StreamEngine® 5160 and 5170 Communication Processors
Ten-Way Multithreaded Processors Optimized for Network Connectivity

1.0   Product Highlights

The StreamEngine 5160 and 5170 Communication Processors (IP51xx) are revolutionary new platforms from Ubicom, designed to provide highly integrated solutions for applications at the "edge" of Internet connectivity, including routers, bridges, gateways, and a wide variety of embedded networked client solutions.

The IP51xx is optimized for efficient network processing in embedded solutions. Its development has led to the definition of a new microprocessor architecture: Multithreaded Architecture for Software I/O (MASI). Many MASI concepts were pioneered in the Ubicom IP2000™ family of processors, but the IP3000 family dramatically extended those techniques by introducing hardware support for eight threads operating with no context switching overhead, as well as three-operand and memory-to-memory operations. The IP5000 family further expands this capability by extending the number of threads from eight to ten.

The IP51xx, then, is a 32-bit CPU supporting ten-way multithreaded operation. It provides for up to ten real-time tasks to execute in a completely deterministic fashion. In essence, the IP51xx supports running a different thread on every clock, but without the overhead for context switching typical with traditional microprocessor architectures. To the system designer, the IP51xx appears as if there were ten processors on the chip.

The multithreaded and deterministic nature of the IP51xx processor provides for integration of numerous functions on chip — some with on-chip hardware assist and some entirely in software — as threads, including the ability to support interfaces such as MII, USB, GPSI, Utopia, PCMCIA, IDE, PCM Highway, RGMII, and PCI interfaces. This yields both a high degree of flexibility and reduces die size, as it eliminates the need for many on-chip dedicated hardware blocks for specific functions.

The IP51xx employs a three-operand and memory-to-memory architecture, utilizing on-chip program and data memory support. This scheme enables highly efficient data movement and processing on data. The result is that the IP51xx is designed to support packet processing and transfers at wire speeds, eliminating the need for large data buffers typically found in use with traditional RISC-based microprocessors.

Key Features:
- 32-bit Multithreaded CPU — 270 MIPS or 350 MIPS
- IP51xx is optimized for communication processing
  - Ten-way fine-grained multithreading
  - Deterministic execution on all threads
  - Zero overhead full context switching
  - Programmable MIPS per thread
  - Instruction Set Architecture optimized for packet processing
    - Memory-to-memory architecture, powerful addressing modes
    - Small, fast instruction set, strong bit manipulation
    - Reduced code size vs. RISC CPUs
- On-chip program / data memory
  - Eliminates cache miss penalties
- Highly configurable I/O support
  - Many combinations of software I/O:
    - Utopia, PCMCIA, IDE / ATAPI
    - PCM Highway, UART, SPI, I^{2}C
  - MII ports of 10 / 100 PHY
  - Two Serdes for fast serial I/O:
    - USB, GPSI
    - SPI, UART
  - RGMII port
  - PCI host
  - High-Speed USB port
- Additional key hardware
  - True random number generator for software-implemented encryption / security (32-bit seed)
- Independent I/O and core CPU clocking
  - Separate phase-locked loops (PLLs)
  - Programmable multipliers & dividers
  - Single low-cost crystal (12 MHz)

To further optimize the IP51xx for networking infrastructure and embedded client solutions, the processor includes several key hardware support blocks, including true random number generator and fixed-point multiply / accumulate (MAC) units. The random number generator facilitates robust software implementation of common encryption / security protocols critical to the continued growth of communication processing.
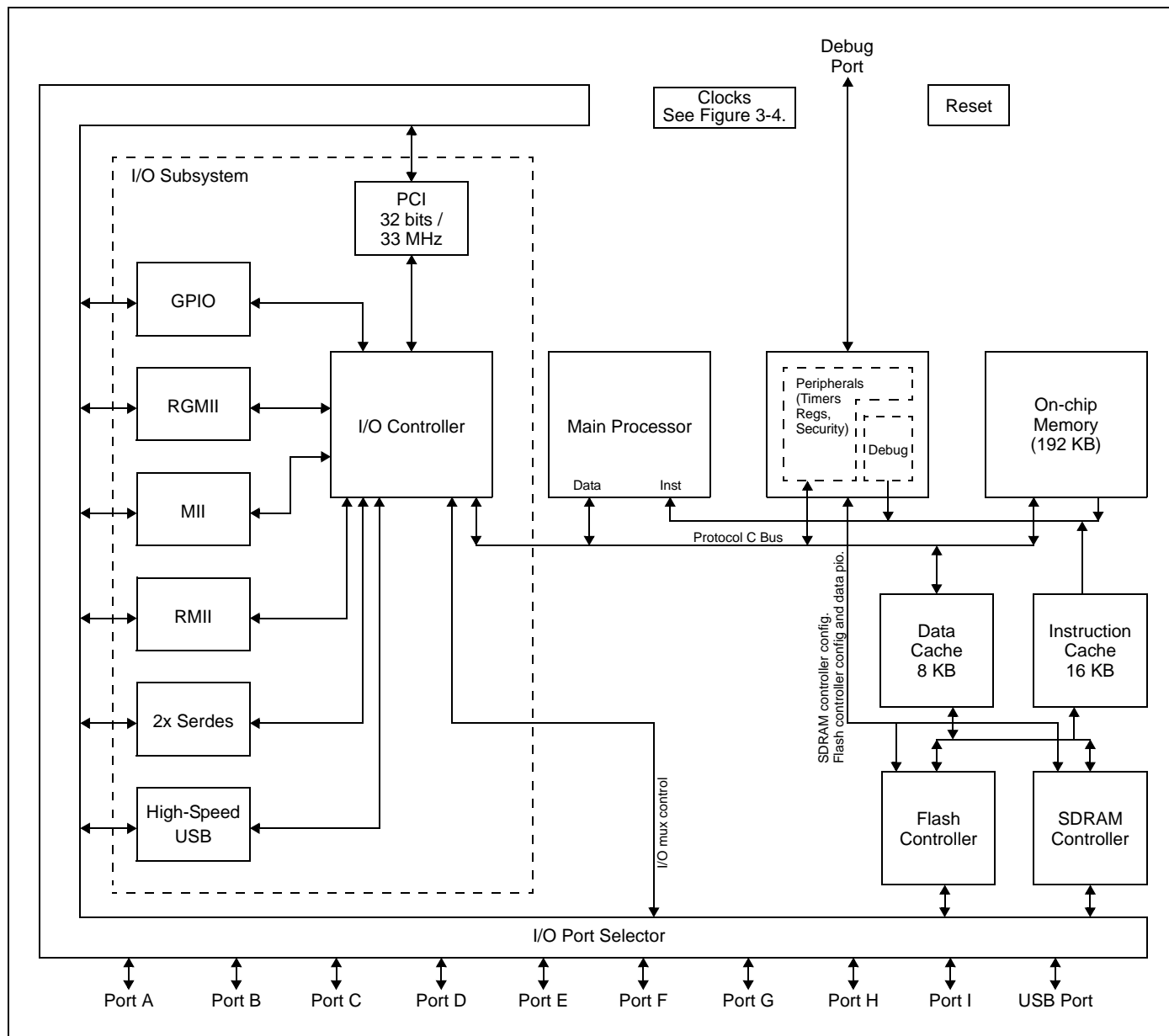
                1

