MARVELL®

Link Street® Integrated Micro Processor (IMP)

Functional Specification

User Guide

Doc. No. MV-S110588-00 Rev. –
June 8, 2015,
CONFIDENTIAL
Document Classification: Proprietary Information

Marvell. Moving Forward Faster

Document Conventions

	Note: Provides related information or information of special importance.
Caution	**Caution:** Indicates potential damage to hardware or software, or loss of data.
	Warning: Indicates a risk of personal injury.

Document Status

	Technical Publication: 1.00

For more information, visit our website at: http://www.marvell.com

Preface

About this Document

The *Link Street Integrated Micro Processor Functional Specification* provides a description of the IMP subsystem that is contained in Link Street® Ethernet devices, typically Ethernet switches. This document covers the CPU system, how to boot it, and each of the peripheral devices attached to it, including the related I/O registers for these peripherals. It also covers the IMP's hardware debugging and communications interface. This document does not cover the CPU core itself, as the IMP uses a Z80 compatible CPU. Documentation of the Z80's capabilities and instruction set are available elsewhere.

This document is for anyone needing to write and debug code that runs on the Link Street IMP CPU.

While this is a stand-alone document for the Link Street IMP, the IMP is a subsystem of a larger device. Refer to that device's datasheet for information on how the entire device works. This document only covers the I/O devices directly accessible by the IMP's CPU as well as how the IMP can access the larger devices registers in a general way.

Table of Contents

List of Tables

List of Figures

1 Introduction

This document provides the technical information about the individual component blocks of the IMP (Integrated Micro Processor) subsystem. The IMP subsystem has the following features:

- 200 MHz Z80 compatible 8-bit CPU
- 60K Bytes of zero wait state on-chip static RAM with parity
- Fast Boot capability from external EEPROM with optional verification
  - Supports 32K bit (24C32 - 4K byte) to 512K bit (24C512 - 64K byte) EEPROM devices
  - More than one EEPROM can be supported if more storage is needed
  - 8K bytes can be transferred into RAM in under 75 mSec when using 1000 KHz EEPROMs
- Slow Boot option using the device’s SMI (System Management Interface) and/or the device’s RMU (Remote Management Unit)
  - Supports booting the IMP from an external CPU without the need of a local EEPROM
- Direct access to all the device’s switch registers
  - Supports simultaneous access to switch registers with an optional external CPU
- Ethernet NIC (Network Interface Controller) connected to a dedicated switch port
  - The switch core provides all buffering, filtering, policing and QoS functions for the IMP’s NIC
- Dedicated local peripherals of the IMP include:
  - Two Timers and a Watch Dog controller
    - Each Timer supports 100 uSec steps with a range of 100 uSec to 6.5 Sec
    - The Watch Dog resets and reboots the IMP - once enabled it cannot be turned off
    - The Watch Dog can trip as early as 1 mSec or as late as 256 Sec
  - Dual-Level 8 source Interrupt controller (with 7 interrupts allocated)
    - The CPU’s NMI can be a higher level (level 2) interrupt that can be masked
    - Three interrupt vector methods are supported by the interrupt controller
    - Simultaneous Level 1 interrupts can be prioritized by the interrupt controller
  - UART-like Communications Port to optional external CPU using SMI registers
  - On-Chip RAM Parity Checker with optional re-boot on a bad parity read
  - EEPROM Speed selections
    - 200 KHz to 1500 KHz EEPROM speeds supported
  - Fast Copy DMA Controller - EEPROM to RAM and RAM to RAM
    - The EEPROM is never part of the CPU’s memory space, but portions of it can be read and copied into the CPU’s RAM using this Fast Copy DMA controller
    - More than 64K bytes of EEPROM can be supported by bank selecting which EEPROM device to read
    - EEPROM Fast Copy with Verification option – to ensure the data was read in correctly
- JTAG-like hardware debug capabilities via SMI (System Management Interface) including:
  - Stop/Run/Reset the CPU
  - Single step or step ‘n’ instructions
  - 4 hardware break points – by range or direct match, memory or I/O
  - Dump all of the CPU’s internal registers
  - Dump/modify all of the CPU’s I/O registers
  - Dump/modify all of the CPU’s RAM
- Marvell’s IMPGUI debug tool supports the GPL SDCC C compiler & assembler

2 CPU, Connections & Conventions

The Integrated Micro Processor (IMP) subsystem (Figure 1) is a block inside of a larger Ethernet switch device (the IMP is the block in the lower right hand corner of Figure 2). The IMP block has the following connections:

- Access to one or more external EEPROMs for booting (section 4.1) and subsequent EEPROM accesses (section 6.5).
- Access to all the registers of the Ethernet switch core (section 7).
- Receives and transmits Ethernet frames via a connection between a port on the switch core and the IMP's NIC (Network Interface Controller – section 6.4).
- Hardware UART-like communications (section 6.1.3) and JTAG-like debug support (section 8) via the switch's registers which connect to the outside world using the device's SMI interface (System Management Interface).

Figure 1: Integrated Micro Processor (IMP) Subsystem Block Diagram

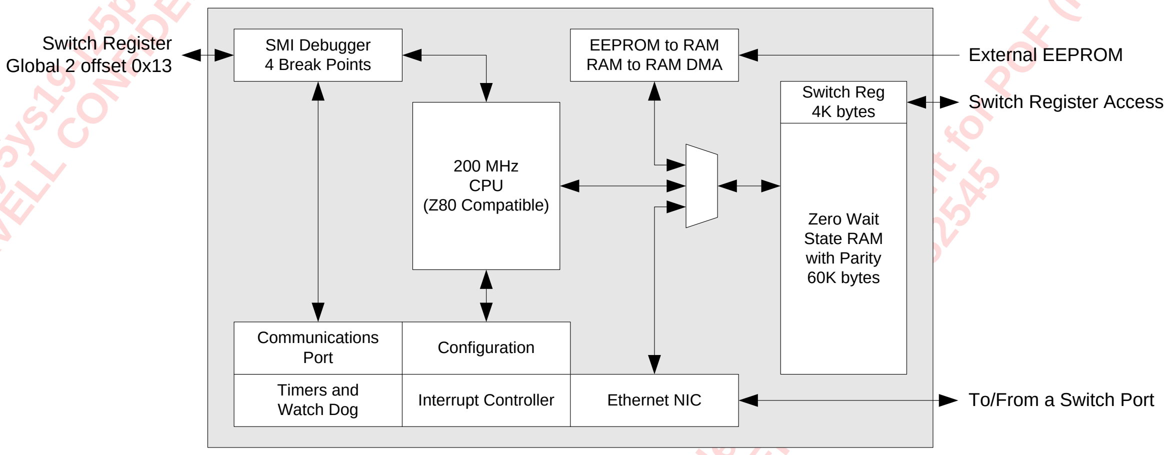
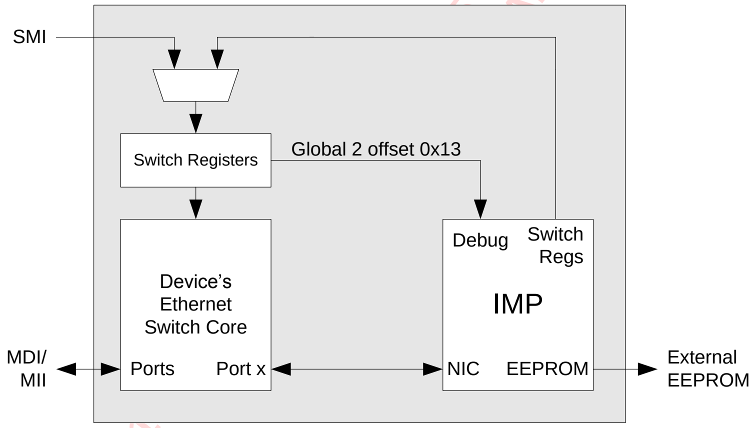


Figure 2: Simplified Ethernet Switch Device Top Level Block Diagram



These connections entail naming conventions used in this document:

- "RESETn" refers to the device's RESETn pin. This reset is a total chip power-on type of reset.
- "Reset" refers to the IMP's reset. Resetting the IMP does not reset the switch core (the switch core has its own levels of reset, one of which is controlled by the IMP (section Table 4).
- The Switch Registers are divided into many functional blocks (see the device's Functional Specification). These are referred to using a syntax of: <block name> offset 0xZZ index 0xYY where:
  - <block name> could be "Port" or "Global 1", etc., and "ZZ" is the address of a register in that <block name>.
  - If the register is indexed then "index 0xYY" indicates the index address 0xYY.
  - An example reference is: Global 2 offset 0x13 index 0x03.
- "IMP" refers to all the components shown in Figure 1.
- "CPU" refers to only the CPU block shown in Figure 1.
- CPU RAM and EEPROM memory addresses are referred to by "addr 0xYYYY" where YYYY is the 16-bit hexadecimal address of the memory.
- CPU Input & Output registers that are only in the IMP's I/O space are referred to as "I/O 0xZZ" where ZZ is the 8-bit hexadecimal address of the I/O.

2.1 CPU Differences

The CPU inside the IMP is a 200 MHz Z80 compatible CPU. There is one difference, however:

- The R, or Refresh, register in the IMP CPU is always zero.

The R register is designed to be used to help refresh DRAMs. Since the RAM inside the IMP is SRAM, the R register is not needed. Writing to the R register has no effect and reading the R register returns zeros.

3 Memory & I/O Maps

The IMP's Z80 compatible CPU supports a 64K byte RAM memory space for code and memory mapped I/O devices (using 16-bit addresses) along with a 256 byte I/O space for peripherals (using 8-bit addresses).

The IMP's implementation allocates the top 4K bytes of the CPU's 64K byte memory space to access all of the registers of the device's switch core (using addresses 0xF000 to 0xFFFF - see Figure 3). This allows the fastest access possible to registers that are frequently used. Switch register access is covered in section 7.

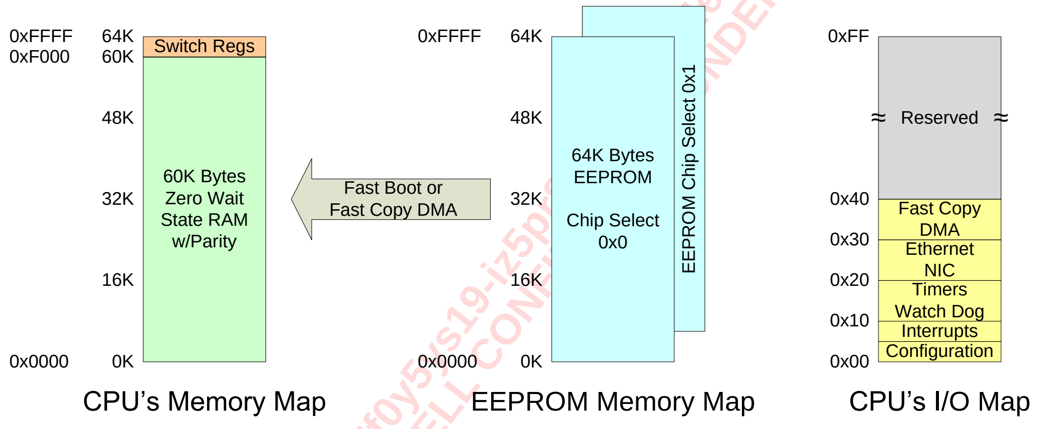
The remaining 60K bytes of the 64K byte memory space is allocated to zero-wait-state, parity protected, SRAM. Zero-wait-state RAM allows the CPU to execute all code and to access all the code's variables as if they were in cache memory, i.e., all reads and writes occur without any added clocking delays. Parity protected RAM ensures that proper code execution occurs 24/7/365 as the IMP can be configured to re-boot whenever a parity error is detected (I/O 0x03).

The CPU's RAM can be loaded from EEPROM (see Figure 3) using Fast Boot (section 4.1) or by using Fast Copy DMA after booting (section 4.1.3). An EEPROM-less Slow Boot mode is also supported (section 4.2). More than one EEPROM device can be supported in the system if more storage is needed by using the methods discussed in section 6.5.4.

The IMP uses the CPU's I/O space to access the IMP's peripherals (see Figure 3). These peripherals are grouped into the following functional blocks:

- Configuration (I/O 0x00 to 0x06) – Section 6.1
- Interrupt Controller (I/O 0x07 to 0x0F) – Section 6.2
- Dual Timers & Watchdog (I/O 0x10 to 0x1F) – Section 6.3
- Ethernet NIC (I/O 0x20 to 0x2F) – Section 6.4
- Fast Copy DMA (I/O 0x30 to 0x3F) – EEPROM to RAM and RAM to RAM – Section 6.5

Figure 3: CPU's Memory & I/O Maps with the EEPROM's Memory Map



4 Booting the IMP

There are two ways to boot the IMP, Fast Boot and Slow Boot.

1. Fast Boot reads the device's external EEPROM into RAM by block reading the data from the EEPROM. This is the recommended way to boot the IMP as 8K bytes of code can be copied in less than 75 mSec (depending upon the speed of the external EEPROM).
2. Slow Boot is usable when there is another (external) processor connected to this device. This method saves the BOM cost of the EEPROM as it allows booting the IMP even when there is no EEPROM connected to this device.

After RESETn the external EEPROM is attempted to be read. The following actions are possible based on the results of the EEPROM read:

- If the 1st byte of the EEPROM is 0xED the Fast Boot process is started – see section 4.1.
- If there is no EEPROM present, or if the EEPROM's 1st byte is 0xFF the IMP CPU and its memory & peripherals are turned off. From this lowest power state the IMP can be Slow Booted – see section 4.2.
- If the 1st byte of the EEPROM is 0x76 (followed by 0x00) the IMP CPU and its memory & peripherals are turned off. In this mode the EEPROM is processed using the switch's Register Loader and the IMP cannot be Slow Booted. See the Switch Functional Specification for more details on its Register Loader.

Note

When using a CPU (internal and/or external) to manage the switch, the device needs to be configured into the CPU Attached mode using the NO_CPU CONFIG pin. This ensures that packets will not flow through the switch until the CPU is ready, as each port resets to the Disabled Port State (Port offset 0x04) and the PHYs are powered down. After the internal CPU boots and configures the switch for its intended application, the CPU can power up the PHYs and change each port's Port State to Forwarding or start an Spanning Tree routine which will use the Port State bits for their intended purpose.

If the NO_CPU CONFIG pin is not set correctly, the switch ports will reset to the Forwarding Port State (and the PHYs are powered up) allowing packets to flow between all ports of the switch until the CPU boots and changes this behavior. This 'packet leaking before boot' can cause unintended behavior in the network.

Note

The CPU's RAM is initialized to zero with proper parity after each device RESETn. This is done in a way such that the boot process is not delayed.

4.1 Fast Boot

Fast Boot is initiated as soon as possible after RESETn whenever the 1st byte of the EEPROM (at EEPROM addr 0x0000) contains a value of 0xED. In this mode the next 10 bytes of the EEPROM define the parameters of the Fast Boot operation. Thus the 1st 11 bytes of the EEPROM are defined as follows:

- 0xED – Indicates the Fast Boot function is to be performed
- 0x<Speed> – The value written to the CPU's EEPROM Speed register indicating the EEPROM's speed & the Switch cores reset state (see EEPROM Speed register description – I/O 0x01)
- 0x<low byte count> – Low 8-bits of the Fast Boot transfer size
- 0x<high byte count> – High 8-bits of the Fast Boot transfer size
- 0x<low dest addr> – Low 8-bits of the Fast Boot transfer destination address (to the RAM)
- 0x<high dest addr> – High 8-bits of the Fast Boot transfer destination address (to the RAM[1])
- 0x<low src addr> – Low 8-bits of the Fast Boot transfer source address (from the EEPROM)
- 0x<high src addr> – High 8-bits of the Fast Boot transfer source address (from the EEPROM)
- 0xC3 – CPU's instruction for an unconditional jump instruction
- 0x<low code addr> – Low 8-bits of the address where the CPU is to start executing code
- 0x<high code addr> – High 8-bits of the address where the CPU is to start executing code

Note

For the fastest possible boot, the 2nd byte in the EEPROM needs to be set to the value that indicates the actual speed of the EEPROM. Care must be used when setting this EEPROM Speed register's value. It must not be set faster than what the EEPROM can be clocked at. The other bits in the Speed register must be set carefully as well. See section Table 4 for a description of the EEPROM Speed register.

Note

To get the fastest possible boot speeds the switch's LEDs are disabled. If the application requires the switch LEDs to work, the loaded code must enabled them by clearing the Skip Columns bits (Port 0 offset 0x16 index 0x00).

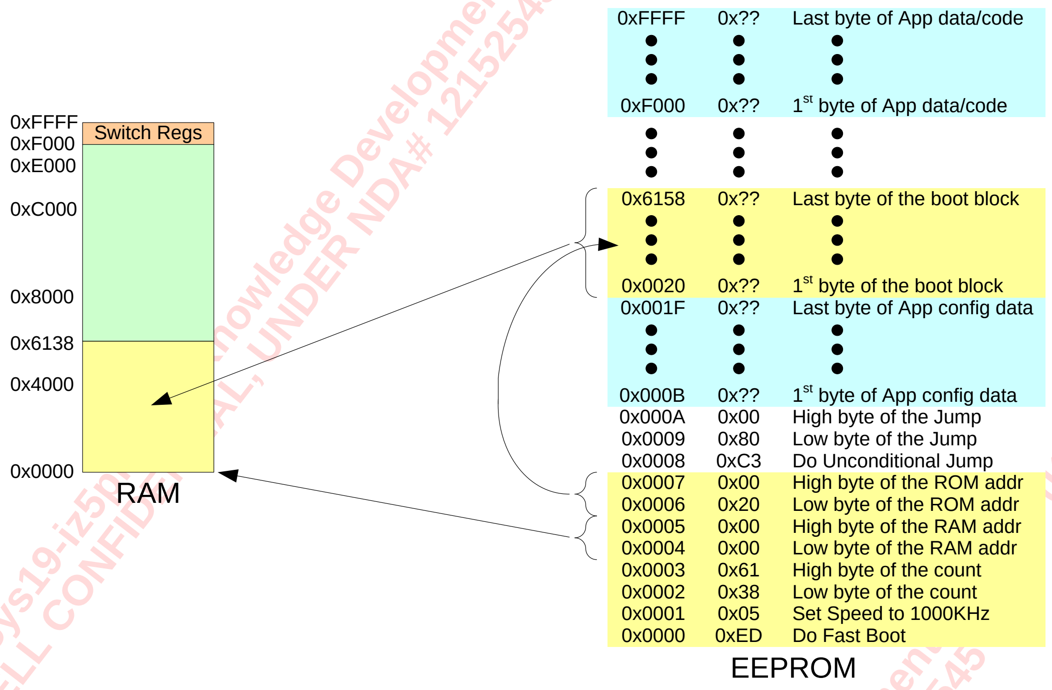
The EEPROM is always based at address 0x0000 as the source address for the Fast Boot operation (i.e., the EEPROM's physical address 0x0000 is always accessed at <src addr> 0x0000). See Figure 3.

The destination address for the Fast Boot operation can be anywhere where RAM exists (i.e., in the lower 60K bytes of address space). For example, a destination address of 0x0100 (0x00 low dest addr & 0x01 high dest addr) is a valid destination address. A typical dest addr may be also be 0x0000, however – see section 5.

An example Fast Boot EEPROM is shown in Figure 4.

[1] The destination of all Fast Boot transfers must be in the lower 60K bytes of the IMP's address space, as writing to the switch registers this way is not supported.

Figure 4: Fast Boot Example


0xFFFF 0x?? Last byte of App data/code
0xF000 0x?? 1st byte of App data/code
0x6158 0x?? Last byte of the boot block
0x0020 0x?? 1st byte of the boot block
0x001F 0x?? Last byte of App config data
0x000B 0x?? 1st byte of App config data
0x000A 0x00 High byte of the Jump
0x0009 0x80 Low byte of the Jump
0x0008 0xC3 Do Unconditional Jump
0x0007 0x00 High byte of the ROM addr
0x0006 0x20 Low byte of the ROM addr
0x0005 0x00 High byte of the RAM addr
0x0004 0x00 Low byte of the RAM addr
0x0003 0x61 High byte of the count
0x0002 0x38 Low byte of the count
0x0001 0x05 Set Speed to 1000KHz
0x0000 0xED Do Fast Boot
EEPROM
0xFFFF
0xF000
0xE000
Switch Regs
0xC000
0x8000
0x6138
0x4000
0x0000
RAM

In the Fast Boot EEPROM example shown in Figure 4, 24,889 bytes (0x6139) of code/data are copied from the EEPROM into the IMP's RAM starting at RAM address 0x0000. This code/data is sourced starting from EEPROM address 0x0020. The speed of the EEPROM is set to 1000 KHz prior to performing this data transfer. After the transfer completes, the CPU starts executing this code starting at RAM address 0x0080.

Note

It is recommended that the CPU start executing code at RAM address 0x0000 as this is where the CPU starts executing whenever it is Reset. The example in Figure 4 used a different start address so the locations of the High & Low addr bytes would be evident.

This example shows a few other items of interest.

- First, the code/data image in the EEPROM does not have to start at EEPROM addr 0x000B. It can be set higher. In this example it is set at EEPROM addr 0x0020 allowing for 21 bytes of Application Configuration Data. This area of the EEPROM is special as it is always at this fixed address regardless of the boot image's size and regardless of the EEPROM's size. This EEPROM area can be used for unit specific information like the device's MAC address and/or serial number, etc. This is discussed more in section 5.
- Second, the 64K byte EEPROM is larger than the 60K bytes of available RAM. This extra EEPROM space is still accessible using the EEPROM to RAM Fast Copy DMA function (section 6.5). Code or data that is not speed critical can be stored in the upper parts of EEPROM and brought into the RAM when needed. More than one EEPROM device can be supported in the system if more storage is needed by using the methods discussed in section 6.5.4. The Fast Boot process copies data from the EEPROM that is selected after RESETn.

4.1.1 Fast Boot Performance

The performance of Fast Boot is optimized to be as fast as possible as block reading of the EEPROM is done. This mode is intended for applications where the boot time is important. Table 1 and Table 2 list the approximate times required to transfer the EEPROM's data in this mode.

Table 1: Generic Fast Boot Times

EEPROM Speed (I/O 0x01)	Overhead for the Transfer	Time to Read each Additional Byte
200 KHz	2790 uSec	45 uSec
400 KHz	1894 uSec	22.5 uSec
500 KHz	1712 uSec	18 uSec
600 KHz	1670 uSec	15 uSec
800 KHz	1670 uSec	11.25 uSec
1000 KHz	1670 uSec	9 uSec
1200 KHz	1670 uSec	7.5 uSec
1500 KHz	1670 uSec	6 uSec

Table 2: Fast Boot Times for 8100 Bytes

EEPROM Speed (IMP I/O 0x01)	Time to Transfer 8100 Bytes
200 KHz	367,290 uSec
400 KHz	184,144 uSec
500 KHz	147,512 uSec
600 KHz	123,170 uSec
800 KHz	92,795 uSec
1000 KHz	74,570 uSec
1200 KHz	62,420 uSec
1500 KHz	50,270 uSec

Note

The EEPROM shares the same device pins as the LEDs. To get the Fast Boot times listed in Table 1 and Table 2, the LEDs are disabled off right after RESETn. If the application requires that the switch LEDs work, the loaded code must enabled them by clearing the Skip Columns bits (Port 0 offset 0x16 index 0x00).

4.1.2 Fast Boot with Verification

The code & data read from the EEPROM during a Fast Boot process can be verified to ensure the data transferred to the IMP's RAM is what was expected. This verification process is enabled when bit 4 in the 2nd byte of the EEPROM is set to a one. This bit enables the ReBootIfFastBootBad function described in I/O 0x01 (section Table 4). When this bit is set to a one, the Fast Boot block of data that is read and transferred to RAM must have a good CRC on it. The CRC used in Ethernet frames is expected. If a good CRC on the block is detected, control is transferred to the IMP CPU at the address defined by bytes 0x0009 & 0x000A of the EEPROM. If a bad CRC is detected, the Fast Boot process is restarted. But this time the EEPROM Speed bits are left at their default value (of 200 KHz) regardless of what the 2nd byte in the EEPROM is. If a bad CRC persist, the Fast Boot re-try process also persists.

