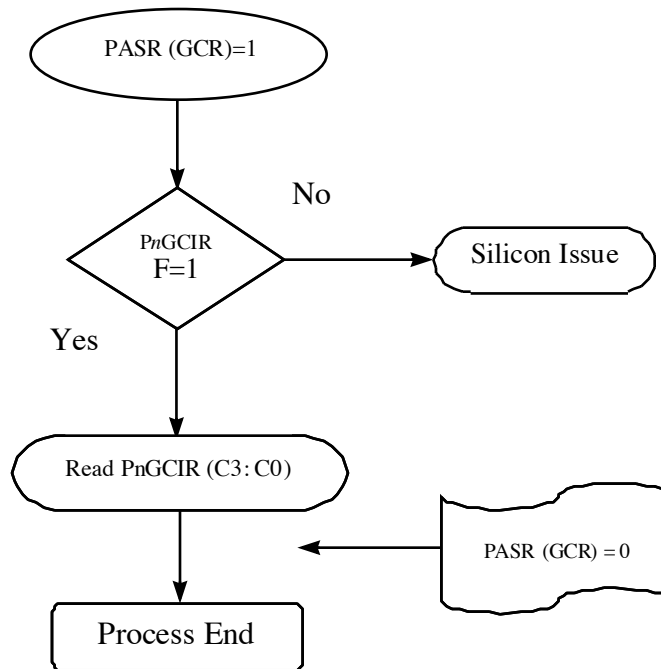


Figure 17. CI Receive Sequence Flow Diagram

Part VII Assembly Code

This section outlines how the ColdFire core will handle the PLIC. Some assembly code will be shown to better describe the sequence flow of the interrupt service routine. An interrupt occurs every 500 microseconds; therefore, all the actions must be performed within this time.

7.1 Interrupt Controller

The MCF5272 microprocessor device has some registers that set the interrupt prioritization levels for each interrupt source.

The on-chip system integration module contains an interrupt controller that is capable of providing up to 32 interrupt sources.

These sources include the following:

- External interrupt signals INT[6:1]
- Software watchdog timer
- Four general-purpose timers
- Two USARTs
- Ethernet controller
- PLIC controller
- DMA controller
- QSPI
- USB module

All external interrupt inputs are edge-sensitive where the active edge is programmable. The active edge is programmable. An interrupt request must be held valid for at least three consecutive CPU clock cycles to be considered a valid input. Each interrupt input can have its priority programmed by programming the xIPL(2-0) bits in the interrupt control registers. When the ColdFire core responds to a request with an interrupt acknowledge cycle, as is standard in MC52xx implementations, the interrupt controller logic will forward the correct vector depending on the original source of the interrupt. Software can clear pending interrupts from any source via the registers in the interrupt controller logic, and can program the location of the block of vectors used for interrupt sources via the programmable interrupt vector register. For an interrupt to be successfully processed, RAM must be available for the stack, and often this RAM will be selected by one of the programmable chip selects. So upon system startup there is a brief period where RAM is not available for the stack. To ensure no problems resulting from interrupts (particularly of priority level 7) during this period, there is an interlock which prevents any interrupt from reaching the ColdFire core until the first write cycle to the programmable interrupt vector register (PIVR). The user should ensure that both RAM chip selects and the system stack are set up prior to this write operation.

The interrupt controller includes daisy-chaining functions in order to avoid contention when the ColdFire core issues an interrupt acknowledge cycle. So if more than one interrupt source has the same interrupt priority level (IPL), they are daisy chained with INT1 being the highest priority. There are four interrupt control registers which control the interrupt priorities for the external general purpose latched interrupt input signals and the internal I/O modules' signals. These registers allow software to reset any pending interrupts from these external interrupt lines or internal modules. There are up to 32 interrupt inputs, each of which has four bits assigned to it in these registers. The registers can be read or written at any time. When read, the data returned is the last value that was written to the register, with the exception of the reset bits, which are transitory functions. The registers can be accessed by either long word (32-bit), word (16-bit), or byte (8-bit) data transfer instructions. An 8-bit write to one-half of a register will leave the other half intact.

7.2 Interrupt Vector Generation

Pending interrupts are presented to the MCF52xx core in order of priority. The core responds to an interrupt request by initiating an interrupt acknowledge cycle to receive a vector number, which allows the core to locate the interrupt's service routine. The interrupt controller is able to identify the source of the interrupt, which is being acknowledged and indicates this to the interrupt module mapper. The mapper determines which slave bus module is to provide the interrupt vector for the identified interrupt source. In most instances it is the interrupt controller itself which will provide the interrupt vector in which case the following procedure is used. The three most significant bits of the interrupt vector are programmed by the user in the programmable interrupt vector register.

7.3 Prioritization Level: ICR2 Register

The interrupt control registers (ICRx) allow the user to define which interrupt priority level (IPL), each of these peripheral sources will use. For those modules whose interrupt sources are mapped to the interrupt controller for the vector source, the programmable interrupt vector register (PIVR) allows the user to define a particular vector number to be presented when the respective module receives an interrupt acknowledge from the MCF52xx core via the interrupt controller logic. The interrupt vector register is initialized upon system reset with the uninitialized interrupt vector (hexadecimal 0x0F), and must be programmed with the required vector number for normal operation. It is important not to use reserved interrupt vector locations for this purpose. The dedicated ICRx for the periodic and aperiodic interrupts is ICR2.

MBASE +0x24

| | | | | | | | | |
|-------|-----------|----------|-----------|-----------|----------|-----------|-----------|----------|
| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| Write | UART1PIR | UART1PL2 | UART1/PL1 | UART1/PL0 | UART2PIR | UART2/PL2 | UART2/PL1 | UART2PLO |
| Read | UART1 | UART1PL2 | UART1/PL1 | UART1/PL0 | UART2 | UART2/PL2 | UART2/PL1 | UART2PLO |
| Reset | 0000_0000 | | | | | | | |

| | | | | | | | | |
|-------|-----------|----------|----------|----------|----------|-----------|-----------|-----------|
| | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Write | PLIPIR | PLI IPL2 | PLI IPL1 | PLI IPL0 | PLI APIR | PLI AIPL2 | PLI AIPL1 | PLI AIPL0 |
| Read | PLIP | PLI IPL2 | PLI IPL1 | PLI IPL0 | PLIA | PLI AIPL2 | PLI AIPL1 | PLI AIPL0 |
| Reset | 0000_0000 | | | | | | | |

| | | | | | | | | |
|-------|-----------|----------|----------|----------|---------|----------|----------|----------|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Write | USB0PIR | USB0IPL2 | ISB0IPL1 | USB0IPL0 | USB1PIR | USB1IPL2 | USB1IPL1 | USB0IPL0 |
| Read | USB0 | USB0IPL2 | ISB0IPL1 | USB0IPL0 | USB1 | USB1IPL2 | USB1IPL1 | USB0IPL0 |
| Reset | 0000_0000 | | | | | | | |

| | | | | | | | | |
|-------|-----------|----------|----------|----------|---------|----------|----------|----------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Write | USB2PIR | USB2IPL2 | USB2IPL1 | USB2IPL0 | USB3PIR | USB3IPL2 | USB2IPL1 | USB2IPL0 |
| Read | ISB2 | USB2IPL2 | USB2IPL1 | USB2IPL0 | USB3 | USB3IPL2 | USB2IPL1 | USB2IPL0 |
| Reset | 0000_0000 | | | | | | | |

Figure 18. Interrupt Controller Register 2

For more explanation about the meaning of those bits, the user should read the *MCF5272 User's Manual*. Here is a brief summary ICRx bits:

xPIR: When set to one, the new IPL value is stored. When set to zero, the corresponding INTx interrupt latch and IPL level is unaffected. Any pending interrupt on that line will remain pending.

xIPL(2:0): Interrupt Priority Level (1-7): When set to zero, the corresponding INTx interrupt line is inhibited and will not generate interrupts. Its state can still be read via the ISR1 register. Otherwise, the corresponding INT1x interrupt line is enabled, and will generate an interrupt to the MCF52xx core with the indicated priority level.

For more explanation about the meaning of these bits, see the *MCF5272 User's Manual*.

7.4 Programmable Interrupt Vector Register: PIVR

This register specifies the vector numbers which will be returned by the interrupt controller in response to interrupt acknowledge cycles for the various peripherals and discrete interrupt sources. The high three bits of the vector number are programmed in the PIVR. The low five bits are provided by the interrupt controller depending on the highest priority source which is currently active for the specific interrupt priority level (IPL) being responded to in the current acknowledge cycle.

| | | | | | | | | |
|-------|-----------|-----|-----|------------|---|---|---|---|
| | | | | MBASE+0x3C | | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Write | IV7 | IV6 | IV5 | — | | | | |
| Read | IV7 | IV6 | IV5 | 0_1111 | | | | |
| Reset | 0000_1111 | | | | | | | |

Figure 19. Programmable Interrupt Vector Register

Table 5. PIV Register Field Descriptions

| Bits | Name | Description |
|------|---------|---|
| 7–5 | IV7–IV5 | Interrupt vectors 7–5. These bits provide the three MSB’s of the interrupt vector for interrupt acknowledge cycles from all sources. To conform to ColdFire interrupt vector allocation these bits should be set equal to or greater than 010. This is the same as writing a value of 0x40 to the register. |
| 4–0 | — | Reserved, should be cleared. |

7.5 MBAR Configuration

This is one of the first registers that should be written after reset. MBAR configuration is arbitrarily set by the user.

7.6 Hardware Configuration

The evaluation board that is used in this setup is the M5272C3. The four ports are available through PCI sockets on the evaluation board. Be sure that, in addition to supplying the evaluation board with power, the user supplies 5V to J7, which supplies power to the daughter cards connected into the PCI sockets. The wire connection between the PC and the board is a BDM wiggler cable. Depending on the tested registers, the external environment could be either MC145572EVK (U Transceiver-oriented board) or MC145574EVK (T transceiver-oriented board). Along with the M5272C3 board, daughter cards will be plugged into these PCI sockets. Most of the time, the T2 daughter boards will be plugged in to evaluate multi-port functionality, D channel access, and other features. The U2 daughter card will be used for IDL BER measurement. The BER tester is the HP1645A, which is directly connected to either the MC145572EVK or MC145574EVK. The hardware setup is as follows:

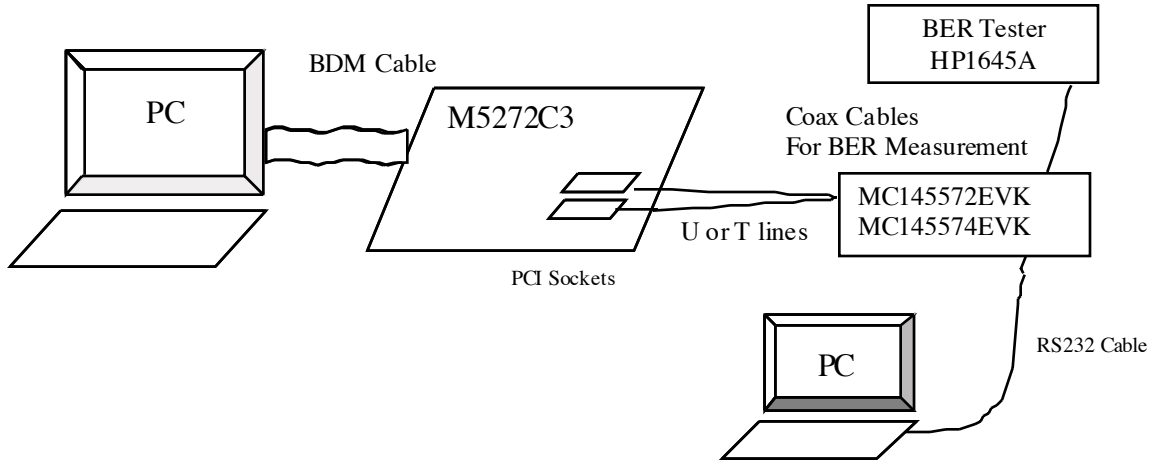


Figure 20. Hardware Configuration

7.7 Software Configuration

This section describes software configuration.