Figure 1-1 IP51xx Block Diagram

2.0 Pin Definitions

2.1 Pin Assignments (256-Pin BGA)

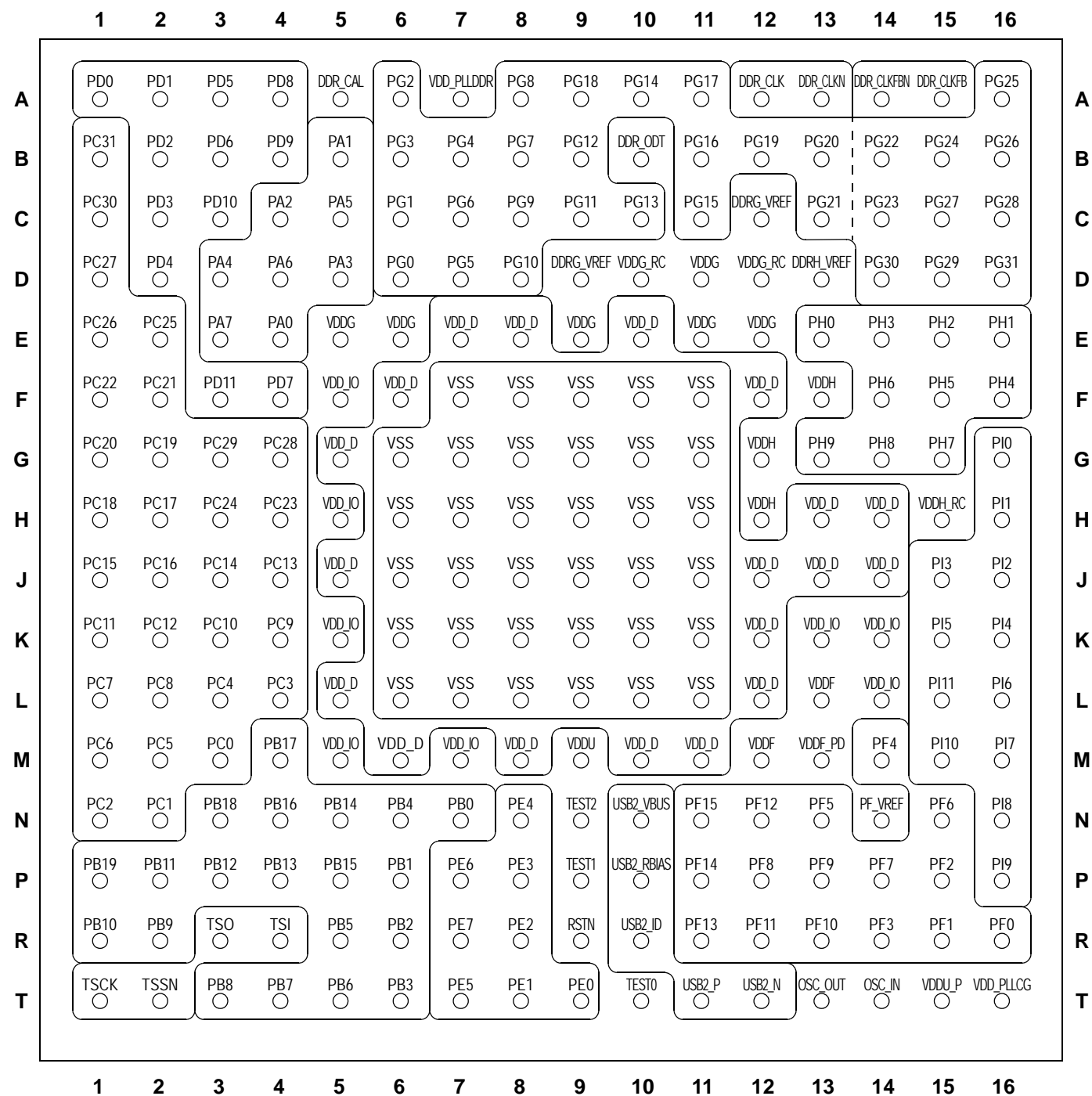|	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	PD0	PD1	PD5	PD8	DDR_CAL	PG2	VDD_PLLDDR	PG8	PG18	PG14	PG17	DDR_CLK	DDR_CLKN	DDR_CLKFBN	DDR_CLKFB	PG25
B	PC31	PD2	PD6	PD9	PA1	PG3	PG4	PG7	PG12	DDR_ODT	PG16	PG19	PG20	PG22	PG24	PG26
C	PC30	PD3	PD10	PA2	PA5	PG1	PG6	PG9	PG11	PG13	PG15	DDRG_VREF	PG21	PG23	PG27	PG28
D	PC27	PD4	PA4	PA6	PA3	PG0	PG5	PG10	DDRG_VREF	VDDG_RC	VDDG	VDDG_RC	DDRH_VREF	PG30	PG29	PG31
E	PC26	PC25	PA7	PA0	VDDG	VDDG	VDD_D	VDD_D	VDDG	VDD_D	VDDG	VDDG	PH0	PH3	PH2	PH1
F	PC22	PC21	PD11	PD7	VDD_IO	VDD_D	VSS	VSS	VSS	VSS	VSS	VDD_D	VDDH	PH6	PH5	PH4
G	PC20	PC19	PC29	PC28	VDD_D	VSS	VSS	VSS	VSS	VSS	VSS	VDDH	PH9	PH8	PH7	PI0
H	PC18	PC17	PC24	PC23	VDD_IO	VSS	VSS	VSS	VSS	VSS	VSS	VDDH	VDD_D	VDD_D	VDDH_RC	PI1
J	PC15	PC16	PC14	PC13	VDD_D	VSS	VSS	VSS	VSS	VSS	VSS	VDD_D	VDD_D	VDD_D	PI3	PI2
K	PC11	PC12	PC10	PC9	VDD_IO	VSS	VSS	VSS	VSS	VSS	VSS	VDD_D	VDD_IO	VDD_IO	PI5	PI4
L	PC7	PC8	PC4	PC3	VDD_D	VSS	VSS	VSS	VSS	VSS	VSS	VDD_D	VDDF	VDD_IO	PI11	PI6
M	PC6	PC5	PC0	PB17	VDD_IO	VDD_D	VDD_IO	VDD_D	VDDU	VDD_D	VDD_D	VDDF	VDDF_PD	PF4	PI10	PI7
N	PC2	PC1	PB18	PB16	PB14	PB4	PB0	PE4	TEST2	USB2_VBUS	PF15	PF12	PF5	PF_VREF	PF6	PI8
P	PB19	PB11	PB12	PB13	PB15	PB1	PE6	PE3	TEST1	USB2_RBIAS	PF14	PF8	PF9	PF7	PF2	PI9
R	PB10	PB9	TSO	TSI	PB5	PB2	PE7	PE2	RSTN	USB2_ID	PF13	PF11	PF10	PF3	PF1	PF0
T	TSCK	TSSN	PB8	PB7	PB6	PB3	PE5	PE1	PE0	TEST0	USB2_P	USB2_N	OSC_OUT	OSC_IN	VDDU_P	VDD_PLLCG

