Application Note 176

Interrupts on MPCore Development Boards

Released on: 28 March, 2007

Application Note 176
Interrupts on MPCore Development Boards

Release Information

The following changes have been made to this application note.

Table 1 Change history

Date	Issue	Change
November 2006	A	Limited release as DRAFT - not fully tested, no example software
December 2006	B	DRAFT including review comments and example software
March 2007	C	Full release - update to binary point explanation, register diagrams, various minor edits

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

1 Introduction

This application note explains how to use interrupts on our MPCore development boards. It shows the interrupt signal routing, lists the memory map, explains the interrupt controller registers and includes example software.

The current MPCore platform is an ARM11 MPCore Core Tile (CT11MPCore) fitted to an Emulation Baseboard (EB). Application note AN152 [2] describes this platform and includes a .bit file that you must program into the EB's FPGA. You can optionally add a Logic Tile for prototyping your own hardware with the MPCore; see application note AN151 for an example design with AXI peripherals in the Logic Tile.

The MPCore has an integrated interrupt controller, called an Interrupt Distributor, which allocates interrupts to one or more of its CPUs.

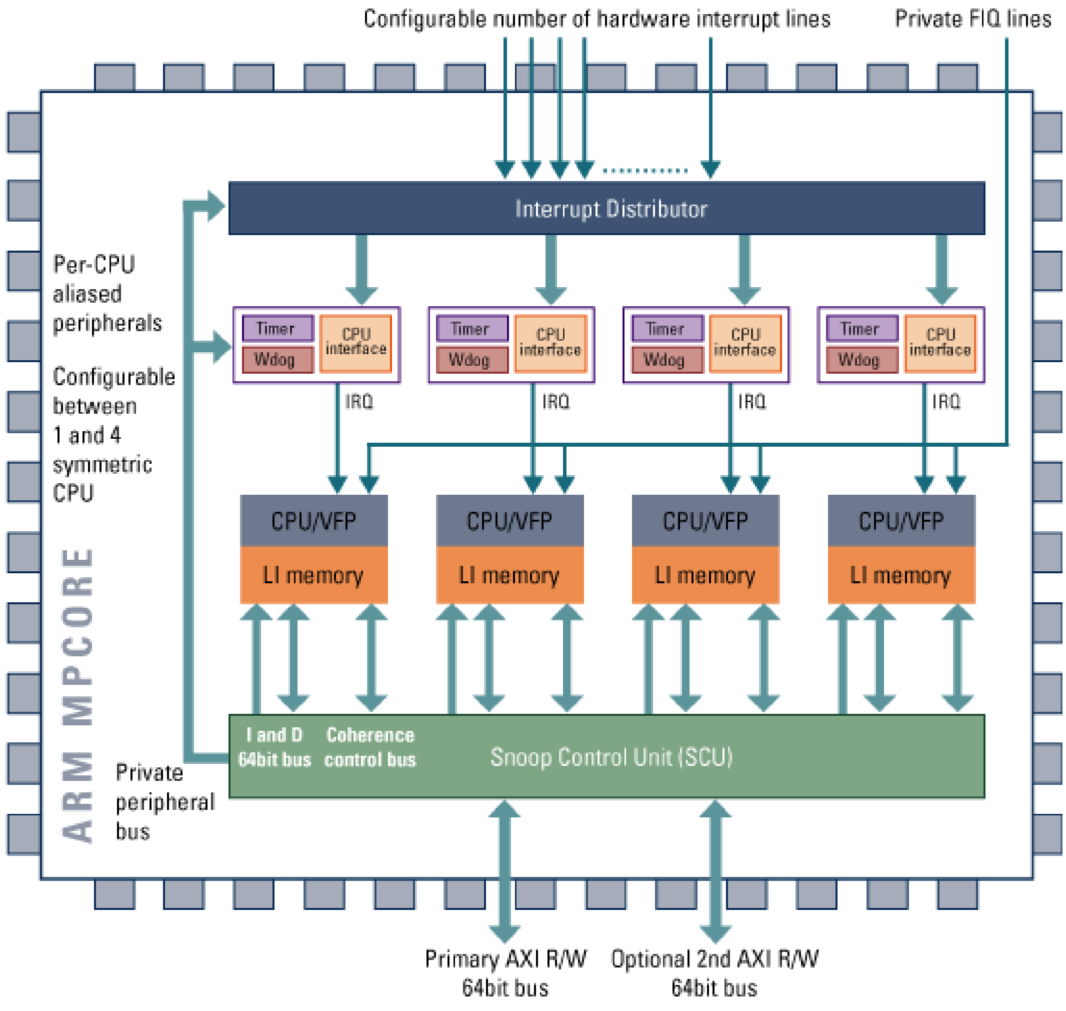

Figure 1 ARM11 MPCore block diagram

For more information about this diagram see the ARM11 MPCore technical reference manual [1]. The Interrupt Distributor is also used in the EB where it is called a Generic Interrupt Controller (GIC) - this document explains both in detail.

2 Getting Started

——— **Warning** ————

Before attaching a Core Tile or Logic Tile to the Emulation Baseboard you must ensure that the FPGA image is correct for the Core Tile you are using. This is because the AHB and AXI buses use different pins, so an incorrect FPGA image may damage the boards. Suitable FPGA images are provided with the application notes on the Versatile CD shipped with the boards. See the Getting Started section of application note AN152 [2] for details.

3 Hardware Description

This section outlines the key ideas behind the design choices for the MPCore platform and presents a block diagram.

The MPCore has a new interrupt architecture: there is one interrupt controller inside the MPCore called an Interrupt Distributor, which allocates interrupts to one or more of its CPUs. This is unlike previous ARM cores which have simple nIRQ and nFIQ interrupt inputs; nor does the MPCore support a Vectored Interrupt Controller (VIC).

This new interrupt controller is called a Generic Interrupt Controller (GIC). It is designed to support symmetric multi-processing (SMP), and to make operating system support simpler.

The MPCore test chip has a limited number of pins, so our development boards use a secondary interrupt controller in the EB FPGA.

- The primary interrupt controller is inside the ARM11 MPCore. The GIC in this test chip was implemented for 4 CPUs, with 32 internal interrupts and 32 external interrupts. The test chip was implemented with 16 external interrupt inputs for validation purposes. The test chip also has 4 legacy nIRQ inputs and 4 nFIQ inputs, one of each for each CPU.
- There are also 4 secondary interrupt controllers inside the EB FPGA. There are 4 GICs, each implemented for 1 CPU (since the MPCore appears to the EB as a single CPU), 32 internal interrupts (not used) and 64 external interrupts.GIC1 is used for normal IRQs, see the table below.

The EB FPGA for the MPCore was designed with 4 GICs to enable a second Core Tile to be added to the EB in future. The table below shows the GICs in the EB FPGA for the MPCore (the design provided in application note AN152 [2]).

Table 2 GIC targets

	AN152
GIC1	nIRQ0 tile site 1
GIC2	nFIQ0 tile site 1
GIC3	nIRQ1 tile site 1[a]
GIC4	nFIQ1 tile site 1[a]

a. Will be deprecated in future so that GIC3 and GIC4 output to tile site 2.

Figure 2 on page 6 shows a block diagram of interrupts in the MPCore + EB development boards. The parts are briefly described in this section, and are covered in more detail in later sub-sections.

