Application Note

AN2333/D
Rev. 0, 10/2002

Maximizing the Performance of Two Fast Ethernet Links on MSC8101 FCCs

by Philippe Chartier

CONTENTS

This application note describes how to implement a dual full-duplex Fast Ethernet driver on the MSC8101ADS board. It examines the on-board Ethernet transceiver configuration and the fast communication controller (FCC) Ethernet configuration on the MSC8101. It considers different ways to optimize these configurations and illustrates them with an example MSC8101 data processing set-up that yields high performance in terms of Ethernet link bandwidth. Example code provided with this application note is reusable and can be rapidly integrated into projects or a real-time operating system (RTOS) board software package (BSP).

1 IEEE 802.3 Ethernet Basics

Ethernet is the most widely used local area network (LAN) technology and is specified in the IEEE 802.3 standard. An Ethernet LAN can use different media: coaxial cable, unshielded twisted pair copper wires, radio frequencies, fiber, and so on. These Ethernet LAN can work at different frequencies, such as 10 Mbps, 100 Mbps, 1 Gpbs, and even 10 Gbps/s. Operating on the LAN are computers, terminal equipment, and other devices that interconnect the LAN between them on a hub or switch. Devices connect to the medium and compete for access using a carrier sense multiple access with collision detection (CSMA/CD) protocol. The Ethernet frame is structured as follows:

- 7-byte preamble of alternating ones and zeros.
- Start frame delimiter (SFD) that marks the beginning of the frame.
- 48-bit destination address and 48-bit source address.
- Ethernet type IEEE 802.3-length field that signifies the protocol.
- Length field that specifies the length of the data portion of the frame. For Ethernet and IEEE 802.3 frames to exist on the same LAN, the length field must be unique on the Ethernet. This requirement limits the length of the data portion of the frame to 1,500 bytes and therefore the total frame length to 1,518 bytes.
- Data (46–1500 bytes).
- Four-byte frame-check sequence (FCS), which is the standard 32-bit CCITT-cyclic reduncancy check (CRC) polynomial used in many protocols.

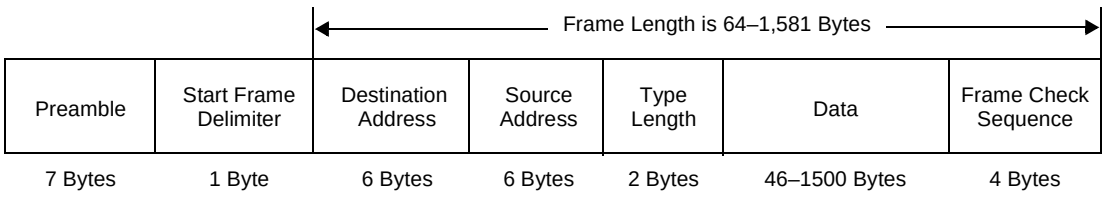

Figure 1. Ethernet Frame Format

1.1 Ethernet Transceiver

In most systems, the Ethernet medium is accessed from the controller through an Ethernet transceiver. This section discusses how the Ethernet transceiver on the MSC8101ADS is configured and also describes the bus interface between this transceiver and the MSC8101.

The MSC8101ADS uses the LXT970 Ethernet transceiver to access a 10/100 Base-T Ethernet port that ends with a standard RJ-45 Ethernet jack. The main features of this Ethernet transceiver are as follows:

- IEEE 802.3 compliant.
- 10 Base-T and 100 Base-TX using a single RJ-45 connection.
- Support for auto negotiation
- Media-independent interface (MII) with extended register capability.
- 100Base-FX fiber optic capable.
- Standard CSMA/CD or full-duplex operation.
- Configurable via MII serial port or external control pins.
- Configurable for DTE or switch applications.
- Integrated LED drivers.

The Ethernet transceiver hardware configuration pins determine the default behavior of the transceiver when it is connected to a line. These pins also provide the transceiver with a 5-bit MII address. The MSC8101 device interconnects through two pins, MDC and MDIO, that provide a bidirectional serial bus to access the device registers for configuration. Several Ethernet transceivers can share the MDC and MDIO serial interface signals as long as they have different MII addresses. Also, Ethernet transceivers from the same manufacturers have in common the base subset of registers, facilitating their control software design.[1] **Table 1** shows the transceiver registers and bits that must be configured.

Table 1. Main Transceiver Registers

Register	Bits
Register 0: Control Register	0.15 Reset chip 0.14 Enable Loopback 0.13 Set Link Speed Selection 0.12 Auto-Negotiation Mode Enable 0.09 Restart Auto-Negotiation Process 0.08 Set Link Duplex Mode
Register 1: Status Register	1.2 Get Link Status (Up/Down)
Register 4: Auto-Negotiation Advertisement Register	4.8 100BASE-TX full-duplex capable 4.7 100BASE-TX half-duplex capable 4.6 10BASE-T full-duplex capable 4.5 10BASE-T half-duplex capable 4.4:0 Selector Field - <00001> = IEEE 802.3
Register 20: Chip Status Register	20.12 Get Duplex Mode 0.11 Get Speed

[1] For details, see the Intel® data sheet for the LXT970/970A Fast Ethernet transceiver, which is available at http://www.intel.com under Networking and Communications Design Components.

1.2 Media Independent Interface (MII)

There are different interconnect bus formats for interfacing Ethernet controllers and transceivers, such as MII, reduced media independent interface (RMII), and serial media independent interface (SMII). The MSC8101 FCCs use the MII bus format. **Table 2** and **Table 3** show the MII device pinout for FCC2 and FCC1. The parentheses delineate the pinout of the MSC8101ADS board.

Table 2. FCC2 MII Device Pinout and Board Pinout

Port Line	Function Set	Direction	Function
Port C28 (D04)	CLK4/TX_CLK	In	FCC2 Tx Ethernet
Port B31 (C01)	TX_ER	Out	FCC2 Tx Ethernet
Port B29 (C03)	TX_EN	Out	FCC2 Tx Ethernet
Port B22 (C10)	TXD0	Out	FCC2 Tx Ethernet
Port B23 (C09)	TXD1	Out	FCC2 Tx Ethernet
Port B24 (C08)	TXD2	Out	FCC2 Tx Ethernet
Port B25 (C07)	TXD3	Out	FCC2 Tx Ethernet
Port C29 (D03)	CLK3/RX_CLK	In	FCC2 Rx Ethernet
Port B28 (C04)	RX_ER	In	FCC2 Rx Ethernet
Port B30 (C02)	RX_EN	In	FCC2 Rx Ethernet
Port B21 (C11)	RXD0	In	FCC2 Rx Ethernet
Port B20 (C12)	RXD1	In	FCC2 Rx Ethernet
Port B19 (C13)	RXD2	In	FCC2 Rx Ethernet
Port B18 (C14)	RXD3	In	FCC2 Rx Ethernet
Port B26 (C06)	CRS	In	FCC2 Control Ethernet
Port B27 (C05)	COL	In	FCC2 Control Ethernet
Port C12 (D20)	MDIO	In/Out	PHY2 Management
Port C13 (D19)	MDC	Out	PHY2 Management