7.7.1 Customer Premises Equipment (CPE)

Depending on the CPE board used, the control software will run differently. This gives the user the ability to control the sourcing transceiver. If the MC145572EVK is used, the RS232 cable is connected and the embedded software will automatically come up with power-on reset. If the MC145574EVK is used, the scp.exe software must be run on a Windows-based PC. (Note: scp.exe software will not run on Windows NT.) Please refer to the specific EVK user’s manual for more detailed information about the hardware and software for each.

7.7.2 ColdFire Port Configuration

Writing into the ColdFire core general purpose registers must configure the TDM pins of the MCF5272. With power-on reset, port1 will be automatically configured. Then port0 and some pins of port3 must be configured as a result of the GPIO register access. Table 6 shows the values.

Table 6. Port Pin Assignments

| Port Pins | Configured by | Field in Control Register | Control Register Value (Binary) | Map BGA Pin |
|-------------|---------------|---------------------------|---------------------------------|-------------|
| FSC0 | PortA | PACNT8 | 01 | J2 |
| DCL0 | PortD | PDCNT0 | 01 | J4 |
| Din0 | PortD | PDCNT1 | 01 | K1 |
| Dout0 | PortD | PDCNT4 | 01 | K4 |
| DREQ0 | PortA | PACNT10 | 01 | K5 |
| DGRANT0 | PortA | PACNT9 | 01 | J3 |
| FSC1 | Reset | N/A | | L4 |
| DCL1 / DCL2 | Reset | N/A | | M1 |

Table 6. Port Pin Assignments

| Port Pins | Configured by | Field in Control Register | Control Register Value (Binary) | Map BGA Pin |
|---------------|---------------|---------------------------|---------------------------------|-------------|
| Din1 / Din2 | Reset | N/A | | N2 |
| Dout1 / Dout2 | Reset | N/A | | N1 |
| DREQ1 | PortA | PACNT14 | 01 | M2 |
| DGRANT1 | PortA | PACNT15 | 01 | M3 |
| FSC2 | PortA | PACNT12 | 01 | L2 |
| FSC3 | PortA | PACNT13 | 01 | L3 |
| Din3* | PortD / Reset | PDCNT5 | 10 / 00 | P3 / N2 |
| Dout3 | PortA / Reset | PACNT 7 | 10 / 00 | P1 / N1 |

Note: The user must keep in mind that ports 1, 2, and 3 share the same resource as long as DCL, Din, and Dout are concerned. This is called the indirect mode. Only the FSC pins are different. In some applications, Din3 and Dout3 might come from other resources, called the direct mode. In this case, PDCNT and PACNT values must be programmed as the above array shows. The MAP BGA pins will, of course, be different.

As a summary, the port control registers values are:

Table 7. Port Control Register Values

| Port Control Register | Indirect Mode (Hex) | Direct Mode (Hex) |
|-----------------------|---------------------|-------------------|
| PACNT (MBAR+0x80) | 55150000 | 55158000 |
| PDCNT (MBAR+0x98) | 00000105 | 00000905 |

7.7.3 Debugger Configuration

Once the code has been compiled with the Diab compiler to the file **main.elf**, the user can download it to the M5272C3 evaluation board via the BDM cable. The configuration file is as follows:

```
set vectbase = 0x00000000
set vectaddr = 0x00000000

@ MBAR = 0x10000001
@ RAMBAR = 0x20000001

# 2MB FLASH on CS0 at 0xFFE00000
write -l 0x10000040=0xFFE00201
write -l 0x10000044=0xFFE00014

# Nothing on CS1 at 0x00000000
write -l 0x10000048=0x00000000
write -l 0x1000004C=0x00000000

# External FSRAM on CS2 at 0x30000000
write -l 0x10000050=0x30000001
write -l 0x10000054=0xFFF80008
```

```
# Nothing on CS3 at 0x00000000
# write -l 0x10000058=0x00000000
# write -l 0x1000005C=0x00000000

# CS7 from address 0x00000000 4M byte SDRAM
write -l 0x10000078 = 0x00000701
write -l 0x1000007C = 0xFFC0007c

# setup SDRAM Timing and Control Registers SDCTR then SDCCR
write -l 0x10000184 = 0x0000f415
write -l 0x10000180 = 0x00004211

# rem initialize SDRAM with a write
write -r 0x00040000=0x55555555 # STARTS SDRAM controller
```

Part VIII Appendix A

This appendix deals with an example of program that has been used to evaluate the MCF5272.

Scope of the program:

The purpose of that program was not to productize the MCF5272, but to simply evaluate the silicon. This is the reason why the internal architecture might not be optimized to the fullest. Some knowledge of Motorola ISDN products is required, in order to fully activate and send data over the data link. For more information, the user must refer to the MC145572 and MC145574 User's Manuals.

Once the hardware has been correctly set up (see Figure 20), the NT-configured MC145574 chip activates the (NR2 to 0x1) down to the TE-configured MC145574 device. When the link is up and running (NR1 to 0x9), the TE must have the B1 and B2 channels on (NR5 to 0xC). Once connected to the NT-configured T chip, the bit error rate tester (HP 1645A) sends data from the NT device through the link down to the TE device. The purpose of MCF5272 is to capture the received data and to perform a loopback in the ColdFire core via the interrupt service routine. The possible loopbacks that can be tested are all combinations of B channels or D channel by itself. Due to the internal architecture and the special processing of the D channel, the 2B+D loopback cannot be performed. The entire loopback process runs in the ISR. The flow diagram of the program (for both IDL and GCI modes, except that there is no aperiodic interrupt in IDL mode) is as follows:

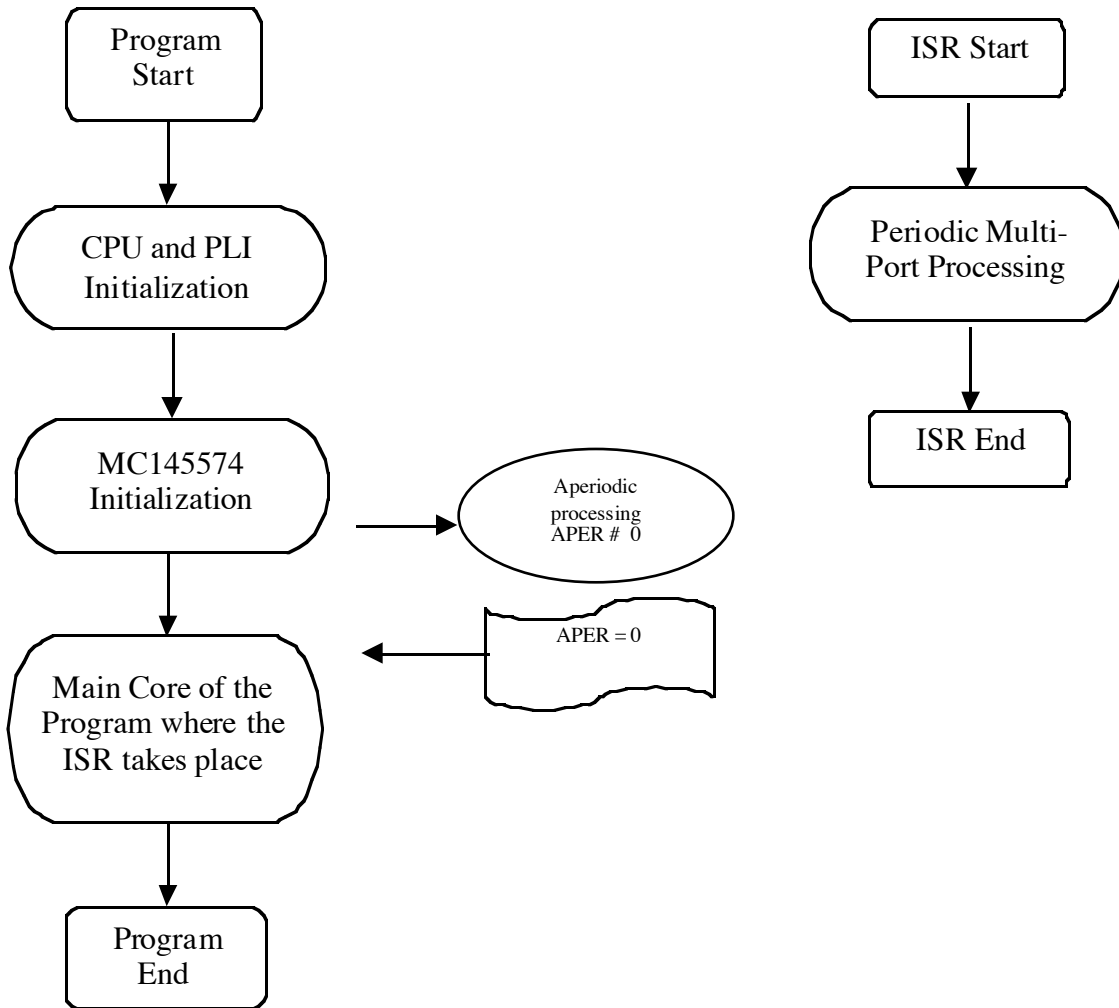


Figure 21. Main Program and ISR Flow Diagram

Here below is the MAIN file, with all the invoked files that are necessary

```

.include "configuration.h"
.include "PLIInterupt.s"
.include "PerIntVector.s"
.include "AperIntVector.s"
.include "CoreInit.s"
.include "Init.s"
.include "CPUInit.s"
.include "Table.s"

;Equates
;Interrupt Vectors
;Periodic Interrupt
;Aperiodic Interrupt
;Main file
;Subroutine to init
;Linked to ColdFire Core
    
```

Note: The user must add a space at each beginning of line to make sure the code will be correctly compiled.

This is the file that must be compiled with the following executable file **makeit**:

```
das -l -g -Xalign-value -tMCF5206eFN:simple -Xasm-debug-on -I@ -o
main.o target.s
dld -m2 -o main.elf main.o -lc > map
ddump -Rv -o try.mot main.elf
```

Once the main.elf file has been generated, it must be downloaded to the board via the BDM. Shown below are all the files included in main.s.