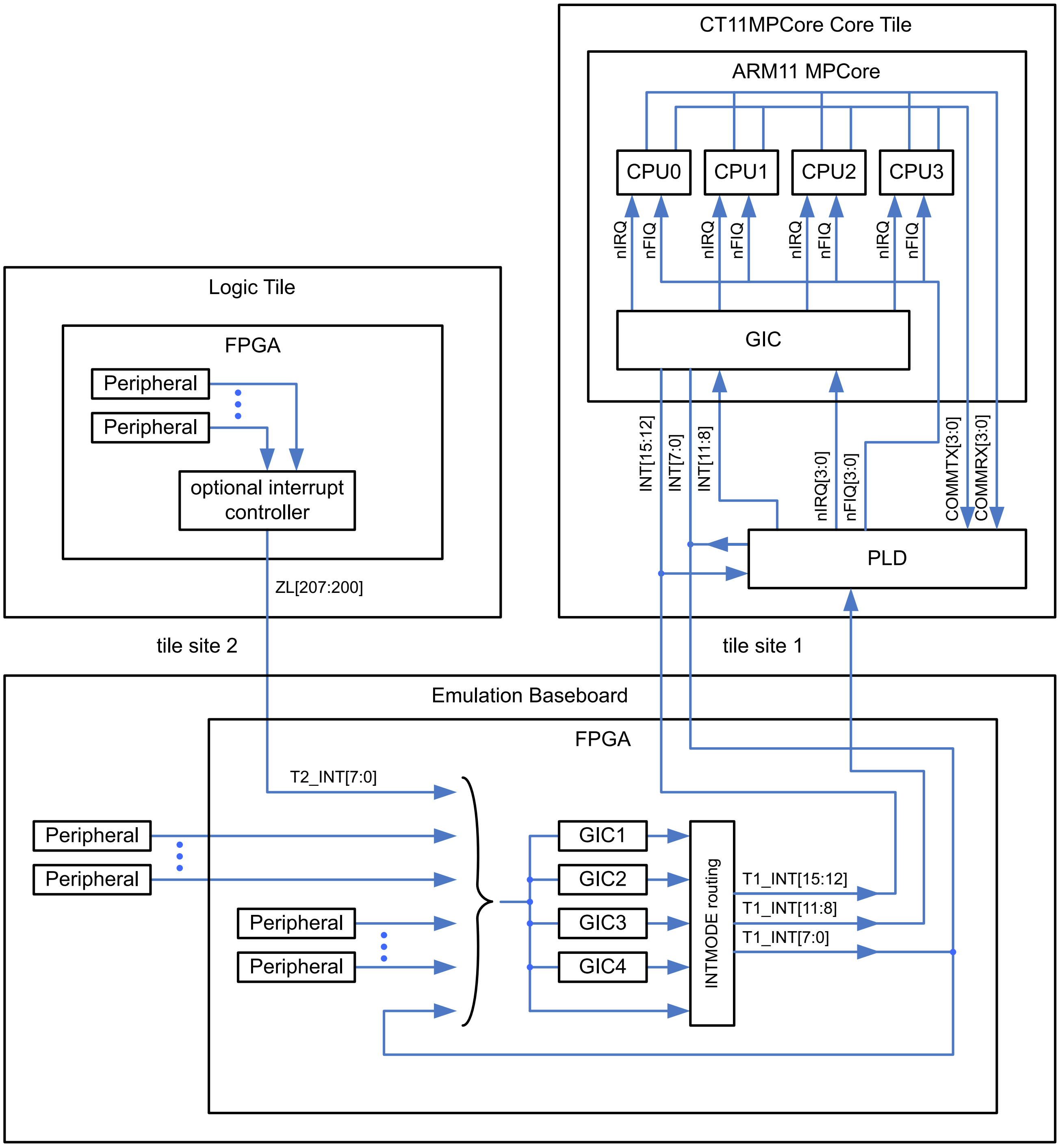

Figure 2 Top-level block diagram of interrupts

You can see from the diagram that interrupt sources may come from peripherals inside the EB FPGA, chips on the EB, and from the design in the Logic Tile FPGA. Each ARM11 CPU also has a COMMRX and COMMTX output that can be used as an interrupt source.

COMMRX and COMMTX signals are used for the Debug Communications Channel (DCC) on ARM processors. The DCC enables you to communicate with your application over the JTAG interface without stopping the processor. DCC can be polled or interrupt driven, but using interrupts is faster.

The interrupt sources are fed into the secondary interrupt controller in the EB FPGA. This consists of four GICs, each with a single output, and some interrupt routing hardware configured by the INTMODE[2:0] bits. In 'legacy mode' the T1_INT[7:0] signals are used as inputs to the FPGA.

Interrupts from the EB are fed to the MPCore Core Tile using 16 signals, T1_INT[15:0]. Once on the Core Tile these are further modified according to the INTMODE[2:0] bits by a PLD.

There is also a GIC in the MPCore that uses the 16 interrupt input pins INT[15:0] on the test chip in 'normal mode' and the 4 legacy interrupt pins nIRQ[3:0] in 'legacy mode'. The interrupts map onto their corresponding CPU number. For example nIRQ0 becomes nIRQ for CPU 0 when CPU 0 is in 'legacy mode'.

The block diagram shows that you can have up to 8 interrupt sources from a Logic Tile fitted to tile site 2. We recommend you use AN151 as a starting point for your Logic Tile design for AXI peripherals on the EB. AN151 includes an interrupt controller that outputs an interrupt on Logic Tile signal ZL200 (signal T2_INT0 in the EB FPGA). This interrupt controller is not a GIC.

3.1 How a GIC works

The GIC is split into two halves: a 'distributor' and 'CPU interface'. In the ARM11 MPCore test chip there is one distributor and 4 CPU interfaces, one for each CPU in the MPCore. See the diagram below. Note there is a private bus so that each CPU can access the GIC registers it needs.

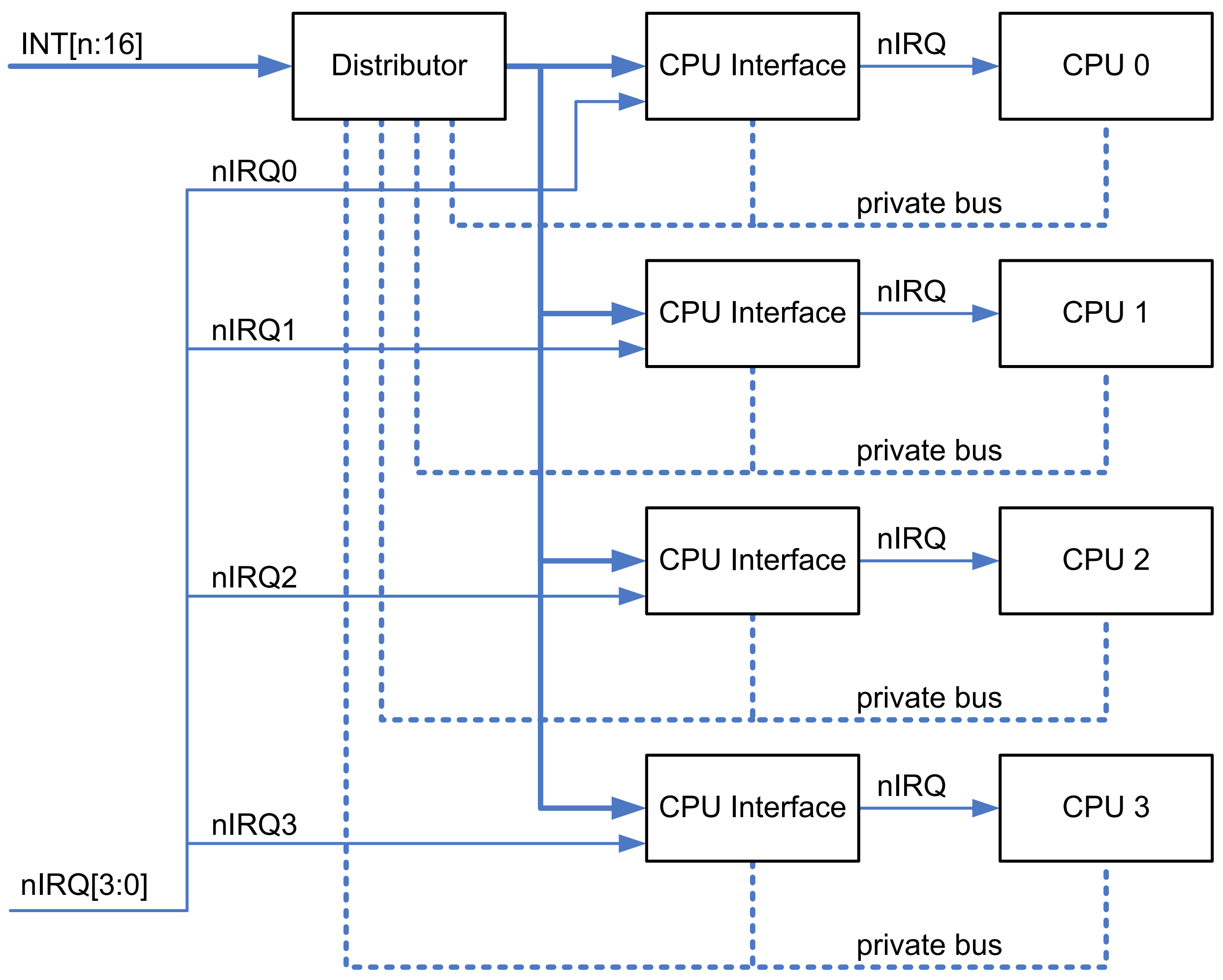

Figure 3 MPCore GIC with 4 CPUs

In the EB's secondary interrupt controller there are 4 GICs. In this case, each GIC has one distributor and one CPU interface. The MPCore appears as one CPU from the EB's point of view.

This document explains the GIC as implemented in the MPCore and EB development boards. It does not cover concepts that may be added in the future.

A GIC allows you to configure an interrupt's priority, enable or disable it, disable all interrupts below a given priority, configure whether it can be pre-empted (interrupted by another interrupt), and optionally what priority is required to pre-empt.

A novel feature of the GIC is the 'Binary Point' register that allows you to reduce the amount of pre-emption in the system. Think of it as a 'pre-emption mask' register. It will be explained in more detail later.

The GIC holds three states for each interrupt source:

Inactive	not asserted
Pending	has been asserted, but processing has not yet started
Active	processing has started but is not yet complete.

Servicing the GIC

Once configured, the GIC operates as follows:

- When an interrupt occurs the ARM11 CPU has to read the Interrupt Acknowledge register to get the interrupt number, then it can use a jump table to call the interrupt service routine.
- When the CPU has finished servicing an interrupt, it writes the interrupt number to the End of Interrupt register.
- If higher-priority interrupt(s) occur while the CPU is in an interrupt handler, the GIC will generate a pre-emptive IRQ and keep track of the previous active interrupt numbers.