Table 3. FCC1 MII Device Pinout and Board Pinout

Port Line	Function Set	Direction	Function
Port D01 (PC31)	CLK2/TX_CLK	In	FCC1 Tx Ethernet
Port B03 (PA29)	TX_ER	Out	FCC1 Tx Ethernet
Port B04 (PA28)	TX_EN	Out	FCC1 Tx Ethernet
Port B14 (PA21)	TXD0	Out	FCC1 Tx Ethernet
Port B13 (PA20)	TXD1	Out	FCC1 Tx Ethernet
Port B12 (PA19)	TXD2	Out	FCC1 Tx Ethernet
Port B11 (PA18)	TXD3	Out	FCC1 Tx Ethernet
Port D02 (PC30)	CLK1/RX_CLK	In	FCC1 Rx Ethernet
Port B06 (PA26)	RX_ER	In	FCC1 Rx Ethernet
Port B05 (PA27)	RX_EN	In	FCC1 Rx Ethernet

Table 3. FCC1 MII Device Pinout and Board Pinout (Continued)

Port Line	Function Set	Direction	Function
Port B15 (PA17)	RXD0	In	FCC1 Rx Ethernet
Port B16 (PA16)	RXD1	In	FCC1 Rx Ethernet
Port B17 (PA15)	RXD2	In	FCC1 Rx Ethernet
Port B18 (PA14)	RXD3	In	FCC1 Rx Ethernet
Port B02 (PA30)	CRS	In	FCC1 Control Ethernet
Port B01 (PA31)	COL	In	FCC1 Control Ethernet
Port C12 (D20)	MDIO	In/Out	PHY1 Management
Port C13 (D19)	MDC	Out	PHY1 Management

1.3 MSC810ADS FCCs Running Ethernet

The MSC8101 communications processor module (CPM) includes two fast communication controllers (FCCs) that can operate in Ethernet mode up to 100Mbits/s. Key features of these FCCs, which make it easy for the SC140 core to handle the Ethernet payload traffic, are as follows:[2]

- MII
- MAC layer functions of fast Ethernet and IEEE 802.3x
- Framing functions
- Full collision support
- Bit rates up to 100 MBPS
- Multi-buffer data structure
- CRC generation
- Logical. 64-bin group address hash table plus broadcast address checking
- Promiscuous mode
- Special RMON counters for monitoring network statistics.[3] This set of counters monitor the receive side of the FCC in Ethernet mode. They provide an approximate measure of the incoming network traffic without the need to implement software counters.

1.4 Control and Data path

The block diagram shown in **Figure 2** gives an overview of the different control and data flows of MSC8101-based systems for Ethernet operation.

[2] For details on the MSC8101 FCCs, consult the *MSC8101 Reference Manual* (MSC8101RM/D).

[3] For details on RMON counters, consult the chapter on FCC Fast Ethernet Controllers in the *MSC8101 Reference Manual* (MSC8101RM/D).

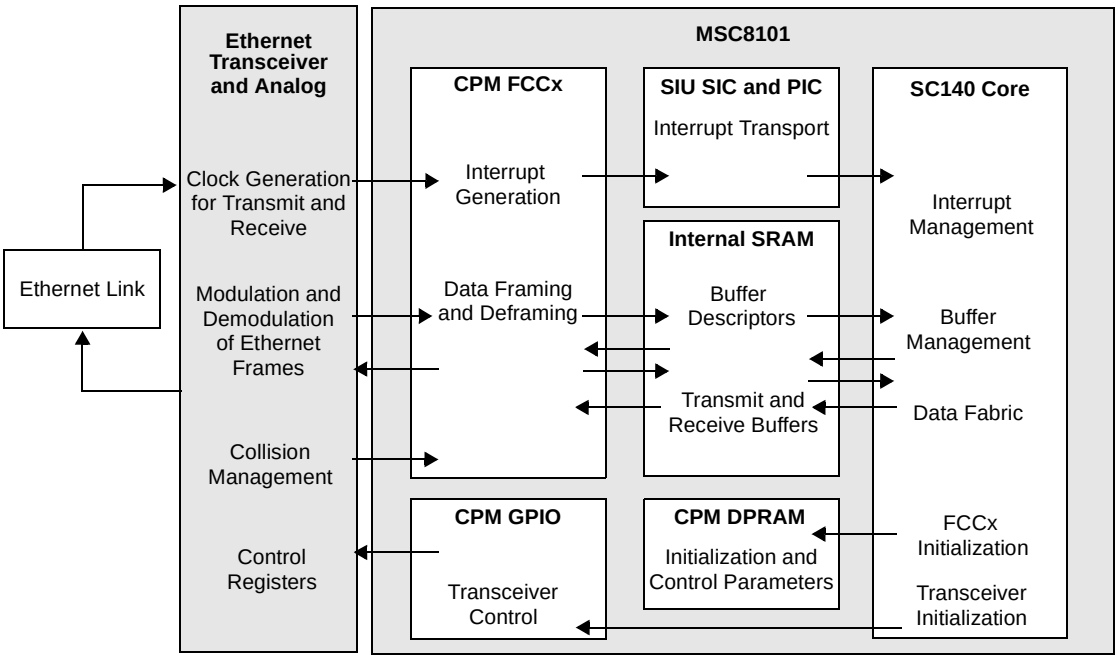

Figure 2. Control and Data Paths

2 2FCCs Project

This section describes the hardware set-up and the software modules of the 2FCCs project.

2.1 Hardware Set-up

There are two example hardware configurations for the 2FCCs project. The first uses two MSC8101ADS boards, so FCC1 is not available and must be disabled in the software project configuration file. The second set-up involves both an MSC8101ADS and an E1 communication (ECOM) board, which is a peripheral extension board designed to connect to the MPC8260ADS.[4] FCC1 uses the ECOM on-board Ethernet transceiver.

2.1.1 Two-MSC8101ADS Set-up

As **Figure 3** shows, the two-MSC8101ADS set-up is defined as follows:

- Crystal = 16,384 MHz; MODCLK40 (yielding SC140 core/CPM/60x-compatible system bus speeds of 196/98/39 MHz). All switches are set to the factory defaults.
- Two PCs with suitable ports (PCI or parallel) for JTAG probes to download, tune, and debug the software.
- A JTAG probe for the MSC8101ADS (Macraigor Systems LLC's OCDemon™ Wiggler)[5]
- One twisted Ethernet cable or one Ethernet switch and two straight Ethernet cables

[4] Documentation on the ECOM board is available on the Motorola web site listed on the back of this document. Consult the MPC8260 product information.

[5] Visit http://www.ocdemon.net/.