Figure 2-1 IP51xx BGA Pin Assignments (Top View Through Package)

Figure 2-2 IP51xx BGA Pin Assignments (Bottom View)

Table 2-1 Pin Assignments (sorted by pin number)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
A1	PD0	E1	PC26	J1	PC15	N1	PC2
A2	PD1	E2	PC25	J2	PC16	N2	PC1
A3	PD5	E3	PA7	J3	PC14	N3	PB18
A4	PD8	E4	PA0	J4	PC13	N4	PB16
A5	DDR_CAL	E5	VDDG	J5	VDD_D	N5	PB14
A6	PG2	E6	VDDG	J6	VSS	N6	PB4
A7	VDD_PLLDDR	E7	VDD_D	J7	VSS	N7	PB0
A8	PG8	E8	VDD_D	J8	VSS	N8	PE4
A9	PG18	E9	VDDG	J9	VSS	N9	TEST2
A10	PG14	E10	VDD_D	J10	VSS	N10	USB2_VBUS
A11	PG17	E11	VDDG	J11	VSS	N11	PF15
A12	DDR_CLK	E12	VDDG	J12	VDD_D	N12	PF12
A13	DDR_CLKN	E13	PH0	J13	VDD_D	N13	PF5
A14	DDR_CLKFBN	E14	PH3	J14	VDD_D	N14	PF_VREF
A15	DDR_CLKFB	E15	PH2	J15	PI3	N15	PF6
A16	PG25	E16	PH1	J16	PI2	N16	PI8
B1	PC31	F1	PC22	K1	PC11	P1	PB19
B2	PD2	F2	PC21	K2	PC12	P2	PB11
B3	PD6	F3	PD11	K3	PC10	P3	PB12
B4	PD9	F4	PD7	K4	PC9	P4	PB13
B5	PA1	F5	VDD_IO	K5	VDD_IO	P5	PB15
B6	PG3	F6	VDD_D	K6	VSS	P6	PB1
B7	PG4	F7	VSS	K7	VSS	P7	PE6
B8	PG7	F8	VSS	K8	VSS	P8	PE3
B9	PG12	F9	VSS	K9	VSS	P9	TEST1
B10	DDR_ODT	F10	VSS	K10	VSS	P10	USB2_RBIAS
B11	PG16	F11	VSS	K11	VSS	P11	PF14
B12	PG19	F12	VDD_D	K12	VDD_D	P12	PF8
B13	PG20	F13	VDDH	K13	VDD_IO	P13	PF9
B14	PG22	F14	PH6	K14	VDD_IO	P14	PF7
B15	PG24	F15	PH5	K15	PI5	P15	PF2
B16	PG26	F16	PH4	K16	PI4	P16	PI9
C1	PC30	G1	PC20	L1	PC7	R1	PB10
C2	PD3	G2	PC19	L2	PC6	R2	PB9
C3	PD10	G3	PC29	L3	PC4	R3	TSO
C4	PA2	G4	PC28	L4	PC3	R4	TSI
C5	PA5	G5	VDD_D	L5	VDD_D	R5	PB5
C6	PG1	G6	VSS	L6	VSS	R6	PB2
C7	PG6	G7	VSS	L7	VSS	R7	PE7
C8	PG9	G8	VSS	L8	VSS	R8	PE2
C9	PG11	G9	VSS	L9	VSS	R9	RSTN
C10	PG13	G10	VSS	L10	VSS	R10	USB2_ID
C11	PG15	G11	VSS	L11	VSS	R11	PF13
C12	DDRG_VREF	G12	VDDH	L12	VDD_D	R12	PF11
C13	PG21	G13	PH9	L13	VDDF	R13	PF10
C14	PG23	G14	PH8	L14	VDD_IO	R14	PF3
C15	PG27	G15	PH7	L15	PI11	R15	PF1
C16	PG28	G16	PI0	L16	PI6	R16	PF0
D1	PC27	H1	PC18	M1	PC6	T1	TSCK
D2	PD4	H2	PC17	M2	PC5	T2	TSSN
D3	PA4	H3	PC24	M3	PC0	T3	PB8
D4	PA6	H4	PC23	M4	PB17	T4	PB7
D5	PA3	H5	VDD_IO	M5	VDD_IO	T5	PB6
D6	PG0	H6	VSS	M6	VDD_D	T6	PB3
D7	PG5	H7	VSS	M7	VDD_IO	T7	PE5
D8	PG10	H8	VSS	M8	VDD_D	T8	PE1
D9	DDRG_VREF	H9	VSS	M9	VDDU	T9	PE0
D10	VDDG_RC	H10	VSS	M10	VDD_D	T10	TEST0
D11	VDDG	H11	VSS	M11	VDD_D	T11	USB2_P
D12	VDDG_RC	H12	VDDH	M12	VDDF	T12	USB2_N
D13	DDRH_VREF	H13	VDD_D	M13	VDDF_PD	T13	OSC_OUT
D14	PG30	H14	VDD_D	M14	PF4	T14	OSC_IN
D15	PG29	H15	VDDH_RC	M15	PI10	T15	VDDU_P
D16	PG31	H16	PI1	M16	PI7	T16	VDD_PLLCG

Table 2-2 Pin Assignments (sorted by signal name)