The sequence diagram below shows a simple case without pre-emption:

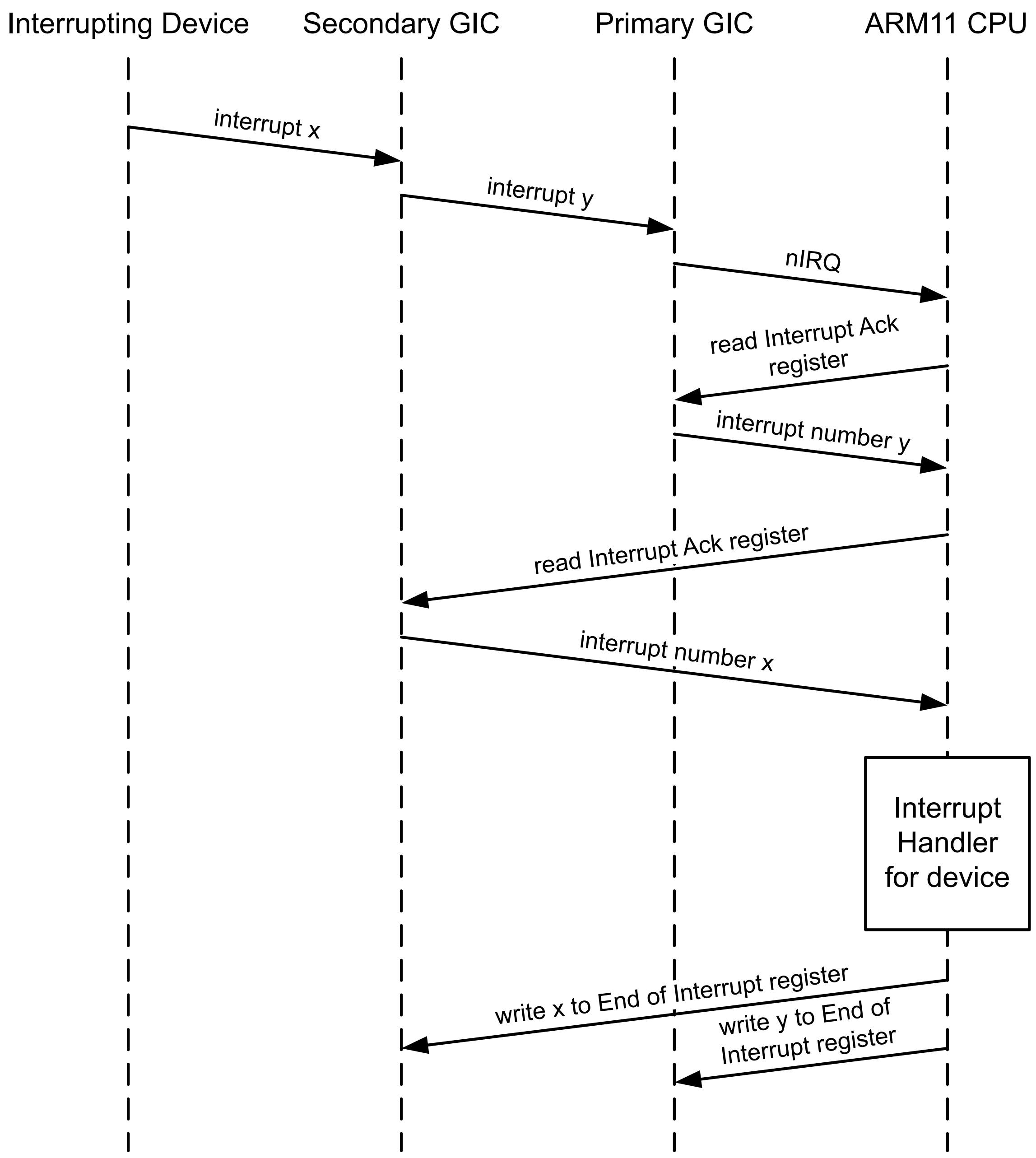
Interrupting Device
Secondary GIC
Primary GIC
ARM11 CPU
interrupt x
interrupt y
nIRQ
read Interrupt Ack register
interrupt number y
read Interrupt Ack register
interrupt number x
Interrupt Handler for device
write x to End of Interrupt register
write y to End of Interrupt register

Figure 4 Interrupt servicing

Calculating the next interrupt

The block diagram in Figure 5 on page 11 shows how the GIC calculates which interrupt to signal next.

The algorithm is described by this pseudo code:

```
if highest pending priority > priority mask
    if no interrupt currently being processed
        issue highest priority pending interrupt
    else if binary point mask calculation > running priority
        pre-empt with highest priority pending interrupt
    end if
end if
```

A diagram of the algorithm is shown in Figure 5 on page 11, below.

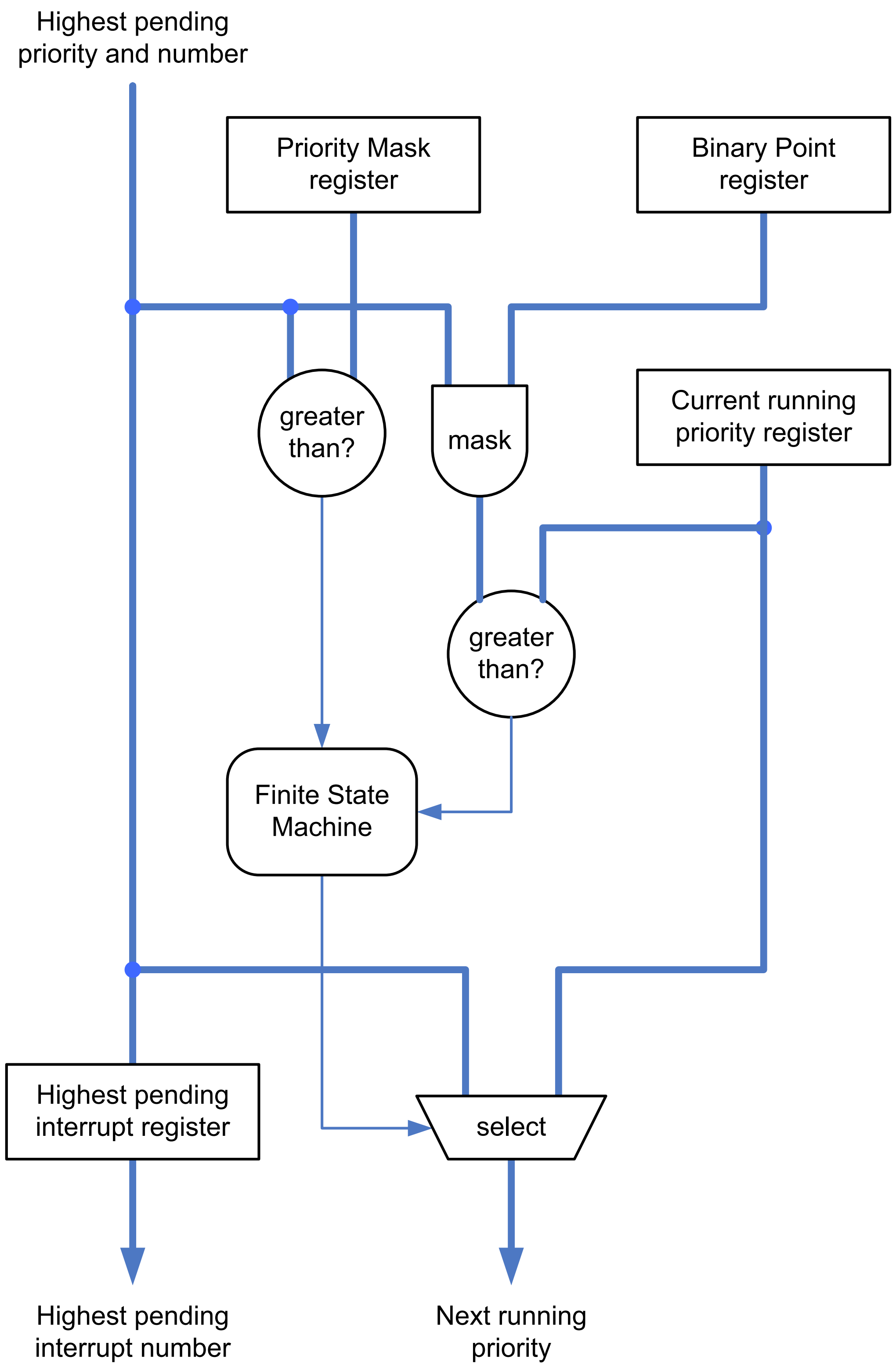
Highest pending priority and number
Priority Mask register
Binary Point register
greater than?
mask
Current running priority register
greater than?
Finite State Machine
Highest pending interrupt register
select
Highest pending interrupt number
Next running priority

Figure 5 Interrupt priority calculation

The finite state machine (FSM) is for each interrupt source x each CPU.

3.2 Primary Interrupt Controller

This section describes the interrupts input to the primary interrupt controller. There are:

- 32 'internal' interrupts 'private' to each CPU. Internal means the source is inside the MPCore. Private means each CPU can only see its own interrupts
- 16 'external' interrupts input to the test chip on signals INT[15:0]. These are sourced from the secondary interrupt controller on the EB and modified by the Core Tile PLD
- 16 'external' interrupts generated inside the test chip by the MPCore's performance monitor unit and the L2 cache.