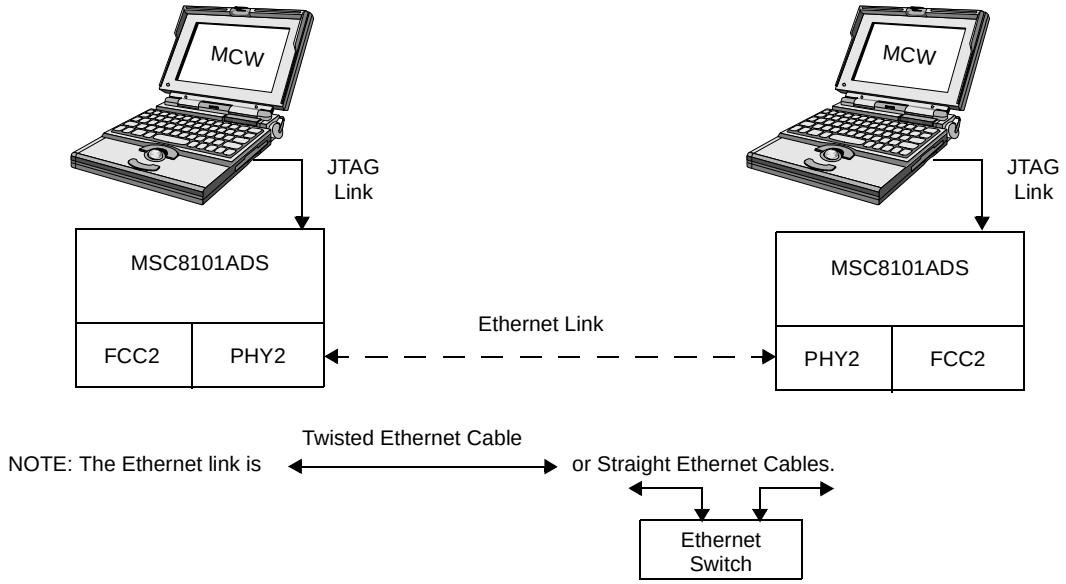

Figure 3. Set-up Using Two MSC8101ADS Boards

2.1.2 MSC8101ADS and ECOM Set-up

The MSC8101ADS-ECOM set-up is preferable to the two-MSC8101ADS set-up because it allows the MSC8101 device to handle two fast Ethernet connections over both FCC1 and FCC2 simultaneously. The MSC8101ADS-ECOM set-up is the one that is used for the 2FCC project described in this application note.

Note: The ECOM can also be replaced by a second ADS with the same board interconnections. However, a small piece of code is required for the second ADS to enable the Ethernet transceiver and disable the local GPIO. The code can be downloaded from a PC or run directly from the on-board Flash memory.

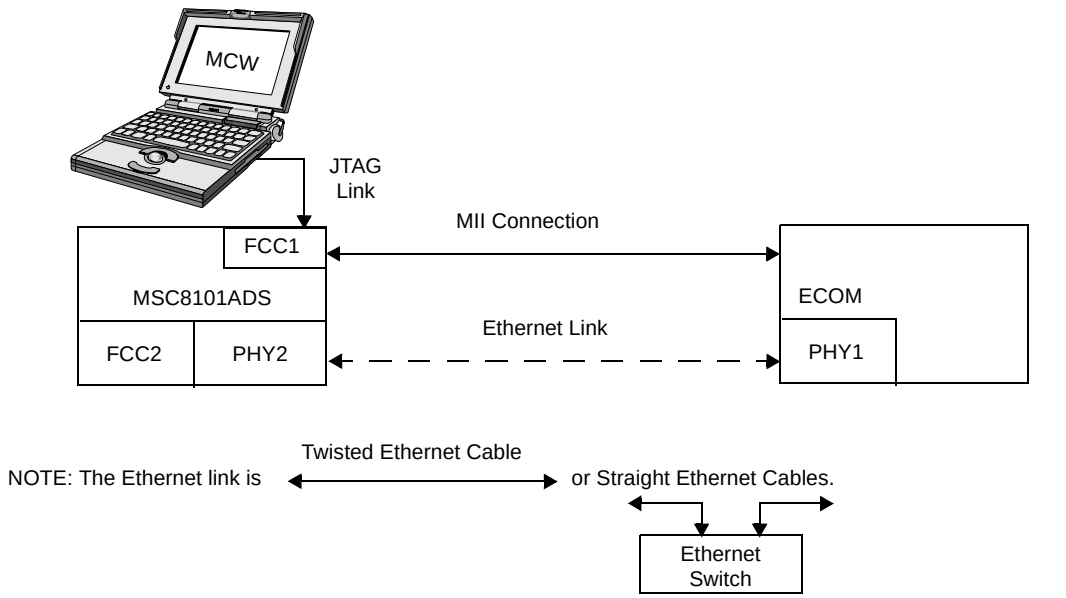

Figure 4. Set-up Using an MSC8101ADS and an ECOM Board

2.2 Software Modules

Table 4. Control and Data Paths

FCC1 on MSC8101ADS			PHY1 (for FCC1) on ECOM		
Signal	**Pin**	**Pin Location (1)**	**Signal**	**Pin**	**Pin Location (1)**
MII TX_ER	PA29	B3	FETHTXER	PA29	B3
MII RX_DV	PA27	B5	FETHRXDV	PA27	B5
MII TX_EN	PA28	B4	FETHTXEN	PA28	B4
MII RX_ER	PA26	B6	FETHRXER	PA26	B6
MII COL	PA31	B1	FETHCOL	PA31	B1
MII CRS	PA30	B2	FETHCRS	PA30	B2
NBL TXD3	PA21	B11	FETHTXD3	PA21	B11
NBL TXD2	PA20	B12	FETHTXD2	PA20	B12
NBL TXD1	PA19	B13	FETHTXD1	PA19	B13
NBL TXD0	PA18	B14	FETHTXD0	PA18	B14
NBL RXD0	PA17	B15	FETHRXD1	PA17	B15
NBL RXD1	PA16	B16	FETHTXD0	PA16	B16
NBL RXD2	PA15	B17	FETHRXD0	PA15	B17
NBL RXD3	PA14	B18	FETHRXD1	PA14	B18
CLK2	PC30	D2	FETHRXD2	PC20	D12
CLK1	PC31	D1	FETHRXD3	PC21	D11
PC13	PC13	D19	FETHTXCK	PC10	D22
PC12	PC12	D20	FETHRXCK	PC09	D23
$\overline{\text{HRESET}}$	$\overline{\text{HRESET}}$	C10 (P1)	RSTBRD	$\overline{\text{HRESET}}$	C10 (P1)
GND	GND	B01–B03 (P1)	GND	GND	B01–B03 (P1)
3.3V	3.3V	A[20–24] (P1)	3.3V	3.3V	A[20–24] (P1)
5.0V	5.0V	A[26–32] (P1)	5.0V	5.0V	A[26–32] (P1)

This section describes the 2FCCs software project, which is available as a zip file. When this file is expanded, all the project files are located in a folder tree, as **Figure 5** shows.