The purpose of Fast Boot with Verification is:

- To ensure data read from the EEPROM is what was expected.
- If reading the EEPROM at faster speeds is failing due to temperature or other issues, the slower speed may result in a successful boot. This can be detected by examination of the EEPROM Speed register (I/O 0x01). Its value will be the default (this could be the normal value if the 2nd byte of the EEPROM does not select a faster EEPROM boot speed, however).
- Indicates the code was changed without updating the CRC (see the Code Update Support section 4.1.3.1).

4.1.3 Boot Loader

A boot loader or a boot strap, brings in a small portion of initial boot code and then this code completes the boot process. The reasons to do are many, with some listed below.

4.1.3.1 Code Update Support

Applications may wish to support the updating of the IMP's code in the field. Using a boot loader can help:

- Use the Fast Boot function (section 4.1) to bring into RAM only the small, boot loader code (Fast Boot with Verification, section 4.1.2, is optional here, but recommended). This small block of code is expected to be simple enough that it NEVER needs to be updated in the field – thus it always boots so that it can perform its job.
- This boot loader then uses the Fast Copy EEPROM to RAM function to bring in the full boot image in one or many segments, each of which as a CRC on it. The Fast Copy EEPROM to RAM performs the same function as Fast Boot except it is control by registers that are under control of the IMP CPU instead using of a table of parameters stored at the beginning of the EEPROM. The block transfer CRC check is also supported (section 6.5.2) but uses a different indicator.
- At the end of each Fast Copy EEPROM to RAM operation, the boot loader checks the GoodFcCRC bit (I/O 0x03). If the CRC is bad, it can be assumed that block of code was not field updated correctly (the update may have been interrupted). In this case the older, back-up image of that block of code (still in the EEPROM) is loaded by the boot loader instead. An indication can then be given by the boot loader to try the code update again.

There is a lot more to the procedure of supporting code updates in the field and it is beyond the scope of this document to fully define this procedure (for example, how to write to the EEPROM in-system, using the Global 2 offset 0x14 & 0x15 registers). The goal here is to show the usefulness of the CRC checker on Fast Copy operations for this application.

4.1.3.2 Quick Static Switch Configuration

Some applications require the switch to be configured in a minimal operational mode as soon as possible after power up. This is supported by booting the device in stages called boot strapping. The 1st stage brings in the minimal amount of code necessary needed to configure the switch along with enough extra code to complete the boot process. The 2nd stage brings in the rest of the boot image and transfers control to it.

Boot strapping is accomplished as follows:

- Use the Fast Boot function (section 4.1) to bring into RAM only the small, critical code, necessary to configure the switch for the target application and then complete the boot process.
- Execute this minimal code to get the switch configured as fast as possible.
- At the end of this critical code, add code to complete the boot process using the Fast Copy EEPROM to RAM function (section 6.5). The Fast Copy EEPROM to RAM function performs the same function as Fast Boot except it is control by registers that are under control of the IMP CPU instead using of a table of parameters stored at the beginning of the EEPROM.
- Execute the now fully loaded application code.

Boot strapping can use the optional Fast Boot with Verification (section 4.1.2) and/or the optional Fast Copy with Verification (section 6.5.2).

4.2 Slow Boot

When there is no EEPROM connected to the device, the IMP CPU is set into its deep sleep mode, saving as much device power as possible (Global 2 offset 0x13 index 0x00 bit 7 will be set to a one).

In this mode the IMP CPU can be Slow Booted using SMI (System Management Interface – the MDC/MDIO pins on the device) or by using the RMU (Remote Management Unit – using Ethernet frames). See the datasheet for this device for more information on using the SMI and RMU interfaces access the switch registers. Regardless of the switch register access method used, the procedure to wake up a deep sleeping IMP CPU and load code that it can execute is as follows (the IMPOp register is at Global 2 offset 0x13 index 0x08):

- Issue a Stop IMPOp – this sets the IMP Debug logic into the Stop state
- Issue a Reset IMPOp – this resets the IMP CPU, re-starts its clocks, but since the IMP Debug logic is in the Stop state, the IMP CPU will not try to re-boot from the EEPROM
- Load the desired code to the IMP RAM using the Examine, Deposit & Deposit Next IMPOps
  - Use Examine to point to the address in RAM to modify
  - Use Deposit to write to the RAM address that is currently being pointed to
  - Use Deposit Next to write to the RAM address that is currently being pointed to and then point to the next higher RAM address
- Execute the just entered code using the Examine & Run IMPOps
  - Use Examine to point to the address in RAM where the CPU is to start executing
  - Use Run to get the CPU to start executing at that address

Note

This is not a fast way to boot, but it does allow use of the IMP CPU in systems that don't contain an EEPROM. If this method is used, it is best to load as small a piece of code as possible that supports booting the rest of the code using the IMP CPU's Ethernet NIC interface (6.4).

5 Recommended RAM/EEPROM Usage

The CPU used in the IMP works in certain ways. This functionality dictates how the IMP's RAM should be used. The rest of the IMP itself works in certain ways. This functionality dictates how the EERPOM should be used. The following are recommendations only, as many other use cases are possible.

5.1 Lower RAM Usage

The 1st 128 bytes of RAM are special for the IMP's CPU:

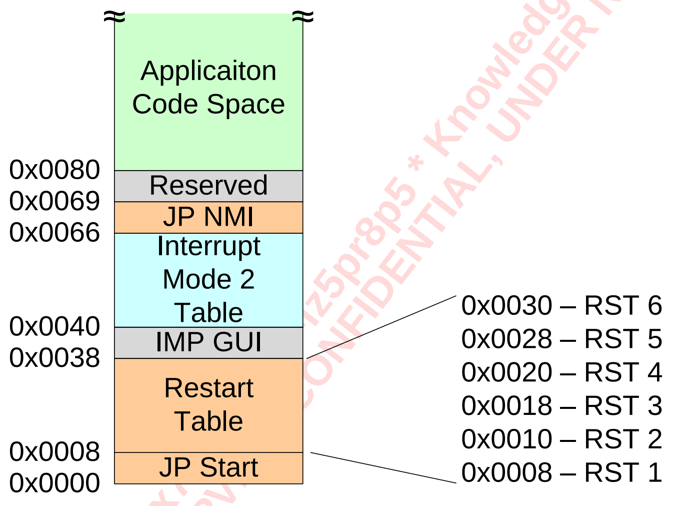
- After a Reset, the CPU starts executing code at addr 0x0000. An unconditional Jump (JP) to the start of the code should be placed at addr 0x0000. This jump recommended to be to addr 0x0080 which is the lowest possible memory address for the application code space.
- The CPU supports eight single-byte RST (restart) instructions that execute a Call to code at addr 0x0000 to 0x0038.
- The NMI (non-maskable interrupt) executes a Call to code at addr 0x0066.

The Marvell IMP GUI (the IMP debugging tools) makes some use of the 1st 128 bytes of RAM too:

- The RST 7 instruction is used to insert breakpoints for C code debugging. The debugger uses RAM from addr 0x0038 to 0x003F for this purpose.
- The debugger uses some RAM for other functions. RAM from addr 0x0070 to 0x007F are reserved for the debugger.

The result of this is the recommended usage of the 1st page in RAM as shown in Figure 5.

Figure 5: Recommended Lower RAM Usage



The color used in Figure 5 means:

- Orange – Can't be moved. If these features are used then these addresses must be used.
- Blue – User usable with some caution – good area for fixed sized variables &/or the Mode 2 Interrupt table (section 6.2).
- Green – Totally open to the user.
- Grey – DO NOT USE – used by the Marvell debug tool.

5.2 Upper RAM Usage

The top 4K bytes of CPU's memory address space is special for the IMP:

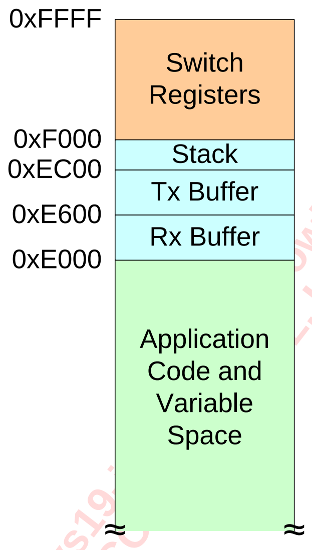
- It is used to access the switch registers (section 7).

In order to support the largest possible consecutive RAM space for the CPU's code, the top of RAM is useful as follows:

- The CPU needs a stack area.
- The NIC needs an Rx buffer and a Tx buffer.
- The code needs a space for variables.

An example allocation of the top of RAM is shown in Figure 6.

Figure 6: Example Upper RAM Usage



The color used in Figure 6 means:

- Orange – Can't be moved. This is how the switch registers are accessed.
- Blue – User usable with modifications per the target application.
- Green – Totally open to the user.

Note The Blue areas in the above figures are examples only and their sizes are application specific.

5.3 Lower EEPROM Usage

Figure 4 shows an example of using the EEPROM for Fast Boot operation (section 4.1). In this mode, the 1st eleven bytes of the EEPROM are specifically defined for proper Fast Boot operation. The next bytes are defined as “App config data” (application configuration data). The size of this data is application specific (i.e., its size is not fixed) and it is targeted for customizable parameters for the application, such as:

- The MAC address of the device which must be unique – allows MAC address assignment in a fixed location of the EEPROM without needing to change the code image.
- Define which switch ports are used – this allows one code image to support many different build options of the device (8 port, 5 port, 2 port, etc.).
- What applications, or options in the application(s), are to be enabled – allows for common code updates for many different product features.

All of these allow customization of the application without needing to change the code image. This is really important if Fast Boot with Verification is being used (section 4.1.2) and it also allows code updates (section 4.1.3.1) without destroying the individual unique parameters of this particular device.

5.4 Upper EEPROM Usage

Figure 3 shows that the maximum EEPROM size can be 64K bytes, which is larger than the RAM size of 60K bytes. Figure 6 shows that the top 8K bytes of the CPU’s memory address space, if not more, is easily allocated for Switch Register Access, NIC buffers, stack, etc. At first look, the top 8K bytes or more of a 64K byte EEPROM appears to be wasted, but this is not true. It is a perfect location for storing portions of the code that are not speed critical and bringing them into RAM when needed. Some examples of this are:

- Text strings used for code debug or application messages.
- Ethernet frame skeletons with many of the fixed protocol fields already filled in.
- Non-speed critical portions of the application code.

Bank switching into RAM portions of the code or data is easily accomplished by using the Fast Copy EEPROM to RAM feature of the IMP (section 6.5). If more code space is needed, more than one EEPROM can be connected to the device (as shown in Figure 1). Simply chip select the desired EEPROM prior to performing the Fast Copy operation (section 6.5.4). A single device GPIO pin and an external inverter is all that is needed to support two EEPROM chip selects. More GPIO pins can be used to support more than two EEPROMs.

6 IMP I/O Peripherals

The IMP CPU has various peripherals under its control. These devices are mapped using the CPU's I/O address space, with the exception of the switch registers, which use the top 4K bytes of the CPU's memory address space (see Figure 3).

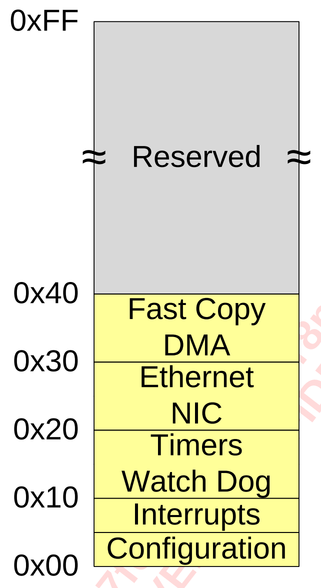
The next sections cover the following I/O mapped peripherals:

- Configuration– Section 6.1
- Interrupt Controller – Section 6.2
- Dual Timers & Watch Dog – Section 6.3
- Ethernet NIC – Section 6.4
- Fast Copy DMA – Section 6.5

As a quick reference, the CPU's I/O Map is as follows – see Figure 7:

- 0x00 to 0x06 = Configuration
- 0x07 to 0x0F = Interrupt Controller
- 0x10 to 0x1F = Dual Timers & Watch Dog
- 0x20 to 0x2F = Ethernet NIC
- 0x30 to 0x3F = Fast Copy DMA
- 0x40 to 0xFF = Reserved for future use

Figure 7: CPU I/O Map



The register descriptions that follow contain a Type field which has the following meanings:

Table 3: Register Field Types

Type	Description
RES	Reserved for future use. All reserved bits are read as zero unless otherwise noted.
RO	Read only.
ROC	Read only clear. After read, register field is cleared to zero.
RWC	Read/Write clear on read. All bits are readable and writable. After reset or after the register is read, the register field is cleared to zero.
RWR	Read/Write reset. All bits are readable and writable. After reset the register field is cleared to zero.
RWS	Read/Write set. All bits are readable and writable. After reset the register field is set to a non-zero value specified in the text.
SC	Self-Clear. Writing a one to this register causes the desired function to be immediately executed, then the register field is cleared to zero when the function is complete.

6.1 Configuration

Configuration includes three groups of I/O registers as follows:

The EEPROM Group:

- 0x01 = EEPROM Speed, switch/device resets & re-boot configuration

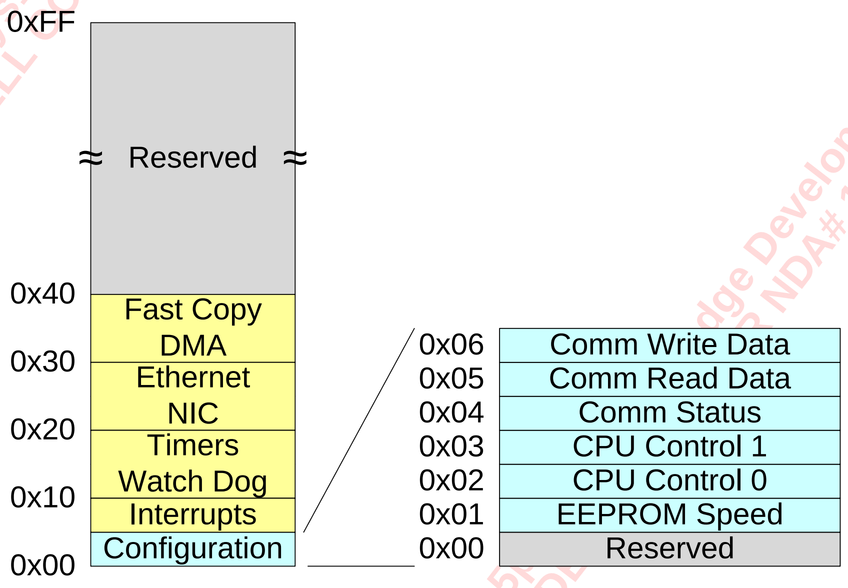
The CPU Control Group:

- 0x02 = Device's INTn pin control & Total IMP Stop
- 0x03 = Misc. Status & IMP Rev

The Communication Group:

- 0x04 = Comm Status
- 0x05 = Comm Read Data
- 0x06 = Comm Write Data

Figure 8: Configuration I/O Map



Each of these groups is defined in more detail below.

EEPROM Registers

The EEPROM register (section 6.1.1) is written to as part of the IMP Fast Boot process as described in section 4.1 (the contents of the 2nd byte of the EEPROM is written to this register) supporting the following functions:

- The Switch Core can be held in a reset state until the software is ready to configure it.
- The entire device, including the IMP itself, can be reset via this register.
- After RESETn, the IMP's RAM is automatically cleared to zero and initialized with proper parity. A parity control bit in this register supports writing bad parity on any writes to the IMP's RAM (i.e., from the CPU, the NIC or the DMA) to facilitate testing of the parity.
- The IMP can be configured to re-boot on a bad Fast Boot or on detection of a parity error.
- The EEPROM's Speed can be configured to optimize the Fast Boot time.

CPU Control Registers

The CPU Control Group registers (section 6.1.2) support the following functions:

- Allows the application to define if the device's INTn pin is an input to the IMP CPU or not (the default mode for the INTn pin is to be an open drain output being driven by the Switch Core interrupts)
  - If there is no other (external) CPU connected to the device, the INTn pin being an input to the IMP CPU allows external devices to wake up/interrupt the IMP CPU.
- Place the IMP into a Total Stop mode to save power. The Total Stop is permanent and takes a Reset to restart the IMP CPU.
  - The Total Stop is intended to be used if the application that boots the CPU only needs to run once after each RESETn and has nothing else it will ever do (like Register Loader type code running on the IMP CPU).
  - If the CPU is in the Total Stop mode, it can be Reset using the debug interface (section 8.2.7).
- Reports to the application the IMP's design revision, the Fast Copy CRC result & if a parity error re-boot ever occurred.
  - Difference between this and future IMP designs will be indicated by changes in the IMP Rev (I/O 0x03).

Communication Registers – UART-Like

The Communication registers (section 6.1.3) are intended to be used like a UART (for debug purposes) or as a mailbox to an external CPU that is connected to the device using the switch's external register interface pins (the MDC/MDIO pins or SMI, for System Management Interface). Interrupts are supported in both directions. The external CPU can be interrupted when the IMP CPU writes a byte to the Comm interface (the device's INTn pin can be activated in this way). Likewise, the IMP CPU can be interrupted when the external CPU writes a byte to its side of the Comm interface. A total of 10 bits of information are available in both directions. These bits are intended to be used as 8-bits of data and 2-bits of control.

6.1.1 EEPROM Registers

Note I/O register 0x01 is modified by the Fast Boot process (see section 4.1).

Table 4: EEPROM Register, I/O Address 0x01

Bits	Field	Type	Description
7	Switch Reset	RWR	Switch Reset. When this bit is set to a one the entire switch core, the switch core's registers and the IMP NIC receives a hardware reset. The IMP CPU and its other peripherals are not reset by this bit. The switch core will be held in the reset state until software clears this bit to a zero. This Switch Reset resets the Ethernet frame datapaths, the switch's databases and the switch's registers (i.e., the ATU, VTU & TCAM are all flushed). If only the switch datapath needs to be reset (without the switch's register settings and databases being wiped out & flushed), use the SWReset bit in Global 1 offset 0x04 (see the switch's Functional Specification for more information on the SWReset bit). Note: This register can be used to hold switch in reset until the IMP software clears this bit. This bit allows emulation of the switch's Register Loader's Release Internal Reset command (for more information on the device's Register Loader see the switch's Functional Specification).
6	Full Reset	SC	Full Reset. Setting this bit to a one resets almost the entire chip, including the PHYs and the IMP (the boot process re-starts as well), as if an external RESETn was applied to the chip. The CONFIG pins are NOT re-read however. The external RESETn pin is not driven in this case but all internal reset nets are driven inside the device. Note: This bit is the only way software can clear the ParityErrReBoot bit (I/O 0x03). If the IMP does not need to be reset, use the Switch Reset bit above or the SWReset bit (Global 1 offset 0x04).

Table 4: EEPROM Register, I/O Address 0x01

Bits	Field	Type	Description
5	BadParity	RWR	Write Bad Parity. 0 = Normal operation – good parity is written to memory 1 = Bad parity is written to memory This bit is needed in order to test the Re-boot if Parity Err bit below. When this bit is set to a one all memory writes will be written with bad parity such that the next read from that location will cause a re-boot if the Re-Boot If Parity Err bit below is set to a one. Note: When this bit is set to a one, ALL writes to the RAM will be written with bad parity. This includes writes by the CPU, the NIC or any DMA.
4	ReBoot If Fast Boot Bad	RWR	Re-boot if the Fast Boot data was Bad. If the Fast Boot block's CRC was bad (section 4.1.2) then the Fast Boot process is restated but at a slowest EEPROM speed. This bit has no effect on Fast Copy operations (section 6.5). 0 = Normal operation – no re-boot if a bad CRC is detected 1 = Re-boot if a bad CRC is detected on the Fast Boot block transfer During the Fast Boot block transfer (section 4.1) a CRC is computed on the data block that was moved from the EEPROM to RAM. This resulting CRC is reported in GoodFcCRC bit (I/O 0x03 bit 4). When this register's bit is set to a one, and if the GoodFcCRC bit is a zero at the end of a Fast Boot operation (i.e., there was a bad CRC on the transfer), the re-boot sequence will re-start from the beginning. However, this time the 2nd byte read from the EEPROM will not modify this register's Speed bits such that the slowest EEPROM speed is used instead for subsequent boot attempts. The re-boot event is indicated by the EEPROM Speed bits (below) being 0x0 instead of the value defined in the 2nd byte of the EEPROM. Note: The EEPROM can decide if its CRC should be checked or not & re-boot in hardware. This is called Fast Boot with Verification (4.1.2).
3	ReBoot If Parity Err	RWR	Re-boot if Parity Error. Re-boot if a Parity Error occurs on a memory read. 0 = Normal operation – no re-boot if parity error detected 1 = Re-boot if bad parity occurs on any memory read When this bit is set to a one, if a parity error occurs during any memory read (by any device), the re-boot sequence will re-start from the beginning (i.e., the 1st byte of the EERPOM will be read, etc.).

Table 4: EEPROM Register, I/O Address 0x01

Bits	Field	Type	Description
2:0	EEPROM Speed	RWR	Speed of the EEPROM. The slowest EEPROM speed is used initially, but software and/or the Fast Boot process can change the EEPROM's speed setting at any time. Valid settings of this register are: 000 = 200 KHz Speed ← Default so all EEPROMs work. 001 = 400 KHz Speed 010 = 500 KHz Speed 011 = 600 KHz Speed 100 = 800 KHz Speed 101 = 1000 KHz Speed 110 = 1200 KHz Speed 111 = 1500 KHz Speed Note: Do not set this register to a value that is a faster speed than what the EEPROM device is rated for.

Note

The following bits in this register are configurable by the 2nd byte of the EEPROM during a Fast Boot (section 4.1):

Bit 7: Switch Reset

Bits 6:5: Must be zero in the 2nd byte of the EEPROM

Bit 4: Re-Boot if Fast Boot Bad

Bit 3: Must be zero in the 2nd byte of the EEPROM

Bits 2:0: EEPROM Speed – only if this Fast Boot is not a Re-Boot if Fast Boot Bad

6.1.2 CPU Control Registers

Table 5: CPU Control 0 Register, I/O Address 0x02

Bits	Field	Type	Description
7	ExtIntPin IsInput	RWR	External Interrupt Pin is Input. 0 = Switch Core Interrupts drive the device’s INTn pin. 1 = The device’s INTn pin is an interrupt input to the IMP CPU. When this bit is cleared to a zero the Switch Core interrupts drive the device’s INTn pin as an open drain output. When this bit is set to a one the Switch Core interrupts are disconnected from driving the device’s INTn pin allowing the INTn pin to be an interrupt input to the IMP CPU connected to the ExtIntPinInt interrupt bit (I/O 0x07). In this mode the IMP CPU can still see and be interrupted by Switch Core interrupts on the SwitchCoreInt interrupt bit (I/O 0x07) allowing the external INTn pin to be an independent interrupt to the IMP CPU from some other external source. The device’s INTn pin structure is shown in Figure 9, below.
6:5	Reserved	RES	Reserved for future use.
4	Total IMP Stop	RWR	Total IMP System Stop. 0 = IMP & its Peripherals work normally 1 = IMP & its Peripherals are totally stopped & can’t re-start Setting this bit to a one stops all the IMP’s internal clocks such that they can’t be started up again without an external device reset (on the reset pin) or an IMP Reset (done by the IMPOp register at Global 2 offset 0x13 index 0x08 – see section 8.2.7). This is intended for lowest power mode when the IMP executes code only once at reset and doesn’t need to execute any more code. Note: Setting this bit to a one will cause the NIC to be disconnected from the switch core (i.e., the NIC’s Port link will be forced down).
3:0	Reserved	RES 0x3	Reserved for future use. Bits 1:0 of this register must always be written with a value of 0x3.