Table 3 Primary interrupt controller sources

ID	Source	Description
63	L220 decode error	Decode error received on master ports from L3
62	L220 slave error	Slave error received on master ports from L3
61	L220 event counter	Event counter overflow/increment
60	MPCore SCU7 PMUIRQ[11]	Performance Monitor Unit (PMU) interrupts for Snoop Control Unit (SCU)
59	MPCore SCU6 PMUIRQ[10]	
58	MPCore SCU5 PMUIRQ[9]	
57	MPCore SCU4 PMUIRQ[8]	
56	MPCore SCU3 PMUIRQ[7]	
55	MPCore SCU2 PMUIRQ[6]	
54	MPCore SCU1 PMUIRQ[5]	
53	MPCore SCU0 PMUIRQ[4]	
52	MPCore CPU3 PMUIRQ[3]	Performance Monitor Unit (PMU) interrupts for CPUs
51	MPCore CPU2 PMUIRQ[2]	
50	MPCore CPU1 PMUIRQ[1]	
49	MPCore CPU0 PMUIRQ[0]	
48	0	Not used
47:32	INT[15:0]	Test chip interrupt inputs
31	Private nIRQ	Each CPU only sees its own legacy nIRQ pin
30	Private watchdog	Each CPU has its own watchdog timer
29	Private timer	Each CPU has its own timer
28:16	0	Spare - available for future private peripherals
15:0	Private inter-processor interrupts	See below

Inter-processor interrupts (IPI) can, for example, be used for interrupt-driven communication between CPUs in an SMP system. The GIC implements IPI using 16 software interrupts. To generate an IPI, an MPCore CPU must write to the primary GIC's Software Interrupt register specifying an interrupt ID in the range 0 to 15 and which CPU(s) will receive the interrupt.

3.3 Secondary to Primary Routing

The EB interrupt controller outputs 16 signals to the Core Tile, called T1_INT[15:0]. The PLD on the Core Tile then presents them to the MPCore test chip according to one of three modes selected by the INTMODE bits in the EB's SYS_PLD_CTRL1 register. The modes are:

Legacy Mode

The ARM11 MPCore supports legacy ARM interrupts where the nIRQ[3:0] signals input to the test chip are fed directly to the corresponding CPU, bypassing the Interrupt Distributor logic. When the INTMODE bits select legacy mode the Core Tile presents the interrupts on the nIRQ[3:0] pins of the test chip, instead of the INT[15:0] pins.

The INTMODE bits do not control whether the individual CPUs are configured for legacy ARM interrupts or not. CPUs that are configured for legacy ARM interrupt s ignore the INT[15:0] inputs.

The T1_INT[7:0] signals are only an input to the EB FPGA in legacy mode, when they are used to feed DCC interrupts from the test chip into the EB FPGA.

Legacy mode is the default for the ARM11 MPCore and the EB, and is used by the boot monitor. You may find legacy mode useful for single CPU operation or if you need to run without the Interrupt Distributor in development.

Normal mode without DCC

This is the recommended mode for SMP operation. It uses the Interrupt Distributor, and the DCC interrupts are not connected.

Other modes

Use of other modes is not recommended.

In addition, the Core Tile can route some interrupts to the nFIQ[3:0] signals, which are connected directly to the 'private' nFIQ input for the corresponding ARM11 CPU. Private means each CPU only sees its own nFIQ signal.

The FIQ or 'fast interrupt' exception is normally used for one low latency interrupt on ARM cores. We feel that most applications will not need to use FIQ as a fast interrupt because the ARM11 has a low latency mode, which allows early termination of multiple store and load memory accesses.

These three interrupt modes and FIQ enable/disable are selected by configuring the INTMODE[2:0] bits on the EB, which are bits 24:22 of the SYS_PLD_CTRL1 register at address 0x10000074. Before writing to this register you must unlock it by writing 0x0000A05F to the SYS_LOCK register at address 0x10000020.

The INTMODE bits are sent to the Core Tile PLD using a serial data stream. The serial data stream runs at 12MHz and sends 60 bits plus a frame sync bit from the EB FPGA to the Core Tile PLD. Data is sent immediately, there is no frame buffer, so there is only a worst-case delay of about 5us for the link plus a few 24MHz clock cycles for synchronization at each end. We recommend you delay the processor for 6us after changing the INTMODE bits.

Table 4 EB interrupt mode control bits

INTMODE	Mode	Comments
b000	Legacy mode without FIQ	Used by boot monitor
b001	Normal mode with DCC and no FIQ	Not recommended
b010	Normal without DCC and no FIQ	Recommended for SMP

Table 4 EB interrupt mode control bits (continued)

INTMODE	Mode	Comments
b011	Reserved for future use	Do not use
b100	Legacy mode with FIQ	Can be useful for a single CPU
b101	Normal mode with DCC and FIQ	Not recommended
b110	Normal mode without DCC and with FIQ	Not recommended
b111	Reserved for future use	Do not use

Note that you must also configure each CPU in the MPCore for 'normal' or 'legacy' mode. This will be explained later. You will probably not use legacy mode for an SMP application.

Table 5 below shows the limitations of the interrupt routing scheme currently implemented on the MPCore platform. Only CPU 0 and CPU 1 can receive IRQ (in legacy mode) and FIQ. We recommend that you use 'normal mode without DCC' for SMP applications.

Table 5 Interrupt routing limitations

EB FPGA	ARM11 MPCore inputs			
	Legacy mode	Normal mode with DCC	Normal mode without DCC	FIQ enabled
GIC1	nIRQ0	not used[a]	INT10	not used[b]
GIC2	not used[c]	INT12	INT12	nFIQ0
GIC3	nIRQ1	not used[a]	INT11	not used[b]
GIC4	not used[c]	INT13	INT13	nFIQ1

a. in 'normal mode with DCC' the nIRQ[3:0] signals are used by COMMRX[3:0].
b. nFIQ2 and nFIQ3 are driven by the MMC/SD card Primecell PL180.
c. in 'legacy mode' nIRQ2 and nIRQ3 are driven by the USB controller and audio codec.

Some interrupts do not need to use a secondary GIC since they are routed directly from the input to the output of the EB's secondary interrupt controller. The three tables below show the interrupts input into the MPCore test chip for the three modes. Note that some signals are not used, see the footnotes.

Table 6 Legacy mode

Interrupt source	EB tile site 1 signal	MPCore input
COMMRX0[a]	Z200	INT0[b]
COMMRX1[a]	Z201	INT1[b]
COMMRX2[a]	Z202	INT2[b]
COMMRX3[a]	Z203	INT3[b]
COMMTX0[a]	Z204	INT4[b]
COMMTX1[a]	Z205	INT5[b]
COMMTX2[a]	Z206	INT6[b]

Table 6 Legacy mode (continued)

Interrupt source	EB tile site 1 signal	MPCore input
COMMTX3[a]	Z207	INT7[b]
0	Z208	INT8[b]
0	Z209	INT9[b]
0	Z210	INT10[b]
0	Z211	INT11[b]
GIC2	Z212	INT12[b]
GIC4	Z213	INT13[b]
MCIINTR0	Z214	INT14[b]
MCIINTR1	Z215	INT15[b]
GIC1	Z208	nIRQ0
GIC3	Z209	nIRQ1
USBnINT	Z210	nIRQ2
AACIINTR	Z211	nIRQ3
GIC2[c]	Z212	nFIQ0
GIC4[c]	Z213	nFIQ1
MCIINTR0[c]	Z214	nFIQ2
MCIINTR1[c]	Z215	nFIQ3

a. these DCC interrupts are input to the EB FPGA but not used.
b. not used by the MPCore in legacy ARM interrupts mode.
c. if FIQs enabled using INTMODE2 = 1, otherwise the nFIQ is logic 1.

Table 7 New mode with DCC

Interrupt source	EB tile site 1 signal	MPCore input	Primary GIC
AACIINTR	Z200	INT0	ID 32
TIMERINT01	Z201	INT1	ID 33
TIMERINT23	Z202	INT2	ID 34
USBnINT	Z203	INT3	ID 35
UARTINT0	Z204	INT4	ID 36
UARTINT1	Z205	INT5	ID 37
RTCINT	Z206	INT6	ID 38
KMIINT0	Z207	INT7	ID 39
COMMTX0	-	INT8	ID 40
COMMTX1	-	INT9	ID 41
COMMTX2	-	INT10	ID 42

Table 7 New mode with DCC (continued)

Interrupt source	EB tile site 1 signal	MPCore input	Primary GIC
COMMTX3	-	INT11	ID 43
GIC2	Z212	INT12	ID 44
GIC4	Z213	INT13	ID 45
MCIINTR0	Z214	INT14	ID 46
MCIINTR1	Z215	INT15	ID 47
COMMRX0[a]	-	nIRQ0	CPU 0 ID 31
COMMRX1[a]	-	nIRQ1	CPU 1 ID 31
COMMRX2[a]	-	nIRQ2	CPU 2 ID 31
COMMRX3[a]	-	nIRQ3	CPU 3 ID 31
GIC2[b]	Z212	nFIQ0	-
GIC4[b]	Z213	nFIQ1	-
MCIINTR0[b]	Z214	nFIQ2	-
MCIINTR1[b]	Z215	nFIQ3	-

a. the MPCore Interrupt Distributor presents these as INT31.
b. if FIQs enabled using INTMODE2 = 1, otherwise the nFIQ is logic 1.