Configuration.h is the file used for all the definition of the equates. In order to save space, only the important and used equates are shown: i.e. the Initialization registers and the PLIC registers.

```
; Default initial register values
```

```
; Most of the values are not used for the PLIC evaluation
```

```
Module_Regs_Addr    EQU    $00300000    ;Addr. of on-chip reg
Init_MBAR           EQU    $10000001    ;MBAR value
Init_SCR            EQU    $0000
Init_PMR            EQU    $0            ;Initial value of PMR
UserProgram         EQU    $00020000    ;Put program in SDRAM
Init_SPR            EQU    $00E8        ;Initial value of SPR
Init_SCFR           EQU    $1211        ;Initial value of SCFR
Init_PIVR           EQU    $40          ;Initial value of PIVR
VBR_Init            EQU    $00000000
SRValue             EQU    $2500
LED                 EQU    $3
; ADDRESSES OF SYSTEM CONFIGURATION REGISTERS - These are absolute addresses
```

```
Sys_Config_Regs    EQU    $00000400    ;Absolute address of sers
MBAR                EQU    Sys_Config_Regs+0    ;Module Base Add Register
SCR                 EQU    Sys_Config_Regs+4    ;System Config Register
SPR                 EQU    Sys_Config_Regs+6    ;System Protection Register
PMR                 EQU    Sys_Config_Regs+8    ;Power Management Register
SCFR                EQU    Sys_Config_Regs+$C    ;Synthesized Clock Freq
DIR                 EQU    Sys_Config_Regs+$10    ;Device ID Register
```

```
; OFFSETS TO MODULES - Offsets w.r.t. contents of MBAR (don't use directly, use
register names below)
```

```
Intc_Reg_Offset    EQU    $000          ;Offset of SIM's IntC
Csel_Reg_Offset    EQU    $040          ;Offset of SIM's CSEL
Port_Reg_Offset    EQU    $080          ;Offset of SIM's Ports
QSPI_Reg_Offset    EQU    $0A0          ;Offset of QSPI module
PWM_Reg_Offset     EQU    $0C0          ;Offset of PWM module
DMA_Reg_Offset     EQU    $0E0          ;Offset of DMA module
Uart1_Reg_Offset   EQU    $100          ;Offset of UART module
Uart2_Reg_Offset   EQU    $140          ;Offset of UART module
SDRAM_Reg_Offset   EQU    $180          ;Offset of SDRAM module
Timer_Reg_Offset   EQU    $200          ;Offset of Timer module
PLIC_Reg_Offset    EQU    $300          ;Offset of PLIC module
Ether_Reg_Offset   EQU    $800          ;Offset of Ethernet module
USB_Reg_Offset     EQU    $1000         ;Offset of USB module
```

Freescale Semiconductor, Inc.

Appendix A Software Configuration

; SIM INTERRUPT CONTROLLER REGISTERS

| | | | |
|------|-----|----------------------|----------------------------|
| ICR1 | EQU | Intc_Reg_Offset+\$20 | ;Int Control Register |
| ICR2 | EQU | Intc_Reg_Offset+\$24 | ;Int Control Register |
| ICR3 | EQU | Intc_Reg_Offset+\$28 | ;Int Control Register |
| ICR4 | EQU | Intc_Reg_Offset+\$2C | ;Int Control Register |
| ISR | EQU | Intc_Reg_Offset+\$30 | ;Interrupt Source Register |
| PITR | EQU | Intc_Reg_Offset+\$34 | ;Prog Interrupt Transition |
| PIWR | EQU | Intc_Reg_Offset+\$38 | ;Prog. Interrupt Wakeup |
| PIVR | EQU | Intc_Reg_Offset+\$3F | ;Prog. Interrupt Vector |

; SIM CHIP SELECT REGISTERS

| | | | |
|-----|-----|----------------------|----------------------------|
| BR0 | EQU | Csel_Reg_Offset+\$0 | ;Chip Select Base Register |
| OR0 | EQU | Csel_Reg_Offset+\$4 | ;CS Option Register |
| BR1 | EQU | Csel_Reg_Offset+\$8 | ;Chip Select Base Register |
| OR1 | EQU | Csel_Reg_Offset+\$C | ;CS Option Register 1 |
| BR2 | EQU | Csel_Reg_Offset+\$10 | ;Chip Select Base Register |
| OR2 | EQU | Csel_Reg_Offset+\$14 | ;CS Option Register |
| BR3 | EQU | Csel_Reg_Offset+\$18 | ;Chip Select Base Register |
| OR3 | EQU | Csel_Reg_Offset+\$1C | ;CS Option Register 3 |
| BR4 | EQU | Csel_Reg_Offset+\$20 | ;CS Base Register 4 |
| OR4 | EQU | Csel_Reg_Offset+\$24 | ;CS Option Register 4 |
| BR5 | EQU | Csel_Reg_Offset+\$28 | ;CS Base Register 5 |
| OR5 | EQU | Csel_Reg_Offset+\$2C | ;CS Option Register 5 |
| BR6 | EQU | Csel_Reg_Offset+\$30 | ;CS Base Register 6 |
| OR6 | EQU | Csel_Reg_Offset+\$34 | ;CS Option Register 6 |
| BR7 | EQU | Csel_Reg_Offset+\$38 | ;CS Base Register 7 |
| OR7 | EQU | Csel_Reg_Offset+\$3C | ;CS Option Register 7 |

* SIM PORTS REGISTERS (PORT A and PORT B)

| | | | |
|-------|-----|----------------------|--------------------------|
| PACNT | EQU | Port_Reg_Offset+\$00 | ;Port A Control Reg |
| PADDR | EQU | Port_Reg_Offset+\$04 | ;Port A Data Direction |
| PADAT | EQU | Port_Reg_Offset+\$06 | ;Port A Data Register |
| PBCNT | EQU | Port_Reg_Offset+\$08 | ;Port B Control Register |
| PBDDR | EQU | Port_Reg_Offset+\$0C | ;Port B Data Direction |
| PBDAT | EQU | Port_Reg_Offset+\$0E | ;Port B Data Register |

; Port C has no CNT register - pins controlled by data bus 16/32-bit mode

| | | | |
|-------|-----|----------------------|--------------------------|
| PCDDR | EQU | Port_Reg_Offset+\$14 | ;Port C Data Direction |
| PCDAT | EQU | Port_Reg_Offset+\$16 | ;Port C Data Register |
| PDCNT | EQU | Port_Reg_Offset+\$18 | ;Port C Control Register |

; Port D has no DDR or DAT register - used for pin assignment only

; PLIC MODULE REGISTERS

| | | | |
|--------|-----|----------------------|-------------------------|
| P0B1RR | EQU | PLIC_Reg_Offset+\$00 | ;B1 Data Receive, Port0 |
| P1B1RR | EQU | PLIC_Reg_Offset+\$04 | ;B1 Data Receive, Port1 |
| P2B1RR | EQU | PLIC_Reg_Offset+\$08 | ;B1 Data Receive, Port2 |
| P3B1RR | EQU | PLIC_Reg_Offset+\$0C | ;B1 Data Receive, Port3 |
| P0B2RR | EQU | PLIC_Reg_Offset+\$10 | ;B2 Data Receive, Port0 |
| P1B2RR | EQU | PLIC_Reg_Offset+\$14 | ;B2 Data Receive, Port1 |

```

P2B2RR      EQU    PLIC_Reg_Offset+$18      ;B2 Data Receive, Port2
P3B2RR      EQU    PLIC_Reg_Offset+$1C      ;B2 Data Receive, Port3
P0DRR       EQU    PLIC_Reg_Offset+$20      ;D Data Receive, Port0
P1DRR       EQU    PLIC_Reg_Offset+$21      ;D Data Receive, Port1
P2DRR       EQU    PLIC_Reg_Offset+$22      ;D Data Receive, Port2
P3DRR       EQU    PLIC_Reg_Offset+$23      ;D Data Receive, Port3
P0B1TR      EQU    PLIC_Reg_Offset+$28      ;B1 Data Transmit, Port0
P1B1TR      EQU    PLIC_Reg_Offset+$2C      ;B1 Data Transmit, Port1
P2B1TR      EQU    PLIC_Reg_Offset+$30      ;B1 Data Transmit, Port2
P3B1TR      EQU    PLIC_Reg_Offset+$34      ;B1 Data Transmit, Port3
P0B2TR      EQU    PLIC_Reg_Offset+$38      ;B2 Data Transmit, Port0
P1B2TR      EQU    PLIC_Reg_Offset+$3C      ;B2 Data Transmit, Port1
P2B2TR      EQU    PLIC_Reg_Offset+$40      ;B2 Data Transmit, Port2
P3B2TR      EQU    PLIC_Reg_Offset+$44      ;B2 Data Transmit, Port3
P0DTR       EQU    PLIC_Reg_Offset+$48      ;D Data Transmit, Port0
P1DTR       EQU    PLIC_Reg_Offset+$49      ;D Data Transmit, Port1
P2DTR       EQU    PLIC_Reg_Offset+$4A      ;D Data Transmit, Port2
P3DTR       EQU    PLIC_Reg_Offset+$4B      ;D Data Transmit, Port3
PLCR0       EQU    PLIC_Reg_Offset+$50      ;GCI/IDL config, Port0
PLCR1       EQU    PLIC_Reg_Offset+$52      ;GCI/IDL config, Port1
PLCR2       EQU    PLIC_Reg_Offset+$54      ;GCI/IDL config, Port2
PLCR3       EQU    PLIC_Reg_Offset+$56      ;GCI/IDL config, Port3
P0ICR       EQU    PLIC_Reg_Offset+$58      ;GCI Int config, Port0
P1ICR       EQU    PLIC_Reg_Offset+$5A      ;GCI Int config, Port1
P2ICR       EQU    PLIC_Reg_Offset+$5C      ;GCI Int config, Port2
P3ICR       EQU    PLIC_Reg_Offset+$5E      ;GCI Int config, Port3
P0GMR       EQU    PLIC_Reg_Offset+$60      ;GCI Monitor RX, Port0
P1GMR       EQU    PLIC_Reg_Offset+$62      ;GCI Monitor RX, Port1
P2GMR       EQU    PLIC_Reg_Offset+$64      ;GCI Monitor RX, Port2
P3GMR       EQU    PLIC_Reg_Offset+$66      ;GCI Monitor RX, Port3
P0GMT       EQU    PLIC_Reg_Offset+$68      ;GCI Monitor TX, Port0
P1GMT       EQU    PLIC_Reg_Offset+$6A      ;GCI Monitor TX, Port1
P2GMT       EQU    PLIC_Reg_Offset+$6C      ;GCI Monitor TX, Port2
P3GMT       EQU    PLIC_Reg_Offset+$6E      ;GCI Monitor TX, Port3
PGMTS       EQU    PLIC_Reg_Offset+$71      ;GCI Monitor TX status
PGMTA       EQU    PLIC_Reg_Offset+$72      ;GCI Monitor TX abort
P0GCIR      EQU    PLIC_Reg_Offset+$74      ;GCI C/I RX, Port0
P1GCIR      EQU    PLIC_Reg_Offset+$75      ;GCI C/I RX, Port1
P2GCIR      EQU    PLIC_Reg_Offset+$76      ;GCI C/I RX, Port2
P3GCIR      EQU    PLIC_Reg_Offset+$77      ;GCI C/I RX, Port3
P0GCIT      EQU    PLIC_Reg_Offset+$78      ;GCI C/I TX, Port0
P1GCIT      EQU    PLIC_Reg_Offset+$79      ;GCI C/I TX, Port1
P2GCIT      EQU    PLIC_Reg_Offset+$7A      ;GCI C/I TX, Port2
P3GCIT      EQU    PLIC_Reg_Offset+$7B      ;GCI C/I TX, Port3
PGCITSR     EQU    PLIC_Reg_Offset+$7F      ;GCI C/I TX status
PDCSR       EQU    PLIC_Reg_Offset+$83      ;D Channel Status
P0PSR       EQU    PLIC_Reg_Offset+$84      ;Port Status, Port0
P1PSR       EQU    PLIC_Reg_Offset+$86      ;Port Status, Port1
P2PSR       EQU    PLIC_Reg_Offset+$88      ;Port Status, Port2
P3PSR       EQU    PLIC_Reg_Offset+$8A      ;Port Status, Port3
PASR        EQU    PLIC_Reg_Offset+$8C      ;Aperiodic Status Reg
PLCR        EQU    PLIC_Reg_Offset+$8F      ;Loopback Control
PDRQR       EQU    PLIC_Reg_Offset+$92      ;D Channel Request
P0SDR       EQU    PLIC_Reg_Offset+$94      ;Sync Delay, Port0
P1SDR       EQU    PLIC_Reg_Offset+$96      ;Sync Delay, Port1
P2SDR       EQU    PLIC_Reg_Offset+$98      ;Sync Delay, Port2
P3SDR       EQU    PLIC_Reg_Offset+$9A      ;Sync Delay, Port3
PCSR        EQU    PLIC_Reg_Offset+$9E      ;Clock Select