Figure 9: INTn Pin Schematic

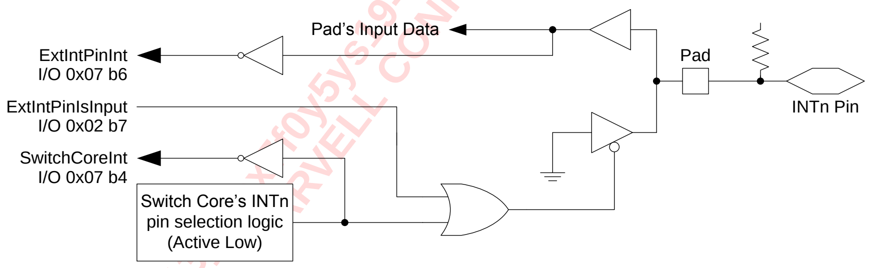


Table 6: CPU Control 1 Register, I/O Address 0x03

Bits	Field	Type	Description
7:6	Reserved	RES	Reserved for future use.
5	ParityErr ReBoot	RO	Parity Error Re-Boot occurred 0 = Normal operation 1 = Parity Error Re-Boot occurred This bit is set to a one if a Parity Error Re-Boot occurred (see I/O 0x01 bit 3). This allows the boot code to test this bit and keep track of the number of Parity Error Re-Boots in a parity error re-boot counter that the code places in any of the switch's scratch registers, since these scratch registers will not get reset during a Parity Error Re-Boot. The only way this bit gets cleared to a zero is by a Full Reset (I/O 0x01) or by asserting the external device's RESETn pin.
4	GoodFc CRC	RO	Good Fast Copy CRC. 0 = The CRC was bad on the last Fast Copy operation 1 = The CRC was good on the last Fast Copy operation This bit is valid only at the end of each Fast Boot or Fast Copy operation. Do not test this bit in the middle of these operations. These operations include EEPROM to RAM transfers generated by the EEPROM (i.e., Fast Boot – section 4.1) and those generated by the CPU (i.e., Fast Copy – section 6.5). CPU RAM to RAM copy transfers (section 6.5) do not affect this bit. If a Fast Copy block of data in the EEPROM does not contain a CRC then the value of this bit becomes don't care.
3:0	IMP Rev	RO	Integrated Micro Processor IP Revision. These bits represent the Revision level of this IMP CPU & its Peripherals. The testing of this bit by the software allows for backwards compatible support when changes are made to this core. This initial IMP Rev is 0x0.

6.1.3 Communications Registers

Table 7: Comm Status Register, I/O Address 0x04

Bits	Field	Type	Description
7	DMADone	RO	Fast Copy DMA Done. 0 = The Fast Copy DMA is not done (or has not started). 1 = The Fast Copy DMA is done. This bit is set to a one whenever a Fast Copy DMA operation completes (i.e., when FastCopyBsy, I/O 0x30, transitions from a one to a zero). This bit being a one can be used to generate a CommDMAInt (I/O 0x07) if enabled. This bit will clear to a zero when a one is written to the FastCopyAck bit (I/O 0x30).
6:5	WtFlags	RWR	Write Data Flags. These two bits can be used for any purpose of communicating extra information between the IMP and the SMI interface. For example, one of these bits can be used to indicate if the WrData contains a Command or Data. These WtFlag bits show up in the SMI's RdFlag bits (Global 2 offset 0x13 index 0x00).
4	WtData Bsy	RO	Write Data to the SMI Interface is Busy. 0 = The WtData register is available for new information 1 = The WtData register is busy with data for the SMI Interface This bit is automatically set to a one whenever the WtData register (I/O 0x06) is written to by the IMP. It automatically clears to a zero whenever the SMI interface reads this data out of its register (Global 2 offset 0x13, index 0x01). This bit being a one can cause the switch's EEInt interrupt to occur if enabled to do so in the switch's register space (see Global 1 offset 0x00). This bit drives the SMI's RdDataRdy bit (Global 2 offset 0x13 index 0x00).
3	RdIntEn	RWR	Enable the RdData Interrupt from the SMI Interface. 0 = The RdData Interrupt from the SMI Interface is not enabled 1 = The RdData Interrupt from the SMI Interface is enabled Setting this bit to a one enables the SMI interface to interrupt the IMP if enabled. The IMP will get a CommDMAInt (I/O 0x07) when this bit is set to a one and whenever the RdDataRdy bit below is also a one. When the RdData register (I/O 0x05) is read by the IMP this interrupt will be cleared until the next write to the SMI's WrData register (Global 2 offset 0x13 index 0x02),

Table 7: Comm Status Register, I/O Address 0x04

Bits	Field	Type	Description
2:1	RdFlags	RO	Read Data Flags. These two bits can be used for any purpose of communicating extra information between the SMI interface and the IMP. For example, one of these bits can be used to indicate if the RdData contains a Command or Data. These RdFlag bits get their data from the SMI's WtFlag bits (Global 2 offset 0x13 index 0x00).
0	RdData Rdy	RO	Data from the SMI interface is ready to be Read. 0 = Data in the RdData register is not valid 1 = Data in the RdData register is valid and should be read This bit being set to a one indicates that the SMI interface has placed a byte of data in the IMP's RdData register (I/O 0x05). When the RdData register is read by the IMP, this bit will automatically clear to zero. This bit being a one can be used to interrupt the IMP via the CommDMAInt (I/O 0x07) if the RdIntEn bit above is set to a one. This bit is a copy of the SMI's WtDataBsy bit (Global 2 offset 0x13 index 0x00).

Table 8: Comm Read Data Register, I/O Address 0x05

Bits	Field	Type	Description
7:0	RdData	RO	Read Data from the Switch's SMI Interface Registers. This register contains the byte of data written to the SMI interface registers destined to go to the IMP (see section 8.5.5). It contains valid data whenever the RdDataRdy bit (I/O 0x04) is set to a one. When this RdData register is read, the RdDataRdy bit will be cleared to a zero indicating to the SMI interface that this register is ready for the next byte of data. The SMI interface can cause an IMP interrupt to occur whenever valid data from the SMI interface is contained in this register. See I/O 0x04. This RdData register gets its data from the SMI's WtData register (Global 2 offset 13 index 0x02).

Table 9: Comm Write Data Register, I/O Address 0x06

Bits	Field	Type	Description
7:0	WrData	RWR	Write Data to the Switch's SMI Interface Registers. This register contains the byte of data the IMP wants to send to the SMI interface registers (see section 8.5.5). Writing to this register automatically sets the WtDataBsy bit (I/O 0x04) which will not be cleared until this data is read using the SMI interface. Therefore, this register should not be written to again without first checking that the WtDataBsy bit is zero. If a write is done with the WtDataBsy bit being a one, the write will occur, overwriting the contents of this register. The writing of this register can cause the switch's EEInt interrupt to occur if enabled to do so in the switch's register space (see Global 2 offset 0x00). This WrData register sends data to the SMI's RdData register (Global 2 offset 0x13 index 0x01).

6.2 Interrupt Controller

Two levels of interrupts from multiple interrupt sources are supported. Interrupt priority for simultaneous interrupts is configurable, and all three CPU Interrupt Modes (0, 1 and 2) are supported. CPU Interrupt Mode 2 is recommended to be used as this frees up the CPU's Restart instructions to be used for the most prevalent subroutine calls, saving code space (Figure 5).

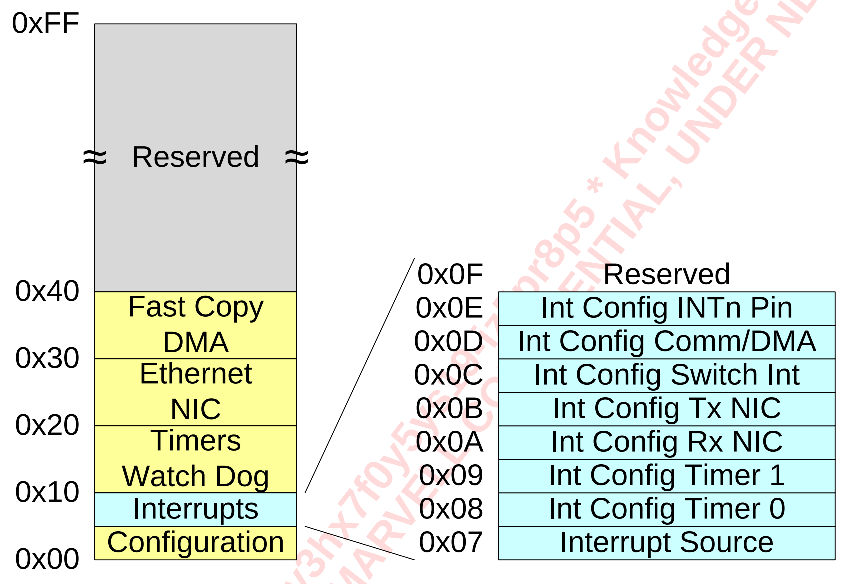
Note Any active and unmasked interrupt will re-start the CPU if it was Halt'ed (i.e., the CPU previously executed a Halt instruction).

The IMP Interrupt Configuration and Control includes the following I/O registers:

- 0x07 = Interrupt Source

- 0x08 = Interrupt Ack 0 Data & Mask – Timer 0
- 0x09 = Interrupt Ack 1 Data & Mask – Timer 1
- 0x0A = Interrupt Ack 2 Data & Mask – Receive Frame Ready (NIC)
- 0x0B = Interrupt Ack 3 Data & Mask – Transmit Frame Done (NIC)
- 0x0C = Interrupt Ack 4 Data & Mask – Switch Core INTn
- 0x0D = Interrupt Ack 5 Data & Mask – Communications Port & Fast Copy DMA Interrupt
- 0x0E = Interrupt Ack 6 Data & Mask – External INTn Pin Interrupt
- 0x0F = Interrupt Ack 7 Data & Mask – Reserved for future use

Figure 10: Interrupt Controller I/O Map



6.2.1 Interrupt Controller Registers

The Interrupt Source register (I/O 0x07) supports a single register to read back all active interrupts in one place (even if they are currently masked). All interrupts are level sensitive. The eight Interrupt Ack registers (I/O 0x08 to 0x0F) are used to:

- Mask or enable each interrupt at the Interrupt Controller level
- Define each interrupt's type (either NMI or INT to the CPU)
- Define each interrupt's priority in case multiple INT interrupts are active at the same time
- Define each interrupt's Interrupt Acknowledge Cycle data presented to the CPU

The CPU supports two different interrupt types, NMI (Non-Maskable Interrupt) & INT (Interrupt). The NMI is non-maskable inside the CPU, and any IMP interrupt defined to be an NMI type is no longer masked by the interrupt controller either. The INT type of interrupts are maskable by the CPU and have to be unmasked by the Enable Interrupt (EI) CPU instruction before the CPU can be interrupted.

The CPU supports three different Interrupt Modes and the Interrupt Controller's Interrupt Ack registers must be configured to match the CPU's Interrupt Mode accordingly.

Note

The CPU's Non-Maskable Interrupt (NMI) is masked in two places after RESETn.

- One of these masks is in the IMP's Interrupt Controller as none of the possible interrupt sources are initially defined as an NMI (which connects that interrupt to the CPU's NMI effectively unmasking it – I/O's 0x08 to 0x0F).
- The second mask is in the IMP's debug interface at Global 2 offset 0x13 index 0x0F bit 0 (Table 63). This bit must be set to a one before the CPU will receive any active NMI's.

Masking & Enabling Specific Interrupt Sources

A value of 0x00 written to an interrupt's Interrupt Ack register masks that specific interrupt source (this is the default value of each Interrupt Ack register).

A value of 0x01 enables the interrupt as a Non Maskable level sensitive Interrupt (NMI – see the Note above to fully unmask the NMI).

All other values (0x02 to 0xFF) define the interrupt as an un-masked, normal, level sensitive, interrupt (INT). In this case the contents of the Interrupt Ack register is used during the active interrupt's Interrupt Acknowledge Cycle to the CPU. Proper values in these registers are required to match the CPU's Interrupt Mode and to match the entry points to the various interrupt service routines. The INT type of interrupts are maskable by the CPU and have to be unmasked by the Enable Interrupt (EI) CPU instruction before the CPU can be interrupted.

Using Interrupt Mode 0

Interrupt Mode 0 (the default mode) requires an interrupt vector table in the 1st 128 bytes of RAM (see the list below). If the CPU is in Interrupt Mode 0 the Interrupt Ack register for an unmasked interrupt must use one of the following values resulting in a subroutine call (Restart) to the listed addresses:

- 0x01 = Uses NMI & the CPU will Restart at 0x0066
- 0xC7 = Uses INT with Priority 7 & the CPU will Restart at 0x0000
- 0xCF = Uses INT with Priority 6 & the CPU will Restart at 0x0008
- 0xD7 = Uses INT with Priority 5 & the CPU will Restart at 0x0010
- 0xDF = Uses INT with Priority 4 & the CPU will Restart at 0x0018
- 0xE7 = Uses INT with Priority 3 & the CPU will Restart at 0x0020
- 0xEF = Uses INT with Priority 2 & the CPU will Restart at 0x0028
- 0xF7 = Uses INT with Priority 1 & the CPU will Restart at 0x0030
- 0xFF = Uses INT with Priority 0 & the CPU will Restart at 0x0038

Using Interrupt Mode 1

Interrupt Mode 1 requires a two entry vector table in the 1st 128 bytes of RAM (see the list below). If the CPU is in Interrupt Mode 1 the Interrupt Ack register for an unmasked interrupt must use one of the following values resulting in a subroutine call (Restart) to the listed addresses:

- 0x01 = Uses NMI & the CPU will Restart at 0x0066
- 0xFF = Uses INT & the CPU will Restart at 0x0038

In Mode 1, the Interrupt Controller will not prioritize the interrupts if more than one is active at the same time. In this situation the interrupt service routine connected to addr 0x0038 can prioritize the active interrupts by reading the Interrupt Source register (I/O 0x07) so see which interrupts are active.

Note

Active but masked interrupts appear as active in the Interrupt Source register as masking only prevents an interrupt from being serviced by the CPU. This approach supports systems that don't use interrupts (polling) as well as other applications.

Using Interrupt Mode 2

Interrupt Mode 2 requires an interrupt vector table but it can be placed just about anywhere in RAM. In this mode the CPU's <I Reg> supplies the high 8-bits of the interrupt vector table's address and the Interrupt Controller supplies the low 8-bits. The IMP Interrupt Controller can support any value for this low 8-bit address, with the exception of the values of 0x00 and 0x01.

If the CPU is in Interrupt Mode 2 the Interrupt Ack register for an unmasked interrupt must use one of the following values:

- 0x01 = Uses NMI & the CPU will Restart at 0x0066
- 0x02 to 0xFF = Uses INT & the CPU will Call at the address defined by <I Reg><IntXAckData>

Interrupts are prioritized by the Interrupt Controller when the CPU is using interrupt Mode 2. If multiple interrupts are active at the same time, the interrupt that has the lowest IntXAckData value will have its IntXAckData value used during the CPU's Interrupt Acknowledge cycle (IntXAckData values 0x00 & 0x01 are not included in this priority resolution case).

Support for Two Levels of Interrupts

The CPU can support two levels of interrupts since the IMP's Interrupt Controller can define each of its interrupt sources as either an INT type or an NMI type of interrupt to the CPU. When the CPU accepts an INT type interrupt (and calls the appropriate interrupt service routine) it automatically masks all other INT type interrupts. But NMI type interrupts can still be serviced by the CPU in this case. Thus:

- INT is a Level 1 Interrupt
- NMI is a Level 2 Interrupt

Where a higher level number interrupt can interrupt a lower level.

Note

Refer to documentation on a Z80 CPU to understand what the software has to do in order to support INT as Level 1 interrupt and NMI as a Level 2 interrupt. But do not use the LD A,R instruction to read the status of the IFF2 interrupt enabled flip-flop as the Refresh register is not implemented in this design. Use the LD A,I instruction instead (load the CPU's register A from the Interrupt Vector register).

Keep the number of NMI type interrupt sources to a minimum as multiple simultaneous NMI interrupts would have to be prioritized in software by examination of the Interrupt Source register (I/O 0x07).

Remember to unmask the NMI's – see the first Note in section 6.2.1 above.

Table 10: Interrupt Source Register, I/O Address 0x07

Bits	Field	Type	Description
7	Reserved	RES	Reserved for future use.
6	ExtIntPin Int	RO	Int 6 – External INTn Pin Interrupt. 0 = The External INTn pin is inactive (high). 1 = The External INTn pin is active (low). This bit will cause an interrupt to the IMP CPU if Int6 is enabled in the Int6AckData register (I/O 0x0E). This bit is an inverted copy of the signal on the device's INTn pin. To prevent this interrupt from being an exact copy of the SwitchCoreInt, the SwitchCoreInt needs to be disconnected from the INTn pin (see the ExtIntPinIsInput bit in I/O 0x02).
5	Comm DMAInt	RO	Int 5 – Communications Port / DMA Interrupt. 0 = Data in the RdData register is not valid & DMA not done. 1 = Data in the RdData register is valid and should be read, or a Fast Copy DMA transfer completed. This bit is an 'OR' of the RdDataRdy and the DMADone bits in I/O 0x04 and this bit being a one will cause an interrupt to the IMP CPU if Int5 is enabled in the Int5AckData register (I/O 0x0D). Since this bit is an 'OR' of two interrupt sources, a subsequent Int5 interrupt will occur unless both interrupt sources are serviced (i.e., both RdDataRdy and DMADone are zero if both are being used). The DMADone interrupt can be masked, however, by leaving FastCopyAck (I/O 0x30) set to a one. In this mode all Int5 interrupts will be from the Communications Port only.
4	Switch CoreInt	RO	Int4 – Switch Core Interrupt. 0 = No Switch Core Interrupts are active. 1 = At least one Switch Core Interrupt is active. This bit will cause interrupt to the IMP CPU if Int4 is enabled in the Int4AckData register (I/O 0x0C). This bit is set to a one whenever any of the switch core interrupts are set to a one (in Global 1 offset 0x00) that are also enabled (in Global 1 offset 0x04).
3	TxFrame DoneInt	RO	Int3 – Transmit Frame Done Interrupt. 0 = The Tx portion of the NIC is busy with a frame. 1 = The Tx portion of the NIC is ready to start a new frame. This bit is a virtual copy of the TxFrameDone bit in I/O 0x28 – except this bit is set to a one on the transition of the TxFrameDone bit from a zero to a one. It is cleared to a zero when the TxFrameDone bit is cleared to a zero. This slight difference in the setting of this bit to a one prevents an interrupt from the Tx NIC after an IMP Reset (as this bit is not set to a one until after the 1st frame is transmitted out the NIC). This bit being a one will cause an interrupt to the IMP CPU if Int3 is enabled in the Int3AckData register (I/O 0x0B).

Table 10: Interrupt Source Register, I/O Address 0x07

Bits	Field	Type	Description
2	RxFrame RdyInt	RO	Int2 – Receive Frame Ready Interrupt. 0 = The switch does not have any Frames for the IMP 1 = The switch has at least one Frame for the IMP This bit is a copy of the RxFrameRdy bit in I/O 0x20 and this bit being a one will cause an interrupt to the IMP CPU if Int2 is enabled in the Int2AckData register (I/O 0x0A).
1	Timer1Int	RO	Int1 – Timer 1 Interrupt. 0 = Timer 1 has not reached a count value of 0x0000. 1 = Timer 1 has reached a count value of 0x0000. This bit is a copy of the Timer1Done bit in I/O 0x11 and this bit being a one will cause an interrupt to the IMP CPU if Int1 is enabled in the Int1AckData register (I/O 0x09).
0	Timer0Int	RO	Int0 – Timer 0 Interrupt. 0 = Timer 0 has not reached a count value of 0x0000. 1 = Timer 0 has reached a count value of 0x0000. This bit is a copy of the Timer0Done bit in I/O 0x10 and this bit being a one will cause an interrupt to the IMP CPU if Int0 is enabled in the Int0AckData register (I/O 0x08).

Table 11: Interrupt Ack 0 Register, I/O Address 0x08

Bits	Field	Type	Description
7:0	Int0Ack Data	RWR	Interrupt Acknowledge Data for Int 0 – the Timer 0 Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 12: Interrupt Ack 1 Register, I/O Address 0x09

Bits	Field	Type	Description
7:0	Int1Ack Data	RWR	Interrupt Acknowledge Data for Int 1 – the Timer 1 Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 13: Interrupt Ack 2 Register, I/O Address 0x0A

Bits	Field	Type	Description
7:0	Int2Ack Data	RWR	Interrupt Acknowledge Data for Int 2 – the RxFrameRdy Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 14: Interrupt Ack 3 Register, I/O Address 0x0B

Bits	Field	Type	Description
7:0	Int3Ack Data	RWR	Interrupt Acknowledge Data for Int 3 – the TxFrameDone Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 15: Interrupt Ack 4 Register, I/O Address 0x0C

Bits	Field	Type	Description
7:0	Int4Ack Data	RWR	Interrupt Acknowledge Data for Int 4 – the SwitchCoreInt Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 16: Interrupt Ack 5 Register, I/O Address 0x0D

Bits	Field	Type	Description
7:0	Int5Ack Data	RWR	Interrupt Acknowledge Data for Int 5 – the CommDMAInt Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 17: Interrupt Ack 6 Register, I/O Address 0x0E

Bits	Field	Type	Description
7:0	Int6Ack Data	RWR	Interrupt Acknowledge Data for Int 6 – the ExtIntPin Interrupt (see I/O 0x07). This register's contents must match the desired interrupt level (NMI or INT), the desired interrupt priority level and the Interrupt Mode of the CPU. See the description in section 6.2.1 to determine the proper value for this register. A value of 0x00 masks this interrupt.

Table 18: Interrupt Ack 7 Register, I/O Address 0x0F

Bits	Field	Type	Description
7:0	Int7Ack Data	RES	Reserved for future use.

6.3 Dual Timers & Watch Dog

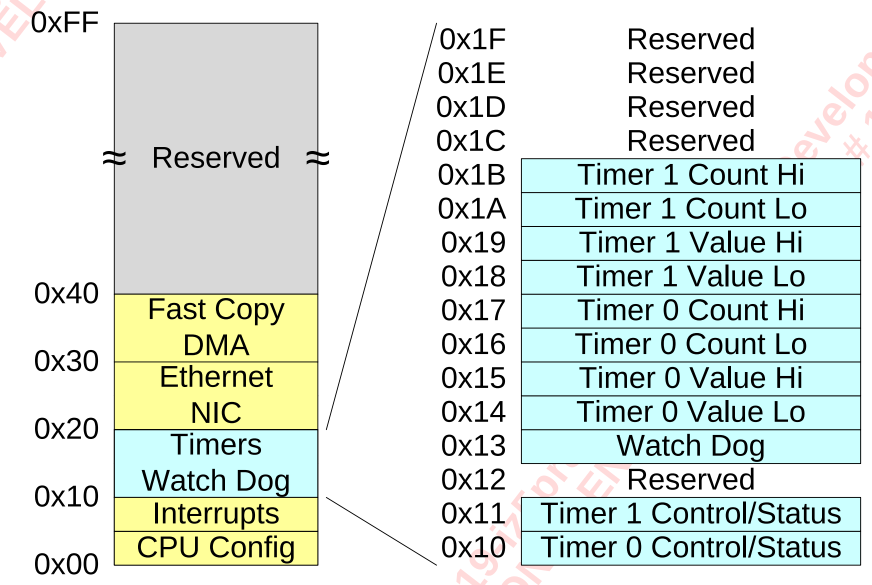
Two independent Timers are supported along with a Watch Dog circuit. The IMP Timers & Watch Dog include the following I/O registers:

- 0x10 = Timer 0 Control/Status
- 0x11 = Timer 1 Control/Status
- 0x12 = Watch Dog Status
- 0x13 = Watch Dog Control

- 0x14 = Timer 0 Value Lo
- 0x15 = Timer 0 Value Hi
- 0x16 = Timer 0 Count Lo
- 0x17 = Timer 0 Count H

- 0x18 = Timer 1 Value Lo
- 0x19 = Timer 1 Value Hi
- 0x1A = Timer 1 Count Lo
- 0x1B = Timer 1 Count Hi

Figure 11: Dual Timers & Watch Dog I/O Map



Each Timer runs at a 100 uSec rate with a 16-bit count down value (supporting a range of 100 uSec to 6.553 Seconds in 100 uSec increments). Each Timer is configurable to count once (one-shot mode) or to re-start itself (continuous mode). When the Timer's counts reaches a value of 0x0000 it can be configured to generate an interrupt to the CPU (I/O 0x07).