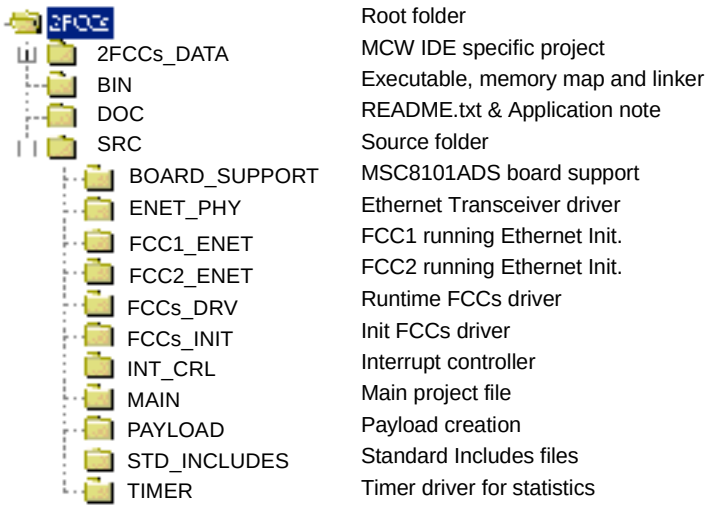

Figure 5. Control and Data Paths

The 2FCCs project was developed using Metrowerks® CodeWarrior® 1.5 for Windows. However, it is compiler independent and therefore runs on any suitable IDE. Moreover, the project is host-independent and can run on a PC or on Solaris, Linux, and so on. When the project is compiled, it should generate neither errors nor warnings. The executable file can be downloaded and run on the MSC8101ADS, where it can enter Debug mode and run step-by-step.

2.2.1 Board Initialization

Before the executable image is downloaded, the MSC8101ADS must be initialized via a command script in the `BIN` directory. This script disables the watchdog timer and configures the memory controller, enabling memory accesses. Then the 2FCCs binary image is downloaded into the MSC8101ADS memory. In the example discussed here, it is downloaded into the MSC8101 internal SRAM. Then the SC140 core executes the bootstrap code (provided by the compiler) until the start of the main function. At this point, the program starts by setting all the parallel ports to act as GPIO inputs, resetting the CPM, and initializing the interrupt controller.

The payloads are dummy Ethernet frames, which contain source and destination MAC addresses, frame length, and random payload. The frame size is determined at compile time, and the FCCs transmit it over the Ethernet link.

2.2.2 Ethernet Transceiver (PHY) Initialization

PHY initialization is performed twice, one for each Ethernet transceiver, as follows:

1. Two CPM GPIO signals are configured to act as MDC and MDIO signals.

   In this set-up the MDC and MDIO signals are common for the two Ethernet transceivers.

2. According to the input parameters, the code configures each FCC for 10 or 100 Mbps speed, half- or full-duplex operation, or auto-negotiation mode.
3. As output, the link status is returned and, if relevant, the results of the auto-negotiation.

2.2.3 FCC Driver Initialization

FCC driver initialization is performed twice, once for each FCC. This process configures the FCC to work in Ethernet mode by programming the following parameters:

1. Configure the external CPM to act as an Ethernet modem.
2. Configure the General FCC Mode Registers (GFMRx), which define all options common to the FCC, regardless of the protocol and select the channel protocol mode.

   In this case, the Ethernet protocol is selected: GFMRx[Mode2] = 1100).

3. Set the FCC clock route.

   Tx and Rx clocks are provided by the Ethernet transceivers and internally routed to the correct FCC.

4. Enable RMON frame counters for statistics and throughput calculation.
5. Specify the events on which the FCCs trigger an interrupt.
6. Connect the FCC interrupt handler to the interrupt controller.
7. Initialize and allocate buffers and buffer descriptor memory.

The code configures the FCC to work in Promiscuous mode, but the mode may need to be changed if the code is reused.

2.2.4 Interrupt controller

The interrupt controller used here is basic, but it performs all the required tasks: context saving, interrupt source determination, interrupt handler address match plus execution, and context restoring. Before the MSC8101 device enters normal run-time mode, the interrupt controller configures the MSC8101 programmable interrupt controller (PIC) and SIU-CPM interrupt controller (SIC) to handle both edge and triggered interrupts. The interrupt controller features a C switch to use either complete or reduced context saving/restoring. The reduced context switching requires fewer cycles, but also requires interrupt handlers to use a reduced set of SC140 registers. You can write the interrupt handlers in assembly language or use compiler options to do so.

As **Figure 4** shows, interrupts are generated by FCCs and then transmitted to the SIC, which manages priorities for serial DMA (SDMA) and CPM peripherals. Then the (PIC) manages the interrupt.[6] At last, the interrupt arrives at the SC140 core.

2.2.5 FCC Driver/Interrupt Handler

The FCC interrupt handler is a short function that first determines the type of the interrupt (received or transmitted frame/buffer, frame discarded, and so on). Its decision logic section depends on the application. In our application, it simply receives and sends frames without any treatment. In other applications, this decision logic might handle buffer management or other tasks. Frame management occurs through the buffer descriptors (BDs), which indicate the buffer to transmit, the received frame, and so on. The interrupt handler acknowledges an interrupt by setting to the appropriate bits to a value of 1. The BD is a memory space to which both the FCC driver and the CPM FCC have access. This memory space can be mapped to a C structure for convenience. A BD contains a set of information on each transmit and receive buffer, such as status (16 bits to set attributes), size (in bytes), and address (pointer to the start of the buffer).

In conjunction with the interrupt controller, there is a C switch to use either complete or reduced context saving/restoring. These two switches must both be set or cleared.

2.2.6 Timer and Statistics

A timer displays the RMON frame counters on the IDE I/O window every 10 seconds. The procedure for initializing this timer is much like that for initializing FCCs. During initialization, the timer registers are configured, and the timer interrupt handler is connected to the interrupt controller. The timer period is crystal-dependent, so a check of the on-board crystal is required. Any changes to the crystal or to MODCLK should be reflected in the appropriate files.

2.3 Software Configuration Options

This section discusses the files containing the global parameters, with listings of the default values, which provide the best throughput.

- `2FCCs.h`. Enable/disable FCC1 or FCC2:

```
#define FCC1_ENABLED 1
#define FCC2_ENABLED 1
#define FCC1_RX_ENABLED 1
#define FCC2_RX_ENABLED 1
#define FCC1_TX_ENABLED 1
#define FCC2_TX_ENABLED 1
```

[6] The PIC also manages the priorities among the rest of the MSC8101 peripherals.