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
DDR_CAL	A5	PC25	E2	PG20	B13	VDD_D	M10
DDR_CLK	A12	PC26	E1	PG21	C13	VDD_D	M11
DDR_CLKFBN	A14	PC27	D1	PG22	B14	VDD_IO	F5
DDR_CLKFB	A15	PC28	G4	PG23	C14	VDD_IO	H5
DDR_CLKN	A13	PC29	G3	PG24	B15	VDD_IO	K5
DDR_ODT	B10	PC30	C1	PG25	A16	VDD_IO	K13
DDRG_VREF	C12	PC31	B1	PG26	B16	VDD_IO	K14
DDRG_VREF	D9	PD0	A1	PG27	C15	VDD_IO	L14
DDRH_VREF	D13	PD1	A2	PG28	C16	VDD_IO	M5
OSC_IN	T14	PD2	B2	PG29	D15	VDD_IO	M7
OSC_OUT	T13	PD3	C2	PG30	D14	VDD_PLLCG	T16
PA0	E4	PD4	D2	PG31	D16	VDD_PLLDDR	A7
PA1	B5	PD5	A3	PH0	E13	VDDF	L13
PA2	C4	PD6	B3	PH1	E16	VDDF	M12
PA3	D5	PD7	F4	PH2	E15	VDDF_PD	M13
PA4	D3	PD8	A4	PH3	E14	VDDG	D11
PA5	C5	PD9	B4	PH4	F16	VDDG	E5
PA6	D4	PD10	C3	PH5	F15	VDDG	E6
PA7	E3	PD11	F3	PH6	F14	VDDG	E9
PB0	N7	PE0	T9	PH7	G15	VDDG	E11
PB1	P6	PE1	T8	PH8	G14	VDDG	E12
PB2	R6	PE2	R8	PH9	G13	VDDG_RC	D10
PB3	T6	PE3	P8	PI0	G16	VDDG_RC	D12
PB4	N6	PE4	N8	PI1	H16	VDDH	F13
PB5	R5	PE5	T7	PI2	J16	VDDH	G12
PB6	T5	PE6	P7	PI3	J15	VDDH	H12
PB7	T4	PE7	R7	PI4	K16	VDDH_RC	H15
PB8	T3	PF_VREF	N14	PI5	K15	VDDU	M9
PB9	R2	PF0	R16	PI6	L16	VDDU_P	T15
PB10	R1	PF1	R15	PI7	M16	VSS	F7
PB11	P2	PF2	P15	PI8	N16	VSS	F8
PB12	P3	PF3	R14	PI9	P16	VSS	F9
PB13	P4	PF4	M14	PI10	M15	VSS	F10
PB14	N5	PF5	N13	PI11	L15	VSS	F11
PB15	P5	PF6	N15	RSTN	R9	VSS	G6
PB16	N4	PF7	P14	TEST0	T10	VSS	G7
PB17	M4	PF8	P12	TEST1	P9	VSS	G8
PB18	N3	PF9	P13	TEST2	N9	VSS	G9
PB19	P1	PF10	R13	TSCK	T1	VSS	G10
PC0	M3	PF11	R12	TSI	R4	VSS	G11
PC1	N2	PF12	N12	TSO	R3	VSS	H6
PC2	N1	PF13	R11	TSSN	T2	VSS	H7
PC3	L4	PF14	P11	USB2_ID	R10	VSS	H8
PC4	L3	PF15	N11	USB2_N	T12	VSS	H9
PC5	M2	PG0	D6	USB2_P	T11	VSS	H10
PC6	M1	PG1	C6	USB2_RBIAS	P10	VSS	H11
PC7	L1	PG2	A6	USB2_VBUS	N10	VSS	J6
PC8	L2	PG3	B6	VDD_D	E7	VSS	J7
PC9	K4	PG4	B7	VDD_D	E8	VSS	J8
PC10	K3	PG5	D7	VDD_D	E10	VSS	J9
PC11	K1	PG6	C7	VDD_D	F6	VSS	J10
PC12	K2	PG7	B8	VDD_D	F12	VSS	J11
PC13	J4	PG8	A8	VDD_D	G5	VSS	K06
PC14	J3	PG9	C8	VDD_D	H13	VSS	K07
PC15	J1	PG10	D8	VDD_D	H14	VSS	K08
PC16	J2	PG11	C9	VDD_D	J5	VSS	K09
PC17	H2	PG12	B9	VDD_D	J12	VSS	K10
PC18	H1	PG13	C10	VDD_D	J13	VSS	K11
PC19	G2	PG14	A10	VDD_D	J14	VSS	L6
PC20	G1	PG15	C11	VDD_D	K12	VSS	L7
PC21	F2	PG16	B11	VDD_D	L5	VSS	L8