The Watch Dog's circuit counts down and when it's count reaches zero, the IMP block is reset. This reset clears the RAM to zeros, resets the I/O registers & the CPU and re-boots the code from the EEPROM. Once enabled the Watch Dog cannot be disabled and software must update/refresh the Watch Dog register before it reaches a value of zero. The Watch Dog circuit supports a time range of 1 mSec to 256 Seconds.

Note

Writing a value of 0x00 to the Watch Dog register (I/O 0x13) after it is enabled will cause the Watch Dog to trip immediately. This is useful for testing purposes.

Table 19: Timer 0 Control Register, I/O Address 0x10

Bits	Field	Type	Description
7:3	Reserved	RES	Reserved for future use.
2	Timer0 Restart	RWR	Timer 0 Restart. 0 = Timer 0 is configured for one-shot mode. 1 = Timer 0 is configured for continuous mode. When this bit is set to a one, the timer is in continuous mode where it automatically reloads its Count (I/O 0x16 & 0x17) with the data in the timer's Value registers (I/O 0x14 & 0x15) and restarts the countdown process whenever the timer's Count reaches a value of 0x0000. When this bit is cleared to a zero, the timer is in one-shot mode where it automatically stops when the Count reaches a value of 0x0000 (the Timer0Start bit below is automatically cleared to a zero).
1	Timer0 Start	RWR	Timer 0 Start. 0 = Timer 0 is not enabled. 1 = Timer 0 is enabled. When this bit transitions from a zero to a one, the data in the timer's Value registers (I/O 0x14 & 0x15) is transferred to the timer's Count register (I/O 0x16 & 0x17) where the Count is decremented once every 100 uSec until the Count reaches 0x0000. At that time the process will repeat (if Timer0Restart above is set to a one) or this bit will automatically be cleared to a zero (if Timer0Restart is cleared to a zero).
0	Timer0 Done	ROC	Timer 0 Done. 0 = Timer 0 has not reached a value of 0x0000. 1 = Timer 0 has reached a value of 0x0000. This bit is set to a one whenever Timer 0 Count (I/O 0x16 & 0x17) transitions from a 0x0001 to 0x0000 value. This bit is cleared to a zero after this register is read. This bit being a one can cause an IMP CPU interrupt (see I/O 0x07).

Table 20: Timer 1 Control Register, I/O Address 0x11

Bits	Field	Type	Description
7:3	Reserved	RES	Reserved for future use.
2	Timer1 Restart	RWR	Timer 1 Restart. 0 = Timer 1 is configured for one-shot mode. 1 = Timer 1 is configured for continuous mode. When this bit is set to a one, the timer is in continuous mode where it automatically reloads its Count (I/O 0x1A & 0x1B) with the data in the timer's Value registers (I/O 0x18 & 0x19) and restarts the countdown process whenever the timer's Count reaches a value of 0x0000. When this bit is cleared to a zero, the timer is in one-shot mode where it automatically stops when the Count reaches a value of 0x0000 (the Timer1Start bit below is automatically cleared to a zero).
1	Timer1 Start	RWR	Timer 1 Start. 0 = Timer 1 is not enabled. 1 = Timer 1 is enabled. When this bit transitions from a zero to a one, the data in the timer's Value registers (I/O 0x18 & 0x19) is transferred to the timer's Count register (I/O 0x1A & 0x1B) where the Count is decremented once every 100 uSec until the Count reaches 0x0000. At that time the process will repeat (if Timer1Restart above is set to a one) or this bit will automatically be cleared to a zero (if Timer1Restart is cleared to a zero).
0	Timer1 Done	ROC	Timer 1 Done. 0 = Timer 1 has not reached a value of 0x0000. 1 = Timer 1 has reached a value of 0x0000. This bit is set to a one whenever Timer 1 Count (I/O 0x1A & 0x1B) transitions from a 0x0001 to 0x0000 value. This bit is cleared to a zero after this register is read. This bit being a one can cause an IMP CPU interrupt (see I/O 0x07).

Table 21: Watch Dog Status Register, I/O Address 0x12

Bits	Field	Type	Description
7	WatchDog Tripped	RO	Watch Dog Tripped. This bit is set to a one whenever the Watch Dog (I/O 0x13) reaches a value of zero and causes an IMP block reset and subsequent Re-Boot. This bit is NOT reset by this IMP block reset so software can determine if the Watch Dog ever tripped. This bit is cleared only when the device's RESETn pin is active or if the Full Reset bit is set to a one (I/O 0x01).
6:0	Reserved	RES	Reserved for future use.

Table 22: Watch Dog Control Register, I/O Address 0x13

Bits	Field	Type	Description
7:6	WatchDog Rate	RWR	Watch Dog Rate. The Watch Dog rate of decrementing determines the minimum and maximum times before a Watch Dog reset occurs as follows: 0x3 = 4096 mSec min time with a 256 Sec max time 0x2 = 256 mSec min time with a 16 Sec max time 0x1 = 16 mSec min time with a 1.0 Sec max time 0x0 = 1 mSec min time with a 63 mSec max time The Watch Dog bits below determine where in the above listed range the Watch Dog reset will occur. The selected min time is the decrement rate applied to the Watch Dog counter bits below. The Watch Dog count bits select how many counts at the Watch Dog Rate can occur before the IMP is reset. Valid Watch Dog count values are 1 to 63 (0x01 to 0x3F).
5:0	WatchDog	RWR	Watch Dog. IMPORTANT! Once this register is written to, the Watch Dog is enabled and cannot be disabled! Once written, the count in this register will decrement and if it reaches a value of 0x00, a hardware reset to the IMP block will occur. This reset will reset the CPU, all of the CPU's I/O registers & peripherals (but not the switch & the switch registers and not the Watch Dog Tripped bit – I/O 0x12), re-initialize the RAM and will cause a re-load of the code from the EEPROM. The rate at which this register is decremented is configured by the Watch Dog Rate bits above. Software can write to this register at any time to re-start/re-initialize the Watch Dog count at the written value, and it must do so to prevent a reset of the CPU and a re-load of the code. The Watch Dog Rate (above) can be changed at any time. Note: While the CPU is Stopped for debugging (section 8) the Watch Dog is also stopped. The Watch Dog will re-start whenever the CPU is re-started for debugging (Run or Single Step). The Watch Dog Rate (above) determines the rate this counter is decremented. The maximum Watch Dog time (a value 0xFF to this 8-bit register) before a reset occurs is 256 Seconds. The minimum time (a value of 0x01 to this 8-bit register) before a reset occurs is 1 mSec.

Table 23: Timer 0 Value Lo Register, I/O Address 0x14

Bits	Field	Type	Description
7:0	Timer0 ValueLo	RWR	Timer 0 count Value[7:0] in 100 uSec increments. This is the lower 8-bits of a 16-bit count value. When Time0Start transitions from a zero to a one (I/O 0x10), the contents of this register, along with the contents of Timer0ValueHi are transferred to the Time0CountLo & Hi registers. Then every 100 uSec the Time0Count is decremented until it reaches 0x0000 at which time the Timer0Done is set to a one (I/O 0x10). If Time0Restart is set to a one (I/O 0x10) the process repeats. The upper 8-bits of this count Value is at I/O 0x15.

Table 24: Timer 0 Value Hi, I/O Address 0x15

Bits	Field	Type	Description
7:0	Timer0 ValueHi	RWR	Timer 0 count Value[15:8] in 100 uSec increments. This is the upper 8-bits of a 16-bit value fully described at I/O 0x14.

Table 25: Timer 0 Count Lo Register, I/O Address 0x16

Bits	Field	Type	Description
7:0	Timer0 CountLo	RO	Timer 0 current Count [7:0]. This is the lower 8-bits of a 16-bit count value. When Time0Start transitions from a zero to a one (I/O 0x10), the contents of the Timer0ValueLo register, along with the contents of Timer0ValueHi are transferred to this register & the Time0CountHi register. Then every 100 uSec the Time0Count is decremented until it reaches 0x0000. If Time0Restart is set to a one (I/O 0x10) the process repeats. The upper 8-bits of this count Value is at I/O 0x17.

Table 26: Timer 0 Count Hi, I/O Address 0x17

Bits	Field	Type	Description
7:0	Timer0 CountHi	RO	Timer 0 current Count[15:8]. This is the upper 8-bits of a 16-bit value fully described at I/O 0x16.

Table 27: Timer 1 Value Lo Register, I/O Address 0x18

Bits	Field	Type	Description
7:0	Timer1 ValueLo	RWR	Timer 1 count Value[7:0] in 100 uSec increments. This is the lower 8-bits of a 16-bit count value. When Time1Start transitions from a zero to a one (I/O 0x11), the contents of this register, along with the contents of Timer1ValueHi are transferred to the Time1CountLo & Hi registers. Then every 100 uSec the Time1Count is decremented until it reaches 0x0000 at which time the Timer1Done bit is set to a one (I/O 0x11). If Time1Restart is set to a one (I/O 0x11) the process repeats. The upper 8-bits of this count Value is at I/O 0x19.

Table 28: Timer 1 Value Hi, I/O Address 0x19

Bits	Field	Type	Description
7:0	Timer1 ValueHi	RWR	Timer 1 count Value[15:8] in 100 uSec increments. This is the upper 8-bits of a 16-bit value fully described at I/O 0x18.

Table 29: Timer 1 Count Lo Register, I/O Address 0x1A

Bits	Field	Type	Description
7:0	Timer1 CountLo	RO	Timer 1 current Count [7:0]. This is the lower 8-bits of a 16-bit count value. When Time1Start transitions from a zero to a one (I/O 0x11), the contents of the Timer1ValueLo register, along with the contents of Timer1ValueHi are transferred to this register & the Time1CountHi register. Then every 100 uSec the Time1Count is decremented until it reaches 0x0000. If Time1Restart is set to a one (I/O 0x11) the process repeats. The upper 8-bits of this count Value is at I/O 0x1B.

Table 30: Timer 1 Count Hi, I/O Address 0x1B

Bits	Field	Type	Description
7:0	Timer1 CountHi	RO	Timer 1 current Count[15:8]. This is the upper 8-bits of a 16-bit value fully described at I/O 0x1A.

6.4 Ethernet NIC

The IMP's Ethernet NIC (Network Interface Controller) receives and transmits Ethernet frames between the switch core and the IMP's RAM using DMA (Direct Memory Access).

The IMP's NIC includes the following I/O registers:

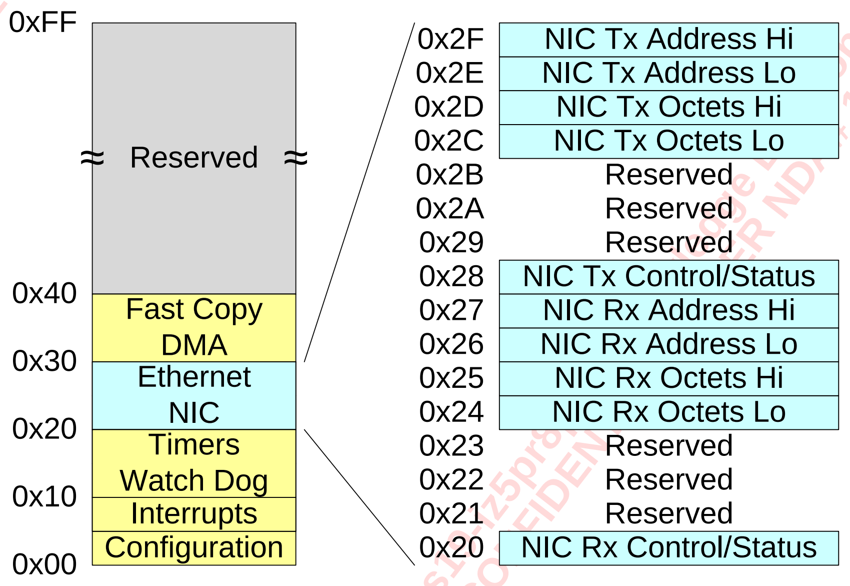
- 0x20 = Ethernet NIC Rx Status/Control

- 0x24 = Ethernet NIC Rx Octets Lo
- 0x25 = Ethernet NIC Rx Octets Hi
- 0x26 = Ethernet NIC Rx Address Lo
- 0x27 = Ethernet NIC Rx Address Hi

- 0x28 = Ethernet NIC Tx Status/Control

- 0x2C = Ethernet NIC Tx Octets Lo
- 0x2D = Ethernet NIC Tx Octets Hi
- 0x2E = Ethernet NIC Tx Address Lo
- 0x2F = Ethernet NIC Tx Address Hi

Figure 12: Ethernet NIC I/O Map



6.4.1 Network Interface Controller (NIC) Overview

The Network Interface Controller (NIC) connected to the IMP CPU is very simple and basic – and thus easy to use. It does not do any MAC address filtering, nor QoS qualifications, nor policing, etc., as the switch core the NIC is connected to can do all this and more. It is the switch's job to protect any CPU connected to it from Denial of Service attacks, intrusions, etc. The switch core also supports Secure Control Technology which performs the QoS mapping of frames mapped to the CPU so the CPU processes the most important frames first. Refer to the switch's Functional Specification for more information on these features of the switch core.

The switch core also contains more packet memory than what is reasonable to allocate for frames in the IMP's RAM. So while the IMP's NIC can transmit and/or receive more than one Ethernet frame from/to RAM, it is recommended to use a single Tx buffer and a single Rx buffer for the Ethernet frames and let the switch core perform the buffering and QoS functions.

Ethernet frames are transmitted and received using DMA (Direct Memory Access). NIC Tx and NIC Rx operations can occur at the same time, at up to gigabit speeds (in both directions at the same time). This is done without impacting the IMP's CPU's memory access in any way – as the CPU's accesses are not slowed down by wait states nor DMA bus requests. This performance is achievable because there is enough RAM bandwidth for all operations to occur concurrently (even Fast Copy DMA operations – section 6.5 – can be occurring at the same time as the NIC transfers without inserting wait states to the CPU's code).

Note

The switch port number that the IMP's NIC is connected to varies from device to device. The capabilities and speed of these switch ports may vary as well. Refer to the switch's Functional Specification for more information.

Note

Received frames in RAM will include every byte received from the switch core, including the frame's CRC. Transmitted frame sizes in RAM must not include the frame's CRC as the IMP NIC will append a good four byte CRC at the end of the data that was DMA'ed into the switch. Frame padding is also supported by the IMP's Tx NIC so software does not need to pad frames up.

Note

The IMP NIC can receive a maximum frame size of 2048 bytes and it can transmit a maximum frame size of 2044 bytes (leaving room for 4 bytes of CRC). The switch port connect to the IMP NIC must be configured to allow frames of this size, however (Port offset 0x08).

6.4.2 NIC Interrupts

The NIC can be configured to generate interrupts to the IMP CPU. Separate interrupt enables are supported for transmit and receive such that one, both or neither can be enabled (see section 6.2). If the same interrupt service routine supports both the transmit and receive interrupts from the NIC, the software can distinguish the source of the interrupt by reading I/O register 0x07 where both the RxFrameRdy and TxFrameDone interrupt status bits exist in the same register (these bits are copied to I/O 0x07 for this purpose).

6.4.3 How to Receive a Frame Via DMA

Before a frame can be received by the IMP's NIC, the NIC must first be configured and enabled:

1. Point RxAddrLo & RxAddrHi (I/O's 0x26 & 0x27) to the start of the receive buffer.
2. Set both RxDMA and TxDMA (I/O's 0x20 & 0x28) to a one – this enables Link up on the port (Port offset 0x00).

When the switch has a frame ready for the CPU, it will transfer the frame to the IMP memory pointed to by the RxAddrLo & RxAddrHi registers (I/O 0x26 & 0x27) and then the NIC will set the RxFrameRdy bit to a one (I/O 0x20). This bit being a one can be used to generate an interrupt to the CPU. Do not set the RxDMA bit to a one unless all the other Rx DMA registers are ready to accept a frame, as the 1st frame could appear immediately.

Note

The Switch's Wake On Frame feature can be used to wake up the CPU if the Switch Core Interrupt (section 6.2) is enabled on the CPU's side. In this case the RxFrameRdy bit must not be set to a one until the Wake On Frame interrupt is serviced, releasing the frame from the switch.

When the RxFrameRdy bit is set to a one by the NIC (i.e., a frame has been DMA'ed into IMP memory):

1. The Ethernet frame is located in IMP memory where RxAddrLo & RxAddrHi originally pointed.
2. RxOctetsLo & RxOctetsHi (I/O 0x24 & 0x25) contain the size of the frame just received (including the four CRC bytes of the frame).
3. RxAddrLo & RxAddrHi currently point to the byte after the last byte received for the frame.
4. CRCErr (I/O 0x20) is a one if the frame had a CRC error.
5. The CPU processes the received frame.
6. When the CPU is ready to receive the next frame it sets RxFrameDone (I/O 0x20) to a one.
7. Setting RxFrameDone to a one:
   a) Clears RxOctetsLo & RxOctetsHi.
   b) Resets RxAddrLo & RxAddrHi to their previous values if RxKeepAddr = 1 (I/O 0x20).
   c) Clears CRCErr.
   d) Clears RxFrameRdy.
   e) Clears RxFrameDone (self clearing the bit).
8. If the switch has another frame for the CPU it will DMA it into IMP memory where RxAddrLo & RxAddrHi point & then set RxFrameRdy to a one.
9. Goto step 1.

The start of the frame is always placed at the memory location pointed to by RxAddrLo & RxAddrHi (I/O 0x26 & 0x27). After that first octet is written to memory the 16-bit address (made up of RxAddrLo & RxAddrHi) will be incremented if RxDecAddr = 0 (I/O 0x20). It will be decremented if RxDecAddr = 1. Subsequent frame data can therefore be placed either direction in memory.

Each subsequent frame will have its first octet placed in the exact same memory location (i.e., the frames will be received into the same frame memory buffer) if RxKeepAddr = 1 (I/O 0x20). If RxKeepAddr = 0, RxAddrLo & RxAddrHi will not be reset to their original values when the CPU sets RxFrameDone to a one. Instead the subsequent frame will be placed in memory right after the last frame. Software needs to prevent the writing of frame data outside its intended buffer in this mode.

If the switch tries to transfer a frame that is larger than 2048 octets in size, the NIC will stop the DMA at octet 2048, set the CRCErr flag and then Skip the rest of the frame. Once the rest of the oversized frame is cleared out, the NIC will set the RxFrameRdy bit to a one so the CPU can process the 1st 2048 octets of the frame if it wishes to.

6.4.4 How to Transmit a Frame Via DMA

Before a frame can be transmitted by the IMP's NIC, the NIC must first be configured and enabled:

1. Point TxAddrLo & TxAddrHi (I/O's 0x2E & 0x2F) to the start of the transmit buffer.
2. Configure the receive portion of the NIC (see section 6.4.3) and then set both RxDMA and TxDMA (I/O's 0x20 & 0x28) to a one – this enables Link up on the port (Port offset 0x00).

When the CPU has a frame ready to be sent into the switch, the CPU sets the TxFrameBsy bit to a one (I/O 0x28) and the NIC will transfer the frame from the IMP RAM to the switch. When the frame is completely inside the switch, the NIC will set the TxFrameDone bit (I/O 0x28) to a one. The transition of the TxFrameDone bit from a zero to a one can be used to generate an interrupt to the CPU.

To Transmit a frame into the switch using DMA:

1. Decide where in the IMP memory to build the frame.
2. Build the Ethernet frame in the IMP memory & note its size. Minimum frame size padding and CRC generation are not to be done by the IMP CPU, nor shall they be included in the frame size.
3. Ensure TxFrameBsy = 0 (I/O 0x28) or wait for its interrupt (TxFrameDoneInt).
4. Write the size of the just built frame (excluding any possible needed padding and its CRC bytes) to TxOctetsLo and TxOctetsHi (I/O 0x2C & 0x2D).
5. Ensure TxAddrLo & TxAddrHi (I/O 0x2E & 0x2F) point to the first octet of the frame in memory.
6. Set TxDecAddr (I/O 0x28) to match the direction of the subsequent octets of the frame in memory.
7. Set TxFrameBsy to a one (I/O 0x28) which starts the DMA transfer of the frame into the switch. This action causes the TxFrameDone bit to be cleared to a zero if it is not already zero.
8. The CPU is done with this frame and can start any other process, including building the next frame by going to step 1 above (as long as this step 1 is done in a separate memory buffer from the frame that is currently being sent into the switch).
9. When the frame is done being DMA'ed into the switch (TxOctetsLo & TxOctetsHi = 0x000) the NIC will (in this order):
   a) Pad the frame with 0x00 octets until at least 60 octets have been written into the switch.
   b) Generate a good 4 byte CRC and send the CRC into the switch.
   c) Allow the switch to fully receive the frame.
   d) Reset TxAddrLo & TxAddrHi to their previous values if TxKeepAddr = 1 (I/O 0x28).
   e) Clear the TxFrameBsy bit.
   f) Set TxFrameDone to a one (I/O 0x28) generating an optional interrupt.
10. At this point the software can transmit the next frame by going to step 1 or if it already built the next frame it goes to step 4.

The start of the frame is always picked up from the memory location pointed to by TxAddrLo & TxAddrHi (I/O 0x2E & 0x2F). After the first octet is read from memory the 16-bit address (made up of TxAddrLo & TxAddrHi) will be incremented if TxDecAddr = 0 (I/O 0x28). It will be decremented if TxDecAddr = 1. Subsequent frame data can therefore be read either direction in memory.

Each subsequent frame will have its first octet read from the same memory location (i.e., the frames will be transmitted from the same frame memory buffer) if TxKeepAddr = 1 (I/O 0x28). If TxKeepAddr = 0, TxAddrLo & TxAddrHi will not be reset to their original values when the NIC sets TxFrameDone to a one. Instead the subsequent frame will be read from memory right after the last frame. Software needs to prevent the reading of frame data outside its intended buffer in this mode.

If the CPU tries to transfer a frame that is larger than 2044 octets in size, the NIC will not be able to place a good CRC onto the frame going into the switch. In this case the NIC will not append a 4 byte CRC and the frame will be seen at the switch 'as-is'.

6.4.5 Marvell Header & DSA Tag

The switch core supports adding additional information to frames to support switch management. These include the Marvell Header and the DSA Tag. While the IMP NIC works if this extra information is in the Ethernet frames or not, it is recommended that both the Marvell Header and Conditional Ethertype DSA Tag are used.

This additional information may increase the frame size by up to 10 bytes. So the application needs to allocated Tx & Rx buffers that are slightly larger than the frames its will use (1536 bytes will support 1522 byte maximum frame sizes). Refer to the switch's Functional Specification for more information on these features of the switch core.

Table 31: Ethernet NIC Rx Register, I/O Address 0x20

Bits	Field	Type	Description
7	CRCErr	RO	Rx Frame CRC Error Status. 0 = The just read frame had a good CRC. 1 = The just read frame had a bad CRC. This CRC Error bit is valid only after the frame is completely received and stays valid until the time the CPU acknowledges that it is done with the frame by writing a one to RxFrameDone (below). Note: A frame is completely received when RxFrameRdy is set to a one when Ethernet Receive DMA is enabled (RxDMA = 1).
6	RxFrame Done	SC	Acknowledge Current Frame Reception is Done. 0 = The current frame is still being processed. 1 = The current frame is done & the next frame can be received. The CPU needs to acknowledge to the NIC that it is done with the current Rx frame and the NIC can start receiving the next frame. This is accomplished by writing a one to this bit (which self clears).
5:4	Reserved	RES	Reserved for future use.
3	RxKeep Addr	RWR	Receive Keep DMA Address. 0 = Use current RxAddrLo & RxAddrHi for start of next frame. 1 = Use original RxAddrLo & RxAddrHi for start of next frame. As each frame is transferred via DMA, the DMA address pointer (RxAddrLo & RxAddrHi, I/O 0x26 & 0x27) gets incremented or decremented (see RxDecAddr below) by the size of the transferred frame. When the end of the frame is acknowledged by the CPU (setting RxFrameDone, above, to a one) the next DMA transfer can start for the next frame. Setting this bit to a one starts the next frame in the exact same place in IMP memory (i.e., the next frame starts where the last frame started). Clearing this bit to a zero allows the next frame to be placed right after the last frame in memory (i.e., the next frame starts where the last frame ended).
2	RxDec Addr	RWR	Receive Decrement DMA Address. 0 = Increment the Rx DMA Address after each octet transferred. 1 = Decrement the Rx DMA Address after each octet transferred. When Ethernet Receive DMA is enabled (RxDMA = 1, below) this bit determines if RxAddrLo & RxAddrHi are incremented or decremented after each octet transfer. Note: Do not change the contents of this register while the RxDMA is active. Disable the RxDMA first (I/O 0x20) or only change this register while RxFrameRdy is set to a one, i.e., between frames. RxFrameRdy will stay set to a one until the software indicates it is ready to receive the next frame by writing a one to RxFrameDone (I/O 0x20).