- `Timer.h`. Reflects the value of the on-board crystal:

```
#define DIV16       1
#define BRG_VALUE 306      // refer to the table
```

- `Payload.h`. Sets the frame size:

```
#define FRAME_SIZE_1 64    // must be an even number
#define FRAME_SIZE_2 64    // must be an even number
```

- `FCCs_Int_Handler.h`. Uses the hand-optimized ASM (reduced register set) or C version:

```
#define INTHANDLER ASM     // C
```

- `FCC1_ENET.h` and `FCC2_ENET.h`. Sets or unsets the internal loopback, sets the FCC MAC address, and determines the number and the size of buffers for both the transmit and receive paths:

```
#define FCCx_INTLPBK 0     //1

#define ENETx_PADDR_H      0x00E0  //Physical Addr.1 (MSB)
#define ENETx_PADDR_M      0x0C12  //Physical Addr.1
#define ENETx_PADDR_L      0x3472  //Physical Addr.1 (LSB)

#define FCCx_TX_BUF_SIZE   0x5EE //1518 in decimal
#define FCCx_RX_BUF_SIZE   0x5EE //1518 in decimal

#define FCCx_TX_NUM_BUF    4
#define FCCx_RX_NUM_BUF    4
```

- `IRQ.h`. Specifies normal or reduced context switching:

```
#define OPTIMIZATIONS ON
```

The parts of the 2FCCs project that can remain as is are as follows:

- Most of the initialization for the board
- Most of the initialization for the Ethernet PHY
- Most of the initialization for the FCCs
- The interrupt controller in normal mode

The parts of the 2FCCs project that may need to be changed and adapted are as follows:

- Buffer descriptor initialization
- Buffer management

3 Performance

MSC8101 target markets include media (voice/fax/data) over packet gateways. In such embedded systems, packet traffic is characterized by moderate to high bandwidth consumption. Packets are usually shorter than 250 bytes. On MSC8101 Ethernet links, this traffic has an impact on frequency interrupt, final throughput, and packet latency. This section discusses how to optimize the Ethernet parameter values. Modifications may be needed if the Ethernet network traffic differs from that described here.

3.1 Accesses to MSC8101 Internal SRAM

The SC140 core should use $\overline{\text{CS0}}$ at memory offset 0x0 to access the internal SRAM. Such an access requires one cycle. In contrast, accessing SRAM via $\overline{\text{CS10}}$ at memory offset 0x02000000 on the system bus requires many more cycles. The $\overline{\text{CS0}}$ setting at a memory offset of 0x0 is the default, which can be reconfigured. Read or write operations can consume up to four bus cycles, and assuming an SC140 core/bus speed ratio of 5 (MODCLK 40), such operations may require up to 20 SC140 core cycles

because read or write requests must cross the Q2PPC bridge and the system bus/local bus bridge. This requirement directly affects MSC8101 internal bus loading and global system speed, as well as all memory space to be accessed by both the SC140 core and the CPM, such as buffers and buffer descriptors.

3.2 Buffers

The MSC8101 buffers enable data transfers to/from the CPM and FCC and to/from memory through SDMA transactions that are transparent to the user. Buffers can be located in three types of memory:

- *Internal SRAM*. Provides rapid one-cycle access to the buffer.
- *External SDRAM*. Accesses to the buffer consume more SC140 core cycles but free the internal memory so that the payload fabric can work in temporary space in the internal SRAM while DMA transfers occur to/from external SDRAM. This use of memory space increases the frame latency.
- *Internal DPRAM*. This is the right choice for applications in which the SC140 core does not access the data and directly forward it to a CPM communications channel. Otherwise, it offers no advantages. This choice has an impact on the RSTATE register in the FCC common parameters.

In a complete MSC8101-based embedded system, the choice of memory location for buffers is a trade-off based upon the available internal SRAM and SDRAM space, frame latency, load on the system bus/local bus, and so on. Buffers can be 4-bye aligned or not, depending on memory space allocation. CPM communications channels (SCC, MCC, FCC) access their buffers and BDs via SDMA. As for all DMA data transfers, the alignment on a 4-byte boundary (for 32-bit systems such as the MSC8101) provides the best performance.

For the 2FCC project, these values yield the best balance between SRAM consumption and throughput. More buffers on the receive side allow a greater burst of frame; more buffers on the transmit side allow larger frame burst generation. Of course, allocating fewer buffers for both receiving and transmitting data saves memory space.

To simplify buffer management, allocate one buffer for one frame rather than several buffers for one frame. This practice increases buffer size so that the use of memory space is less flexible. However, depending on the Ethernet protocol, if a partial chunk checksum must be computed (such as for IP and UDP), allocating one buffer per frame results in faster and simpler algorithms because the data is always contiguous.

We set buffer size to 1500 (Ethernet MTU) + 18 (Ethernet encapsulation). When the system is connected to an unfamiliar environment, this is the safest choice. When the system is connected to an entirely defined environment, setting the buffer size to a known maximum frame size saves memory space. If you are allocating several buffers to one frame, you can set the size to a smaller value than the known maximum frame size (for example, 64 bytes). Because the 2FCCs project must handle frame sizes in the range of 64–1500 bytes, it is mandatory that we set the buffer size to 1500 (Ethernet MTU) + 18 (Ethernet encapsulation).

Note: Once the BD that points to a buffer is full or not empty (for transmit buffers) or empty or not fully (for receive buffers), neither the SC140 core nor any MSC8101 peripherals should write or even read the contents of the buffer. Doing so may result in a crash and freeze of the CPM and/or SC140 core.

3.3 Buffer Descriptors

A BD is a set of structures containing information on the buffer size, status, properties, and pointers. BDs provide the means to manage buffers. The CPM/FCC reads and writes to BDs via SDMA transfers that are transparent to the user. BD rings should be located in internal SRAM because they do not consume much memory and they allow flexibility in buffer servicing methods for the FCC interrupt handler. Locating BDs in external SDRAM decreases system performance without offering any advantages. Locating BDs in internal DPRAM may offer the advantage of low interrupt frequency, but accesses to DPRAM increase SC140 core cycle consumption and latency. Also, locating BDs in internal DPRAM has an impact on the RSTATE in the FCC common parameters. As noted in **Section 3.2**, BD rings should be 4-byte aligned.

The transmit buffer descriptor INT bit is set in one of three ways, and the setting has a direct impact on the interrupt frequency and thus on the system global performance:

- Set for each transmit buffer. As soon as a smart buffer management or buffer queuing system is implemented, this choice is a must. The 2FCCs project uses this option in order to count transmitted frames.
- Set for the last buffer of the transmit ring, which is suitable for basic systems.
- Set for none, which simplifies the design of the FCC interrupt handler.

The receive buffer descriptor INT bit is set in one of three ways:

- Set for each receive buffer. As soon as a smart buffer management or buffer queuing system is implemented, this choice is a must.
- Set for the last buffer of the receive ring, which is suitable for basic systems.
- Set for none: polling. In systems in which the main task is intensive and repetitive (such as traffic aggregation), polling received buffers can be advantageous. The interrupt mechanism is then no longer mandatory, but a system scheduler must be implemented.