PC22	F1	PG17	A11	VDD_D	L12	VSS	L9
PC23	H4	PG18	A9	VDD_D	M6	VSS	L10
PC24	H3	PG19	B12	VDD_D	M8	VSS	L11

www.ubicom.com

2.2 Pin Descriptions

Type Codes: I = Digital Input, AI = Analog Input, O = Output, Z = Tristateable, P = Power, PU = On-Chip Pullup,

PD = On-Chip Pulldown, ST = Schmitt Trigger, NS = Non-Slew Rate Limited

Table 2-3 Pin Descriptions

Name	Type	Output Current	Input Voltage	Supply	Description
DDR_CAL	I/O				Calibrator pin for input and output impedance calibration
DDR_CLK	O,Z				Clock output to DDR SDRAM
DDR_CLKN	O,Z				Clock output to DDR SDRAM (inverted)
DDR_CLKFB	I				Clock input from DDR SDRAM
DDR_CLKFBN	I				Clock input from DDR SDRAM (inverted)
DDR_ODT	O,Z				Output to DDR SDRAM (also sensed by the IP51xx to determine internal configuration).
DDRG_VREF	I				DDR SDRAM voltage reference from port G
DDRH_VREF	I				DDR SDRAM voltage reference from port H
OSC_IN	AI				Crystal clock input
OSC_OUT	O				Crystal clock output
PA[6,4:0]	I/O	8mA	0-5.5V	VDD_IO	Port A. Refer to Table 2-5.
PA[7,5]	I/O, NS	8mA	0-5.5V	VDD_IO	Port A. Refer to Table 2-5.
PB[19,17:0]	I/O	8mA	0-5.5V	VDD_IO	Port B. Refer to Table 2-6.
PB18	I/O, NS	12mA	0-5.5V	VDD_IO	Port B. Refer to Table 2-6.
PC[31:0]	I/O	8mA	0-5.5V	VDD_IO	Port C. Refer to Table 2-7.
PD[11:0]	I/O	8mA	0-5.5V	VDD_IO	Port D. Refer to Table 2-8.
PE[7:0]	I/O	8mA	0-5.5V	VDD_IO	Port E. Refer to Table 2-9.
PF_VREF	I				Input threshold level — reference voltage level for input comparator
PF[15:0]	I/O	8.1mA	not 5V tolerant	VDDF / VDDF_PD	Port F. Refer to Table 2-10.
PG[31:0]	I/O	8.1mA	not 5V tolerant	VDDG / VDDG_RC	Port G. Refer to Table 2-11.
PH[9:0]	I/O	8.1mA	not 5V tolerant	VDDH / VDDH_RC	Port H. Refer to Table 2-12.
PI[11:0]	I/O	8mA	0-5.5V	VDD_IO	Port I. Refer to Table 2-13.
RSTN	I,ST,PU		0-5.5V	VDD_IO	Assert to 0 for chip reset. See Note 1.
TEST0, TEST1, TEST2	I,PD		0-5.5V	VDD_IO	Test mode pins. Connect to VSS. See Note 1.
TSCK	I,ST,PD		0-5.5V	VDD_IO	Debug Interface Clock (used only for in-system programming and debug).
TSI	I,ST,PU		0-5.5V	VDD_IO	Debug Interface Serial Data Input (used only for in-system programming and debug).