Table 31: Ethernet NIC Rx Register, I/O Address 0x20

Bits	Field	Type	Description
1	RxDMA	RWR	Ethernet Receive Direct Memory Access. 0 = Ethernet Rx NIC is disabled. 1 = Enable Rx Frames by DMA into IMP memory from the NIC. When this bit is cleared to a zero the receive path of the IMP's Ethernet NIC is disabled, the Link on the internal switch port is brought down (Port offset 0x00) & any remaining frames in the internal switch port's egress queue will be drained (i.e., discarded). When this bit is set to a one, received frame octets are transferred to IMP memory pointed to by RxAddrLo & RxAddrHi (I/O 0x26 & 0x27). The number of octets transferred will be reflected in RxOctetsLo & RxOctetsHi (I/O 0x24 & 0x25). Note: Do not set this bit to a one until all the other Rx DMA registers are ready to accept their 1st frame.
0	RxFrame Rdy	RO	An Ethernet Frame from the switch is Ready. 0 = The switch does not have any Frames for the IMP 1 = The switch has at least one Frame for the IMP in RAM When the Ethernet Receive DMA is enabled (RxDMA = 1) this bit being a one indicates that the switch frame has been completely transferred to IMP memory via DMA. Once the CPU is ready to receive the next frame it sets the RxFrameDone bit (above) to a one which clears this bit to a zero and starts the receive process for the next frame. This bit being a one can generate an interrupt to the IMP (if enabled – see I/O 0x07).

Table 32: Ethernet Rx Octets Lo Register, I/O Address 0x24

Bits	Field	Type	Description
7:0	RxOctets Lo	RO	Receive Frame Data byte count from the switch. This is the lower 8-bits of a 12-bit counter that counts receive octets. The upper 4-bits of this counter are at I/O 0x25. When an Ethernet frame is Received via DMA (RxDMA = 1) this counter indicates the size of the Ethernet frame that was just transferred into IMP memory by DMA.

Table 33: Ethernet Rx Octets Hi Register, I/O Address 0x25

Bits	Field	Type	Description
7:4	Reserved	RES	Reserved for future use.
3:0	RxOctets Hi	RO	Receive Frame Data byte count from the switch. This is the upper 4-bits of a 12-bit counter fully described above (I/O 0x24).

Table 34: Ethernet Rx DMA Addr Lo Register, I/O Address 0x26

Bits	Field	Type	Description
7:0	RxAddr Lo	RWR	Receive Frame Data (from the switch) DMA Address Pointer. This is the lower 8-bits of a 16-bit address pointer which indicates where the receive octets for the Ethernet frame are to be put into IMP memory. This address is only used for Ethernet Rx DMA operations. The upper 8-bits of this counter are at I/O address 0x27. Note: Do not change the contents of this or the RxAddrHi register below while the RxDMA is active. Disable the RxDMA first (I/O 0x20) or only change these registers while RxFrameRdy is set to a one, i.e., between frames. RxFrameRdy will stay set to a one until the software indicates it is ready to receive the next frame by writing a one to RxFrameDone (I/O 0x20).

Table 35: Ethernet Rx DMA Addr Hi Register, I/O Address 0x27

Bits	Field	Type	Description
7:0	RxAddr Hi	RWR	Receive Frame Data (from the switch) DMA Address Pointer. This is the upper 8-bits of a 16-bit counter fully described above (I/O 0x26).

Note

While DMA transfers are occurring the address in these registers will point to the next location to store the next byte of the frame. If RxKeepAddr = 1 (I/O 0x20) the value in these registers will revert back to their previous start value when RxFrameDone (I/O 0x20) is set to a one.

Table 36: Ethernet NIC Tx Register, I/O Address 0x28

Bits	Field	Type	Description
7	Reserved	RES	Reserved for future use.
6	TxFrame Done	ROC	The Ethernet Frame Transmitter is ready for the next frame. 0 = The Tx portion of the NIC is busy with a frame. 1 = The Tx portion of the NIC is ready to start a new frame. This bit is set to a one whenever the Transmit portion of the NIC is ready to start processing a new frame (i.e., it power-on resets to a one). The transition of this bit from a zero to a one can be enabled to generate an interrupt to the IMP CPU (see I/O 0x07). This bit is cleared to a zero whenever it is read (to clear out the interrupt at the end of the interrupt service routine), or the TxFrameBsy bit, below, is set to a one. This bit is set to a one when the complete frame has been sent into the switch (after TxFrameBsy is cleared a zero). Note: Use the TxFrameBsy bit below as an indication that the Tx NIC is ready for the next frame if interrupts are not being used (as reading this register will clear this bit (& thus the TxFrameDoneInt bit too).
5:4	Reserved	RES	Reserved for future use.
3	TxKeep Addr	RWR	Transmit Keep DMA Address. 0 = Use current TxAddrLo & TxAddrHi for start of next frame. 1 = Use original TxAddrLo & TxAddrHi for start of next frame. As each frame is transferred via DMA, the DMA address pointer (TxAddrLo & TxAddrHi, I/O 0x2E & 0x2F) gets incremented or decremented (see TxDecAddr below) by the size of the transferred frame. When the end of the Tx frame transmission is acknowledged (by the NIC setting of TxFrameDone, above, to a one) the next DMA transfer can start for the next frame. Setting this bit to a one starts the next frame in the exact same place in IMP memory (i.e., the next frame starts where the last frame started). Clearing this bit to a zero allows the next frame to be placed right after the last frame in memory (i.e., the next frame starts where the last frame ended).
2	TxDec Addr	RWR	Transmit Decrement DMA Address. 0 = Increment the Tx DMA Address after each octet transferred. 1 = Decrement the Tx DMA Address after each octet transferred. When Ethernet Transmit DMA is enabled (TxDMA = 1, below) this bit determines if TxAddrLo & TxAddrHi are incremented or decremented after each octet transfer. Note: Do not change the contents of this register while the TxDMA is active. Disable the TxDMA first (I/O 0x28) or only change this register while TxFrameBsy is a zero.

Table 36: Ethernet NIC Tx Register, I/O Address 0x28

Bits	Field	Type	Description
1	TxDMA	RWR	Ethernet Transmit Direct Memory Access. 0 = Ethernet Tx NIC is disabled. 1 = Enable Tx Frames by DMA from IMP memory. When this bit is cleared to a zero the transmit path of the IMP's Ethernet NIC is disabled. When this bit is set to a one, frame octets in IMP memory pointed to by TxAddrLo & TxAddrHi (I/O 0x2E & 0x2F) are transferred to the switch as an Ethernet frame. The number of octets transferred is determined by the values in TxOctetsLo & TxOctetsHi (I/O 0x2C & 0x2D).
0	TxFrame Bsy	RWR	An Ethernet Frame from the IMP is being sent into the NIC. 0 = The Tx portion of data into the NIC is idle. 1 = The Tx portion of the NIC is transmitting a frame into the switch. When the Ethernet Transmit DMA is enabled (TxDMA = 1), setting this bit to a one will cause the TxFrameDone bit, above, to be cleared to a zero. Then the data pointed to by TxAddrLo & TxAddrHi (I/O 0x2E & 0x2F) will be DMA'ed into the switch as an Ethernet frame for the number of octets defined in TxOctetsLo & TxOctets Hi (I/O 0x2C & 0x2D). At the end of this transfer the frame will be padded with zeros (if necessary) and a good 4 byte CRC will be appended to the frame and transferred into the switch. When all this is completed, this bit will self clear to a zero and the TxFrameDone bit (above) will be set to a one (which can be enabled to generate an interrupt).

Caution

Do not write a zero to the TxFrameBsy bit unless it is already zero.

Table 37: Ethernet Tx Octets Lo Register, I/O Address 0x2C

Bits	Field	Type	Description
7:0	TxOctets Lo	RWR	Transmit Frame Data byte count into the switch. This is the lower 8-bits of an 11-bit counter that counts Tx octets. The upper 3-bits of this counter are at I/O 0x2D. When Ethernet Transmit DMA is enabled (TxDMA = 1) this counter indicates the size of the Ethernet frame that is to be transferred into the switch from IMP memory via DMA. If less than 60 octets are transferred, automatic small frame padding with zeros will be done. Do not include the frame's CRC in this count as a good 4 byte CRC will automatically be computed and placed on the end of the frame (after any padding is done). Note: Do not change the contents of this or the TxOctetsHi register below while the TxDMA is active. Disable the TxDMA first (I/O 0x28) or only change these registers while TxFrameBsy is a zero.

Table 38: Ethernet Tx Octets Hi Register, I/O Address 0x2D

Bits	Field	Type	Description
7:3	Reserved	RES	Reserved for future use.
2:0	TxOctets Hi	RWR	Transmit Frame Data byte count into the switch. This is the upper 3-bits of an 11-bit counter fully described above (I/O 0x2C).

Table 39: Ethernet Tx DMA Addr Lo Register, I/O Address 0x2E

Bits	Field	Type	Description
7:0	TxAddr Lo	RWR	Transmit Frame Data (to the switch) DMA Address Pointer. This is the lower 8-bits of a 16-bit address pointer which indicates where the transmit octets for the Ethernet frame are to be read from IMP memory. This address is only used for Ethernet Tx DMA operations. The upper 8-bits of this counter are at I/O address 0x2F. Note: Do not change the contents of this or the TxAddrHi register below while the TxDMA is active. Disable the TxDMA first (I/O 0x28) or only change these registers while TxFrameBsy is a zero.

Table 40: Ethernet Tx DMA Addr Hi Register, I/O Address 0x2F

Bits	Field	Type	Description
7:0	TxAddr Hi	RWR	Transmit Frame Data (to the switch) DMA Address Pointer. This is the upper 8-bits of a 16-bit counter fully described above (I/O 0x2E).

Note

While DMA transfers are occurring the address in these registers will point to the next location to retrieve the next byte for the frame. If TxKeepAddr = 1 (I/O 0x28) the value in these registers will revert back to their previous start value when TxFrameDone (I/O 0x28) is set to a one.

6.5 Fast Copy DMA

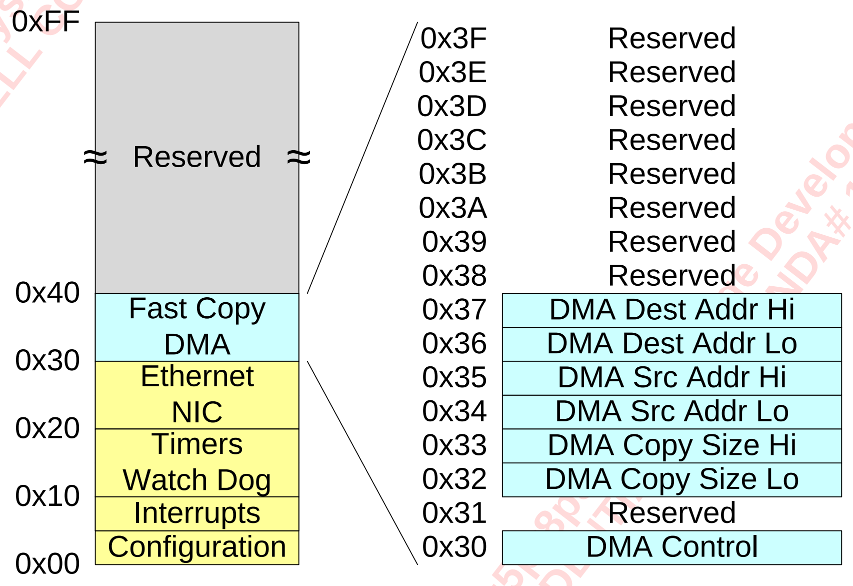
After a boot loader has completed its initialization job and its next job is to complete the fast booting process from the EEPROM (section 4.1.3) or if the CPU simply wants to copy the contents of a portion of the RAM or the EEPROM to another area of RAM, the Fast Copy DMA function can be used.

The IMP Fast Copy DMA function includes the following I/O registers:

- 0x30 = Fast Copy Control

- 0x32 = Fast Copy Transfer Size Lo
- 0x33 = Fast Copy Transfer Size Hi
- 0x34 = Fast Copy Source Address Lo
- 0x35 = Fast Copy Source Address Hi
- 0x36 = Fast Copy Destination Address Lo
- 0x37 = Fast Copy Destination Address Hi

Figure 13: Fast Copy DMA I/O Map



The IMP's Fast Copy DMA function will copy a block of data from the EEPROM or from its RAM. The destination of these transfers is always the IMP's RAM. The selection of which source device to use (RAM or EEPROM) is configured by writing to the FastCopySrcSel bit (I/O 0x30).

When the EEPROM is selected as the source, its full 64K bytes of address space can be accessed. The EEPROM's first byte (at EEPROM address 0x0000) is always accessed using a FastCopySrcAddr (I/O's 0x34 & 0x35) of 0x0000.

The Fast Copy DMA has the following options (all are independently selectable – see I/O 0x30):

- Re-use the last Source Address on the next Fast Copy.
- Re-use the last Destination Address on the next Fast Copy.
- Increment or Decrement the Source Address after each byte.
  - If the source device is the EEPROM the Source Address can only be incremented.
- Increment or Decrement the Destination Address after each byte.
- Cause an interrupt to be generated at the end of the Fast Copy operation.

6.5.1 Fast Copy EEPROM to RAM Performance

The performance of Fast Copy DMA from the EEPROM to RAM (when FastCopySrcSel is zero – I/O 0x30) is optimized to be as fast as possible as block reading of the EEPROM is done. Table 41 and Table 42 list the approximate times required to read data in this mode.

Table 41: Generic Fast Copy DMA EEPROM Read Times

EEPROM Speed (I/O 0x01)	Overhead for the Transfer	Time to Read each Additional Byte
200 KHz	310 uSec	45 uSec
400 KHz	182 uSec	22.5 uSec
500 KHz	156 uSec	18 uSec
600 KHz	150 uSec	15 uSec
800 KHz	150 uSec	11.25 uSec
1000 KHz	150 uSec	9 uSec
1200 KHz	150 uSec	7.5 uSec
1500 KHz	150 uSec	6 uSec

Table 42: Fast Copy DMA EEPROM Read Times for 2048 Bytes

EEPROM Speed (IMP I/O 0x01)	Time to Transfer 2048 Bytes
200 KHz	92,470 uSec
400 KHz	46,262 uSec
500 KHz	37,020 uSec
600 KHz	30,870 uSec
800 KHz	23,190 uSec
1000 KHz	18,582 uSec
1200 KHz	15,510 uSec
1500 KHz	12,438 uSec

Note

The EEPROM shares the same device pins as the LEDs. To get the Fast Copy times listed in Table 41 and Table 42 the LEDs must be disabled. This is accomplished by setting the Skip Columns bits (Port 0 offset 0x16 index 0x00). If the LEDs are enabled add ~250 uSec of Overhead for each LED Column that is enabled (range will be variable between 0 to 500 uSec per LED Column).

6.5.2 Fast Copy EEPROM to RAM with Verification

The code & data read from the EEPROM during a Fast Copy operation can be verified to ensure the data transferred to the IMP's RAM is what was expected. The verification is always enabled in hardware. At the end of all Fast Copy EEPROM to RAM operations, the GoodFcCRC bit (I/O 0x03) will be set to a one if the block of data that was just read from the EEPROM contained a good CRC. The CRC used in Ethernet frames is expected.

It is up to the software to test the GoodFcCRC bit after those Fast Copy operations where a good CRC is expected.

The purpose of Fast Copy EEPROM to RAM with Verification is:

- To ensure data read from the EEPROM is what was expected.
- If reading the EEPROM at faster speeds is failing due to temperature or other issues, the software could choose a slower EEPROM speed. The speed of the EEPROM is controlled with the EEPROM Speed register (I/O 0x01).
- Indicates the code might have been changed without the CRC being updated (see the Code Update Support section 4.1.3.1).

6.5.3 Fast Copy RAM to RAM Performance

The performance of Fast Copy DMA from the RAM to RAM (when FastCopySrcSel is one – I/O 0x30) is optimized to be as fast as possible. The RAM to RAM copy performance works out to be approximately 10.7 ns per byte for a transfer size of 4096 bytes.

This mode is intended for applications where large size RAM to RAM data transfers need to be done as quickly as possible.

6.5.4 Fast Copy EEPROM to RAM with Multiple EEPROMs

If more code space is needed, the Fast Copy EEPROM to RAM operation works with more than one EEPROM connected to the device's EEPROM pins as long as only one EEPROM responds at a time. This is accomplished by connecting the EEPROM chip select pins to the device's GPIO pins – either directly or through inverters.

Which GPIO pins are used to select which EEPROM is device build specific. Just be 100% sure that the EEPROM with the Fast Boot image (section 4.1) is selected after RESETn.

Table 43: Fast Copy Control Register, I/O Address 0x30

Bits	Field	Type	Description
7	FastCopy Ack	RWS	Fast Copy Acknowledge. Writing this bit to a one will clear the DMADone bit (I/O 0x04) and mask the FastCopyDone interrupt. The clearing of the DMADone bit needs to be done in order to re-arm the CommDMAInt interrupt (I/O 0x07) so software needs to toggle this bit to clear the interrupt. Note: If Fast Copy interrupts are not being used this bit should be set to a one.
6	Reserved	RES	Reserved for future use.
5	SrcAddr Keep	RWR	Fast Copy DMA Source Address Keep. 0 = Use current FastCopySrcAddr (Lo & Hi) for next fast copy. 1 = Use original FastCopySrcAddr (Lo & Hi) for next fast copy. As each fast copy operation is done, the fast copy source address pointer (FastCopySrcAddr Lo & Hi, I/O 0x34 & 0x35) get incremented or decremented (See SrcAddrDec below) by the size of the copied data. Setting this bit to a one has each copy operation starting in the exact same source address. Clearing this bit to a zero allows the source address of next copy operation started right after the last copy operation.
4	DstAddr Keep	RWR	Fast Copy DMA Destination Address Keep. 0 = Use current FastCopyDstAddr (Lo & Hi) for next fast copy. 1 = Use original FastCopyDstAddr (Lo & Hi) for next fast copy. As each fast copy operation is done, the fast copy destination address pointer (FastCopyDstAddr Lo & Hi, I/O 0x36 & 0x37) get incremented or decremented (See DstAddrDec below) by the size of the copied data. Setting this bit to a one has each copy operation starting in the exact same destination address. Clearing this bit to a zero allows the destination address of next copy operation started right after the last copy operation.
3	SrcAddr Dec	RWR	Fast Copy DMA Source Address Decrement. 0 = Increment the source address after each octet transferred. 1 = Decrement the source address after each octet transferred. When Fast Copy Busy is enabled (FastCopyBsy = 1, below) this bit determines if FastCopySrcAddr (Lo & Hi – I/O 0x34 & 0x35) are incremented or decremented after each octet transfer. Note: This bit must be zero for EEPROM to RAM Fast Copy operations as the EEPROM can only support incrementing addresses.

Table 43: Fast Copy Control Register, I/O Address 0x30

Bits	Field	Type	Description
2	DstAddr Dec	RWR	Fast Copy DMA Destination Address Decrement. 0 = Increment the destination address after each octet transferred. 1 = Decrement the destination address after each octet transferred. When Fast Copy Busy is enabled (FastCopyBsy = 1, below) this bit determines if FastCopyDstAddr (Lo & Hi – I/O 0x36 & 0x37) are incremented or decremented after each octet transfer.
1	FastCopy SrcSel	RWR	Fast Copy DMA Source Select. 0 = Copy data from EEPROM to RAM 1 = Copy data from RAM to RAM When this bit is cleared to a zero the Fast Copy Source Address (FastCopySrcAddr at indexes 0x34 & 0x35) point to a physical address in the EEPROM. When this bit is set to a one the Fast Copy Source Address (FastCopySrcAddr at indexes 0x34 & 0x35) point to the IMP's RAM. This supports fast memory to memory copies of any size at much higher speeds than what the CPU can do on its own.
0	FastCopy Bsy	SC	Fast Copy DMA Operation Busy. 0 = The Fast Copy function is idle. 1 = The Fast Copy function is busy transferring data. When this bit is set to a one, a fast copy operation is started using the parameters in the Fast Copy registers (0x30 to 0x37). This bit will automatically clear to a zero when the fast copy operation is done. The transition of this bit from a one to a zero can be used to generate an interrupt to the IMP CPU. This bit can be used to abort the current fast copy operation by clearing this bit to a zero.

Table 44: Fast Copy Transfer Size Lo Register, I/O Address 0x32

Bits	Field	Type	Description
7:0	FastCopy SizeLo	RWR	Fast Copy Size[7:0] of a 16-bit register. These bits define the size in bytes of a Fast Copy DMA transfer. This register must be configured prior to the enabling a transfer (i.e., before FastCopyBsy is set to a one – index 0x30 above). These bits will decrement during the Fast Copy transfer and the transfer will stop as soon as these bits reach a value of zero (FastCopyBsy will be cleared to a zero). At this point these bits will be restored to their previous value so they are ready for the next Fast Copy operation. The upper 8-bits of the Fast Copy Size are at I/O 0x33, below.

Table 45: Fast Copy Transfer Size Hi Register, I/O Address 0x33

Bits	Field	Type	Description
7:0	FastCopy SizeHi	RWR	Fast Copy Size[15:8] of a 16-bit register. The description of these bits can be found at I/O 0x32, above, where the lower 8-bits of this register are defined.

Table 46: Fast Copy Source Address Lo Register, I/O Address 0x34

Bits	Field	Type	Description
7:0	FastCopy SrcAddrLo	RWR	Fast Copy Source Address[7:0] of a 16-bit register. These bits define the source address of a Fast Copy DMA transfer. This register must be configured prior to the enabling a transfer (i.e., before FastCopyBsy is set to a one – index 0x30 above). These bits will decrement or increment during the Fast Copy transfer depending upon the setting of the SrcAddrDec bit (index 0x30 above). At the end of the Fast Copy transfer these bits will be restored to their last written value if the SrcAddrKeep bit is set to a one (index 0x30 above). This allows back-to-back copies from the same area in memory. The upper 8-bits of the Fast Copy Source Address are at I/O 0x35, below. Note: If the source of a Fast Copy operation is the EEPROM (FastCopySrcSel = 0 in index 0x30) the source address must always be based on the physical address of the EEPROM (where address 0x0000 of the EEPROM is at Source Address 0x0000). Note: The Source Address must not start in nor pass through the top 4K addresses of the IMP's address space unless the Fast Copy source is the EEPROM (FastCopySrcSel, I/O 0x30, is cleared to a zero). This is where the switch I/O registers are located and using them as a source address will result in undefined results.

Table 47: Fast Copy Source Address Hi Register, I/O Address 0x35

Bits	Field	Type	Description
7:0	FastCopy SrcAddrHi	RWR	Fast Copy Source Address[15:8] of a 16-bit register. The description of these bits can be found at I/O 0x34, above, where the lower 8-bits of this register are defined.