Note: Once the BD that points to a buffer is is full or not empty (for transmit buffers) or empty or not fully (for receive buffers), neither the SC140 core nor any MSC8101 peripherals should write or even read the contents of the buffer. Doing so may result in a crash and freeze of the CPM and/or SC140 core.

3.4 Interrupt Service Routine

An interrupt service routine (ISR) runs when the SIU-CPM interrupt controller (SIC) generates an interrupt. The ISR determines the resource(s) responsible for the interrupt and launches the appropriate resource interrupt handler. The routine is encapsulated by context saving and restoring functions. Context switching can be either complete or reduced, as follows:

- *Complete*. In normal conditions, complete register context saving/restoring consumes 56 cycles.
- *Reduced*. In the 2FCCs project, the number of cycles was decreased to 26 by saving/restoring registers (r[0–7] and d[0–7]).

The choice between reduced or complete context switching has a direct impact on the Ethernet frame latency and on the real-time aspects of the global system. Context switching is a great source of errors, so care must be taken in designing this module. Other important aspects of ISRs that warrant consideration at design time are reentrance and whether the ISR is edge or level triggered.

The ISR decision logic is implemented through either the SIU Interrupt Vector Register (SIVEC) or through the SIU Interrupt Pending Register (SIPNR_H and SIPNR_L), as follows:

- *SIVEC*. This register contains a 6-bit code representing the unmasked interrupt source of the highest priority level. Use of SIVEC results in interrupt service that is generic and independent of peripherals.
- *SIPNR_H and SIPNR_L*. Each bit in the interrupt pending registers corresponds to an interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. These registers are used in conjunction with their respective mask (SIMR_H and SIMR_L). Use of these registers can slightly improve performance.

On most compilers, the ISR is defined by adding a line in the source code just before the function definition, as follows:

```
"#pragma interrupt Name_of_the_ISR_function"
```

An interrupt handler function differs from other functions in two main ways. Context saving and restoring surrounds the ISR function, and it runs in Exception mode, forcing the compiler to generate instructions that differ slightly from the instructions issued in Normal mode.

3.5 FCC Interrupt Handler

For the 2FFC project, the FCC interrupt handler is a simple BD manager. In a complete system, it would also provide hooks for upper protocol layers (software) and function as the entry point for a payload fabric. The coding language is either assembly or C. The use of assembly language permits a reduction in SC140 core cycles and is required when the ISR is working with reduced context switching. Assembly code is a great source of errors, especially for the ISR, so care must be taken in writing Assembly code. The C language is the default option and is more portable and more easily maintainable.

There are many possible buffer servicing methods for the interrupt handler. Calls to the FCC interrupt handler consume SC140 core cycles. The servicing method directly determines the ISR call frequency. Thus the choice of the servicing method and the FCC interrupt handler design are critical in a system design. Buffers can be serviced one at a time so that each received or transmitted buffer generates an interrupt. This method can result in a non-serviced buffer, which then breaks the BD rings and stops reception or transmission. Such breakages can occur at high throughputs as more than one buffer can be transmitted or received between two FCC Interrupt handler calls.

Alternatively, all buffers can be serviced at once so that each received or transmitted buffer generates an interrupt. This method consumes many SC140 core cycles but ensures that all buffers are serviced so that no BD rings are broken. Yet a third buffer service option is to use a minimum FCCINTHandler in conjunction with dynamic buffer allocation/management. A flag/semaphore and separate handler launched by scheduler/RTOS can also be added to decrease the cycles spent in the handler, thus improving the real-time aspect of the global system.

The goal of the 2FCCs project is to provide a simple example with high Ethernet throughput. Therefore, it implements the method of servicing the buffers one at a time because of its simplicity. In real, more complex systems, using a minimum FCCINTHandler in conjunction with dynamic buffer allocation/management (or a similar schema) would be necessary because of the flexibility. The maximum reachable throughput should be similar, regardless of the buffer servicing method.

Note: For the reasons provided here, the FCC interrupt handler should reduce its access to FCCE and other DPRAM registers/memory areas to a minimum and make use of optimized code.

3.6 Timer and Statistics

In its default configuration, the 2FCC project works with a reduced context switching ISR in conjunction with an ASM FCC interrupt handler. However, the timer interrupt handler is not written in ASM and can potentially use registers that are not yet saved when a timer interrupt occurs. In most systems, this would lead to crashes, but in the 2FCC project, the main program is an empty infinite loop that executes no instructions. Thus, the fact that the timer interrupt handler is not written in ASM has no impact.

4 Results and Conclusions

The direct application of the 2FCCs project is the benchmarking for Voice over IP (VoIP) systems. The array shown in **Table 5** gives the Ethernet frame size for VoIP/G711 communications. The test runs the project for different frame sizes, from 64 to 1500 bytes with several steps including the values shown. The hardware and software assumptions for this test are described in the previous sections of this document.

Table 5. G711 Figures for Frame Size

G711 X (Ms)	2.5	5	10	20
Frequency (Hz)	400	200	100	50
Voice payload (bytes)	20	40	80	160
RTP (bytes)	12	12	12	12
UDP (bytes)	8	8	8	8
IP (bytes)	20	20	20	20
Ethernet MAC @ (bytes)	14	14	14	14
Ethernet CRC (bytes)	4	4	4	4
Total size of a packet (bytes)	78	98	138	218
Throughput for one full duplex (Tx and Rx) channel (kbits/s)	249.6	156.8	110.4	87.2

The validity of the test is limited. In fact, the use of a software tool called CPM_Perf (see **Section 5**), indicates that the CPM RISC processor does not run under normal conditions (saturation) when the frame size is less than approximately 138 bytes. The maximum MSC8101 speed configuration is SC140 core/CPM/system bus = 300/150/100 MHz, and the minimum frame size is approximately 64 bytes. See the calculations for these two scenarios in the results matrix of the 2FCCs test (**Table 7**).

Table 6. 2FFCs Throughput Performances

G711 X (Ms)	X	2.5	5	10	20	X	X	X
Ethernet frame length	64	78	98	138	218	500	1000	1500
FCC1 throughput (Mbits/s)	54.9	65.4	77.8	84	89.3	94.8	96.8	97.2
FCC2 throughput (Mbits/s)	60.5	60.4	78.4	84	89.3	94.8	96.8	97.2
Total throughput (Mbits/s)	115.4	125.8	156.2	168	178.6	189.6	193.6	194.4
Interrupt frequency (1/s)	1803125	1612821	1593878	1217391	819266	379200	193600	129600
Number of channels	X	504	996	1522	2048	X	X	X

The throughput divergence between the FCCs is a confirmation of the expected limit (138 bytes) given by the CPM_Perf tool. At full speed, this limitation disappears. Sometimes the debugger I/O window displayed the message `discarded frames = 2` because of BD resynchronization after the timer

interrupt, itself due to the basic buffer management. As expected, the interrupt frequency increases when frame size decreases. This demonstrates the need for care in both the interrupt controller and FCC interrupt handler design.