Table 2-3 Pin Descriptions (continued)

Name	Type	Output Current	Input Voltage	Supply	Description
TSO	O,Z	8 mA	5V tolerant when Z	VDD_IO	Debug Interface Serial Data output (used only for in-system programming and debug; high Z unless TSSN low). 5V tolerant when tristated.
TSSN	I,ST,PU		0-5.5V	VDD_IO	Debug Interface Slave Select, active low (used only for in-system programming and debug). See Note 1.
USB2_ID	I				If this input is high, then this port is a USB peripheral port. If this input is low, then this port is a USB host port.
USB2_N	I/O		5V tolerant		USB data on USB port (negative)
USB2_P	I/O		5V tolerant		USB data on USB port (positive)
USB2_RBIAS	O				Tie this to VSS with a 9.1 Kohm (± 1%) resistor. This will give an output of 1.2V ± 3%.
USB2_VBUS	I/PD		5V tolerant		The IP51xx senses whether this is 5V or floating — if it is floating, then the IP51xx can output on a GPIO to turn on a circuit to supply 5V to the USB2 bus. A 200 ohm resistor is recommended between 5V and the USB2_VBUS pin, so that transients in the 5V supply won't cause the USB2_VBUS pin to go above 5.25V.
VDD_D	P 1.2V				VDD for digital core
VDD_IO	P 3.3V				VDD for GPIO I/Os
VDD_PLLCG	P 1.2V				VDD for core and I/O PLLs (clock generator)
VDD_PLLDDR	P 1.2V				VDD for DDR SDRAM and DDR SDRAM Deskew PLLs
VDDF	2.5/3.3V				VDD for Port F: 2.5V for Gigabit Ethernet, 3.3V for GPIO.
VDDF_PD	3.3V				VDD for Port F Gigabit Ethernet predriver
VDDG	1.8/2.5V				VDDQ (I/O supply) for Port G SDRAM DDRL
VDDG_RC	2.5V				VDD for DDRL Receiver
VDDH	1.8/2.5V				VDDQ (I/O supply) for Port H SDRAM DDRH
VDDH_RC	2.5V				VDD for DDRH Receiver
VDDU	3.3V				VDD for USB2 PHY I/O
VDDU_P	P 1.2V				VDD for USB2 PHY PLL
VSS	P 0V				VSS for all digital core, I/Os, and PLLs
Note 1: Ubicom recommends not relying on internal pullup or pulldown.					

2.3   I/O Ports Signal Maps

The ten I/O ports are designated Port A, Port B, ... , Port I, and USB Port. Every port is capable of multiple functions, except the USB port, which is dedicated to USB. Programs select the function of a port by programming the port's function select register. Behavior of each I/O port's signals depends on the function selected for that port.

Table 2-4 shows the overall organization of the I/O ports. Table 2-5 through Table 2-14 show the signal assignments for each function of each port. Refer also to Section 6.0 on page 145 for more detail and for explanations of terms.

Table 2-4 Port Function Summary

Port	Port Width	Function 0	Function 1	Function 2	Function 3
A	8	GPIO	Flash / INT / Clock	GPIO / INT / Clock	GPIO / INT
B	20	GPIO	PCI	---	---
C	32	GPIO	PCI (I/O only)	Reserved	---
D	12	GPIO	Serdes (240 MHz)	Reserved	---
E	8	GPIO	Serdes (250 MHz)	Reserved	MII / RMII
F	16	GPIO	GMAC (MII / RMII / RGMII)	---	---
G	32	GPIO	DDR SDRAM	---	---
H	10	GPIO	DDR SDRAM	---	---
I	12	GPIO	N/A	Reserved	MII (Port E Extension)
USB Port	2	N/A	High-Speed USB	N/A	N/A

Table 2-5 Port A Signal Map

Bit # PA[n]	Function 0 (GPIO)	Function 1 (Flash, INT, Clock)	Function 2 (GPIO, INT, Clock)	Function 3 (GPIO, INT)
0	GPIO	SI	GPIO	GPIO
1	GPIO	SO	GPIO	GPIO
2	GPIO	SCK	GPIO	GPIO
3	GPIO	CE_N	GPIO	GPIO
4	GPIO	GPIO / INT_0	GPIO / INT_0	GPIO / INT_0
5	GPIO	GPIO / INT_1 / CLOCK_0 (250 MHz)	GPIO / INT_1 / CLOCK_0 (250 MHz)	GPIO / INT_1
6	GPIO	GPIO / INT_2	GPIO / INT_2	GPIO / INT_2
7	GPIO	GPIO / CLOCK_1 (Core Clock)	GPIO / CLOCK_1 (Core Clock)	GPIO

Table 2-6 Port B Signal Map

Bit # PB[n]	Function 0 (GPIO)	Function 1 (PCI)	Function 2 (reserved)	Function 3 (reserved)
0	GPIO	DEVSEL_N	---	---
1	GPIO	PERR_N	---	---
2	GPIO	STOP_N	---	---
3	GPIO	SERR_N	---	---
4	GPIO	TRDY_N	---	---
5	GPIO	FRAME_N	---	---
6	GPIO	IRDY_N	---	---
7	GPIO	PAR	---	---
8	GPIO	CBE[0]	---	---
9	GPIO	CBE[1]	---	---
10	GPIO	CBE[2]	---	---
11	GPIO	CBE[3]	---	---
12	GPIO	REQ0_N	---	---
13	GPIO	GNT0_N	---	---
14	GPIO	REQ1_N	---	---
15	GPIO	GNT1_N	---	---
16	GPIO	RST_N	---	---
17	GPIO	CLK	---	---
18	GPIO	CLK_OUT	---	---
19	GPIO	INTA	---	---