```

Shown below are the interrupt vector file. As long as the purpose of this evaluation is PLIC oriented, not all vectors need to correspond to a real ISR address:

| | org | VBR_Init |
|------------|------|----------------|
| reset_vec | DC.L | Init_SSP |
| | DC.L | Code_Start |
| berr_vec | DC.L | berr_handler |
| aerr_vec | DC.L | aerr_handler |
| illeg_vec | DC.L | illeg_handler |
| divz_vec | DC.L | divz_handler |
| chk_vec | DC.L | chk_handler |
| trapv_vec | DC.L | trapv_handler |
| privv_vec | DC.L | privv_handler |
| trace_vec | DC.L | trace_handler |
| aline_vec | DC.L | aline_handler |
| fline_vec | DC.L | fline_handler |
| res12_vec | DC.L | rsrv_handler |
| res13_vec | DC.L | rsrv_handler |
| res14_vec | DC.L | rsrv_handler |
| uninit_vec | DC.L | uninit_handler |
| res16_vec | DC.L | rsrv_handler |
| res17_vec | DC.L | rsrv_handler |
| res18_vec | DC.L | rsrv_handler |
| res19_vec | DC.L | rsrv_handler |
| res20_vec | DC.L | rsrv_handler |
| res21_vec | DC.L | rsrv_handler |
| res22_vec | DC.L | rsrv_handler |
| res23_vec | DC.L | rsrv_handler |
| spuri_vec | DC.L | spuri_handler |
| res25_vec | DC.L | rsrv_handler |
| res26_vec | DC.L | rsrv_handler |
| res27_vec | DC.L | rsrv_handler |
| res28_vec | DC.L | rsrv_handler |
| res29_vec | DC.L | rsrv_handler |
| res30_vec | DC.L | rsrv_handler |
| res31_vec | DC.L | rsrv_handler |
| trap0_vec | DC.L | trap0_handler |
| trap1_vec | DC.L | trap1_handler |
| trap2_vec | DC.L | trap2_handler |
| trap3_vec | DC.L | trap3_handler |
| trap4_vec | DC.L | trap4_handler |
| trap5_vec | DC.L | trap5_handler |
| trap6_vec | DC.L | trap6_handler |
| trap7_vec | DC.L | trap7_handler |
| trap8_vec | DC.L | trap8_handler |
| trap9_vec | DC.L | trap9_handler |
| trap10_vec | DC.L | trap10_handler |
| trap11_vec | DC.L | trap11_handler |
| trap12_vec | DC.L | trap12_handler |
| trap13_vec | DC.L | trap13_handler |
| trap14_vec | DC.L | trap14_handler |
| trap15_vec | DC.L | trap15_handler |
| res48_vec | DC.L | rsrv_handler |
| res49_vec | DC.L | rsrv_handler |
| res50_vec | DC.L | rsrv_handler |
| res51_vec | DC.L | rsrv_handler |
| res52_vec | DC.L | rsrv_handler |
| res53_vec | DC.L | rsrv_handler |
| res54_vec | DC.L | rsrv_handler |
| res55_vec | DC.L | rsrv_handler |
| res56_vec | DC.L | rsrv_handler |
| res57_vec | DC.L | rsrv_handler |
| res58_vec | DC.L | rsrv_handler |
| res59_vec | DC.L | rsrv_handler |
| res60_vec | DC.L | mbar_handler |
| res61_vec | DC.L | mbar_handler |


```

res62_vec      DC.L      mbar_handler
res63_vec      DC.L      mbar_handler

i_spur_vec     DC.L      i_spur_handler
int1_vec       DC.L      int1_handler
int2_vec       DC.L      int2_handler
int3_vec       DC.L      int3_handler
int4_vec       DC.L      int4_handler
i_tim1_vec     DC.L      i_tim1_handler      ;Timer interrupt handler
i_tim2_vec     DC.L      i_tim2_handler      ;Timer interrupt handler
i_tim3_vec     DC.L      i_tim3_handler      ;Timer interrupt handler
i_tim4_vec     DC.L      i_tim4_handler      ;Timer interrupt handler
i_uart1_vec    DC.L      i_uart1_handler     ;UART interrupt handler
i_uart2_vec    DC.L      i_uart2_handler     ;UART interrupt handler
i_plic_per_vec DC.L      i_PLIC_Periodic     ;PLIC periodic interrupt
i_plic_aper_vec DC.L      i_PLIC_Aperiodic    ;PLIC Aperiodic interrupt
i_usb0_vec     DC.L      i_usb0_handler
i_usb1_vec     DC.L      i_usb1_handler
i_usb2_vec     DC.L      i_usb2_handler
i_usb3_vec     DC.L      i_usb3_handler
i_usb4_vec     DC.L      i_usb4_handler
i_usb5_vec     DC.L      i_usb5_handler
i_usb6_vec     DC.L      i_usb6_handler
i_usb7_vec     DC.L      i_usb7_handler
i_dma_vec      DC.L      i_dma_handler
i_ether_rx_vec DC.L      i_ether_rx_handler
i_ether_tx_vec DC.L      i_ether_tx_handler
i_ether_ntc_vec DC.L      i_ether_ntc_handler
i_qspi_vec     DC.L      i_qspi_handler
int5_vec       DC.L      int5_handler
int6_vec       DC.L      int6_handler

```

```

berr_handler:
aerr_handler:
illeg_handler:
divz_handler:
chk_handler:
trapv_handler:
privv_handler:
trace_handler:
aline_handler:
fline_handler:
rsrv_handler:
uninit_handler:
spuri_handler:
trap0_handler:
trap1_handler:
trap2_handler:
trap3_handler:
trap4_handler:
trap5_handler:
trap6_handler:
trap7_handler:
trap8_handler:
trap9_handler:
trap10_handler:
trap11_handler:
trap12_handler:
trap13_handler:
trap14_handler:
trap15_handler:
mbar_handler:
i_spur_handler:
int1_handler:

```

```

int2_handler:
int3_handler:
int4_handler:
i_tim1_handler:
i_tim2_handler:
i_tim3_handler:
i_tim4_handler:
i_uart1_handler:
i_uart2_handler:
i_usb0_handler:
i_usb1_handler:
i_usb2_handler:
i_usb3_handler:
i_usb4_handler:
i_usb5_handler:
i_usb6_handler:
i_usb7_handler:
i_dma_handler:
i_ether_rx_handler:
i_ether_tx_handler:
i_ether_ntc_handler:
i_qspi_handler:
int5_handler:
int6_handler:
                nop
                stop        #$2700

                org        UserProgram

User_Ram:      ds.b        10                ;Leave 1K for user variables
Stack:        ds.b        10                ;Supervisor Stack 1Kbytes
Init_SSP      ;Initial SSP

```

The file below deals with the general periodic interrupt service routine including all of the bit handling. Not all of the overrun and underrun bits are implemented in this file. Nevertheless, those bits have been verified. Further more, this file has been written as dynamic as possible.

```

;*****
; The program below has been written in the dynamic way. That means
; every time an Interrupt has been handled, the program counter exits
; In case that other bits are set, the program counter will enter the
; ISR again.
;*****

; D0 is used for Data
; D1 is used for recovering the PnPSR
; D2 is used for checking the T/RIE bits by reading PnICR
; D3 is used for checking bits
; D4 is used for B2 Data in case of crossing the B1/B2 data
; D5 is used LED if necessary
; D6 is used for D channel
; D7 for comparing values

i_PLIC_Periodic:
Port0Test:move.w P0ICR(A5),D1; Interrupt Config Register

                andi.l        #$00008000,D1        ; Mask the IE bit
                cmp.l        #$00008000,D1        ; compare
                bne          Port1Test             ; Go to Port1
                move.w       P0PSR(A5),D1         ; Port0 is the cause
                move.l       D1,D7                ; Save D1 into D7

```

```

andi.l    #$0000003F,D7    ; mask with all Port0 bits
cmp.l    #$0,D7           ; compare to 0
beq      Port1Test        ; go to Port1 if not equal

Port0Int:
move.w   P0ICR(A5),D2     ; Read ICR0
andi.l   #$00000001,D2   ; to make sure B1RIE is set
cmp.l    #$00000001,D2
bne      EndPort0RxB1    ; if not, go to next interrupt
move.l   D1,D7           ; P0PSR check
andi.l   #$00000001,D7   ; D1 has the PLPSP0 value
cmp.l    #$00000001,D7
beq      Port0ReadB1     ; Go to read B1 if bit set

EndPort0RxB1:
move.w   P0ICR(A5),D2     ; Read ICR0 to make sure B1TIE
andi.l   #$00000008,D2   ; is set
cmp.l    #$00000008,D2
bne      EndPort0TxB1
move.l   D1,D7           ; B1TDE Test
andi.l   #$00000008,D7
cmp.l    #$00000008,D7
beq      Port0TransmitB1 ; go to Transmit B1 if bit set

EndPort0TxB1:
move.w   P0ICR(A5),D2     ; Read ICR0 to make sure B2RIE
andi.l   #$00000002,D2   ; is set
cmp.l    #$00000002,D2
bne      EndPort0RxB2
move.l   D1,D7           ; R2RDF Test
andi.l   #$00000002,D7
cmp.l    #$00000002,D7
beq      Port0ReadB2     ; Go to read B2 if bit set

EndPort0RxB2:
move.w   P0ICR(A5),D2     ; Read ICR0 to make sure B2RIE
andi.l   #$00000010,D2   ; is set
cmp.l    #$00000010,D2
bne      Port0D
move.l   D1,D7           ; B2TDE Test
andi.l   #$00000010,D7
cmp.l    #$00000010,D7
beq      Port0TransmitB2 ; go to Transmit B2 if bit set

Port0D:  move.w   P0ICR(A5),D2     ; Read ICR0 to make sure DRIE
andi.l   #$00000004,D2   ; is set
cmp.l    #$00000004,D2
bne      EndPort0Rx      ; if not, go to next source
move.l   D1,D7           ; DRIE is set, needs to
andi.l   #$00000004,D7   ; receive the D data
cmp.l    #$00000004,D7
beq      Port0RxD        ; Go to Read D if bit set

EndPort0Rx:
move.w   P0ICR(A5),D2     ; Read ICR0(DTIE) to make
andi.l   #$00000020,D2   ; sure the bit is set
cmp.l    #$00000020,D2
bne      Port1Test      ; go to next port
move.l   D1,D7           ; DTDE Test
andi.l   #$00000020,D7
cmp.l    #$00000020,D7
beq      Port0TxD

Port1Test:
move.w   P1ICR(A5),D1     ; IE Test on Port1
    
```

Freescale Semiconductor, Inc.