Table 8 New mode without DCC

Interrupt source	EB tile site 1 signal	MPCore input	Primary GIC
AACIINTR	Z200	INT0	ID 32
TIMERINT01	Z201	INT1	ID 33
TIMERINT23	Z202	INT2	ID 34
USBnINT	Z203	INT3	ID 35
UARTINT0	Z204	INT4	ID 36
UARTINT1	Z205	INT5	ID 37
RTCINT	Z206	INT6	ID 38
KMIINT0	Z207	INT7	ID 39
KMIINT1	Z208	INT8	ID 40
ETHINTR	Z209	INT9	ID 41
GIC1	Z210	INT10	ID 42
GIC3	Z211	INT11	ID 43
GIC2	Z212	INT12	ID 44
GIC4	Z213	INT13	ID 45
MCIINTR0	Z214	INT14	ID 46

Table 8 New mode without DCC (continued)

Interrupt source	EB tile site 1 signal	MPCore input	Primary GIC
MCIINTR1	Z215	INT15	ID 47
1	-	nIRQ0	CPU 0 ID 31
1	-	nIRQ1	CPU 1 ID 31
1	-	nIRQ2	CPU 2 ID 31
1	-	nIRQ3	CPU 3 ID 31
GIC2[a]	Z212	nFIQ0	-
GIC4[a]	Z213	nFIQ1	-
MCIINTR0[a]	Z214	nFIQ2	-
MCIINTR1[a]	Z215	nFIQ3	-

a. if FIQs enabled using INTMODE2 = 1, otherwise the nFIQ is logic 1.

For completeness, signal names for the interrupt outputs to the MPCore are listed in the table below. The EB signal 'T1Z200' means tile site 1 signal name Z200, which is header Z (HDRZ) pin 112 on the schematic. You do not need to keep track of the actual pin number because we refer to signal names like Z200 on the Logic Tile and Core Tile schematics.

Table 9 Interrupts to the Core Tile

FPGA signal	EB signal	HDRZ pin
T1_INT0	T1Z200	112
T1_INT1	T1Z201	110
T1_INT2	T1Z202	108
T1_INT3	T1Z203	106
T1_INT4	T1Z204	104
T1_INT5	T1Z205	102
T1_INT6	T1Z206	100
T1_INT7	T1Z207	98
T1_INT8	T1Z208	96
T1_INT9	T1Z209	94
T1_INT10	T1Z210	92
T1_INT11	T1Z211	90
T1_INT12	T1Z212	88
T1_INT13	T1Z213	86
T1_INT14	T1Z214	84
T1_INT15	T1Z215	82

3.4 Secondary Interrupt Controller

The table below shows the interrupt sources for the secondary interrupt controller in the EB FPGA. Note that the interrupt ID here is different to the interrupt ID in the primary interrupt controller; and the secondary GIC input sources 31:0 are not connected.

Table 10 Secondary interrupt controller sources

ID	Source	Description
95:84	0	Spare
83	PCI D	Interrupts from PCI expansion bus
82	PCI C	
81	PCI B	
80	PCI A	
79	T2_INT7	Interrupts from tile site 2
78	T2_INT6	
77	T2_INT5	
76	T2_INT4	
75	T2_INT3	
74	T2_INT2	
73	T2_INT1	
72	T2_INT0	
71	T1_INT7	Not used and set to 0
70	T1_INT6	
69	T1_INT5	
68	T1_INT4	
67	T1_INT3	
66	T1_INT2	
65	T1_INT1	
64	T1_INT0	
63	TSnKPADIRQ	Touch screen keypad interrupt
62	TSnPENIRQ	Touch screen pen down interrupt
61	USB	USB controller IC interrupt
60	Ethernet	Ethernet controller IC interrupt
59	DOC	Disk-on-Chip flash interrupt (Disk-on-Chip flash not fitted to EB rev D and later)
58	PISMO	Memory expansion board interrupt
57	0	Not used (intended for power fail)
56	DMAC	DMA controller interrupt

Table 10 Secondary interrupt controller sources (continued)

ID	Source	Description
55	CLCD	Color LCD display (from adapter board)
54	LCD	Character LCD display
53	KMI1	Keyboard / Mouse Interface interrupts
52	KMI0	
51	AACI	Audio CODEC controller interrupt
50	MCI1	Multimedia Card Interface interrupts
49	MCI0	
48	SCI	Smart Card Interface interrupt
47	UART3	Serial port interrupts
46	UART2	
45	UART1	
44	UART0	
43	SSP	Synchronous Serial Port interrupt
42	RTC	Real Time Clock interrupt
41	0	Reserved
40	GPIO2	General Purpose I/O controller interrupts
39	GPIO1	
38	GPIO0	
37	Timer 2 or 3	Timer interrupts
36	Timer 1 or 0	
35	COMMTX	Not used and set to 0.
34	COMMRX	
33	Software	Software interrupt. Enabling and disabling the software interrupt is done with the Enable Set and Enable Clear registers, for this bit. Triggering the interrupt is done by writing to the Soft Interrupt Set register.
32	Watchdog	Watchdog timer interrupt
31:0	0	Not used

4 Programmer's Reference

This section describes the MPCore platform interrupts from a software viewpoint.

4.1 Memory Map

To access an interrupt register use the base address in the table below plus the offset listed for the register's description. Note that the ARM core's CPSR register is not memory mapped.

Table 11 Interrupt control register addresses

Registers	Base Address
EB control registers	0x10000000
EB GIC1 CPU interface	0x10040000
EB GIC1 distributor	0x10041000
EB GIC2 CPU interface	0x10050000
EB GIC2 distributor	0x10051000
EB GIC3 CPU interface	0x10060000
EB GIC3 distributor	0x10061000
EB GIC4 CPU interface	0x10070000
EB GIC4 distributor	0x10071000
MPCore CPU interface	0x1F000100
MPCore alias for CPU interface 0[a]	0x1F000200
MPCore alias for CPU interface 1[a]	0x1F000300
MPCore alias for CPU interface 2[a]	0x1F000400
MPCore alias for CPU interface 3[a]	0x1F000500
MPCore distributor	0x1F001000

a. Aliased registers are provided to assist debug.

4.2 Primary GIC

This is the MPCore's Interrupt Distributor, see the MPCore technical reference manual [1] for details. Please note the following:

- When you enable the CPU interface you disable the ARM legacy interrupt (nIRQ pin) for that CPU
- When a CPU is in normal mode, an interrupt on its legacy nIRQ pin is mapped onto interrupt ID 31 for that CPU
- When FIQ is made non-maskable it means exactly that. It still causes an FIQ exception; it does not generate a non-maskable IRQ exception.

The register addresses implemented for the primary GIC are listed below. You can read the register descriptions for the secondary GIC in the next section. The registers are identical apart from the number of each and the number of CPUs supported.

Table 12 Differences between primary and secondary GIC

	Interrupt ID used	Number of CPUs
Primary GIC	0 - 63	4
Secondary GIC	32 - 95	1

CPU Interface registers

The register addresses below are given as offsets from the MPCore CPU Interface base address listed in Table 11 on page 20, above. For debug purposes, each CPU also has access to the CPU Interface registers of other CPUs, using the aliased addresses.

0x000	CPU Control
0x004	Priority Mask
0x008	Binary Point
0x00C	Interrupt Acknowledge
0x010	End of Interrupt (EOI)
0x014	Running Interrupt
0x018	Highest Pending Interrupt

Distribution registers

The register addresses below are given as offsets from the MPCore Distributor base address listed in Table 11 on page 20, above.

0x1000	Distributor Control
0x1004	Controller Type
0x1100 - 0x1104	Set Enable
0x1180 - 0x1184	Clear Enable
0x1200 - 0x1204	Set Pending
0x1280 - 0x1284	Clear Pending
0x1300 - 0x1304	Active
0x1400 - 0x143C	Priority
0x1800 - 0x183C	CPU Targets
0x1C00 - 0x1C0C	Configuration
0x1F00	Software Interrupt
0x1FE0 - 0x1FEC	Peripheral Identification

0x1FF0 - 0x1FFC
PrimeCell Identification

4.3 Secondary GIC

This section describes the registers for the secondary interrupt controller in the EB's FPGA. Remember, each of the 4 GICs has one distributor and one CPU interface.

Meanings are for the secondary GIC not the MPCore's primary GIC, although the diagrams are the same.

Where a register configures more than one interrupt, the lowest interrupt ID number is configured by bit 0 of the register.

The address for each register, offset from the base address, is shown on the left.

SBZ means 'should be zero'.

CPU Interface registers

The register addresses below are given as offsets from the EB GIC CPU Interface base address listed in Table 11 on page 20.

0x0000 CPU Control

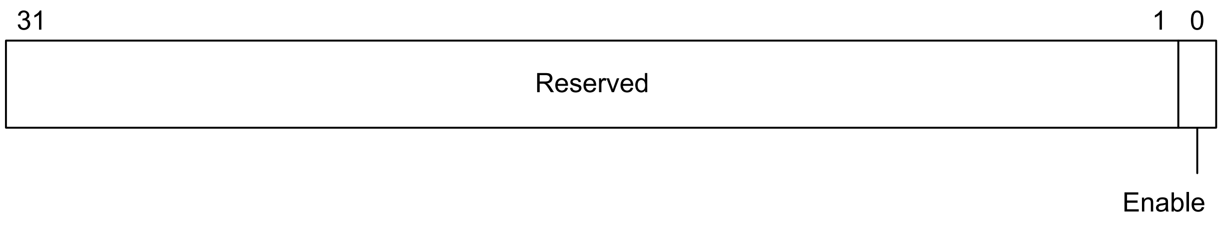

Figure 6 0x0000 CPU control

Write a 1 to bit 0 to enable the CPU interface for this GIC.