Table 2-7 Port C Signal Map

Bit # PC[n]	Function 0 (GPIO)	Function 1 (PCI)	Function 2 (reserved)	Function 3 (reserved)
0	GPIO	AD[00]	---	---
1	GPIO	AD[01]	---	---
2	GPIO	AD[02]	---	---
3	GPIO	AD[03]	---	---
4	GPIO	AD[04]	---	---
5	GPIO	AD[05]	---	---
6	GPIO	AD[06]	---	---
7	GPIO	AD[07]	---	---
8	GPIO	AD[08]	---	---
9	GPIO	AD[09]	---	---
10	GPIO	AD[10]	---	---
11	GPIO	AD[11]	---	---
12	GPIO	AD[12]	---	---
13	GPIO	AD[13]	---	---
14	GPIO	AD[14]	---	---
15	GPIO	AD[15]	---	---
16	GPIO	AD[16]	---	---
17	GPIO	AD[17]	---	---
18	GPIO	AD[18]	---	---
19	GPIO	AD[19]	---	---
20	GPIO	AD[20]	---	---
21	GPIO	AD[21]	---	---
22	GPIO	AD[22]	---	---
23	GPIO	AD[23]	---	---
24	GPIO	AD[24]	---	---
25	GPIO	AD[25]	---	---
26	GPIO	AD[26]	---	---
27	GPIO	AD[27]	---	---
28	GPIO	AD[28]	---	---
29	GPIO	AD[29]	---	---
30	GPIO	AD[30]	---	---
31	GPIO	AD[31]	---	---

Table 2-8 Port D Signal Map

Bit # PD[n]	Function 0 (GPIO)	Function 1 (Serdes) (240 MHz)	Function 2 (reserved)	Function 3 (reserved)
0	GPIO	RXD	---	---
1	GPIO	RXM	---	---
2	GPIO	RXP	---	---
3	GPIO	CLK	---	---
4	GPIO	TXME	---	---
5	GPIO	TXM	---	---
6	GPIO	TXP	---	---
7	GPIO	TXPE	---	---
8	GPIO	GPIO	---	---
9	GPIO	GPIO	---	---
10	GPIO	GPIO	---	---
11	GPIO	GPIO	---	---

Table 2-9 Port E Signal Map

Bit # PE[n]	Function 0 (GPIO)	Function 1 (Serdes) (250 MHz)	Function 2 (reserved)	Function 3 (MII / RMII)
0	GPIO	RXD	---	RX_CLK / REF_CLK
1	GPIO	RXM	---	RXD[0] / TXD[0]
2	GPIO	RXP	---	RXD[1] / TXD[1]
3	GPIO	CLK	---	RXD[2] / TX_EN
4	GPIO	TXME	---	RXD[3] / RX_ER
5	GPIO	TXM	---	RX_DV / CRS_DV
6	GPIO	TXP	---	RX_ER / RXD[0]
7	GPIO	TXPE	---	COL / RXD[1]

Table 2-10 Port F Signal Map

Bit # PF[*n*]	Function 0 (GPIO)	Function 1 (MII / RMII RGMII)	Function 2 (reserved)	Function 3 (reserved)
0	GPIO	TXD[0] / TXD[0] / TXD[0]	---	---
1	GPIO	TXD[1] / TXD[1] / TXD[1]	---	---
2	GPIO	TXD[2] / N/A / TXD[2]	---	---
3	GPIO	TXD[3] / N/A / TXD[3]	---	---
4	GPIO	TX_ER / N/A / N/A	---	---
5	GPIO	TX_EN / TX_EN / TX_CTL	---	---
6	GPIO	TX_CLK / REF_CLK_I / N/A	---	---
7	GPIO	COL / REF_CLK_O/ TX_CLK_O	---	---
8	GPIO	RXD[0] / RXD[0] / RXD[0]	---	---
9	GPIO	RXD[1] / RXD[1] / RXD[1]	---	---
10	GPIO	RXD[2] / N/A / RXD[2]	---	---
11	GPIO	RXD[3] / N/A / RXD[3]	---	---
12	GPIO	RX_ER / RX_ER / N/A	---	---
13	GPIO	RX_DV / N/A / RX_CTL	---	---
14	GPIO	RX_CLK / N/A / RX_CLK	---	---
15	GPIO	CRS / CRS_DV N/A	---	---

N/A: Not applicable for that mode (RMII and/or RGMII)