The results obtained show the high performance of the MSC8101 CPM. They prove that the MSC8101 is suitable for most applications requiring high throughput. There are technical ways to increase the Ethernet throughput. For example, several IP packets can be concatenated into one Ethernet frame, with additional software mechanisms to control the packet latency. Another way would be to implement CPM microcode for IP and, even upper layers, encapsulation/desencapsulation in the CPM RISC processor.

5 CPM Performance Tool

This section describes how to test PowerQUICC II product-based systems such as the MSC8101and MPC8260. In four steps, it is possible to know the loads on the CPM RISC processor and the two internal buses (local and 60x-compatible system bus):

1. Launch the CPM_Perf application, read the first screen, and then acknowledge.

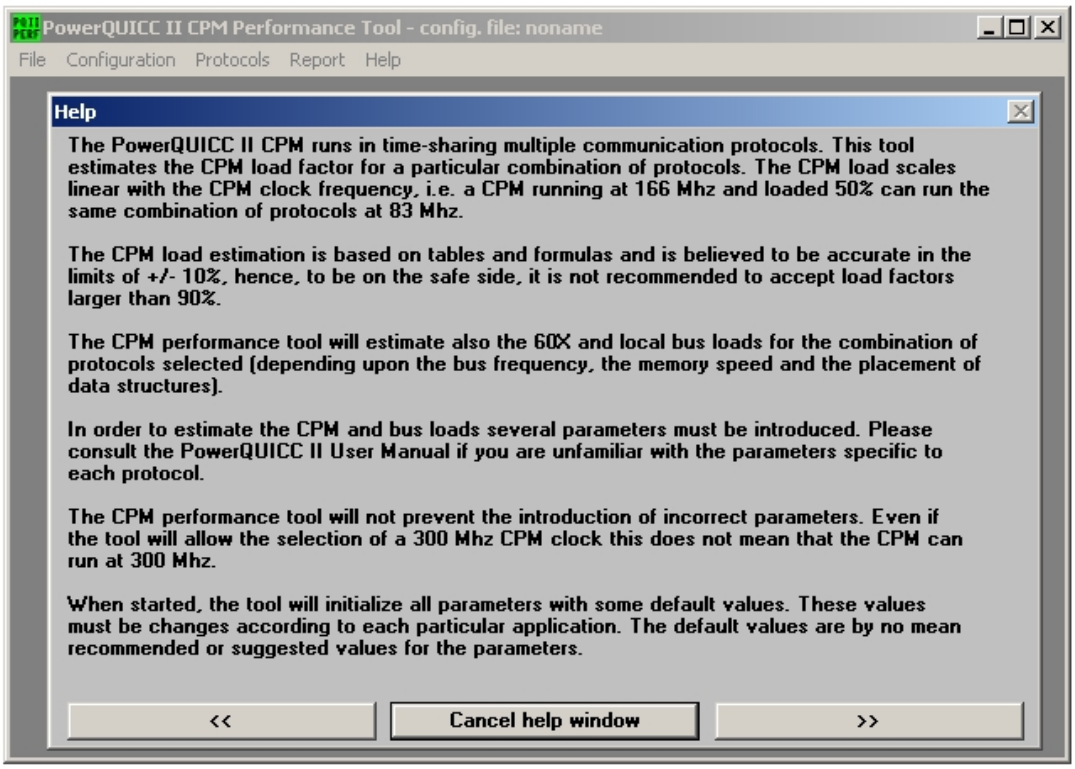

Figure 6. CPM Performance Tool, Opening Screen

2. On the main screen, set CPM speed, bus speed, and the access scheme. Then, specify the location of the buffer and BDs on the local bus (SRAM) or not (SDRAM, DPRAM).