Bits 31:1 are not used and should be set to zero.

0x0004 Priority Mask

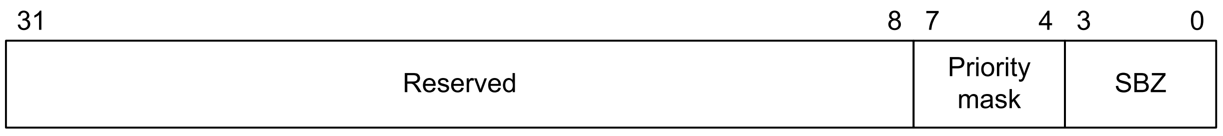

Figure 7 0x0004 Priority mask

The priority mask is stored in bits 7:4 of this register. Bits 31:8 are not used and should be set to zero. Bits 3:0 may be used in the future, so software should write 0xFF to this register then read it back to determine how many bits are implemented. Note that zero is the highest priority.

The Priority Mask is used to prevent interrupts from being sent to the processor. The CPU Interface asserts an interrupt request if the priority of the highest pending interrupt sent by the Interrupt Distributor is greater than the priority in the Priority Mask register. For example, a priority mask value of 0x0 means all interrupts are masked; and a priority value of 0xF means interrupts with priority 0xF are masked but priorities 0x0 to 0xE are not masked.

0x0008 Binary Point

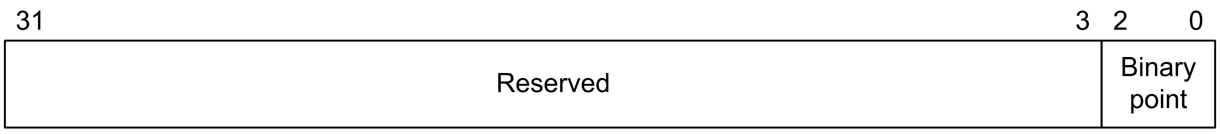

Figure 8 0x0008 Binary point

The Binary Point register sets the position of a 'binary point' that controls which bits of an interrupt's priority are compared for pre-emption purposes. This allows software to adjust the level of interrupt pre-emption in the system.

You could think of this register as a 'pre-emption mask'. The Priority Mask register sets the priority below which an interrupt does not occur; and the Binary Point register sets the priority below which an interrupt does not cause pre-emption.

Table 13 Binary Point bit values assignment

Bit value	Meaning
b011	All priority bits are compared for pre-emption
b100	Only bits [7:5] of priority are compared for pre-emption
b101	Only bits [7:6] of priority are compared for pre-emption
b110	Only bit [7] of priority is compared for pre-emption
b111	No pre-emption is performed

For an example of the Binary Point register in action, see Figure 9, below, and the description on the next page.

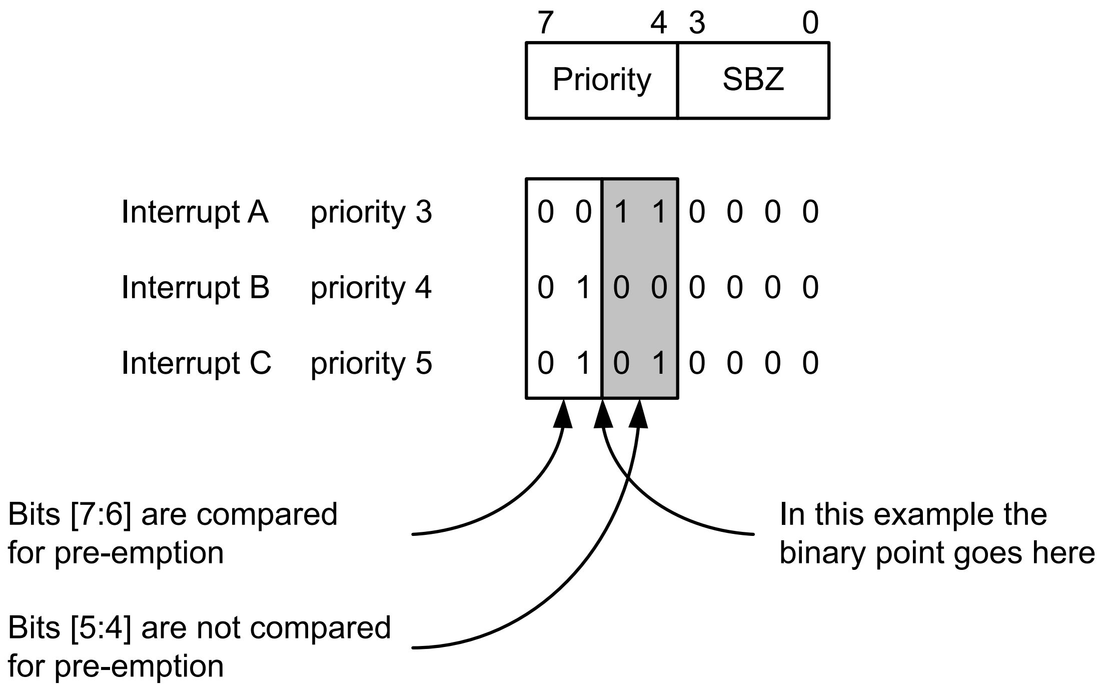
7
4
3
0
Priority
SBZ
Interrupt A priority 3 0 0 1 1 0 0 0 0
Interrupt B priority 4 0 1 0 0 0 0 0 0
Interrupt C priority 5 0 1 0 1 0 0 0 0
Bits [7:6] are compared for pre-emption
Bits [5:4] are not compared for pre-emption
In this example the binary point goes here

Figure 9 Binary point example

In this example there are three interrupts A, B, C that use 4 priority bits, and the Binary Point register is set to 0x00000005 so the 'binary point' is set between bits 6 and 5.

Zero (b0000) is the highest priority, so A is a higher priority than B and C. For pre-emption, any bits to the right of the 'binary point' are ignored, so A can interrupt B or C, but B cannot interrupt C.

If interrupt A is active and interrupts B and C are pending, when A has completed B will be taken as it has a higher priority than C.

0x000C Interrupt Acknowledge

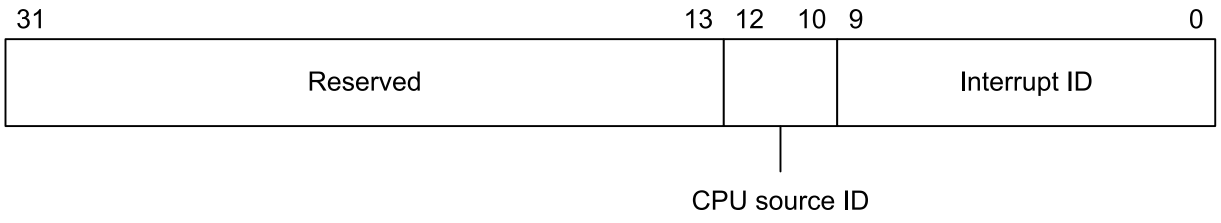

Figure 10 0x000C Interrupt acknowledge

The processor reads the interrupt number from here. The CPU source ID is not used in the EB; it is only used in the MPCore GIC for inter-processor interrupt IDs 0 - 15. Read only.

The processor responds to an interrupt by reading this register from the interrupting GIC that returns the interrupt number. Pre-empted interrupts are recorded as 'active'.

In the event the interrupt priorities are changed before the processor reads the interrupt number, and the interrupt has become a lower priority, the interrupt number returned is 1023 meaning 'spurious' interrupt.

0x0010 End of Interrupt (EOI)

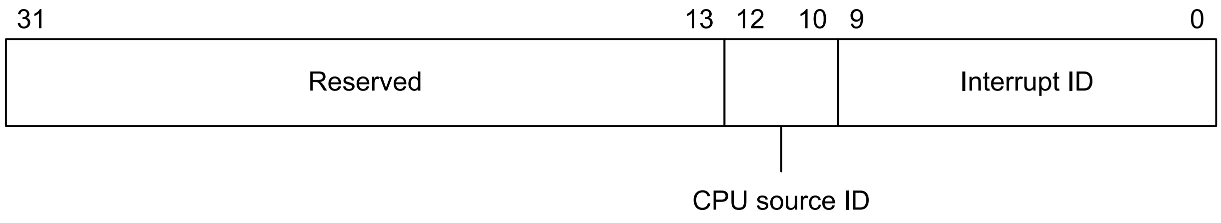

Figure 11 0x0010 End of interrupt

When a CPU writes the interrupt number to this register it ends the interrupt. Write only.

When the interrupt has been completed by the processor, it writes the interrupt number to this register in the interrupting GIC.

0x0014 Running Interrupt

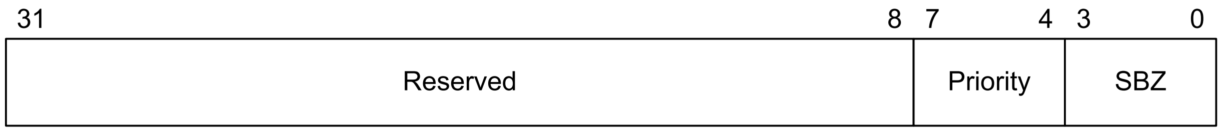

Figure 12 0x0014 Running interrupt

Contains the priority level of the currently running interrupt. Read only.