Table 48: Fast Copy Destination Address Lo Register, I/O Address 0x36

Bits	Field	Type	Description
7:0	FastCopy DstAddrLo	RWR	Fast Copy Destination Address[7:0] of a 16-bit register. These bits define the destination address of a Fast Copy DMA transfer. This register must be configured prior to the enabling a transfer (i.e., before FastCopyBsy is set to a one – index 0x30 above). These bits will decrement or increment during the Fast Copy transfer depending upon the setting of the DstAddrDec bit (index 0x30 above). At the end of the Fast Copy transfer these bits will be restored to their last written value if the DstAddrKeep bit is set to a one (index 0x30 above). This allows back-to-back copies to the same area in memory. The upper 8-bits of the Fast Copy Destination Address are at I/O 0x37, below. Note: The Destination Address must not start in nor pass through the top 4K addresses of the IMP's address space. This is where the switch I/O registers are located and using them as a destination address will result in undefined results.

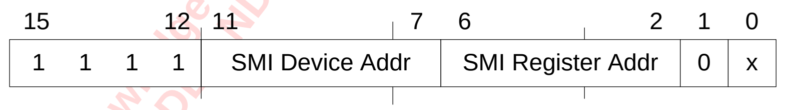
Table 49: Fast Copy Destination Address Hi Register, I/O Address 0x37

Bits	Field	Type	Description
7:0	FastCopy DstAddrHi	RWR	Fast Copy Destination Address[15:8] of a 16-bit register. The description of these bits can be found at I/O 0x36, above, where the lower 8-bits of this register are defined.

7 Memory Mapped Switch Registers

The top 4K bytes of the IMP CPU's memory map accesses the switch core's registers (see section 3). The CPU's 16 address lines are decoded to access the desired Switch Register as follows:

Figure 14: CPU Address to Switch Register Mapping



- The upper 4 bits (bits 15:12) must always be 1's (the top 4K addresses).
- Bits 11:7 select the SMI Device Address (1 of 32 possible devices).
- Bits 6:2 select the SMI Register Address (1 of 32 possible registers for the selected device).
- Bit 1 is a zero as the current Switch Registers are 16-bit wide registers.
- Bit 0 is used to define which byte of the selected 16-bit register is being accessed.

Please refer to the Switch Functional Specification document that describes the switch registers description for an understanding of how the SMI Device Addr and SMI Register Addr fields are used to select specific Switch Registers.

Important: All Switch Registers are 16-bits and must be read and written 16-bits at a time. Since the IMP is an 8-bit CPU, the following procedures must be used when accessing the Switch Registers:

Caution When interrupts are used, care is needed to ensure the CPU can complete the 16-bit data reads & writes to the switch registers before an interrupt is serviced.

- Writing to a Switch Register:
  - Write the desired data for Switch Register bits 7:0 to <SMI Device Addr><SMI Register Addr> with address bits 1:0 cleared to zero. This holds the lower 8-bits ready in the interface without writing them to the Switch Register.
  - Write the desired data for Switch Register bits 15:8 to <SMI Device Addr><SMI Register Addr> with address bit 1 cleared to zero and address bit 0 set to a one (i.e., the next higher byte address from the address used above). This writes the full 16-bits (including the previously saved lower 8-bits) to the Switch Register.

- Reading from a Switch Register:
  - Read the desired data for Switch Register bits 7:0 from <SMI Device Addr><SMI Register Addr> with address bits 1:0 cleared to zero. This causes the 16-bit switch register to be read in the interface, passing the lower 8-bits to the CPU while holding the upper 8-bits in the interface.

- Read the desired data for Switch Register bits 15:8 to <SMI Device Addr><SMI Register Addr> with address bit 1 cleared to zero and address bit 0 set to a one (i.e., the next higher byte address from the address used above). This reads the held data without reading the Switch Register again. This sequence is critical as some bits are cleared after read from the Switch Registers and the CPU must get all 16-bits of the data from this read.

Software cannot read the upper 8-bits of a 16-bit register without first reading the lower 8-bits, and it cannot write the upper 8-bits of a 16-bit register without first writing the lower 8-bits.

8 Code Debugging Support

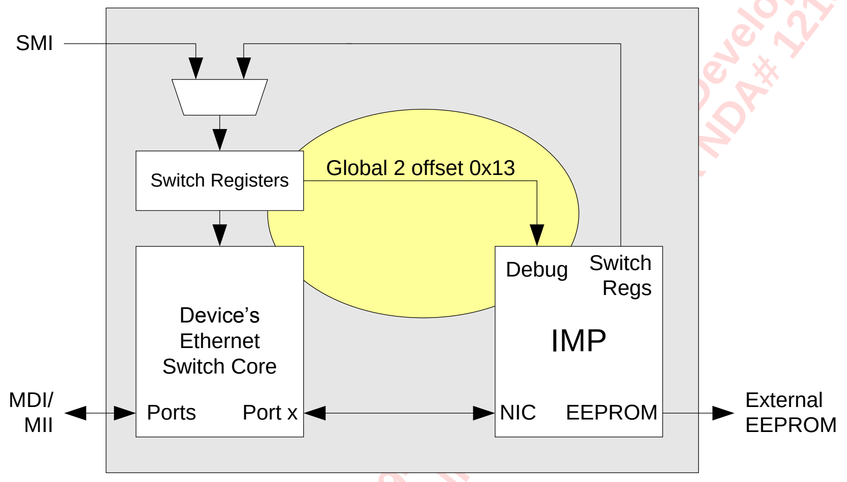
Since all code needs to be debugged, the device supports extensive debugging capabilities in both hardware and software. The software portion runs under Windows and it is called IMPGUI. The software portion is not discussed here, but it uses all of the hardware features that are discussed below.

The device supports JTAG-like hardware debug capabilities via SMI (System Management Interface) including:

- Stop/Run/Reset the CPU
- Single step or step 'n' instructions
- 4 hardware break points – by range or direct match, memory or I/O
- Dump all of the CPU's internal registers
- Dump/modify all of the CPU's I/O registers
- Dump/modify all of the CPU's RAM

The SMI interface is used to access the switch's registers. A set of these switch registers access the debug features of the device. All of these switch registers are at Global 2 offset 0x13 (the area highlighted in yellow in Figure 15).

Figure 15: Device Top Level Block Diagram – Debug Support



The debug registers and controls are NOT in the IMP CPU's address space (as they are used to debug the CPU). They are in the Switch Register space so an external device using the MDC/MDIO register access method (SMI) can be used to debug the IMP.

8.1 Accessing the CPU’s Internal Registers

The CPU has the following internal registers which are accessible using the debug Interface:

Figure 16: CPU’s Internal Registers

Main Register Set

7 ... 0	7 ... 0
A (Index 0x10)	F (Index 0x11)
B (Index 0x12)	C (Index 0x13)
D (Index 0x14)	E (Index 0x15)
H (Index 0x16)	L (Index 0x17)

Alternate Register Set

7 ... 0	7 ... 0
A’ (Index 0x18)	F’ (Index 0x19)
B’ (Index 0x1A)	C’ (Index 0x1B)
D’ (Index 0x1C)	E’ (Index 0x1D)
H’ (Index 0x1E)	L’ (Index 0x1F)

Special Purpose Registers

15 ... 0	
Index Register (IX) (Indexes 0x20 & 0x21)	
Index Register (IY) (Indexes 0x22 & 0x23)	
Stack Pointer (SP) (Indexes 0x24 & 0x25)	
Program Counter (PC) (Indexes 0x26 & 0x27)	
I (Index 0x28)	R (Index 0x29)

All of the CPU’s internal registers are readable at any time (all of the indexes listed below are in the Global 2 offset 0x13 SMI register space):

- The Main Register set is readable from indexes 0x10 to 0x17.
- The Alternate Register set is readable from indexes 0x18 to 0x1F.
- The Special Purpose registers are readable from indexes 0x20 to 0x29.
- The CPU Status is readable from index 0x09.

Note

The CPU’s registers are read only in the hardware. But the Marvell debug tool (IMPGUI) supports writing to the CPU’s registers (while the CPU is stopped). This support is one reason why the IMP’s RAM from addr 0x00F0 to 0x00FF cannot be used by applications.

8.2 Controlling the CPU & its RAM

The IMP's CPU can be controlled by the IMP Control registers at indexes 0x08 to 0x0F. The sections below cover the features of the IMP Control registers.

Figure 17: Debug Control Registers

	7 0
Index 0x08	IMP Control
Index 0x09	IMP Status
Index 0x0A	Addr Lo
Index 0x0B	Addr Hi
Index 0x0C	WrData
Index 0x0D	RdData
Index 0x0E	SSCount
Index 0x0F	Debug Control

8.2.1 Stopping the CPU

The CPU can be stopped by performing a CPU Stop IMPOp command (index 0x08). Stop will stop the CPU as soon as it is safe to do so. If a Stop command is issued while the CPU is Halt'ed (index 0x09), the Stop will take effect on the 1st instruction following the Interrupt Acknowledge cycle that woke up the CPU. This 'pending' Stop can be canceled by issuing a Run command (section 8.2.2). In all cases the CPU will be Stopped on the next Op Code fetch that occurs.

Note

While the CPU is stopped it is placed into constant wait states until a CPU Run IMPOp command is done. During this time interrupt processing is stopped (in fact both interrupts, NMI and INT are masked while the CPU is stopped).

8.2.2 Starting the CPU

The CPU can be started (assuming it is stopped) by performing a CPU Run IMPOp command (index 0x08). Run will restart the CPU running at the current Program Counter's location or cancel a Stop request that has not occurred yet (section 8.2.1). The Program Counter's location can be modified before a Run by using the Examine (section 8.2.3), Examine Next or Deposit Next commands (section 8.2.6). A Run command will allow the CPU to continue processing after a Break Point (section 8.3) as well.

8.2.3 Setting the CPU's Program Counter

The CPU's Program Counter can be modified (assuming the CPU is stopped) by performing a CPU Examine IMPOp command (index 0x08). Examine will move the CPU's Program Counter to the address defined in the AddrHigh and AddrLow registers (indexes 0x0A & 0x0B). The CPU will then be stopped at this new address and the contents of that memory location will be readable in the RdData register (index 0x0D). If the CPU is stopped in the middle of executing an instruction (OpCodeFetch = 0 in index 0x08) this Examine does nothing (see Single Step – section 8.2.4 to fix this situation). A Run command after a (successful) Examine will cause the CPU to start the execution of code at the new address.

8.2.4 Single Stepping the CPU

The CPU's program can be executed one instruction[2] step at a time or N steps (assuming the CPU is stopped) by performing a CPU Single Step IMPOp command (index 0x08). If the CPU is at the beginning of a new instruction (OpCodeFetch = 1 in index 0x08) the Single Step command will let the CPU execute N instructions as defined in the SSCount register (index 0x0E). If SSCount is at its default value of 0x01, then the CPU executes one instruction per Single Step command.

If the CPU is stopped in the middle of an instruction (OpCodeFetch = 0 in index 0x08) which can occur if a Break Point (section 8.3) is set in the middle of an instruction, then the Single Step command will complete the current instruction only, regardless of the value of the SSCount register.

If a Break Point (section 8.3) occurs while Single Step is executing N instructions, the Break Point will terminate the Single Step operation.

8.2.5 Resetting the CPU

The CPU and all of its peripherals can be reset by performing a CPU Reset IMPOp command (index 0x08). This command resets all of the CPU's internal registers and all if its I/O registers back to their default values (these debug index registers are not part of the CPU's I/O registers as they are in the SMI address space – so they are not reset).

The CPU's RAM will NOT be initialized, however, by this reset command if the CPU is in Stop mode (section 8.2.1). If the CPU was Running (section 8.2.2) it will fetch its next instruction at address 0x0000.

To perform a reset without resetting the CPU's peripherals and I/O registers, Stop the CPU (section 8.2.1), Examine address 0x0000 (section 8.2.3), and then Run the CPU (section 8.2.2).

8.2.6 Accessing the CPU's RAM

The CPU RAM is accessible using the following set of IMPOp commands (index 0x08). All of these commands require that the CPU be stopped (section 8.2.1) at the beginning of execution of the next instruction (OpCodeFetch = 1 in index 0x08 – see section 8.2.4 if this is not the case):

- Examine – Examine moves the CPU's Program Counter to the address pointed to by the AddrLo & AddrHi registers (indexes 0x0A & 0x0B) and the data found at that address is readable in the RdData register (index 0x0D) – see section 8.2.3).
- Examine Next – Examine Next will move the CPU's Program Counter up by one byte. The CPU will then be stopped at this new address and the contents of that memory location will be readable in the RdData register below (index 0x0D).
- Deposit – Deposit causes the contents of the WrData register (index 0x0C) to be written to the memory currently pointed to by the CPU's Program Counter. After this write takes place the RdData register (index 0x0D) will reflect the new memory contents (if the write was to RAM).
- Deposit Next – Deposit Next is a simple combination of the Deposit command followed by the Examine Next command.

Note

The methods described above for accessing the CPU's RAM can be used to download code to a CPU – even to a CPU that never booted. See Slow Boot (section 4.2).

[2] CPU instructions that contain multiple byte Op Code bytes are counted as one instruction.

8.2.7 Debugging an Idle CPU

When the CPU is not running due to:

- The CPU never booted, i.e., the RunState (index 0x08) is 0x3, or
- The CPU shut itself down, i.e., the Total IMP Stop bit (index 0x4) is 1, or
- The EEPROM TimedOut (index 0x04) is a 1 (even if EEPROM Read is a 1).

The CPU and its peripherals can be accessed using this debug interface as follows:

- Issue a Stop IMPOp (index 0x08). This will set the RunState (index 0x08) to 0x1.
- Issue a Reset IMPOp (index 0x08). This will reset the IMP CPU and its peripherals giving control to the debugger.
- Restart the CPU by:
  - Downloading the desired code into RAM using the Examine, Deposit & Deposit Next IMPOps (index 0x08).
  - Run the code by using the Examine & then the Run IMPOps (index 0x08).

All of these individual IMPOps are discussed in more detail in this section (section 8.2) above.

Note

If the EEPROM times out when reading other than the 1st byte of the EEPROM (i.e., there is not EEPROM installed), the Stop IMPOp may not be able set the RunState to 0x1. This is an indication of an intermittent EEPROM interface or EEPROM device.

8.3 Break Points

The device support four programmable break points to facilitate code debug. Each break point has the following features:

- Support a range of addresses defined by a Low address limit and a High address limit
- An exact match is defined if both High and Low addresses are the same
- Supports breaking on a memory fetch or an I/O fetch
- Supports breaking on a read, write, either or neither operation
- Supports a 32-bit counter that counts once on each access to the define compare range

Figure 18: Break Point Registers

	7 … 0
Index 0x30	Break 0 Control
Index 0x31	Break 1 Control
Index 0x32	Break 2 Control
Index 0x33	Break 3 Control
Index 0x38	Break Int Status

	15 … 0
Indexes 0x40 & 41	Break 0 Low Addr Limit
Indexes 0x42 & 43	Break 0 High Addr Limit
Indexes 0x44 & 45	Break 0 Counter[15:0]
Indexes 0x46 & 47	Break 0 Counter[31:16]
Indexes 0x48 & 49	Break 1 Low Addr Limit
Indexes 0x4A & 4B	Break 1 High Addr Limit
Indexes 0x4C & 4D	Break 1 Counter[15:0]
Indexes 0x4E & 4F	Break 1 Counter[31:16]
Indexes 0x50 & 51	Break 2 Low Addr Limit
Indexes 0x52 & 53	Break 2 High Addr Limit
Indexes 0x54 & 55	Break 2 Counter[15:0]
Indexes 0x56 & 57	Break 2 Counter[31:16]
Indexes 0x58 & 59	Break 3 Low Addr Limit
Indexes 0x5A & 5B	Break 3 High Addr Limit
Indexes 0x5C & 5D	Break 3 Counter[15:0]
Indexes 0x5E & 5F	Break 3 Counter[31:16]

8.3.1 Enabling Break Points

Enable Break Points using the following procedure:

- Configure the Break Point’s Low Addr Limit.
- Configure the Break Point’s High Addr Limit.
- Configure the Break Point conditions in the Break Point’s Control register.

The Break Point’s Control register is used to determine the Break Point’s mode as follows:

- Use BkPtXMem to select between I/O accesses (a zero) or memory accesses (a one).
- Use BkPtXRd to select between don’t break (a zero) or break on a read (a one).
- Use BkPtXWt to select between don’t break (a zero) or break on a write (a one).
- Use ClrCtrX to clear that breaks point’s counter
- Use BkXIntEn to have an active break point generate an INTn interrupt.

Note Break point interrupts vector through the device's EEInt. Reading index 0x03 will indicate that it is a Break interrupt and then index 0x38 will indicate the active Break Point number that caused the interrupt.

The Break Point's mode allows each break point to monitor just memory or just I/O address (only the lower 8-bit of the Limit registers are used for I/O address monitoring), to break on reads only, writes only, either reads or writes, or not to break at all.

When an active break point occurs, the source of the break point is indicated (index 0x38) and the CPU is Stopped (section 8.2.1). The Break Point's Counter is incremented as well.

8.3.2 Program Performance Tuning

The last Break Point mode of not breaking at all (i.e., an non-active Break Point) can be used for program performance tuning as follows:

- Stop the CPU (section 8.2.1).
- Set the various Break Point Limits to the sections of code that need to be performance tuned.
- Clear all the Break Point counters and restart the CPU (section 8.2.2).
- Stop the CPU again after a while or use one of the Break Points to stop the CPU
- At this point the non-active Break Point Counters will indicate how many times their sections of code were executed.

8.3.3 Clearing the Break Point Counters

There are two ways to clear each Break Point's counters.

- An individual Break Point counter is cleared when a one is written to that Break Point's ClrCtrX register. The ClrCtrX bits are self clearing bits contained inside each Break Point's Control register (index 0x30 to 0x33).
- All the Break Point counters are cleared when a CPU Clear All Break Point Counters command is issued (index 0x08).

8.4 Controlling the CPU's I/O

The IMP's I/O can be accessed using index 0x2F and indexes 0x70 to 0x7F. Index 0x2F defines the upper 4 address bits of the I/O Page to access. Indexes 0x70 to 0x7F then access the 16 I/O addresses of that I/O Page.

Reading of the any I/O register can be done at any time. But writing to the I/O register can only be done while the CPU is Stopped (section 8.2.1).

Note

Reading of an I/O register using the debug interface will not clear any Clear on Read bits. These bits can only be cleared when the IMP CPU does the actual reading of the register.

8.5 Communications Port

The debug interface supports a UART-like communications port connected to the CPU. This Comm port can be used like a UART to send out debug messages and to send commands to the IMP CPU. Marvell's IMPGUI debug software supports this usage with a terminal type window.

Implementations that have an external processor connected to this device can use the comm port to send commands &/or interrupts between the external processor and the IMP CPU as interrupts are supported in both directions.

The Comm port supports 10 bits of data to & from the CPU with associated busy bits. The 10 bits of data can be used in any way. The 10-bits are separated with 8-bits in a data register and 2-bits in a status register. One use case for these 10-bits is to use 8-bits for data and 2-bits to indicate what kind of data the other 8-bits are.

The Comm registers are at indexes 0x00 to 0x03.

Figure 19: Comm Registers

	7 … 0
Index 0x00	Comm Status
Index 0x01	Comm Read Data
Index 0x02	Comm Write Data
Index 0x03	Interrupt Source

8.5.1 Comm Status

The Comm Status contains:

- WtFlags – two of the ten bits that are to be sent to the IMP CPU.
- WtDataBsy – an indicator that the IMP CPU has read out the previous data.
- RdIntEn – interrupt enable to the external processor when the IMP writes Comm data.
- RdFlags – two of the ten bits that are written by the IMP to its Comm port.
- RdDataRdy – an indicator that the IMP CPU has written data to its Comm port.

8.5.2 Comm Read Data

The Comm Read data register contains eight of the 10 bits that are written by the IMP to its Comm port. When the IMP writes data (to its WrData register – section Table 9) that data shows up in this register and the RdDataRdy bit in the Comm Status register, above, will be set to a one. This can be used to generate an interrupt on the device's INTn pin (section 8.5.4).

8.5.3 Comm Write Data

The Comm Write data register contains eight of the 10 bits that are to be sent to the IMP on its Comm port. Data written here can be read by the IMP (from its Rdata register – section Table 8) Writing to this register causes the WtDataBsy bit in the Comm Status register, above, to be set to a one.

8.5.4 Interrupt Source

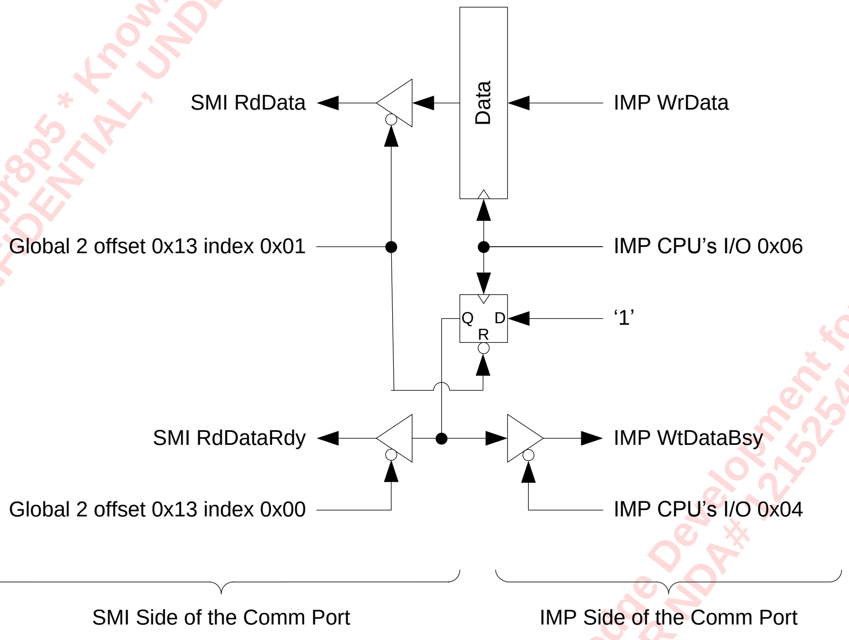
When Comm interrupts are being used, the external CPU can determine that the cause of the interrupt was from the Comm port as that register's RdRdy bit will be set to a one.

8.5.5 Comm Port Structure

The Comm port is a connection between the IMP CPU's I/O register space and the device's SMI register space. Registers and register bits are named 'read' or 'write' and 'busy' and 'ready' relative to the interface that is accessing that register. In Figure 20, the IMP's WrData register becomes the SMI's RdData register. When the IMP writes data to its WrData register the IMP 'sees' a WtDatBsy but the SMI 'sees' and RdDataRdy bit. When the SMI side reads out the RdData, it clears both the SMI's RdDataRdy bit and the IMP's WrDataBsy bit.

The reverse direction is not shown, but it is the same structure.

Figure 20: Comm Port Schematic


SMI RdData
Data
IMP WrData
Global 2 offset 0x13 index 0x01
IMP CPU's I/O 0x06
Q
D
R
'1'
SMI RdDataRdy
IMP WtDataBsy
Global 2 offset 0x13 index 0x00
IMP CPU's I/O 0x04
SMI Side of the Comm Port
IMP Side of the Comm Port

8.6 Debug Registers

The debug registers are accessed using the device's SMI interface using an indexed register approach. The base register is at Global 2 offset 0x13.

The upper 8-bits of this 16-bit register is used to indicate the specific 8-bit index register's address and if this registers is being written to or read from.

The lower 8-bits is the data to be written to the register or the data that was read from it.

8.6.1 Writing to an Index Register

Writing an 8-bit index register requires a single SMI write operation to Global 2 offset 0x13 with:

- Bit 15, the Update bit, set to a one.
- Bits 14:8, the Pointer bits, set to the index value of the register to write, and
- Bits 7:0, the Data, set to the data value to write to the register.

8.6.2 Reading from an Index Register

Reading an 8-bit index register requires two SMI operations to Global 2 offset 0x13. The first operation is an SMI write with:

- Bit 15, the Update bit, set to a zero.
- Bits 14:8, the Pointer bits, set to the index value of the register to read, and
- Bits 7:0, the Data, set to any value as these bits are ignored.

The second operation is an SMI read from Global 2 offset 0x13 where:

- Bit 15, the Update bit, will be zero.
- Bits 14:8, the Pointer bits, will be set to the index value of the register that was read, and
- Bits 7:0, the Data, set to the value read from the above index.