Appendix A Software Configuration

```

        andi.l    #$00008000,D1
        cmp.l    #$00008000,D1
        bne                                ; go to Port2 if not equal

Port1Int:nop
        move.w   PnPSR1(A5),D1                ; Port1
        move.l   D1,D7
        andi.l   #$0000003F,D7
        cmp.l    #$0,D7
        beq      Port2Test                    ; go to Port2 if not equal
        move.w   P1ICR(A5),D2
        andi.l   #$00000001,D2
        cmp.l    #$00000001,D2
        bne      EndPort1RxB1
        move.l   D1,D7                        ; B1RDF Test
        andi.l   #$00000001,D7
        cmp.l    #$00000001,D7
        beq      Port1ReadB1

EndPort1RxB1:
        move.w   P1ICR(A5),D2
        andi.l   #$00000008,D2
        cmp.l    #$00000008,D2
        bne      EndPort1TxB1
        move.l   D1,D7                        ; B1TDE Test
        andi.l   #$00000008,D7
        cmp.l    #$00000008,D7
        beq      Port1TransmitB1

EndPort1TxB1:
        move.w   P1ICR(A5),D2
        andi.l   #$00000002,D2
        cmp.l    #$00000002,D2
        bne      EndPort1RxB2
        move.l   D1,D7                        ; B2RDF Test
        andi.l   #$00000002,D7
        cmp.l    #$00000002,D7
        beq      Port1ReadB2

EndPort1RxB2:
        move.w   P1ICR(A5),D2
        andi.l   #$00000010,D2
        cmp.l    #$00000010,D2
        bne      Port1D
        move.l   D1,D7                        ; B2TDE Test
        andi.l   #$00000010,D7
        cmp.l    #$00000010,D7
        beq      Port1TransmitB2

Port1D:  move.w   P1ICR(A5),D2
        andi.l   #$00000004,D2
        cmp.l    #$00000004,D2
        bne      EndPort1Rx
        move.l   D1,D7                        ; DRDF Test
        andi.l   #$00000004,D7
        cmp.l    #$00000004,D7
        beq      Port1RxD

EndPort1Rx:
        move.w   P1ICR(A5),D2
        andi.l   #$00000020,D2
        cmp.l    #$00000020,D2
        bne      Port2Test
        move.l   D1,D7                        ; DTDE Test
    
```

```

    andi.l    #$00000020,D7
    cmp.l    #$00000020,D7
    beq

Port2Test:
    move.w   P2ICR(A5),D1           ; IE Test on Port2
    andi.l   #$00008000,D1
    cmp.l    #$00008000,D1
    bne

Port2Int:nop
    move.w   P2PSR(A5),D1           ; Port2
    move.l   D1,D7
    andi.l   #$0000003F,D7
    cmp.l    #$0,D7
    beq
    move.w   P2ICR(A5),D2
    andi.l   #$00000001,D2
    cmp.l    #$00000001,D2
    bne
    move.l   EndPort2RxB1
    move.l   D1,D7                 ; B1RDF Test
    andi.l   #$00000001,D7
    cmp.l    #$00000001,D7
    beq
    Port2ReadB1

EndPort2RxB1:
    move.w   P2ICR(A5),D2
    andi.l   #$00000008,D2
    cmp.l    #$00000008,D2
    bne
    move.l   EndPort2TxB1
    move.l   D1,D7                 ; B1TDE Test
    andi.l   #$00000008,D7
    cmp.l    #$00000008,D7
    beq
    Port2TransmitB1

EndPort2TxB1:
    move.w   P2ICR(A5),D2
    andi.l   #$00000002,D2
    cmp.l    #$00000002,D2
    bne
    move.l   EndPort2RxB2
    move.l   D1,D7                 ; B2RDF Test
    andi.l   #$00000002,D7
    cmp.l    #$00000002,D7
    beq
    Port2ReadB2

EndPort2RxB2:
    move.w   P2ICR(A5),D2
    andi.l   #$00000010,D2
    cmp.l    #$00000010,D2
    bne
    move.l   Port2D
    move.l   D1,D7                 ; B2TDE Test
    andi.l   #$00000010,D7
    cmp.l    #$00000010,D7
    beq
    Port2TransmitB2

Port2D:
    move.w   P2ICR(A5),D2
    andi.l   #$00000004,D2
    cmp.l    #$00000004,D2
    bne
    move.l   EndPort2Rx
    move.l   D1,D7                 ; DRDF Test
    andi.l   #$00000004,D7
    cmp.l    #$00000004,D7
    beq
    Port2RxD

```

Appendix A Software Configuration

```

EndPort2Rx:
    move.w        P2ICR(A5),D2
    andi.l        #$00000020,D2
    cmp.l         #$00000020,D2
    bne          Port3Test
    move.l        D1,D7                ; DTDE Test
    andi.l        #$00000020,D7
    cmp.l         #$00000020,D7
    beq          Port2TxD

Port3Test:
    move.w        P3ICR(A5),D1        ;Port3 IE Test
    andi.l        #$00008000,D1
    cmp.l         #$00008000,D1
    bne          EndSR

Port3Int:move.w   P3PSR(A5),D1
    move.l        D1,D7
    andi.l        #$0000003F,D7
    cmp.l         #$0,D7
    beq          EndSR
    move.w        P3ICR(A5),D2
    andi.l        #$00000001,D2
    cmp.l         #$00000001,D2
    bne          EndPort3RxB1
    move.l        D1,D7                ; B1RDF Test
    andi.l        #$00000001,D7
    cmp.l         #$00000001,D7
    beq          Port3ReadB1

EndPort3RxB1:
    move.w        P3ICR(A5),D2
    andi.l        #$00000008,D2
    cmp.l         #$00000008,D2
    bne          EndPort3TxB1
    move.l        D1,D7                ; B1TDE Test
    andi.l        #$00000008,D7
    cmp.l         #$00000008,D7
    beq          Port3TransmitB1

EndPort3TxB1:
    move.w        P3ICR(A5),D2
    andi.l        #$00000002,D2
    cmp.l         #$00000002,D2
    bne          EndPort3RxB2
    move.l        D1,D7                ; B2RDF Test
    andi.l        #$00000002,D7
    cmp.l         #$00000002,D7
    beq          Port3ReadB2

EndPort3RxB2:
    move.w        P3ICR(A5),D2
    andi.l        #$00000010,D2
    cmp.l         #$00000010,D2
    bne          Port3D
    move.l        D1,D7                ; B2TDE test
    andi.l        #$00000010,D7
    cmp.l         #$00000010,D7
    beq          Port3TransmitB2

Port3D:  nop
    move.w        P3ICR(A5),D2
    andi.l        #$00000004,D2
    cmp.l         #$00000004,D2
    bne          EndPort3Rx
    move.l        D1,D7                ; DRDF Test

```

```

    andi.l    #$00000004,D7
    cmp.l    #$00000004,D7
    beq      Port3RxD

EndPort3Rx:
    move.w   P3ICR(A5),D2
    andi.l   #$00000020,D2
    cmp.l    #$00000020,D2
    bne     EndSR
    move.l   D1,D7                ; DTDE Test
    andi.l   #$00000020,D7
    cmp.l    #$00000020,D7
    beq      Port3TxD

EndSR:    rte

;*****
;                               *
;           End of Interrupt Procedure *
;*****
;                               *
;           Read Procedure for each port *
;*****

Port0ReadB1:
    move.l   P0B1RR(A5),D0; Read B1 on Port0
    jsr     Port0B1RDFReset; test if RDF is reset
    bra     EndSR      ; go to ISR end

Port0ReadB2:
    move.l   P0B2RR(A5),D0; Read B2 on Port0
    jsr     Port0B2RDFReset; test if RDF is reset
    bra     EndSR      ; go to ISR end

Port0RxD:
    move.b   P0DRR(A5),D6
    jsr     Port0DRDFReset
    bra     EndSR

Port1ReadB1:
    move.l   P1B1RR(A5),D0
    jsr     Port1B1RDFReset
    bra     EndSR

Port1ReadB2:
    move.l   P1B2RR(A5),D0
    jsr     Port1B2RDFReset
    bra     EndSR

Port1RxD:
    move.b   P1DRR(A5),D6
    jsr     Port1DRDFReset
    bra     EndSR

Port2ReadB1:
    move.l   P2B1RR(A5),D0
    jsr     Port2B1RDFReset
    bra     EndSR

Port2ReadB2:
    move.l   P2B2RR(A5),D0
    jsr     Port2B2RDFReset
    bra     EndSR

Port2RxD:
    move.b   P2DRR(A5),D6
    jsr     Port2DRDFReset
    bra     EndSR

```

Appendix A Software Configuration

```

Port3ReadB1:
    move.l    P3B1RR(A5),D0
    jsr      Port3B1RDFReset
    bra      EndSR

Port3ReadB2:
    move.l    P3B2RR(A5),D0
    jsr      Port3B2RDFReset
    bra      EndSR

Port3RxD:
    move.b    P3DRR(A5),D6
    jsr      Port3DRDFReset
    bra      EndSR

;*****
;          Transmit Procedure                               *
;*****

Port0TransmitB1:
    jsr      Port0B1TDESet          ; Make sure CPU can write
    move.l    D0,P0B1TR(A5)        ; move to the register
    jsr      Port0B1TDEReset       ; Make sure the bit is reset
    bra      EndSR                ; go to ISR end

Port0TransmitB2:
    jsr      Port0B2TDESet          ; Make sure CPU can write
    move.l    D0,P0B2TR(A5)        ; move to the register
    jsr      Port0B2TDEReset       ; Make sure the bit is reset
    bra      EndSR                ; go to ISR end

Port0TxD: jsr      Port0DTDESet
    move.b    D6,P0DTR(A5)
    jsr      Port0DTDEReset
    bra      EndSR

Port1TransmitB1:
    jsr      Port1B1TDESet
    move.l    D0,P1B1TR(A5)
    jsr      Port1B1TDEReset
    bra      EndSR

Port1TransmitB2:
    jsr      Port1B2TDESet
    move.l    D0,P1B2TR(A5)
    jsr      Port1B2TDEReset
    bra      EndSR

Port1TxD: jsr      Port1DTDESet
    move.b    D6,P1DTR(A5)
    jsr      Port1DTDEReset
    bra      EndSR

Port2TransmitB1:
    jsr      Port2B1TDESet
    move.l    D0,P2B1TR(A5)
    jsr      Port2B1TDEReset
    bra      EndSR

Port2TransmitB2:
    jsr      Port2B2TDESet
    move.l    D0,P2B2TR(A5)
    jsr      Port2B2TDEReset
    bra      EndSR

```



```

Port2TxD: jsr          Port2DTDESet
          move.b       D6,P2DTR(A5)
          jsr          Port2DTDEReset
          bra          EndSR

Port3TransmitB1:
          jsr          Port3B1TDESet
          move.l       D0,P3B1TR(A5)
          jsr          Port3B1TDEReset
          bra          EndSR

Port3TransmitB2:
          jsr          Port3B2TDESet
          move.l       D0,P3B2TR(A5)
          jsr          Port3B2TDEReset
          bra          EndSR

Port3TxD: jsr          Port3DTDESet
          move.b       D6,P3DTR(A5)
          jsr          Port3DTDEReset
          bra          EndSR

;*****
;                               *
;          End of Transmit Bx   *
;*****
;                               *
;          Bx RDF reset        *
;*****

Port0B1RDFReset:
          nop
          move.w       P0PSR(A5),D3          ; To make sure B1RDF is reset
          andi.l       #$00000001,D3
          cmp.l        #$00000000,D3
          bne          Port0B1RDFReset
          rts

Port0B2RDFReset:
          nop
          move.w       P0PSR(A5),D3          ; To make sure B2RDF is reset
          andi.l       #$00000002,D3
          cmp.l        #$00000000,D3
          bne          Port0B2RDFReset
          rts

Port0DRDFReset:
:          nop
          move.w       P0PSR(A5),D3
          andi.l       #$00000004,D3
          cmp.l        #$00000000,D3
          bne          Port0DRDFReset
          rts

Port1B1RDFReset:
          nop
          move.w       PnPSR1(A5),D3
          andi.l       #$00000001,D3
          cmp.l        #$00000000,D3
          bne          Port1B1RDFReset
          rts

Port1B2RDFReset:
          nop
          move.w       PnPSR1(A5),D3
          andi.l       #$00000002,D3