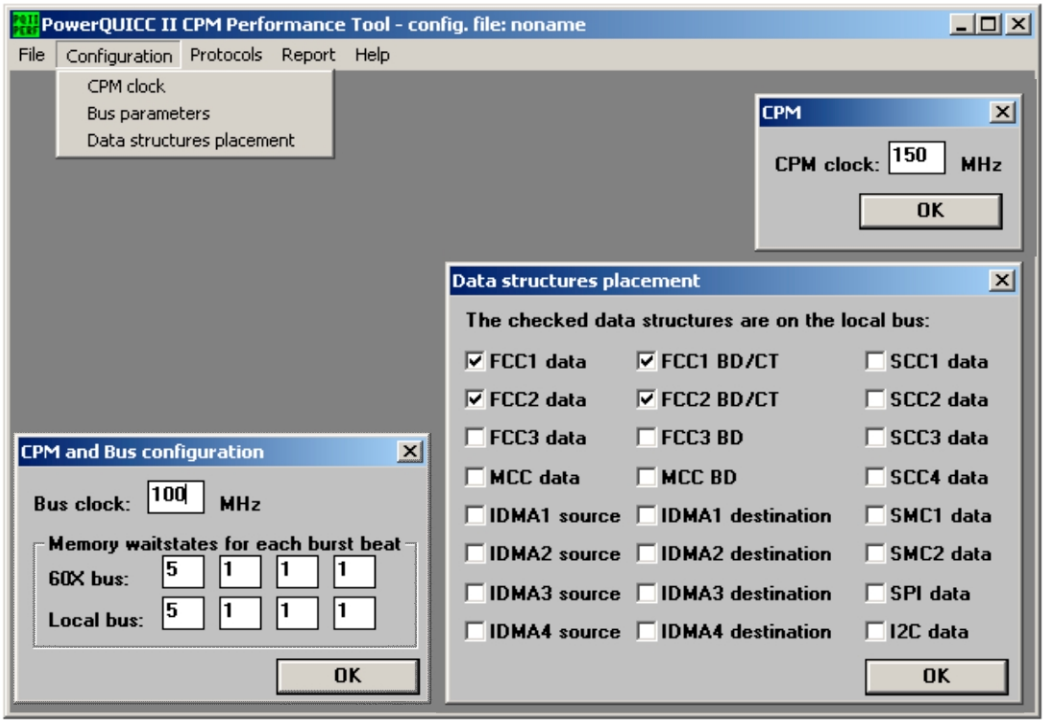

Figure 7. CPM Performance Tool, Screen 2

3. Set the speed, buffer size and other parameters for both FCC1 and FCC2.

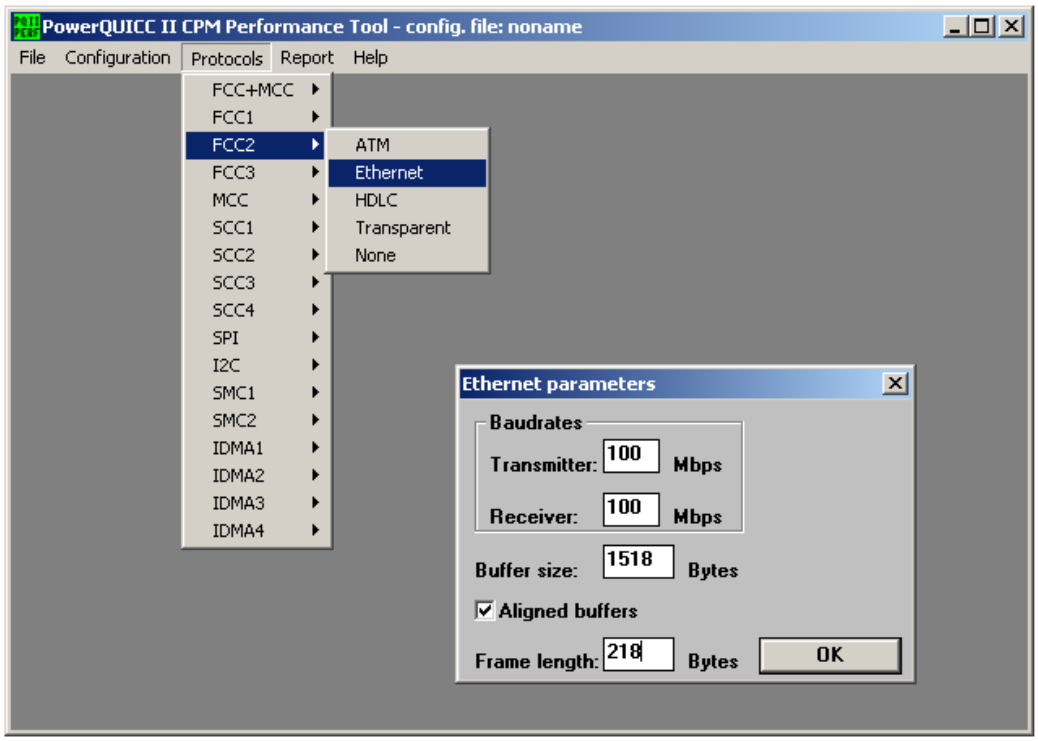

Figure 8. CPM Performance Tool, Screen 3

4. Directly watch the results or save them into a text file.

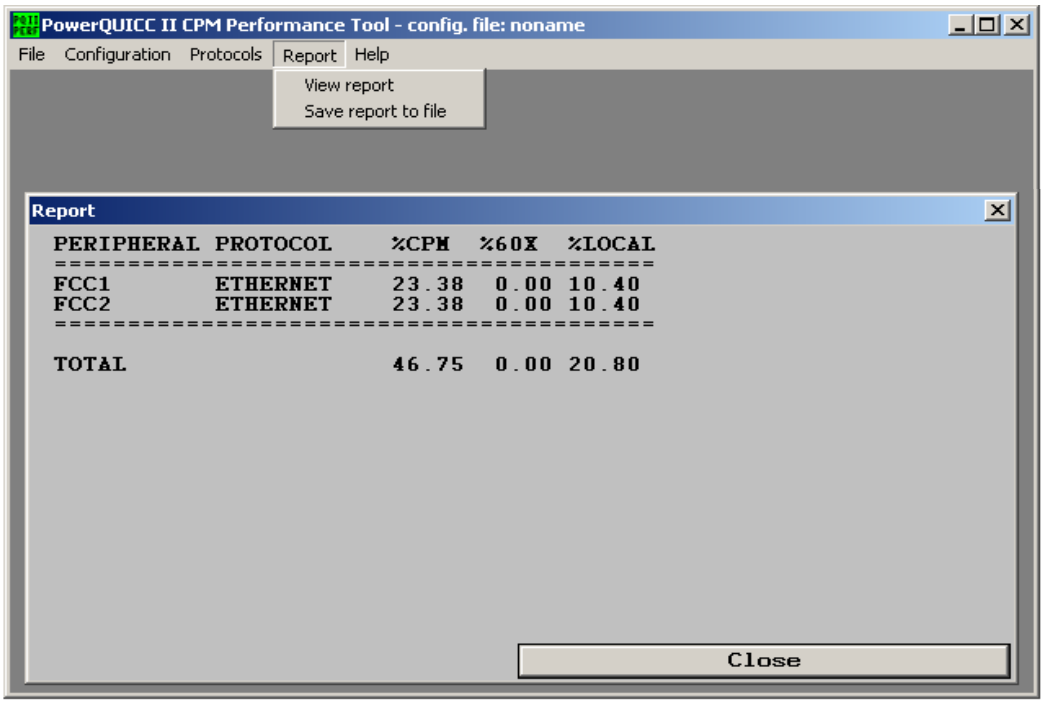

Figure 9. CPM Performance Tool, Screen 4

The assumptions underlying the 2FCCs project settings are as follows:

- CPM bus access schemes:
  - System bus memory wait states = 5, 1, 1, 1
  - Local bus memory wait states = 5, 1, 1, 1
- CPM FCC configuration:
  - Data is placed on the local bus (internal SRAM)
  - Buffer descriptors are placed on the local bus (internal SRAM)
  - Receiver baud rate = 100 Mbps (Fast Ethernet)
  - Transmitter baud rate = 100 Mbps (Fast Ethernet)
  - Buffer length = 1500 bytes (no frame fragmentation)
  - Unaligned buffers = YES (optimum)

Table 7 shows the results of simulations for two scenarios.

Table 7. CPM Performance Tool Results

Speed Configuration	CPM/Buses Clock = 98/39 MHz		CPM/Buses Clock = 150/100 MHz	
Frame Size (Bytes)	CPM Busy (%)	FCC Busy (%)	CPM Busy (%)	FCC Busy (%)
64	150.22	75.11	98.15	49.07
98	122.26	61.13	79.87	39.94
138	96.10	48.05	62.79	31.39
218	71.56	35.78	46.75	23.38

Table 7. CPM Performance Tool Results (Continued)

Speed Configuration	CPM/Buses Clock = 98/39 MHz		CPM/Buses Clock = 150/100 MHz	
Frame Size (Bytes)	CPM Busy (%)	FCC Busy (%)	CPM Busy (%)	FCC Busy (%)
500	50.82	25.41	33.20	16.60
1000	42.62	21.31	27.84	13.92
1500	39.14	19.57	25.57	12.79

6 Related Reading

[1] *MSC8101 Reference Manual*, MSC8101RM/D, available at the website shown on the back cover of this document.

[2] *MSC8101ADS User's Manual*, available at the website shown on the back cover of this document.

[3] Ethernet quick reference, inspired from the source available at http://www.whatis.com

[4] Intel LXT970 data sheet (PDF document available at http://developer.intel.com).

[5] Metrowerks Enterprise compiler documentation available with Metrowerks CodeWarrior® for StarCore®.

NOTES:

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

AN2333/D