8.6.3 The Debug Registers

A complete description of the IMP debug registers follows. They are called Comm/Debug as these registers support both the Communications port (section 8.5) and the IMP debug registers (sections 8.1, 8.2, 8.3 and 8.4).

Table 50: IMP Comm/Debug Register, Offset: 0x13 or decimal 19 – G2

Bits	Field	Type	Description
15	Update	SC	Update Data. When this bit is set to a one the data written to bits 7:0 will be loaded into the IMP Communications/Debug register selected by the Pointer bits below. After the write has taken place this bit self clears to zero.
14:8	Pointer	RWS to 0x03[3]	Pointer to the desired octet of IMP Communications/Debug. These bits select one of the possible IMP Communications/Debug registers for both read and write operations. A write operation occurs if the Update bit is a one (the IMP Communications/Debug registers can be written to by writing this register in a single operation). Otherwise a read of the current Pointer occurs and the data found there is placed in the Data bits below (the IMP Communications/Debug register can be read by first writing to this register, with Update = 0, and then reading this register). The Pointer bits are used to access the Index registers as follows[4]: 0x00 to 0x04: Communication Interface to/from the CPU The following registers are also supported, and are intended to used in downloading and debugging IMP code: 0x05 to 0x07: Reserved for future use 0x08 to 0x0F: Debug Control & Status 0x10 to 0x1F: CPU's Main and Alternate Registers (always readable) 0x20 to 0x29: CPU's Special Purpose Registers (always readable) 0x2A to 0x2E: Reserved for future use 0x2F : IMP I/O Page Register 0x30 to 0x33: Break Point Control 0x38 : Break Point Status 0x39 to 0x3F: Reserved for future use 0x40 to 0x47: Break Point 0 0x48 to 0x4F: Break Point 1 0x50 to 0x57: Break Point 2 0x58 to 0x5F: Break Point 3 0x60 to 0x6F: Reserved for future use 0x70 to 0x7F: IMP I/O Registers (always readable)
7:0	Data	RWR	IMP Communications/Debug data. This is the data read or written to the register pointed to by the Pointer bits above.

The individual registers accessed by the IMP Communications & Debug register are described below and are indicated with a light green background heading color.

[3] The Pointer resets to 0x03 so that the sources from the EEInt (Global 1 offset 0x00) can be accessed quickly.
[4] These registers are documented here, but they are also documented in the device's Functional Specification.

Table 51: Comm Status, Index: 0x00 of G2 Offset 0x13

Bits	Field	Type	Description
7	CPU Stopped	RO	CPU clocks are Stopped. 0 = CPU is running 1 = CPU clocks are stopped This bit is set to a one whenever the IMP CPU's clocks are stopped. The clocks can be stopped by the IMP CPU writing to its I/O register 0x02 (by setting the Total IMP Stop bit to a one), or by never starting in the first place (i.e., no EEPROM is present or the EEPROM's was not indicated to be used by the IMP CPU), or by executing a Halt instruction (the clocks will be stopped until the next unmasked interrupt becomes active).
6:5	WtFlags	RWR	Write Data Flags. These two bits can be used for any purpose of communicating extra information between this SMI interface and the IMP. For example, one of these bits can be used to indicate if the WrData contains a Command or Data.
4	WtDataBsy	RO	Write Data to the IMP is Busy. 0 = The WtData register is available for new information 1 = The WtData register is busy with data for the IMP This bit is automatically set to a one whenever the SMI's WtData register (Index 0x02) is written to by this interface. It automatically clears to a zero whenever the IMP reads this data out of its register (I/O 0x05). This bit being a one can be used to interrupt the IMP (via its Comm/DMAInt at I/O 0x07).
3	RdIntEn	RWR	Enable the RdData Interrupt from the IMP. 0 = The RdData Interrupt from the IMP is not active 1 = The RdData Interrupt from the IMP is active Setting this bit to a one enables the IMP to drive this device's INTn interrupt pin through the EEInt interface (Global 1 offset 0x00 & Index 0x03 below). The device's interrupt pin will go low (if EEInt is enabled in Global 1 offset 0x04) when this bit is set to a one and whenever the RdDataRdy bit below is also a one. When the SMI's RdData register (Index 0x01) is read by this interface this interrupt will be cleared until the next write to the SMI's RdData register by the IMP (when the IMP writes to its I/O 0x06).
2:1	RdFlags	RO	Read Data Flags. These two bits can be used for any purpose of communicating extra information between the IMP and this SMI interface. For example, one of these bits can be used to indicate if the RdData contains a Command or Data.

Table 51: Comm Status, Index: 0x00 of G2 Offset 0x13

Bits	Field	Type	Description
0	RdDataRdy	RO	Data from the IMP is ready to be Read. 0 = Data in the RdData register is not valid 1 = Data in the RdData register is valid and should be read This bit being set to a one indicates that the IMP has placed a byte of data in the SMI's RdData register (Index 0x01). When the SMI RdData register is read by this interface, this bit will automatically clear to zero. This bit being a one can be used to drive this device's INTn interrupt pin if the RdIntEn bit above is set to a one.

Table 52: Comm Read Data, Index: 0x01 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	RdData	RO	Read Data from the IMP. This register contains the byte of data the IMP has sent to this interface (when the IMP writes to its I/O 0x06). It contains valid data whenever the RdDataRdy bit (Index 0x00) is set to a one. When this SMI RdData register is read, the RdDataRdy bit will be cleared to a zero indicating to the IMP that this register is ready for the next byte of data. The IMP can cause an EEInt interrupt to occur whenever valid data from the IMP is contained in this register. See Index 0x00.

Table 53: Comm Write Data, Index: 0x02 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	WrData	RWR	Write Data to the IMP. This register contains the byte of data this SMI interface wants to send to the IMP. Writing to this register automatically sets the SMI WtDataBsy bit which will not be cleared until the IMP reads this register's contents (by reading its I/O 0x05). Therefore, this register should not be written to again without first checking that the SMI WtDataBsy bit is zero. If a write is done with the WtDataBsy bit being a one, the write will occur, overwriting the contents of this register. This interface can cause an interrupt to occur to the IMP CPU whenever valid data to the IMP is contained in this register. See RdIntEn at IMP I/O 0x04.

Table 54: EE Interrupt Source, Index: 0x03 of G2 Offset 0x13

Bits	Field	Type	Description
7:5	Reserved	RES	Reserved for future use.
4	EEPROM Done	ROC	EEPROM operation Done. This bit gets set to a one whenever the EEBusy bit (Global 2 offset 0x14) transitions from a one to a zero. This bit self clears when read.
3	Break	ROC	IMP Break. This bit is set to a one whenever a Break occurs for a Break Point that has its BkXIntEn bit set to a one (where X is the number of the Break Point – at Indexes 0x30 to 0x33). This bit self clears when read. Which Break Point is active (and therefore the one that caused this bit to be set to a one) can be seen by reading Index 0x38.
2	RdRdy	RO	Read Data Ready from the IMP. This bit is set to a one whenever the SMI's RdDataRdy bit is a one and the RdIntEn bit is a one (both bits are in Index 0x00 above).
1	RLDone	ROC	Register Loader Done. This bit is set to a one whenever the Register Loader was given control by the EEPROM and the Register Loader has executed a HALT instruction. This bit self clears when read.
0	NoEEP	ROC	No EEPROM. This bit is set to a one whenever the first byte read from the EEPROM was 0xFF, meaning no EEPROM was present. This bit self clears when read.

Note

If any of these bits in this register are a one the EEInt in Global 1 offset 0x00 will be set to a one, which in turn will drive the device's INTn pin low if the EEIntEn in Global 1 offset 0x04 is also set to a one.

Table 55: IMP State, Index: 0x04 of G2 Offset 0x13

Bits	Field	Type	Description
7:4	Reserved	RES	Reserved for future use.
3	EEPROM Read	RO	EEPROM Read in process. This bit is set to a one whenever the EEPROM is performing a read operation. This bit will be cleared to zero once the EEPROM read completes or if the EEPROM read timed out. If an EEPROM read time out occurred on other than the 1st EEPROM read, the timed out EEPROM read will be re-tried (and this bit will be set to a one again).
2	EEPROM TimedOut	RO	EEPROM Timed Out. This bit is set to a one whenever any EEPROM read operation timed out. This bit will be cleared to a zero whenever a Reset IMPOp occurs (Global 2 offset 0x13 index 0x08).
1	Total IMP Stop	RO	Total IMP Stop. This bit is a copy of the Total IMP Stop bit at IMP I/O address 0x02 except this bit is read only. This bit will be cleared to a zero whenever a Reset IMPOp occurs (Global 2 offset 0x13 index 0x08).
0	No EEPROM	RO	No EEPROM. This bit is a copy of the NoEEP bit in index 0x03, except that this bit does not clear on read. This bit will be cleared to a zero whenever a Reset IMPOp occurs (Global 2 offset 0x13 index 0x08).

Table 56: IMP Control, Index: 0x08 of G2 Offset 0x13

Bits	Field	Type	Description
7:6	RunState	RO	The IMP CPU's running status as follows: 00 = The CPU is Running normally and executing code 01 = The CPU is Stopped being held in wait states by the Debugger 10 = The CPU is Running normally but Halt'ed 11 = The CPU never booted – no EEPROM or it wasn't for the CPU The CPU will stop running only when a Stop IMPOp is executed (below) or if an enabled Break Point occurs (Indexes 0x30 to 0x33). It will start running again when a Run IMPOp is executed or it will run for one or 'n' instructions when a Single Step IMPOp is executed. Most IMPOps, below, can be done only while the CPU is stopped.
5	OpCode Fetch	RO	Op Code Fetch by the CPU core. 0 = The CPU is not accessing the 1st byte of an Op Code Fetch 1 = The CPU is accessing the 1st byte of an Op Code Fetch When this bit is set to a one, it indicates that the CPU is fetching the 1st byte of an instruction from memory. Some IMPOps, below, can only be done while the CPU is performing an Op Code Fetch.
4	Reserved	RES	Reserved for future use.
3:0	IMPOp	RWR	Integrated Micro Processor Debug Operation. This register contains the desired IMP Debug Operation. Each operation occurs once each time this register is written to. The supported operations are: 0x0 = No Operation 0x1 = Run - only works if the CPU is Stopped or if a Stop was requested 0x2 = Stop - only works if the CPU is Running 0x3 = Examine - only works if the CPU is Stopped at OpCodeFetch 0x4 = Examine Next - only works if the CPU is Stopped at OpCodeFetch 0x5 = Deposit - only works if the CPU is Stopped at OpCodeFetch 0x6 = Deposit Next - only works if the CPU is Stopped at OpCodeFetch 0x7 = Single Step - only works if the CPU is Stopped 0x8 to 0xD = Reserved for future use 0xE = Clear all Break Point Counters 0xF = Reset - Issue a hard Reset the IMP CPU Each of these operations are described in more detail below.

A full description of each IMPOp follows.

- 0x0 = No Op – This command does nothing

- 0x1 = Run – Run will restart the CPU running at the current Program Counter's location or cancel a Stop request that has not occurred yet. The Program Counter's location can be modified before a Run by using the Examine, Examine Next or Deposit Next IMPOps. A Run IMPOp will allow the CPU to continue processing after a Break Point as well.

- 0x2 = Stop – Stop will stop the CPU as soon as it is safe to do so. If a Stop operation is issued while the CPU is Halt'ed, the Stop will take effect on the 1st instruction following the Interrupt Acknowledge cycle that woke up the CPU. This 'pending' Stop can be canceled by issuing a Run operation. In all cases the CPU will be Stopped on the next Op Code fetch that occurs (i.e., when OpCodeFetch = 1).

- 0x3 = Examine (when OpCodeFetch = 1) – Examine will move the CPU's PC (Program Counter) to the address defined in the AddrHi and AddrLo registers below (Indexes 0x0A & 0x0B). The CPU will then be Stopped at this new PC address and the contents of that memory location will be readable in the RdData register below (Index 0x0D). If OpCodeFetch = 0 this Examine does nothing (see Single Step below). A Run IMPOp after an Examine will cause the CPU to start execution of code at the new PC's address.

- 0x4 = Examine Next (when OpCodeFetch = 1) – Examine Next will move the CPU's PC (Program Counter) up by one byte. The CPU will then be Stopped at this new PC address and the contents of that memory location will be readable in the RdData register below (Index 0x0D). If OpCodeFetch = 0 this Examine Next does nothing (see Single Step below). A Run IMPOp after an Examine Next will cause the CPU to start execution of code at the new PC's address.

- 0x5 = Deposit (when OpCodeFetch = 1) – Deposit causes the contents of the WrData register below (Index 0x0C) to be written to the memory currently pointed to by the CPU's Program Counter (PC). After this write takes place the RdData register (Index 0x0D) will reflect the new memory contents (assuming the write was to RAM). If OpCodeFetch = 0 this Deposit does nothing (see Single Step below).

- 0x6 = Deposit Next (when OpCodeFetch = 1) – Deposit Next is a simple combination of the Deposit IMPOp followed by the Examine Next IMPOp. If OpCodeFetch = 0 this Deposit Next does nothing (see Single Step below).

- 0x7 = Single Step – Single Step causes the CPU to execute 1 or N instructions[5] before it is Stopped again and it will advance a Break Point to the next OpCode Fetch (where OpCodeFetch = 1). If OpCodeFetch = 1 then the Single Step IMPOp will let the CPU execute N instructions as defined in the SSCount register below (Index 0x0E). If OpCodeFetch = 0 (typically due to a Break Point) then the Single Step IMPOp will Stop at the beginning of next Op Code fetch regardless of the value of the SSCount register. If a Break Point occurs while Single Step is executing N instructions, the Break Point will terminate this Single Step operation.

[5] CPU instructions that contain multiple byte Op Code bytes are counted as one instruction.

- 0xE = Clear All Break Point Counters – Causes all the Break Point counters to be reset to a value of 0x0000. These counters are at Indexes 0x44 to 0x47 for Break 0, 0x4C to 0x4F for Break 1, 0x54 to 0x57 for Break 2, and 0x5C to 0x5F Break 3.

- 0xF = Reset – Issue a hardware reset to the IMP CPU and all of its peripherals, resetting all of the CPU's I/O registers back to their default values (these Debug Index registers are not part of the CPU's I/O registers as they are in the SMI address space). The CPU's SRAM will NOT be initialized by this Reset if the CPU is in Stop mode. If the CPU was Running it will fetch its next instruction at address 0x0000. If the CPU was stopped it will be Stopped again at its first Op Code fetch at address 0x0000. To perform a software reset without resetting the CPU's peripherals and I/O registers, Stop the CPU, Examine address 0x0000, and then Run the CPU.

Note

When the CPU is Stopped the Watch Dog and Timers are also stopped so that interrupts or resets don't occur while the CPU is Stopped. When the CPU is Single Step'ed the Watch Dog and Timers will count for the appropriate number of cycles.

Table 57: IMP Status Index: 0x09 of G2 Offset 0x13

Bits	Field	Type	Description
7	NMI	RO	The value of the NMI (Non Maskable Interrupt) signal going to the CPU core. 0 = Non-Maskable Interrupt to the CPU is not active 1 = Non-Maskable Interrupt to the CPU is active The NMI is maskable by an SMI register (Index 0x0F bit 0) and this signal is after that mask!
6	INT	RO	The value of the INT (Interrupt) signal going to the CPU core. 0 = Maskable Interrupt to the CPU is not active 1 = Maskable Interrupt to the CPU is active
5	HALT	RO	The value of the HALT signal directly from the CPU core. 0 = The CPU is not Halt'ed 1 = The CPU is Halt'ed
4	M1	RO	The value of the M1 (Machine Cycle One) signal from the CPU core. 0 = M1 from the CPU is not active 1 = M1 form the CPU is active
3	MREQ	RO	The value of the MREQ (Memory Request) signal from the CPU core. 0 = MREQ from the CPU is not active 1 = MREQ form the CPU is active
2	IORQ	RO	The value of the IORQ (Input/Output Request) signal from the CPU core. 0 = IORQ from the CPU is not active 1 = IORQ form the CPU is active
1	RD	RO	The value of the RD (Read) signal from the CPU core. 0 = RD from the CPU is not active 1 = RD form the CPU is active
0	WR	RO	The value of the WR (Write) signal from the CPU core. 0 = WR from the CPU is not active 1 = WR form the CPU is active

Table 58: Debug Addr Lo Index: 0x0A of G2 Offset 0x13

Bits	Field	Type	Description
7:0	AddrLo	RWR	Address Lo. The contents of this register are used by the Debugger's Examine IMPOp command (see Index 0x08). It is used to set the CPU's Program Counter's Lo address byte.

Table 59: Debug Addr Hi Index: 0x0B of G2 Offset 0x13

Bits	Field	Type	Description
7:0	AddrHi	RWR	Address Hi. The contents of this register are used by the Debugger's Examine IMPOp command (see Index 0x08). It is used to set the CPU's Program Counter's Hi address byte.

Table 60: Debug Write Data Index: 0x0C of G2 Offset 0x13

Bits	Field	Type	Description
7:0	WrData	RWR	Write Data. The contents of this register are used by the Debugger's Deposit & Deposit Next IMPOp commands (see Index 0x08). It is used as the data to write to memory.

Table 61: Debug Read Data Index: 0x0D of G2 Offset 0x13

Bits	Field	Type	Description
7:0	RdData	RO	Read Data. The contents of this register are used by the Debugger's Break Points, Stop, Examine, Examine Next, Deposit & Deposit Next IMPOp commands (see Index 0x08). It is used to return the data found on the CPU's input Data bus from the device pointed to by the CPU's Address & Control bus.

Table 62: Single Step Count Index: 0x0E of G2 Offset 0x13

Bits	Field	Type	Description
7:0	SSCount	RWS to 0x01	Single Step Count. The contents of this register are used by the Debugger's Single Step IMPOp command (see Index 0x08) when OpCodeFetch = 1 at the start of the Single Step IMPOp. It is used to define the number of instructions (OpCodeFetch cycles) the CPU can execute before the Debugger Stops the CPU again. A value of 0x00 equals 256 instructions. Note: If the CPU is executing multiple instructions during a Single Step IMPOp and a Break Point occurs, the Break Point will terminate the Single Step IMPOp so that the Break Point event can be examined.

Table 63: Debug Control Index: 0x0F of G2 Offset 0x13

Bits	Field	Type	Description
7	SwitchLoop Back	RWR	Switch port Loop Back test mode. 0 = Normal operation 1 = Switch port loop back test mode enabled When this bit is set to a one any frame mapped by the switch to the internal CPU's port will get echoed back to the switch via the internal CPU's switch port. No software is needed to be running on the IMP CPU for this to work & it will echo packets back into the switch at full wire speed. Setting this bit to a one also enables Link on the internal CPU's port (Port offset 0x00 - but it does not change its Ports State - Port offset 0x04). Each switch frame is echo'ed back byte for byte as it was received with one exception: If the Header mode is enabled on the internal CPU's port (Header = 1 in Port offset 0x04) then the frame's Header bytes are zero'ed. Doing this prevents the frame's echo'ed Header from being written to Port offset 0x06 allowing Port offset 0x06 to be configured to force the echo'ed frames out a specific switch port that has been isolated for this purpose. Note: When this bit is set to a one the internal CPU will not receive any frames from the switch and any frames the CPU tries to transmit into the switch will be transmitted to nowhere.
6:3	Reserved	RES	Reserved for future use.
2	RunTimers	RWR	Run the Timers. Normally when the CPU is Stopped all the Timers are also stopped. Setting this bit to a one will allow the Times to continue running even when the CPU is Stopped. This could be useful to help debugging. Note: When this bit is set to a one only the Timers are allowed to continue counting while the CPU is Stopped. This bit has no effect on the IMP Watch Dog (i.e., the IMP Watch Dog is always stopped while the CPU is Stopped).
1	MaskINT	RWR	Mask INT. The CPU's INT signal is normally masked whenever the CPU is Stopped by the Debugger, but setting this bit to a one will mask the INT signal all the time. This could be useful to help debugging.
0	UnMask NMI	RWR	UnMask NMI. The CPU's NMI signal is normally masked whenever the CPU is Stopped by the Debugger, but clearing this bit to a zero will mask the NMI signal all the time. Masking NMI is required for Booting (you don't want an NMI being serviced prior to its code being loaded. This feature could also be useful to help debugging.

Table 64: CPU Register A, Index: 0x10 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	A	RO	The current contents of CPU's Main register A.

Table 65: CPU Flags, Index: 0x11 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	Flags	RO	The current contents of CPU's Main flags.

Table 66: CPU Register B, Index: 0x12 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	B	RO	The current contents of CPU's Main register B.

Table 67: CPU Register C, Index: 0x13 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	C	RO	The current contents of CPU's Main register C.

Table 68: CPU Register D, Index: 0x14 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	D	RO	The current contents of CPU's Main register D.

Table 69: CPU Register E, Index: 0x15 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	E	RO	The current contents of CPU's Main register E.

Table 70: CPU Register H, Index: 0x16 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	H	RO	The current contents of CPU's Main register H.

Table 71: CPU Register L, Index: 0x17 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	L	RO	The current contents of CPU's Main register L.

Table 72: CPU Register A’, Index: 0x18 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	A’	RO	The current contents of CPU’s Alternate register A.

Table 73: CPU Flags’, Index: 0x19 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	Flags’	RO	The current contents of CPU’s Alternate flags.

Table 74: CPU Register B’, Index: 0x1A of G2 Offset 0x13

Bits	Field	Type	Description
7:0	B’	RO	The current contents of CPU’s Alternate register B.

Table 75: CPU Register C’, Index: 0x1B of G2 Offset 0x13

Bits	Field	Type	Description
7:0	C’	RO	The current contents of CPU’s Alternate register C.

Table 76: CPU Register D’, Index: 0x1C of G2 Offset 0x13

Bits	Field	Type	Description
7:0	D’	RO	The current contents of CPU’s Alternate register D.

Table 77: CPU Register E’, Index: 0x1D of G2 Offset 0x13

Bits	Field	Type	Description
7:0	E’	RO	The current contents of CPU’s Alternate register E.

Table 78: CPU Register H’, Index: 0x1E of G2 Offset 0x13

Bits	Field	Type	Description
7:0	H’	RO	The current contents of CPU’s Alternate register H.

Table 79: CPU Register L’, Index: 0x1F of G2 Offset 0x13

Bits	Field	Type	Description
7:0	L’	RO	The current contents of CPU’s Alternate register L.

Table 80: CPU Register IXH, Index: 0x20 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	IXH	RO	The current contents of CPU’s Special Purpose register IX high byte.

Table 81: CPU Register IXL, Index: 0x21 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	IXL	RO	The current contents of CPU’s Special Purpose register IX low byte.

Table 82: CPU Register IYH, Index: 0x22 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	IYH	RO	The current contents of CPU’s Special Purpose register IY high byte.

Table 83: CPU Register IYL, Index: 0x23 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	IYL	RO	The current contents of CPU’s Special Purpose register IY low byte.

Table 84: CPU Register SPH, Index: 0x24 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	SPH	RO	The current contents of CPU’s Special Purpose register SP high byte.

Table 85: CPU Register SPL, Index: 0x25 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	SPL	RO	The current contents of CPU’s Special Purpose register SP low byte.

Table 86: CPU Register PCH Index: 0x26 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	PCH	RO	The current contents of CPU's Special Purpose register PC high byte.

Table 87: CPU Register PCL, Index: 0x27 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	PCL	RO	The current contents of CPU's Special Purpose register PC low byte.

Table 88: CPU Register IV Index: 0x28 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	IV	RO	The current contents of CPU's Special Purpose Interrupt Vector register.

Table 89: CPU Reg Refresh Index: 0x29 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	Refresh	RO	The current contents of CPU's Special Purpose Refresh register.

Note

The Refresh register is always 0x00 as there is no need in this design to support DRAM refreshes.