```

Appendix A Software Configuration

```

    cmp.l          #$00000000,D3
    bne           Port1B2RDFReset
    rts

Port2B1RDFReset:
    nop
    move.w       P2PSR(A5),D3
    andi.l       #$00000001,D3
    cmp.l        #$00000000,D3
    bne         Port2B1RDFReset
    rts

Port2B2RDFReset:
    nop
    move.w       P2PSR(A5),D3
    andi.l       #$00000002,D3
    cmp.l        #$00000000,D3
    bne         Port2B2RDFReset
    rts

Port2DRDFReset:
    nop
    move.w       P2PSR(A5),D3
    andi.l       #$00000004,D3
    cmp.l        #$00000000,D3
    bne         Port2DRDFReset
    rts

Port3B1RDFReset:
    nop
    move.w       P3PSR(A5),D3
    andi.l       #$00000001,D3
    cmp.l        #$00000000,D3
    bne         Port3B1RDFReset
    rts

Port3B2RDFReset:
    nop
    move.w       P3PSR(A5),D3
    andi.l       #$00000002,D3
    cmp.l        #$00000000,D3
    bne         Port3B2RDFReset
    rts

Port3DRDFReset:
    nop
    move.w       P3PSR(A5),D3
    andi.l       #$00000004,D3
    cmp.l        #$00000000,D3
    bne         Port3DRDFReset
    rts
;*****
;*           End of Bx RDF Procedures           *
;*****
;*           Bx TDE Set                         *
;*****
Port0B1TDESet:nop
    move.w       P0PSR(A5),D3          ; to make sure B1TDE is set
    andi.l       #$00000008,D3
    cmp.l        #$00000008,D3
    bne         Port0B1TDESet
    rts

Port0B2TDESet:nop
    move.w       P0PSR(A5),D3          ; to make sure B2TDE is set
    andi.l       #$00000010,D3
    cmp.l        #$00000010,D3

```

```

        bne          Port0B2TDESet
        rts
Port0DTDESet:nop
        move.w      P0PSR(A5),D3
        andi.l      #$00000020,D3
        cmp.l       #$00000020,D3
        bne          Port0DTDESet
        rts
Port1B1TDESet:nop
        move.w      P1PSR(A5),D3
        andi.l      #$00000008,D3
        cmp.l       #$00000008,D3
        bne          Port1B1TDESet
        rts
Port1B2TDESet:nop
        move.w      P1PSR(A5),D3
        andi.l      #$00000010,D3
        cmp.l       #$00000010,D3
        bne          Port1B2TDESet
        rts
Port1DTDESet:nop
        move.w      P1PSR(A5),D3
        andi.l      #$00000020,D3
        cmp.l       #$00000020,D3
        bne          Port1DTDESet
        rts
Port2B1TDESet:nop
        move.w      P2PSR(A5),D3
        andi.l      #$00000008,D3
        cmp.l       #$00000008,D3
        bne          Port2B1TDESet
        rts
Port2B2TDESet:nop
        move.w      P2PSR(A5),D3
        andi.l      #$00000010,D3
        cmp.l       #$00000010,D3
        bne          Port2B2TDESet
        rts
Port2DTDESet:nop
        move.w      P2PSR(A5),D3
        andi.l      #$00000020,D3
        cmp.l       #$00000020,D3
        bne          Port2DTDESet
        rts
Port3B1TDESet:nop
        move.w      P3PSR(A5),D3
        andi.l      #$00000008,D3
        cmp.l       #$00000008,D3
        bne          Port3B1TDESet
        rts
Port3B2TDESet:nop
        move.w      P3PSR(A5),D3
        andi.l      #$00000010,D3
        cmp.l       #$00000010,D3
        bne          Port3B2TDESet
        rts
Port3DTDESet:nop
        move.w      P3PSR(A5),D3
        andi.l      #$00000020,D3
        cmp.l       #$00000020,D3
        bne          Port3DTDESet
        rts
;*****
;          End of Bx TDE Set procedures          *

```

Freescale Semiconductor, Inc.

Appendix A Software Configuration

```
*****
;*      Bx TDE Reset      *
*****
Port0B1TDEReset:nop
    move.w      P0PSR(A5),D3          ; to make sure B1TDE is Reset
    andi.l      #$00000008,D3
    cmp.l       #$00000000,D3
    bne        Port0B1TDEReset
    rts

Port0B2TDEReset:nop
    move.w      P0PSR(A5),D3          ; to make sure B2TDE is Reset
    andi.l      #$00000010,D3
    cmp.l       #$00000000,D3
    bne        Port0B2TDEReset
    rts

Port0DTDEReset:nop
    move.w      P0PSR(A5),D3
    andi.l      #$00000020,D3
    cmp.l       #$00000000,D3
    bne        Port0DTDEReset
    rts

Port1B1TDEReset:nop
    move.w      P1PSR(A5),D3
    andi.l      #$00000008,D3
    cmp.l       #$00000000,D3
    bne        Port1B1TDEReset
    rts

Port1B2TDEReset:nop
    move.w      P1PSR(A5),D3
    andi.l      #$00000010,D3
    cmp.l       #$00000000,D3
    bne        Port1B2TDEReset
    rts

Port1DTDEReset:nop
    move.w      P1PSR(A5),D3
    andi.l      #$00000020,D3
    cmp.l       #$00000000,D3
    bne        Port1DTDEReset
    rts

Port2B1TDEReset:nop
    move.w      P2PSR(A5),D3
    andi.l      #$00000008,D3
    cmp.l       #$00000000,D3
    bne        Port2B1TDEReset
    rts

Port2B2TDEReset:nop
    move.w      P2PSR(A5),D3
    andi.l      #$00000010,D3
    cmp.l       #$00000000,D3
    bne        Port2B2TDEReset
    rts

Port2DTDEReset:nop
    move.w      P2PSR(A5),D3
    andi.l      #$00000020,D3
    cmp.l       #$00000000,D3
    bne        Port2DTDEReset
    rts

Port3B1TDEReset:nop
    move.w      P3PSR(A5),D3
    andi.l      #$00000008,D3
    cmp.l       #$00000000,D3
    bne        Port3B1TDEReset
    rts
```

```

Port3B2TDEReset:nop
    move.w        P3PSR(A5),D3
    andi.l        #$00000010,D3
    cmp.l         #$00000000,D3
    bne          Port3B2TDEReset
    rts
Port3DTDEReset:nop
    move.w        P3PSR(A5),D3
    andi.l        #$00000020,D3
    cmp.l         #$00000000,D3
    bne          Port3DTDEReset
    rts
    
```

The following file describes all the aperiodic interrupts that only occur in GCI mode of operation. Given that the ports are supposed to work in conjunction with periodic process, this example below does not show IE tests. The file is as follows:

```

; PLIC Aperiodic Interrupt Subroutine
; Can handle up to 4 GCI Ports in a dynamic way
; As soon as the action has been taken, the program counter quits
; the interrupt. In case of many interrupts at the same time, the
; program counter will re-enter the Aperiodic Interrupt.
; All the comments will be written for the Port0. For the other Ports, the
; same comments apply.
; Use of register
; D1: PASR Value
; D2: No use
; D3: Monitor Channel Receive or Transmit register
; D4: Monitor Channel Receive or Transmit Buffer of D3
; D5: First Byte of the Received Monitor Channel Register
; D6: Second Byte of the Received Monitor Channel Register
; D7: Buffer of D1
    
```

```

i_PLIC_Aperiodic:
    move.w        PASR(A5),D1        ; Store PASR into D1
    move.w        D1,D7              ; Buffering D1 into D7
    andi.l        #$0000000F,D7     ; mask D7
    cmp.l         #$0,D7            ; compare with 0
    bne          AperPort0         ; go to AperPort0 otherwise
    move.w        D1,D7              ; Store PASR into D7
    andi.l        #$000000F0,D7     ; mask D7
    cmp.l         #$0,D7            ; compare with 0
    bne          AperPort1         ; go to AperPort1 otherwise
    move.w        D1,D7              ; Store PASR into D7
    andi.l        #$00000F00,D7     ; mask D7
    cmp.l         #$0,D7            ; compare with 0
    bne          AperPort2         ; go to AperPort2 otherwise
    move.w        D1,D7              ; Store PASR into D7
    andi.l        #$0000F000,D7     ; mask D7
    cmp.l         #$0,D7            ; compare with 0
    bne          AperPort3         ; go to AperPort3 otherwise
    bra          EndASR             ; exit w/o taking action
    
```

```

AperPort0:
    move.w        D1,D7              ; Store PASR into D7
    andi.l        #$00000008,D7     ; mask D7
    cmp.l         #$00000008,D7     ; compare with $8
    beq          Port0CommandIndRx ; if equal, go to CI Receive
    move.w        D1,D7              ; Store PASR into D7
    andi.l        #$00000004,D7     ; mask D7
    
```

Freescale Semiconductor, Inc.