0x0018 Highest Pending Interrupt

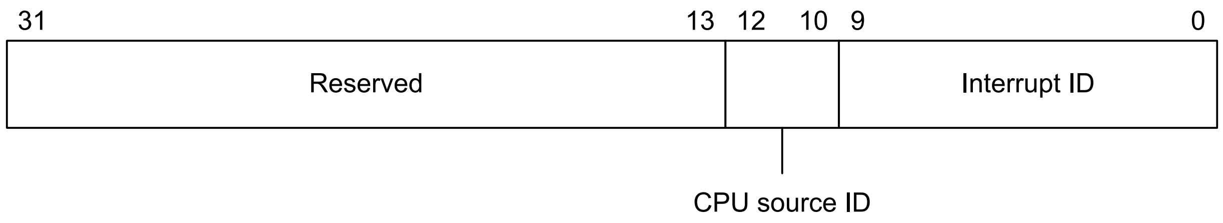

Figure 13 0x0018 Highest pending interrupt

Contains the number of the highest pending interrupt.

0x001C - 0x00FF

Reserved

0x0100 - 0x0FFF

Reserved

This area of memory is not fully address decoded and contains aliases (copies) of the CPU Interface registers.

Distribution registers

The register addresses below are given as offsets from the EB GIC Distributor base address listed in *Interrupt control register addresses* on page 20.

0x1000 Distributor Control

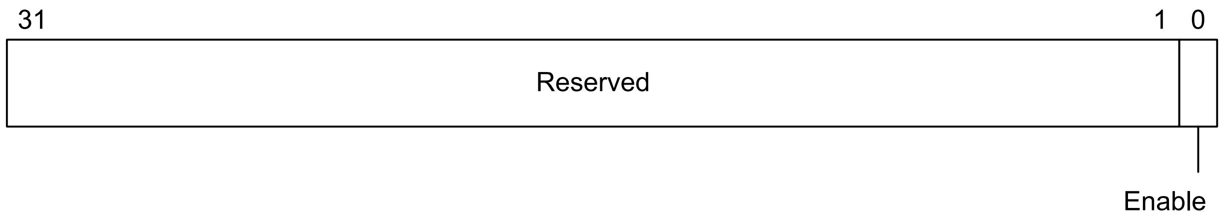

Figure 14 0x1000 Distributor control

Write a 1 to bit 0 to enable interrupts. Bits 31:1 are not used and should be set to zero.

0x1004 Controller Type

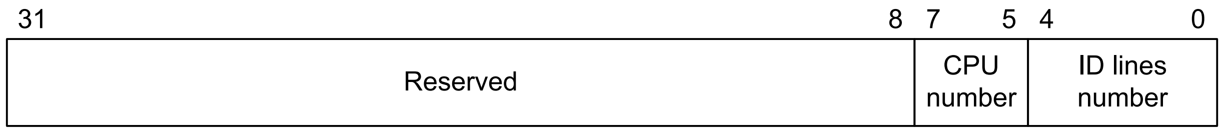

Figure 15 0x1004 Controller type

Identifies the number of CPUs and interrupts in this implementation of the GIC. In this case it has a fixed value of 0x00000002, which means 1 CPU and 64 external interrupt inputs. Read only.

0x1008 - 0x10FC

Reserved

0x1100 - 0x1108

Set Enable

Read from these registers to determine which interrupts are enabled. A bit set to 1 means enabled. Write a 1 to a bit in a Set Enable register to enable the corresponding interrupt.

The first register at offset 0x1100 is for interrupt ID 0-31. Note that ID 0 - 31 are not used in the EB GIC. Bit 0 corresponds to interrupt ID 0, 32 or 64.

0x110C - 0x117C

Reserved

0x1180 - 0x1188

Clear Enable

Write a 1 to a bit in a Clear Enable register to disable the corresponding interrupt. You can read which interrupts are enabled from these registers, the same as the Set Enable registers.

The first register at offset 0x1180 is for interrupt ID 0-31. Note that ID 0 - 31 are not used in the EB GIC. Bit 0 corresponds to interrupt ID 0, 32 or 64.

0x118C - 0x11FC

Reserved

0x1200 - 0x1208

Set Pending

Read from these registers to determine which interrupts are Pending. A bit set to 1 means the corresponding interrupt is in Pending state. Write a 1 to a bit in a Set Pending register to put the corresponding interrupt into Pending state.

The first register at offset 0x1200 is for interrupt ID 0-31. Note that ID 0 - 31 are not used in the EB GIC. Bit 0 corresponds to interrupt ID 0, 32 or 64.

0x120C - 0x127C

Reserved

0x1280 - 0x1288

Clear Pending

Write a 1 to a bit in a Clear Pending register to make an interrupt in Pending state change to Inactive state. You can read which interrupts are in Pending state from these registers, the same as the Set Pending registers.

The first register at offset 0x1280 is for interrupt ID 0-31. Note that ID 0 - 31 are not used in the EB GIC. Bit 0 corresponds to interrupt ID 0, 32 or 64.

0x128C - 0x12FC

Reserved

0x1300 - 0x1308

Active

Read from these registers to determine which interrupts are Active. A bit set to 1 means the corresponding interrupt is in Active state.

The first register at offset 0x1300 is for interrupt ID 0 - 31. Note that ID 0 - 31 are not used in the EB GIC. Bit 0 corresponds to interrupt ID 0, 32 or 64. Read only.

0x130C - 0x13FC

Reserved

0x1400 - 0x145C

Priority

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
ID n+3 Priority	SBZ	ID n+2 Priority	SBZ	ID n+1 Priority	SBZ	ID n Priority	SBZ

Figure 16 0x1400 – 0x145C Priority

These registers set the priority level for each interrupt.

Priority is a 4-bit number where zero is the highest priority. Future implementations of the GIC may use 8 bits for priority, so the priority is stored in the most significant bits of an 8-bit field. In this case the priority is stored in bits 7:4, while bits 3:0 are not used and set to zero.

Each 32-bit register holds the priority for 4 interrupts. The first register at offset 0x1400 is for interrupt ID 0 - 3. Note that ID 0 - 31 are not used in the EB GIC. For example, for interrupt ID 32 - 35 stored in the register at address offset 0x1420:

- Interrupt ID 32 is stored in bits 7:4, bits 3:0 are unused and set to zero
- Interrupt ID 33 is stored in bits 15:12, bits 11:8 are unused and set to zero
- Interrupt ID 34 is stored in bits 23:20, bits 19:16 are unused and set to zero
- Interrupt ID 35 is stored in bits 31:28, bits 27:24 are unused and set to zero.

0x1460 - 0x17FC

Reserved

0x1800 - 0x185C

CPU Targets

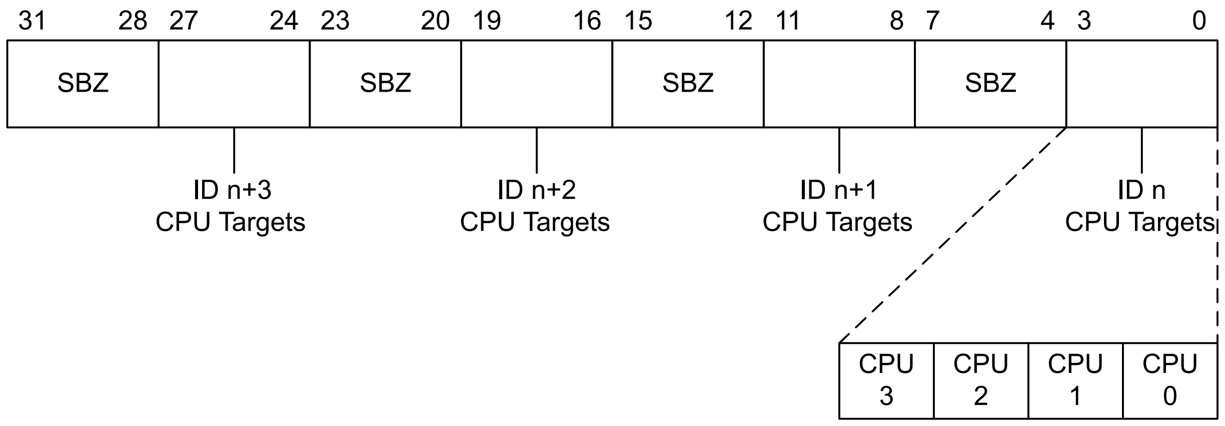

Figure 17 0x1800 – 0x185C CPU targets

These registers configure which CPU can receive an interrupt, for each interrupt ID. The first register is for interrupt ID 0 - 3. Note that ID 0 - 31 are not used in the EB GIC.

Each register can store a bit-map for 4 interrupts x 8 CPUs. The EB GICs have been implemented for only one CPU (CPU 0) so you must initialize these registers to 0x01010101.