Table 90: IMP I/O Page Index: 0x2F of G2 Offset 0x13

Bits	Field	Type	Description
7:4	IOPage	RWR	I/O Page address when Debug is accessing the CPU’s I/O registers. The IMP CPU supports an I/O Address space of 256 addresses. This register is used to set the upper 4-bits of the 8-bit I/O address when this debug interface is used to access the CPU’s I/O registers using Indexes 0x70 to 0x7F below.
3:0	Reserved	RES	Reserved for future use.

Table 91: Break 0 Control, Index: 0x30 of G2 Offset 0x13

Bits	Field	Type	Description
7	ClrCtr0	SC	Clear Break Point 0's Counter. Setting this bit to a one will clear to zero the 32-bit counter that is associated with this Break Point (Indexes 0x44 to 0x47) & then this bit will self clear.
6	Bk0IntEn	RWR	Break Point 0 Interrupt Enable. 0 = This Break Point will not generate an EEInt Interrupt 1 = This Break Point will generate an EEInt Interrupt When this bit is set to a one and a Break occurs due to this Break Point an EEInt Interrupt will occur. A single bit called Break (Index 0x03) is set to a one if any Break occurs with an enabled interrupt. Reading Index 0x38 will indicate which Break caused this interrupt.
5:3	Reserved	RES	Reserved for future use.
2	BkPt0Mem	RWR	Break Point 0 Matches Memory Accesses. 0 = This Break Point monitors I/O accesses 1 = This Break Point monitors Memory accesses Note: When this bit is cleared to a zero, only the lower 8-bits of this Break Point's address range bits are used. In this case the upper 8-bits of this Break Point address registers must be 0x00.
1	BkPt0Wr	RWR	Break Point 0 Matches Write Accesses. 0 = This Break Point doesn't monitor Write accesses 1 = This Break Point monitors Write accesses When this bit is set to a one, a Break will occur whenever the CPU writes to the address type defined by BkPt0Mem above, within the address range defined by this Break Points address registers (Indexes 0x40 to 0x43). See BkPt0Rd below for more information.
0	BkPt0Rd	RWR	Break Point 0 Matches Read Accesses. 0 = This Break Point doesn't monitor Read accesses 1 = This Break Point monitors Read accesses When this bit is set to a one, a Break will occur whenever the CPU read from the address type defined by BkPt0Mem above, within the address range defined by this Break Points address registers (Indexes 0x40 to 0x43). A Break stops the CPU similar to a Stop IMPOp (Index 0x08) but it may not occur at an Op Code fetch (see Single Step in Index 0x08).

Note

If both BkPt0Rd and BkPt0Wt are set to a one, a break will occur for either a read or a write to the Break Point's address range/type.

If both BkPt0Rd and BkPt0Wt are cleared to a zero, a break will NOT occur but the Break Point's range counter (Indexes 0x44 to 0x47) will still increment for each access to the Break Point's defined range and type (for both reads and writes). The Break Point's type is controlled by the BkPt0Mem bit above.

Table 92: Break 1 Control, Index: 0x31 of G2 Offset 0x13

Bits	Field	Type	Description
7	ClrCtr1	SC	Clear Break Point 1's Counter. Setting this bit to a one will clear to zero the 32-bit counter that is associated with this Break Point (Indexes 0x4C to 0x4F) & then this bit will self clear.
6	Bk1IntEn	RWR	Break Point 1 Interrupt Enable. 0 = This Break Point will not generate an EEInt Interrupt 1 = This Break Point will generate an EEInt Interrupt When this bit is set to a one and a Break occurs due to this Break Point an EEInt Interrupt will occur. A single bit called Break (Index 0x03) is set to a one if any Break occurs with an enabled interrupt. Reading Index 0x38 will indicate which Break caused this interrupt.
5:3	Reserved	RES	Reserved for future use.
2	BkPt1Mem	RWR	Break Point 1 Matches Memory Accesses. 0 = This Break Point monitors I/O accesses 1 = This Break Point monitors Memory accesses Note: When this bit is cleared to a zero, only the lower 8-bits of this Break Point's address range bits are used. In this case the upper 8-bits of this Break Point address registers must be 0x00.
1	BkPt1Wr	RWR	Break Point 1 Matches Write Accesses. 0 = This Break Point doesn't monitor Write accesses 1 = This Break Point monitors Write accesses When this bit is set to a one, a Break will occur whenever the CPU writes to the address type defined by BkPt1Mem above, within the address range defined by this Break Points address registers (Indexes 0x48 to 0x4B). See BkPt1Rd below for more information.
0	BkPt1Rd	RWR	Break Point 1 Matches Read Accesses. 0 = This Break Point doesn't monitor Read accesses 1 = This Break Point monitors Read accesses When this bit is set to a one, a Break will occur whenever the CPU read from the address type defined by BkPt1Mem above, within the address range defined by this Break Points address registers (Indexes 0x48 to 0x4B). A Break stops the CPU similar to a Stop IMPOp (Index 0x08) but it may not occur at an Op Code fetch (see Single Step in Index 0x08).

Note

If both BkPt1Rd and BkPt1Wt are set to a one, a break will occur for either a read or a write to the Break Point's address range/type.

If both BkPt1Rd and BkPt1Wt are cleared to a zero, a break will NOT occur but the Break Point's range counter (Indexes 0x4C to 0x4F) will still increment for each access to the Break Point's defined range and type (for both reads and writes). The Break Point's type is controlled by the BkPt1Mem bit above.

Table 93: Break 2 Control, Index: 0x32 of G2 Offset 0x13

Bits	Field	Type	Description
7	ClrCtr2	SC	Clear Break Point 2's Counter. Setting this bit to a one will clear to zero the 32-bit counter that is associated with this Break Point (Indexes 0x54 to 0x57) & then this bit will self clear.
6	Bk2IntEn	RWR	Break Point 2 Interrupt Enable. 0 = This Break Point will not generate an EEInt Interrupt 1 = This Break Point will generate an EEInt Interrupt When this bit is set to a one and a Break occurs due to this Break Point an EEInt Interrupt will occur. A single bit called Break (Index 0x03) is set to a one if any Break occurs with an enabled interrupt. Reading Index 0x38 will indicate which Break caused this interrupt.
5:3	Reserved	RES	Reserved for future use.
2	BkPt2Mem	RWR	Break Point 2 Matches Memory Accesses. 0 = This Break Point monitors I/O accesses 1 = This Break Point monitors Memory accesses Note: When this bit is cleared to a zero, only the lower 8-bits of this Break Point's address range bits are used. In this case the upper 8-bits of this Break Point address registers must be 0x00.
1	BkPt2Wr	RWR	Break Point 2 Matches Write Accesses. 0 = This Break Point doesn't monitor Write accesses 1 = This Break Point monitors Write accesses When this bit is set to a one, a Break will occur whenever the CPU writes to the address type defined by BkPt2Mem above, within the address range defined by this Break Points address registers (Indexes 0x50 to 0x53). See BkPt2Rd below for more information.
0	BkPt2Rd	RWR	Break Point 2 Matches Read Accesses. 0 = This Break Point doesn't monitor Read accesses 1 = This Break Point monitors Read accesses When this bit is set to a one, a Break will occur whenever the CPU read from the address type defined by BkPt2Mem above, within the address range defined by this Break Points address registers (Indexes 0x50 to 0x53). A Break stops the CPU similar to a Stop IMPOp (Index 0x08) but it may not occur at an Op Code fetch (see Single Step in Index 0x08).

Note

If both BkPt2Rd and BkPt2Wt are set to a one, a break will occur for either a read or a write to the Break Point's address range/type.

If both BkPt2Rd and BkPt2Wt are cleared to a zero, a break will NOT occur but the Break Point's range counter (Indexes 0x54 to 0x57) will still increment for each access to the Break Point's defined range and type (for both reads and writes). The Break Point's type is controlled by the BkPt2Mem bit above.

Table 94: Break 3 Control, Index: 0x33 of G2 Offset 0x13

Bits	Field	Type	Description
7	ClrCtr3	SC	Clear Break Point 3's Counter. Setting this bit to a one will clear to zero the 32-bit counter that is associated with this Break Point (Indexes 0x5C to 0x5F) & then this bit will self clear.
6	Bk3IntEn	RWR	Break Point 3 Interrupt Enable. 0 = This Break Point will not generate an EEInt Interrupt 1 = This Break Point will generate an EEInt Interrupt When this bit is set to a one and a Break occurs due to this Break Point an EEInt Interrupt will occur. A single bit called Break (Index 0x03) is set to a one if any Break occurs with an enabled interrupt. Reading Index 0x38 will indicate which Break caused this interrupt.
5:3	Reserved	RES	Reserved for future use.
2	BkPt3Mem	RWR	Break Point 3 Matches Memory Accesses. 0 = This Break Point monitors I/O accesses 1 = This Break Point monitors Memory accesses Note: When this bit is cleared to a zero, only the lower 8-bits of this Break Point's address range bits are used. In this case the upper 8-bits of this Break Point address registers must be 0x00.
1	BkPt3Wr	RWR	Break Point 3 Matches Write Accesses. 0 = This Break Point doesn't monitor Write accesses 1 = This Break Point monitors Write accesses When this bit is set to a one, a Break will occur whenever the CPU writes to the address type defined by BkPt3Mem above, within the address range defined by this Break Points address registers (Indexes 0x58 to 0x5B). See BkPt3Rd below for more information.
0	BkPt3Rd	RWR	Break Point 3 Matches Read Accesses. 0 = This Break Point doesn't monitor Read accesses 1 = This Break Point monitors Read accesses When this bit is set to a one, a Break will occur whenever the CPU read from the address type defined by BkPt3Mem above, within the address range defined by this Break Points address registers (Indexes 0x58 to 0x5B). A Break stops the CPU similar to a Stop IMPOp (Index 0x08) but it may not occur at an Op Code fetch (see Single Step in Index 0x08).

Note

If both BkPt3Rd and BkPt3Wt are set to a one, a break will occur for either a read or a write to the Break Point's address range/type.

If both BkPt3Rd and BkPt3Wt are cleared to a zero, a break will NOT occur but the Break Point's range counter (Indexes 0x5C to 0x5F) will still increment for each access to the Break Point's defined range and type (for both reads and writes). The Break Point's type is controlled by the BkPt3Mem bit above.

Table 95: Break Int Status, Index: 0x38 of G2 Offset 0x13

Bits	Field	Type	Description
7:4	Reserved	RES	Reserved for future use.
3	Bk3Active	RO	Break Point 3 is Active. 0 = This Break Point does not match the CPU's activity 1 = This Break Point matches the CPU's activity and Stopped the CPU This bit is set to a one whenever the CPU has been Stopped due to this Break Point. It is cleared to a zero by a Run or Single Step IMPOp (Index 0x08) assuming the Break conditions no longer match the CPU's activity.
2	Bk2Active	RO	Break Point 2 is Active. 0 = This Break Point does not match the CPU's activity 1 = This Break Point matches the CPU's activity and Stopped the CPU This bit is set to a one whenever the CPU has been Stopped due to this Break Point. It is cleared to a zero by a Run or Single Step IMPOp (Index 0x08) assuming the Break conditions no longer match the CPU's activity.
1	Bk1Active	RO	Break Point 1 is Active. 0 = This Break Point does not match the CPU's activity 1 = This Break Point matches the CPU's activity and Stopped the CPU This bit is set to a one whenever the CPU has been Stopped due to this Break Point. It is cleared to a zero by a Run or Single Step IMPOp (Index 0x08) assuming the Break conditions no longer match the CPU's activity.
0	Bk0Active	RO	Break Point 0 is Active. 0 = This Break Point does not match the CPU's activity 1 = This Break Point matches the CPU's activity and Stopped the CPU This bit is set to a one whenever the CPU has been Stopped due to this Break Point. It is cleared to a zero by a Run or Single Step IMPOp (Index 0x08) assuming the Break conditions no longer match the CPU's activity.

Table 96: Break 0 Low Addr[7:0], Index: 0x40 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0Low Addr[7:0]	RWR	Break Point 0's Low Address bits [7:0]. This register defines the low 8-bits of the low break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 97: Break 0 Low Addr[15:8], Index: 0x41 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0Low Addr[15:8]	RWR	Break Point 0's Low Address bits [15:8]. This register defines the high 8-bits of the low break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x30) to this low address or higher (up to the value of this Break Point's High Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 98: Break 0 High Addr[7:0], Index: 0x42 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0High Addr[7:0]	RWR	Break Point 0’s High Address bits [7:0]. This register defines the low 8-bits of the high break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 99: Break 0 High Addr[15:8], Index: 0x43 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0High Addr[15:8]	RWR	Break Point 0’s High Address bits [15:8]. This register defines the high 8-bits of the high break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x30) to this low address or lower (down to the value of this Break Point’s Low Addr value), a Break will occur. This address is also used to increment this Break Point’s counter (see the Indexes after this Break Point’s addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 100: Break 0 Counter[7:0], Index: 0x44 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0 Ctr[7:0]	RWR	Break Point 0's Counter bits [7:0]. The upper 24-bits of this counter are in the registers below where the full register description can be found.

Table 101: Break 0 Counter[15:8], Index: 0x45 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0 Ctr[15:8]	RWR	Break Point 0's Counter bits [15:8]. The lower 8-bits of this counter are in the register above and the upper 16-bits is in the registers below where the full register description can be found.

Table 102: Break 0 Counter[23:16], Index: 0x46 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0 Ctr[23:16]	RWR	Break Point 0's Counter bits [23:16]. The lower 16-bits of this counter are in the registers above and the upper 8-bits is in the register below where the full register description can be found.

Table 103: Break 0 Counter[31:24], Index: 0x47 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt0 Ctr[31:24]	RWR	Break Point 0's Counter bits [31:24]. This Break Point counter always counts once each and every time the defined access occurs (read and write to memory or I/O – see Index 0x30) between or equal to, this Break Point's Low Addr value and its High Addr value (defined in the Indexes 0x40 to 0x43). If a Break is also enabled (Index 0x30) the break will occur. If a Break is not enabled this counter counts on both read and write accesses and the CPU continues processing. This mode is useful to help determine where the CPU is spending most of its processing time. The lower 24-bits of this counter are in the registers above.

Table 104: Break 1 Low Addr[7:0], Index: 0x48 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1Low Addr[7:0]	RWR	Break Point 1's Low Address bits [7:0]. This register defines the low 8-bits of the low break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 105: Break 1 Low Addr[15:8], Index: 0x49 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1Low Addr[15:8]	RWR	Break Point 1's Low Address bits [15:8]. This register defines the high 8-bits of the low break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x31) to this low address or higher (up to the value of this Break Point's High Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 106: Break 1 High Addr[7:0], Index: 0x4A of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1High Addr[7:0]	RWR	Break Point 1's High Address bits [7:0]. This register defines the low 8-bits of the high break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 107: Break 1 High Addr[15:8], Index: 0x4B of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1High Addr[15:8]	RWR	Break Point 0's High Address bits [15:8]. This register defines the high 8-bits of the high break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x31) to this low address or lower (down to the value of this Break Point's Low Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 108: Break 1 Counter[7:0], Index: 0x4C of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1 Ctr[7:0]	RWR	Break Point 1's Counter bits [7:0]. The upper 24-bits of this counter are in the registers below where the full register description can be found.

Table 109: Break 1 Counter[15:8], Index: 0x4D of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1 Ctr[15:8]	RWR	Break Point 1's Counter bits [15:8]. The lower 8-bits of this counter are in the register above and the upper 16-bits is in the registers below where the full register description can be found.

Table 110: Break 1 Counter[23:16], Index: 0x4E of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1 Ctr[23:16]	RWR	Break Point 1's Counter bits [23:16]. The lower 16-bits of this counter are in the registers above and the upper 8-bits is in the register below where the full register description can be found.

Table 111: Break 1 Counter[31:24], Index: 0x4F of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt1 Ctr[31:24]	RWR	Break Point 1's Counter bits [31:24]. This Break Point counter always counts once each and every time the defined access occurs (read and write to memory or I/O – see Index 0x31) between or equal to, this Break Point's Low Addr value and its High Addr value (defined in the Indexes 0x48 to 0x4B). If a Break is also enabled (Index 0x31) the break will occur. If a Break is not enabled this counter counts on both read and write accesses and the CPU continues processing. This mode is useful to help determine where the CPU is spending most of its processing time. The lower 24-bits of this counter are in the registers above.

Table 112: Break 2 Low Addr[7:0], Index: 0x50 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2Low Addr[7:0]	RWR	Break Point 2's Low Address bits [7:0]. This register defines the low 8-bits of the low break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 113: Break 2 Low Addr[15:8], Index: 0x51 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2Low Addr[15:8]	RWR	Break Point 2's Low Address bits [15:8]. This register defines the high 8-bits of the low break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x32) to this low address or higher (up to the value of this Break Point's High Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 114: Break 2 High Addr[7:0], Index: 0x52 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2High Addr[7:0]	RWR	Break Point 2's High Address bits [7:0]. This register defines the low 8-bits of the high break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 115: Break 2 High Addr[15:8], Index: 0x53 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2High Addr[15:8]	RWR	Break Point 2's High Address bits [15:8]. This register defines the high 8-bits of the high break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x32) to this low address or lower (down to the value of this Break Point's Low Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 116: Break 2 Counter[7:0], Index: 0x54 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2 Ctr[7:0]	RWR	Break Point 2's Counter bits [7:0]. The upper 24-bits of this counter are in the registers below where the full register description can be found.

Table 117: Break 2 Counter[15:8], Index: 0x55 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2 Ctr[15:8]	RWR	Break Point 2's Counter bits [15:8]. The lower 8-bits of this counter are in the register above and the upper 16-bits is in the registers below where the full register description can be found.

Table 118: Break 2 Counter[23:16], Index: 0x56 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2 Ctr[23:16]	RWR	Break Point 2's Counter bits [23:16]. The lower 16-bits of this counter are in the registers above and the upper 8-bits is in the register below where the full register description can be found.

Table 119: Break 2 Counter[31:24], Index: 0x57 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt2 Ctr[31:24]	RWR	Break Point 2's Counter bits [31:24]. This Break Point counter always counts once each and every time the defined access occurs (read and write to memory or I/O – see Index 0x32) between or equal to, this Break Point's Low Addr value and its High Addr value (defined in the Indexes 0x50 to 0x53). If a Break is also enabled (Index 0x32) the break will occur. If a Break is not enabled this counter counts on both read and write accesses and the CPU continues processing. This mode is useful to help determine where the CPU is spending most of its processing time. The lower 24-bits of this counter are in the registers above.

Table 120: Break 3 Low Addr[7:0], Index: 0x58 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3Low Addr[7:0]	RWR	Break Point 3's Low Address bits [7:0]. This register defines the low 8-bits of the low break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 121: Break 3 Low Addr[15:8], Index: 0x59 of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3Low Addr[15:8]	RWR	Break Point 3's Low Address bits [15:8]. This register defines the high 8-bits of the low break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x33) to this low address or higher (up to the value of this Break Point's High Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 122: Break 3 High Addr[7:0], Index: 0x5A of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3High Addr[7:0]	RWR	Break Point 3's High Address bits [7:0]. This register defines the low 8-bits of the high break point address for this Break Point. The upper 8-bits of this address are in the register below where the full register description can be found.

Table 123: Break 3 High Addr[15:8], Index: 0x5B of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3High Addr[15:8]	RWR	Break Point 0's High Address bits [15:8]. This register defines the high 8-bits of the high break point address for this Break Point. Any enabled access (read &/or write to memory or I/O – configured in Index 0x33) to this low address or lower (down to the value of this Break Point's Low Addr value), a Break will occur. This address is also used to increment this Break Point's counter (see the Indexes after this Break Point's addresses) where it counts even if a CPU stopping break is not enabled to occur. The lower 8-bits of this address are in the register above.

Table 124: Break 3 Counter[7:0], Index: 0x5C of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3 Ctr[7:0]	RWR	Break Point 3's Counter bits [7:0]. The upper 24-bits of this counter are in the registers below where the full register description can be found.

Table 125: Break 3 Counter[15:8], Index: 0x5D of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3 Ctr[15:8]	RWR	Break Point 3's Counter bits [15:8]. The lower 8-bits of this counter are in the register above and the upper 16-bits is in the registers below where the full register description can be found.

Table 126: Break 3 Counter[23:16], Index: 0x5E of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3 Ctr[23:16]	RWR	Break Point 3's Counter bits [23:16]. The lower 16-bits of this counter are in the registers above and the upper 8-bits is in the register below where the full register description can be found.

Table 127: Break 3 Counter[31:24], Index: 0x5F of G2 Offset 0x13

Bits	Field	Type	Description
7:0	BkPt3 Ctr[31:24]	RWR	Break Point 3's Counter bits [31:24]. This Break Point counter always counts once each and every time the defined access occurs (read and write to memory or I/O – see Index 0x33) between or equal to, this Break Point's Low Addr value and its High Addr value (defined in the Indexes 0x58 to 0x5B). If a Break is also enabled (Index 0x33) the break will occur. If a Break is not enabled this counter counts on both read and write accesses and the CPU continues processing. This mode is useful to help determine where the CPU is spending most of its processing time. The lower 24-bits of this counter are in the registers above.

Table 128: IMP I/O Regs, Index: 0x70 to 0x7F of G2 Offset 0x13

Bits	Field	Type	Description
7:0	IMP I/O	RW	IMP I/O Register access. The IMP CPU supports an I/O Address space of 256 addresses. These 16 registers are used to define the lower 4-bits address bits of the 8-bit I/O address (the upper 4-bits of the I/O address is set in Index 0x2F above) and then return the current contents found in that IMP I/O register or allow the contents of the selected IMP I/O register to be written to. Reading this register can be done at any time without affecting the operation of the IMP CPU as this action simply samples the current contents of the selected IMP I/O registers. IMP I/O registers that are defined as Clear on Read will not be cleared by this debug interface reading of those registers. Registers that clear other bits when read do not affect those status bits either when these registers are read by the debug interface. For example: The reading of the IMD's RdData register (IMD I/O 0x05) by this interface will not clear the IMD's RdDataRdy bit (IMD I/O 0x04) if it was set. Writing to this register will only write to the selected I/O register if the IMP CPU is Stopped (use the Stop IMPOp in index 0x08). See section 6 for a description of the IMP's I/O address map, followed by description of each of its I/O registers.

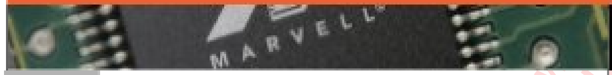
A Acronyms and Abbreviations

Name	Description
CPU	Central Processing Unit.
DMA	Direct Memory Access – a block that quickly transfers blocks of data typically between a CPU's RAM and some other block.
DRAM	Dynamic Random Access Memory – RAM that needs to be refreshed.
EEPROM	Electrically Erasable Programmable Read Only Memory – a device where a CPU's code is typically stored when the device is powered off.
GPIO	General Purpose Input Output – usually a mode on a multi-function device pin.
GUI	Graphical User Interface
IMP	Integrated Micro Processor – the block this document defines
NIC	Network Interface Controller – the block that DMA's Ethernet frames between a CPU's RAM and the NIC's MAC.
MAC	Media Access Controller – the part of a NIC or switch that allows Ethernet frames access to the wire.
MDC	Management Data Clock – the clock wire of the 2-wire SMI interface
MDI	Media Dependent Interface – the device pins used to connect to the magnetics that connect to the Ethernet cable.
MDIO	Management Data Input Output – the data wire of the 2-wire SMI interface
MII	Media Independent Interface – the device pins used to connect to an Ethernet PHY or an external CPU's NIC.
RAM	Random Access Memory – fast memory typically where the code for a CPU resides.
SMI	System Management Interface – a 2 wire register interface, sometimes referred to as MDC/MDIO, used to communicate between IEEE 802.3 MAC's and PHY's. Also used as the register interface in Marvell's Link Street switches.
SRAM	Static Random Access Memory – RAM that does not need to be refreshed.
UART	Universal Asynchronous Receiver Transmitter – an I/O port off a CPU used for byte-wide communications.

B Revision History

Table 129: Revision History

Revision	Date	Description
0.1	5/15/15	Initial Release by dpannell
0.2	05/27/15	Consistency and text editing.
0.3	06/04/15	Updated per review comments
Rev. --	06/08/15	ECN approved, document released.



Marvell Technology Group

http://www.marvell.com

Marvell. **Moving Forward Faster**