Appendix A Software Configuration

```
    cmp.l      #$00000004,D7      ; compare with $4
    beq       Port0CommandIndTx   ; if equal, go to CI Transmit
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000002,D7      ; mask D7
    cmp.l     Port0MonitorChanRx  ; compare with $2
    beq       D1,D7              ; if equal, go to MC receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000001,D7      ; mask D7
    cmp.l     Port0MonitorChanTx  ; compare with $1
    beq       D1,D7              ; if equal, go to MC Transmit

AperPort1:
    move.w    PASR(A5),D1        ; Store PASR into D7
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000080,D7      ; mask D7
    cmp.l     Port1CommandIndRx   ; compare with $8
    beq       D1,D7              ; if equal, go to CI Receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000040,D7      ; mask D7
    cmp.l     Port1CommandIndTx   ; compare with $4
    beq       D1,D7              ; if equal, go to CI Transmit
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000020,D7      ; mask D7
    cmp.l     Port1MonitorChanRx  ; compare with $2
    beq       D1,D7              ; if equal, go to MC receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000010,D7      ; mask D7
    cmp.l     Port1MonitorChanTx  ; compare with $1
    beq       D1,D7              ; if equal, go to MC Transmit

AperPort2:
    move.w    PASR(A5),D1        ; Store PASR into D7
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000800,D7      ; mask D7
    cmp.l     Port2CommandIndRx   ; compare with $8
    beq       D1,D7              ; if equal, go to CI Receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000400,D7      ; mask D7
    cmp.l     Port2CommandIndTx   ; compare with $4
    beq       D1,D7              ; if equal, go to CI Transmit
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000200,D7      ; mask D7
    cmp.l     Port2MonitorChanRx  ; compare with $2
    beq       D1,D7              ; if equal, go to MC receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00000100,D7      ; mask D7
    cmp.l     Port2MonitorChanTx  ; compare with $1
    beq       D1,D7              ; if equal, go to MC Transmit

AperPort3:
    move.w    PASR(A5),D1        ; Store PASR into D7
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00008000,D7      ; mask D7
    cmp.l     Port3CommandIndRx   ; compare with $8
    beq       D1,D7              ; if equal, go to CI Receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00004000,D7      ; mask D7
    cmp.l     Port3CommandIndTx   ; compare with $4
    beq       D1,D7              ; if equal, go to CI Transmit
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00002000,D7      ; mask D7
    cmp.l     Port3MonitorChanRx  ; compare with $2
    beq       D1,D7              ; if equal, go to MC receive
    move.w    D1,D7              ; Store PASR into D7
    andi.l    #$00001000,D7      ; mask D7
    cmp.l     Port3MonitorChanTx  ; compare with $1
    beq       D1,D7              ; if equal, go to MC Transmit
```

```

EndASR    nop
          move.l    #$0,D1          ; clear all registers
          move.l    #$0,D2
          move.l    #$0,D3
          move.b    #$0,D4
          move.b    #$0,D5
          move.b    #$0,D6
          move.l    #$0,D7
          rte          ; End of ISR
;*****
;*          Port0 Subroutines          *
;*****
;*          Monitor Channel Subroutine  *
;*****
Port0MonitorChanTx:
          move.b    PGMTS(A5),D3    ; read PGMTS to clear the bit
          jsr      Port0GMTCheck    ; to make sure GMT=0
          bra      EndASR          ; ends the CI Tx
Port0MonitorChanRx:
          move.w    POGMR(A5),D3    ; read GMR
          jsr      Port0GMRCheck    ; to make sure R=0
          move.w    D3,D4          ; Save D3 into D4
          andi.l    #$000000FF,D4
          cmp.l    #$00000035,D4    ; to access to NR5 (MC145574)
          beq      FirstByte       ; if equal go to FirstByte
          move.w    D3,D4          ; otherwise continue
          andi.l    #$000000FF,D4
          cmp.l    #$000000CF,D4    ; $CF is NR5 Value of MC145574
          beq      SecondByte      ; if equal go to SecondByte
          bra      EndASR          ; ends the ISR
FirstByte:
          move.w    D3,D5          ; move D3 into D5 to check
          bra      EndASR
SecondByte:
          move.w    D3,D6          ; move D3 into D6 to check
          bra      EndASR
;*****
;*          Command Indicate Subroutine *
;*****
Port0CommandIndTx:
          move.l    PGCITSR(A5),D3  ; clear the bit
          jsr      Port0Rcheck      ; to make sure R=0
          bra      EndASR
Port0CommandIndRx:
          move.l    #0,D3
          move.b    POGCIR(A5),D3   ; Move GCR0 into D3
          andi.l    #$000000FF,D3
          cmp.l    #$00000010,D3    ; Deactivation Request
          beq      Port0DeacReq
          cmp.l    #$00000018,D3    ; Activation Indication Value
          beq      Port0ActInd
          cmp.l    #$0000001C,D3    ; Activation Confirmed
          beq      EndASR          ; nothing to do
          cmp.l    #$0000001F,D3    ; Deactivation Indication
          beq      Port0DeacInd
          bra      EndASR
Port0DeacReq:
          move.b    #$1F,D0         ; Send Deactivation Indication
          move.b    D0,PGCIT(A5)   ; in response of dea Request
          jsr      Port0Rcheck      ; to make sure R=0
          bra      EndASR
Port0DeacConf:
          nop          ; nothing else to do

```

Freescale Semiconductor, Inc.

Appendix A Software Configuration

```

        bra                EndASR
Port0ActInd:
        move.b            #$1C,D0                ; Send Activation Indication
        move.b            d0,P0GCIT(A5)
        jsr               Port0RCheck
        bra                EndASR
Port0DeacInd:
        move.b            #$1F,D0                ; Send Deactivation Indication
        move.b            D0,P0GCIT(A5)
        jsr               Port0RCheck
        bra                EndASR
;*****
;*                Checking Procedure                *
;*****

Port0GMTCheck:
        move.w            PASR(A5),D3            ; to make sure GMT=0
        andi.l            #$00000001,D3
        cmp.l             #$0,D3
        bne               Port0GMTCheck
        rts
Port0GMRCheck:
        move.w            PASR(A5),D4            ; to make sure GMR=0
        andi.l            #$00000002,D4
        cmp.l             #$0,D4
        bne               Port0GMRCheck
        rts
Port0LCheck:
        move.w            P0GMT(A5),D3          ; to make sure L=0
        andi.l            #$00000200,D3
        cmp.l             #$0,D3
        bne               Port0RTxCheck
        rts
Port0RTxCheck:
        move.w            P0GMT(A5),D3          ; to make sure R=0
        andi.l            #$00000100,D3
        cmp.l             #$0,D3
        bne               Port0RTxCheck
        rts
Port0RCheck:
        move.b            P0GCIT(A5),D3        ; to make sure R=0
        andi.l            #$00000010,D3
        cmp.l             #$0,D3
        bne               Port0RCheck
        rts
;*****
;*                Port1 Subroutines                *
;*****
Port1MonitorChanTx:
        move.b            PGMTS(A5),D3
        jsr               Port1GMTCheck        ;to make sure GMT=0
        bra                EndASR
Port1MonitorChanRx:
        move.w            P1GMR(A5),D3
        jsr               Port1GMRCheck        ; to make sure GMR=0
        move.w            D3,D4
        andi.l            #$000000FF,D4
        cmp.l             #$00000035,D4
        beq               FirstByte1
        move.w            D3,D4
        andi.l            #$000000FF,D4
        cmp.l             #$000000CF,D4
        beq               SecondByte1
        bra                EndASR

```



```

FirstByte1:
    move.w        D3,D5
    bra          EndASR
SecondByte1:
    move.w        D3,D6
    bra          EndASR
Port1CommandIndTx:
    move.b        PGCITSR(A5),D3
    jsr          Port1RCheck
    bra          EndASR
;*****
;*          Command Indicate Subroutine          *
;*****
Port1CommandIndRx:
    move.l        #0,D3
    move.b        P1GCIR(A5),D3
    andi.l        #$000000FF,D3
    cmp.l         #$00000010,D3
    beq          Port1DeacReq
    cmp.l         #$00000018,D3
    beq          Port1ActInd
    cmp.l         #$0000001C,D3
    beq          EndASR
    cmp.l         #$0000001F,D3
    beq          Port1DeacInd
    bra          EndASR
Port1DeacReq:
    move.b        #$1F,D0
    move.b        D0,P1GCIT(A5)
    jsr          Port1RCheck
    bra          EndASR
Port1DeacConf:
    nop
    bra          EndASR
Port1ActInd:
    move.b        #$1C,D0
    move.b        d0,P1GCIT(A5)
    jsr          Port1RCheck
    bra          EndASR
Port1DeacInd:
    move.b        #$1F,D0
    move.b        D0,P1GCIT(A5)
    jsr          Port1RCheck
    bra          EndASR
;*****
;*          Checking Procedures          *
;*****
Port1GMTCheck:
    move.w        PASR(A5),D4          ; to make sure GMT=0
    andi.l        #$00000010,D4
    cmp.l         #$0,D4
    bne          Port1GMTCheck
    rts
Port1GMRCheck:
    move.w        PASR(A5),D4          ; to make sure GMR=0
    andi.l        #$00000020,D4
    cmp.l         #$0,D4
    bne          Port1GMRCheck
    rts
Port1LCheck:
    move.w        P1GMT(A5),D4          ; to make sure L=0
    andi.l        #$00000200,D4
    cmp.l         #$0,D4

```

Appendix A Software Configuration

```

        bne          Port1LCheck
        rts
Port1RTxCheck:
        move.w      P1GMT(A5),D4          ; to make sure R=0
        andi.l      #$00000100,D4
        cmp.l       #$0,D4
        bne          Port1RTxCheck
        rts
Port1RCheck:
        move.b      P1GCIT(A5),D4        ; to make sure R=0
        andi.l      #$00000010,D4
        cmp.l       #$0,D4
        bne          Port1RCheck
        rts
;*****
;*          Port2 Subroutines          *
;*****
;          Monitor Channel Subroutines *
;*****

Port2MonitorChanTx:
        move.b      PGMTS(A5),D3
        jsr         Port2GMTCheck        ; to make sure GMT=0
        bra         EndASR
Port2MonitorChanRx:
        move.w      P2GMR(A5),D3
        jsr         Port2GMRCheck        ; to make sure GMR=0
        move.w      D3,D4
        andi.l      #$000000FF,D4
        cmp.l       #$00000035,D4
        beq         FirstByte2
        move.w      D3,D4
        andi.l      #$000000FF,D4
        cmp.l       #$000000CF,D4
        beq         SecondByte2
        bra         EndASR
FirstByte2:
        move.w      D3,D5
        bra         EndASR
SecondByte2:
        move.w      D3,D6
        bra         EndASR
;*****
;*          Command Indicate Subroutine *
;*****

Port2CommandIndTx:
        move.b      PGCITSR(A5),D3
        jsr         Port2RCheck
        bra         EndASR

Port2CommandIndRx:
        move.l      #0,D3
        move.b      P2GCIR(A5),D3
        andi.l      #$000000FF,D3
        cmp.l       #$00000010,D3
        beq         Port2DeacReq
        cmp.l       #$00000018,D3
        beq         Port2ActInd
        cmp.l       #$0000001C,D3
        beq         EndASR
        cmp.l       #$0000001F,D3
        beq         Port2DeacInd
        bra         EndASR

```

```

Port2DeacReq:
    move.b    #$1F,D0
    move.b    D0,P2GCIT(A5)
    jsr      Port2RCheck
    bra      EndASR
Port2DeacConf:
    bra      EndASR
Port2ActInd:
    move.b    #$1C,D0
    move.b    d0,P2GCIT(A5)
    jsr      Port2RCheck
    bra      EndASR
Port2DeacInd:
    move.b    #$1F,D0
    move.b    D0,P2GCIT(A5)
    jsr      Port2RCheck
    bra      EndASR
;*****
;*          Checking Subroutines          *
;*****

Port2GMTCheck:
    move.w    PASR(A5),D4                ; to make sure GMT=0
    andi.l    #$00000100,D4
    cmp.l     #$0,D4
    bne      Port2GMTCheck
    rts
Port2GMRCheck:
    move.w    PASR(A5),D4                ; to make sure GMR=0
    andi.l    #$00000200,D4
    cmp.l     #$0,D4
    bne      Port2GMRCheck
    rts
Port2LCheck:
    move.w    P2GMT(A5),D4              ; to make sure L=0
    andi.l    #$00000200,D4
    cmp.l     #$0,D4
    bne      Port2LCheck
    rts
Port2RTxCheck:
    move.w    P2GMT(A5),D4              ; to make sure R=0
    andi.l    #$00000100,D4
    cmp.l     #$0,D4
    bne      Port2RTxCheck
    rts
Port2RCheck:
    move.b    P2GCIT(A5),D4             ; to make sure R=0
    andi.l    #$00000010,D4
    cmp.l     #$0,D4
    bne      Port2RCheck
    rts
;*****
;*          Port3 Subroutines          *
;*****
;*          Monitor Channel Subroutines          *
;*****

Port3MonitorChanTx:
    move.b    PGMTS(A5),D3
    jsr      Port3GMTCheck                ; to make sure GMT=0
    bra      EndASR
Port3MonitorChanRx:
    move.w    P3GMR(A5),D3
    jsr      Port3GMRCheck                ; to make sure GMR=0
    move.w    D3,D4