0x1860 - 0x1BFC

Reserved

0x1C00 - 0x1C14

Configuration

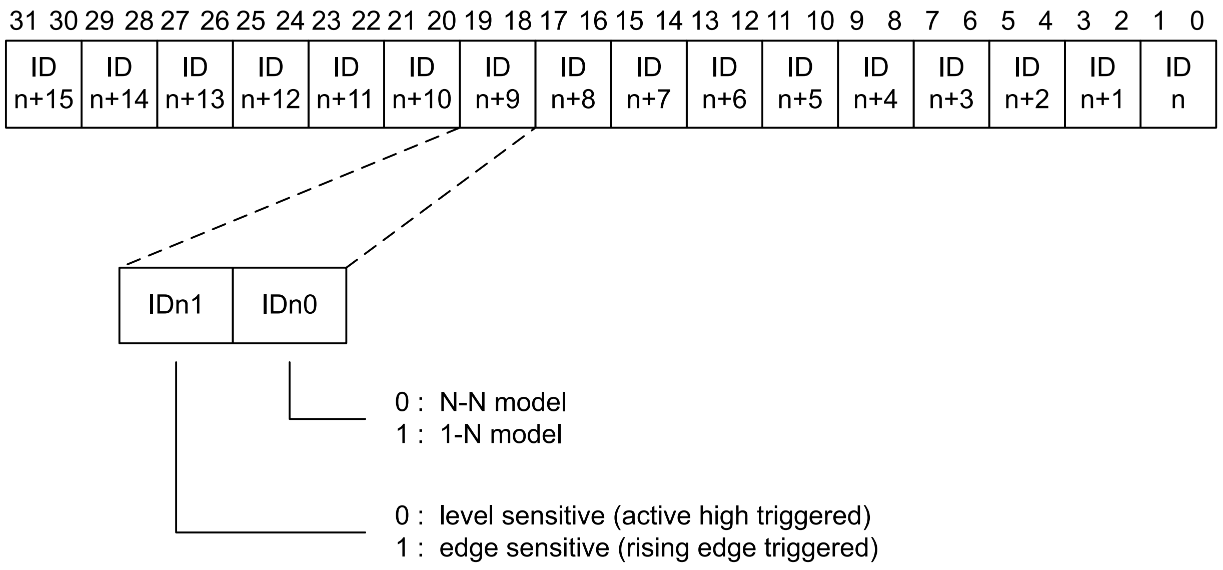

Figure 18 0x1C00 – 0x1C14 Configuration

The Interrupt Configuration registers have two bits IDn[1:0] for each interrupt ID n. These two bits set each interrupt to be level or edge sensitive, and which 'software model' is used:

- 1-N model means only one CPU takes the interrupt. An interrupt that is taken on any CPU clears the pending status on all CPUs. The EB GICs can only use this option
- N-N model will be deprecated in future versions of the GIC and should not be used.

For example, for interrupt ID 32 set bits 1:0 of the register at offset 0x1C08 to b01 for level-sensitive or b11 for edge-sensitive. ID33 uses bits 3:2, ID34 uses bits 5:4, and so on. If you want to make all the interrupts level sensitive (this is the default in the EB boot monitor) then write 0x55555555 to the Configuration registers.

The first register at offset 0x1C00 is for interrupt ID 0 - 15. Note that ID 0 - 31 are not used in the EB GIC.

0x1C18 - 0x1EFC

Reserved

0x1F00

Software Interrupt

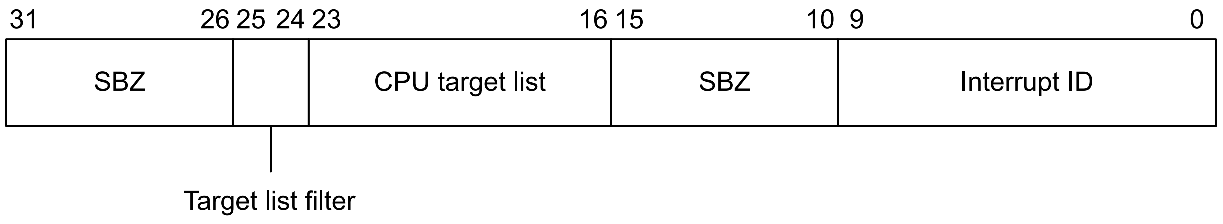

Figure 19 0x1F00 Software interrupt

Write only. Write to this register to trigger an interrupt ID 32 - 95.

If you want to trigger an inter-processor interrupt (IPI) then you must write to the Software Interrupt register in the MPCore's Interrupt Distributor, specifying interrupt ID 0 - 15.

For example, write 0x02000021 to the EB's Software Interrupt register to trigger interrupt ID 33. Bits 9:0 contain the interrupt ID, bits 25:24 are set to 10 (to ignore the list of CPU targets in bits 23:16), and the remaining bits (31:26 and 23:10) should be set to zero. You should see the Set Pending register at offset 0x1204 be set to 0x00000002.

There is only 1 CPU in the EB GIC so features such as 'CPU target list' are only needed in the MPCore. For full details see the ARM11 MPCore technical reference manual [1].

0x1F04 - 0x1FDC

Reserved

0x1FE0 - 0x1FEC

Peripheral Identification

0x1FF0 - 0x1FFC

PrimeCell Identification

Eight 8-bit read only registers have been implemented at addresses 0x1FE0, 0x1FE4, 0x1FE8, 0x1FEC, 0x1FF0, 0x1FF4, 0x1FF8, 0x1FFC. These registers contain the Primecell number, version and company.

5 Example Software

For an example of interrupts in 'legacy mode' see the EB boot monitor file sys_interrupts_eb.c provided on the Versatile 3.0 CD and later versions.

This section describes the example software provided with this application note, which uses 'normal mode' interrupts. The files in the example software provided are as follows:

Table 14 Example software

build.bat	Builds the software
EB_GIC.c	Sets up a secondary interrupt controller (GIC 1) in the EB FPGA to generate an IRQ to CPU 0
EB_GIC.h	Register definitions for secondary interrupt controllers (GIC 1 - 4) in the EB FPGA
EB_GIC_test.axf	Executable image
EB_timer.c	Sets up EB Timer0 to run at 32.768kHz and generate an interrupt 2 times per second
EB_timer.h	Register definitions for EB Timer0
handlers.s	Dummy exception vector handlers. The IRQ handler is in a separate C module
InitStacks.s	Initialise the stack pointer and status register (CPSR)
IRQ_handler.c	Simple IRQ handler that clears the interrupt on the primary and secondary interrupt controllers and Timer0. To show that interrupts are running it flashes EB general purpose LED 0 (next to DIP switch S6)
main.c	To show that the program is running it flashes EB general purpose LED 7
MPCore_GIC.c	This code sets up the primary interrupt controller in the MPCore test chip to generate an IRQ to CPU 0, from the output of GIC1 on the EB motherboard
MPCore_GIC.h	Register definitions for primary interrupt controller in the MPCore test chip
retarget.c	This code retargets the __user_initial_stackheap function, and calls functions that enable the timer and interrupt controllers before entering main()
scatter.scat	Scatter-loader description file to place the example program, stacks and heap all in the EB's SDRAM
vectors.s	Exception vector table and dummy exception handlers

6 Glossary

Table 15 Glossary

AMP	Asymmetric Multiprocessing Similar to symmetric multiprocessing (SMP) but one or more CPUs are not perfectly symmetrical. The asymmetric CPU(s) might run different software or have dedicated I/O such as an interrupt signal
CPU	Central Processing Unit An MPCore can contain one or more CPUs, for example 4 x ARM11
DCC	Debug Communications Channel DCC enables you to communicate with your application over the JTAG interface, without stopping the processor. DCC is implemented in the ARM core as a 32-bit transmit (TX) register and transmit flag, and a 32-bit receive (RX) register and receive flag. A small program has to run in target memory, which either polls the TX and RX flags or responds to the COMMTX and COMMRX interrupt signals generated by the DCC logic. Using interrupts is faster than polling.
EB	Emulation Baseboard
FIQ	Fast Interrupt When the nFIQ signal is active and enabled, the CPU takes an FIQ exception. FIQ is designed to provide a guaranteed low-latency interrupt for time-critical applications. If FIQ is used in a system it is normally only used for one interrupt
GIC	Generic Interrupt Controller GIC refers to the Interrupt Distributor or distributed interrupt controller that is part of the ARM11 MPCore
IPI	Inter-Processor Interrupt
IRQ	An interrupt generated by one CPU and received by one or more other CPU(s)
NMI	Non-Maskable Interrupt The ARM11 MPCore's nFIQ signals can be made non-maskable
Private	An area of memory or an interrupt signal that can only be used by one CPU
SMP	Symmetric Multiprocessing A symmetric multiprocessor has two or more identical processors connected to a single shared memory and I/O system
WFI	Wait For Interrupt This ARM11 MPCore instruction makes the CPU suspend execution until it receives an interrupt. The CPU's clock is stopped to save power

7 References

Here is a list of useful reference material, some of which are referred to in this document:

Table 16 References

[1]	ARM11 MPCore Processor Technical Reference Manual ARM DDI 0360
[2]	Application Note 152 : Using a CT11MPCore with the RealView Emulation Baseboard ARM DAI 0152
[3]	Core Tile for ARM11 MPCore User Guide ARM DUI 0318
[4]	Emulation Baseboard User Guide ARM DUI 0303
[5]	Technical Support FAQs http://www.arm.com/support/versatile.html