```

Appendix A Software Configuration

```

    andi.l    #$000000FF,D4
    cmp.l    #$00000035,D4
    beq      FirstByte3
    move.w   D3,D4
    andi.l    #$000000FF,D4
    cmp.l    #$000000CF,D4
    beq      SecondByte3
    bra      EndASR
FirstByte3:
    move.w   D3,D5
    bra      EndASR
SecondByte3:
    move.w   D3,D6
    bra      EndASR
;*****
;*          Command Indicate Subroutine          *
;*****

Port3CommandIndTx:
    move.b   PGCITSR(A5),D3
    jsr      Port3RCheck
    bra      EndASR

Port3CommandIndRx:
    move.l   #0,D3
    move.b   P3GCIR(A5),D3
    andi.l   #$000000FF,D3
    cmp.l   #$00000010,D3
    beq     Port3DeacReq
    cmp.l   #$00000018,D3
    beq     Port3ActInd
    cmp.l   #$0000001C,D3
    beq     EndASR
    cmp.l   #$0000001F,D3
    beq     Port3DeacInd
    bra     EndASR

Port3DeacReq:
    move.b   #$1F,D0
    move.b   D0,P3GCIT(A5)
    jsr      Port3RCheck
    bra      EndASR

Port3DeacConf:
    nop
    bra      EndASR

Port3ActInd:
    move.b   #$1C,D0
    move.b   d0,P3GCIT(A5)
    jsr      Port3RCheck
    bra      EndASR

Port3DeacInd:
    move.b   #$1F,D0
    move.b   D0,P3GCIT(A5)
    jsr      Port3RCheck
    bra      EndASR
;*****
;*          Checking Subroutines          *
;*****

Port3GMTCheck:
    move.w   PASR(A5),D4          ; to make sure GMT=0
    andi.l   #$00001000,D4
    cmp.l   #$0,D4
    bne     Port3GMTCheck
    rts

```

```

Port3GMRCheck:
    move.w        PASR(A5),D4        ; to make sure GMR=0
    andi.l        #$00002000,D4
    cmp.l         #$0,D4
    bne          Port3GMRCheck
    rts

Port3LCheck:
    move.w        P3GMT(A5),D4      ; to make sure L=0
    andi.l        #$00000200,D4
    cmp.l         #$0,D4
    bne          Port3LCheck
    rts

Port3RTxCheck:
    move.w        P3GMT(A5),D4      ; to make sure R=0
    andi.l        #$00000100,D4
    cmp.l         #$0,D4
    bne          Port3RTxCheck
    rts

Port3RCheck:
    move.b        P3GCIT(A5),D4     ; to make sure R=0
    andi.l        #$00000010,D4
    cmp.l         #$0,D4
    bne          Port3RCheck
    rts
    
```

The following file is called CoreInit.s. It is the core of the program with the program counter starting at the address defined by the user. Obviously, this program will require modifications depending on the evaluation the user wants to perform. This example shows a very generic flow. Users are welcome to make further modifications, which should not affect the core of the program itself.

```

Code_Start:
    nop
    move.l        #$40000000,A3      ; address allocation
    move.l        #$10000000,A6
    move.l        #$10000000,A5
    move.l        #$20001000,A7
    jsr          IntInit             ; Interrupt Initialization
    jsr          RegisterInit        ; Register Initialization
    jsr          GCIInit             ; if used, GCI mode on
    jsr          GCIIIntEnable       ; GCI Interrupt on
    jsr          WaitLoop            ; loop to configure in GCI
    jsr          CI2F                ;
    jsr          MonitorAbort        ; Monitor initialization
    jsr          CICCommand          ; to Initialization CI
    jsr          ST1BchannelEn       ; to Send Value to MC
    jsr          ReadPort0MC
    rts
    
```

The file shown below is the initialization file configuring some registers in order to perform the right tests. Before performing any tests, the user definitely needs to know the PLIC registers set to make sure the PLIC configuration will match the test requirements. The following example tests Port1 in GCI Slave Mode. Some monitor channel information is sent and the 2kHz rate works. The other ports are off.

```

;*****
;* Different Subroutine used for init and checking GCI/IDL *
;*****
GCIInit:
    
```

```

move.w      #$0003,d0      ; port0 off, GCI, B1,B2 on
move.w      d0,PLCR0(A5)  ;
move.w      #$A203,D0     ;port1 on,S, FSM, GCI,B1,B2 on
move.w      D0,PLCR1(A5)  ;
move.w      #$0003,D0     ; port3 off,Slave,GCI, B1,B2 on
move.w      D0,PLCR3(A5)  ;
move.w      #$0003,D0     ; port2 off,Slave,GCI, B1,B2 on
move.w      D0,PLCR2(A5)  ;
move.b      PGMITS(A5),D0
move.b      PGCITSR(A5),D0
move.b      POGCIR(A5),D0
move.w      #$0000,D0     ; NPM disabled
move.w      D0,PCSR(A5)   ; DCL=512kHz, MULT=64, MUX=FSC
move.w      #$0000,D0
move.w      D0,P0SDR(A5)
move.w      #$0000,D0
move.w      D0,P1SDR(A5)  ; $1E delay,Max delay at 512kHz
move.w      #$0000,D0    ; $40
move.w      D0,P2SDR(A5)  ; Total delay=$20 but not used
move.w      #$0000,D0
move.w      D0,P3SDR(A5)  ; Total delay=$20 but not used
move.w      #$0000,D0    ; D channel off
move.w      D0,PDRQR(A5)
rts

```

GCIIntEnable:

```

move.w      #$0F00,D0     ; IE=0, GCI Interrupt Off
move.w      D0,P0ICR(A5)  ; B1,B2, D Interrupt Off
move.w      #$001B,D0     ; Interrupt off on Port2
move.w      D0,P2ICR(A5)
move.w      #$0000,D0     ; Interrupt off on Port3
move.w      D0,P3ICR(A5)
move.w      #$8F1B,D0     ; IE=1, GCI Interrupts On,
move.w      D0,P1ICR(A5)  ; B1,B2 Interrupts On
rts

```

MonitorAbort:

```

move.b      #$20,D2
move.b      D2,PGMTA(A5)  ; Abort previous Port1 GCI MC
rts

```

CI2F:

```

move.b      #$1F,D0       ; Port1 Deactivation Request
move.b      D0,P1GCIT(A5)
rts

```

ST1BChannelEn:

```

; Send Monitor Channel

```

```

lea.l      B1B2En,A2
Loop0:     nop
move.w      (A2)+,D0
jsr        RTxCheck1
move.w      D0,P1GMT(A5)
and.l      #$000000AA,D0
cmp.l      #$000000AA,D0
beq        EndLoop2
bra        Loop0
EndLoop2:  nop
rts

```

RTxCheck1:

```

move.w      P1GMT(A5),D2
andi.l     #$00000100,D2

```

```

        cmp.l      #$0,D2
        bne       RTxCheck1
        rts

CICCommand:nop
        lea.l     CICom,A2
        jsr      RCheck
LoopCI:  move.b   (A2)+,D0
        andi.l   #$000000FF,D0
        jsr      RCheck
        move.b   D0,P1GCIT(A5)
        cmp.l   #$1C,D0
        bne     LoopCI
        rts

CheckACK:nop
        move.b   PGCITSR(A5),D1
        andi.l   #$02,D1
        cmp.l   #$02,D1
        bne     CheckACK
        rts

RCheck:  nop
        move.b   P1GCIT(A5),D2
        andi.l   #$00000010,D2
        cmp.l   #$00000010,D2
        beq     RCheck
        rts

WaitLoop:
        nop
        Move.l   #$FFF,D0
LoopGCI: sub.l   #1,D0
        cmp.l   #0,D0
        bne     LoopGCI
        rts
    
```

The following file is used to configure the ColdFire[®] CPU core. Before using this file, the user must check to make sure the configuration matches his requirements. The file is as follows:

```

RegisterInit;; to initialize the registers
        clr.l    D0
        clr.l    D1
        clr.l    D2
        clr.l    D3
        clr.l    D4
        clr.l    D5
        clr.l    D6
        clr.l    D7
        rts

IntInit:
        move.l   #VBR_Init,D0          ; to set the vector base reg.
        movec   D0,VBR
        move.w   #$2400,D0             ; to set the status register
        move.w   D0,SR
        move.b   #$40,D0              ; to point the interrupts
        move.b   D0,PIVR(A6)
        move.l   #$88888888,D0        ; Disable all type of ISR
        move.l   D0,ICR1(A6)
        move.l   D0,ICR3(A6)
        move.l   D0,ICR4(A6)          ; reset all types of interrupt
        move.l   #$88EF8888,D0        ; PLIC APer Interrupt on, level 7
        move.l   D0,ICR2(A6)          ; PLIC Per Interrupt on level 6
    
```

Freescale Semiconductor, Inc.

Appendix A Software Configuration

```

move.l    #$0,D0                ; this case.
move.l    D0,PIWR(A6)           ; No wake up process yet
move.l    #i_PLIC_Periodic,D0  ; Point to the address vector
move.l    D0,i_plic_per_vec
move.l    #i_PLIC_Aperiodic,D0 ; Point to the address vector
move.l    D0,i_plic_aper_vec
rts

```

The final file represents the table allocation of configuration data that can be sent to the peripheral to control various functions:

```

B1B2En:
DC.W      $0125                ; MC145574 NR5 access
DC.W      $01C0                ; Enabling the B1/B2 Channels
DC.W      $03AA                ; End of sending (should see $FF)
DC.W      $0126                ; MC145574 NR5 access
DC.W      $0180                ; Enabling Loopabck
DC.W      $03FF                ; End of sending
DC.W      $0105                ; Access to BR5
DC.W      $01EE                ; Send $EE
DC.W      $03FF                ; End of sending

B1B2Read:
DC.W      $0135                ; Read NR5
DC.W      $03FF                ; End of Process

B1Send:
DC.L      $00112233            ; Value written to TBx
DC.L      $A00A5FF5            ; Value written to TBx
DC.L      $F708D728            ; Value written to TBx
DC.L      $FFFFFFFF            ; Value written to TBx

CICom:   DC.B      $18          ; Command Indicate Activation Request
          DC.B      $1C          ; Command Indicate Activation Confirmed

TxData:  DC.B      $95          ;
          DC.B      $A5          ;
          DC.B      $AA          ;
;        DC.W      $03FF        ; End of sending

B1B2Read:
DC.W      $0135                ; Read NR5
DC.W      $03FF                ; End of Process

B1Send:
DC.L      $00112233            ; Value written to TBx
DC.L      $A00A5FF5            ; Value written to TBx
DC.L      $F708D728            ; Value written to TBx
DC.L      $FFFFFFFF            ; Value written to TBx

CICom:   DC.B      $18          ; Command Indicate Activation Request
          DC.B      $1C          ; Command Indicate Activation Confirmed

TxData:  DC.B      $95          ;
          DC.B      $A5          ;
          DC.B      $AA          ;


```


Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

DigitalDNA and ColdFire are trademarks of Motorola, Inc.

Information in this document is provided solely to enable system and software implementers to use ColdFire microprocessors. There are no express or implied copyright licenses granted hereunder to design or fabricate ColdFire 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  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

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: ColdFire Applications Engineering

WORLD WIDE WEB ADDRESSES: <http://www.motorola.com/semiconductors>



MOTOROLA

**For More Information On This Product,
Go to: www.freescale.com**