Table 2-11 Port G Signal Map

Bit # PG[*n*]	Function 0 (GPIO)	Function 1 (DDR SDRAM)	Function 2 (reserved)	Function 3 (reserved)
0	GPIO	ADDR[13]	---	---
1	GPIO	ADDR[12]	---	---
2	GPIO	ADDR[11]	---	---
3	GPIO	ADDR[09]	---	---
4	GPIO	ADDR[08]	---	---
5	GPIO	ADDR[07]	---	---
6	GPIO	ADDR[06]	---	---
7	GPIO	ADDR[05]	---	---
8	GPIO	ADDR[04]	---	---
9	GPIO	ADDR[03]	---	---
10	GPIO	ADDR[02]	---	---
11	GPIO	ADDR[01]	---	---
12	GPIO	ADDR[00]	---	---
13	GPIO	ADDR[10]	---	---
14	GPIO	CS_N	---	---
15	GPIO	CAS_N	---	---
16	GPIO	BA2	---	---
17	GPIO	BA1	---	---
18	GPIO	BA0	---	---
19	GPIO	RAS_N	---	---
20	GPIO	WE_N	---	---
21	GPIO	CKE	---	---
22	GPIO	DM / LDM	---	---
23	GPIO	DQ[7]	---	---
24	GPIO	DQ[6]	---	---
25	GPIO	DQ[5]	---	---
26	GPIO	DQ[4]	---	---
27	GPIO	LDQS	---	---
28	GPIO	DQ[3]	---	---
29	GPIO	DQ[2]	---	---
30	GPIO	DQ[1]	---	---
31	GPIO	DQ[0]	---	---

Table 2-12 Port H Signal Map

Bit # PH[*n*]	Function 0 (GPIO)	Function 1 (DDR SDRAM)	Function 2 (reserved)	Function 3 (reserved)
0	GPIO	UDM	---	---
1	GPIO	DQ[15]	---	---
2	GPIO	DQ[14]	---	---
3	GPIO	DQ[13]	---	---
4	GPIO	DQ[12]	---	---
5	GPIO	UDQS	---	---
6	GPIO	DQ[11]	---	---
7	GPIO	DQ[10]	---	---
8	GPIO	DQ[09]	---	---
9	GPIO	DQ[08]	---	---

Table 2-13 Port I Signal Map

Bit # PI[*n*]	Function 0 (GPIO)	Function 1 (reserved)	Function 2 (reserved)	Function 3 (MII)
0	GPIO	---	---	TX_CLK / TX_CLK_OUT
1	GPIO	---	---	TXD[0]
2	GPIO	---	---	TXD[1]
3	GPIO	---	---	TXD[2]
4	GPIO	---	---	TXD[3]
5	GPIO	---	---	TX_EN
6	GPIO	---	---	TX_ER
7	GPIO	---	---	CRS
8	GPIO	---	---	GPIO
9	GPIO	---	---	GPIO
10	GPIO	---	---	GPIO
11	GPIO	---	---	GPIO

Table 2-14 USB Port Signal Map

Pin Name	Function 1 (USB)
USB2_ID	ID
USB2_N	dataN
USB2_P	dataP
USB2_RBIAS	rBias
USB2_VBUS	vbus

3.0 System Architecture

The central feature of the IP51xx architecture is hardware multithreading, with zero-overhead context switching between hardware threads. All registers that contain context-specific information are duplicated for each of ten hardware threads. The CPU hardware is capable of switching from one hardware thread to another, on a cycle-by-cycle basis with no switching delay. This design enables deterministic and extremely efficient interrupt response, which in turn supports the creation of *software peripherals*. A *software peripheral* is a combination of simple peripheral I/O hardware, and control logic implemented in software, rather than custom peripheral hardware.

3.1 CPU Registers

The IP51xx features 16 general-purpose 32-bit data registers, eight 32-bit address registers (A0-A6, A7/SP), multiply/multiply-accumulate (ACC) registers, and various other registers. These registers reside in the register address space, an address space separate from the indirect registers and memory. Instructions reference the registers within the register address space directly (as opposed to indirectly through offsets from an address base register). There is no capability for indirect referencing of registers in the register address space.

Every register in the register address space is 32 bits wide.

There are two distinct groups of registers in the register address space:

- Per-Thread Registers
- Global Registers

Some registers are described as read-only. Do not write to a read-only register. Writes to these registers do not change the state of the register, but may cause undesirable side effects.

Some registers are described as write-only. Reads of these registers return undefined results.

3.1.1 Per-Thread Registers

Per-thread registers define the architectural state of one hardware thread. The first 64 registers are per-thread; that is, to support immediate context switching (without the overhead of saving and restoring these registers in software), the per-thread register set is duplicated for each of the ten hardware-supported threads, as shown in Figure 3-1. Table 3-1 shows the locations of these registers in the register space.

Refer also to Section 7.2 for detailed register descriptions.

Table 3-1 Per-Thread Register Map

| Address	Register(s)	Description
000-03C	D0–D15	General-purpose data registers.
040-07C	Reserved	
080-098	A0–A6	32-bit address registers.
09C	A7 or SP	32-bit stack pointer, also referred to as A7.
0A0	ACC0_HI	High 32 bits of MAC, DSP, and multiplier result.
0A4	ACC0_LO	Low 32 bits of MAC, DSP, and multiplier result.
0A8	MAC_RC16	Multiply-accumulate result, rounded and clipped.
0AC	SOURCE3	Implicit third source operand for certain instructions.
0B0	INST_CNT	Count of executed instructions.
0B4	CSR	Control codes and status register.
0B8	ROSR	Read-only status register.
0BC	IREAD_DATA	IREAD result
0C0	INT_MASK0	Thread interrupt mask.
0C4	INT_MASK1	Thread interrupt mask.
0C8-0CC	Reserved	
0D0	PC	32-bit Program Counter.
0D4	TRAP_CAUSE	Cause of most recent trap.
0D8	ACC1_HI	High 32 bits of DSP result.
0DC	ACC1_LO	Low 32 bits of DSP result.
0E0	PREVIOUS_PC	Program Counter value for last successfully executed instruction for this thread.
0E4-0FC	Reserved	
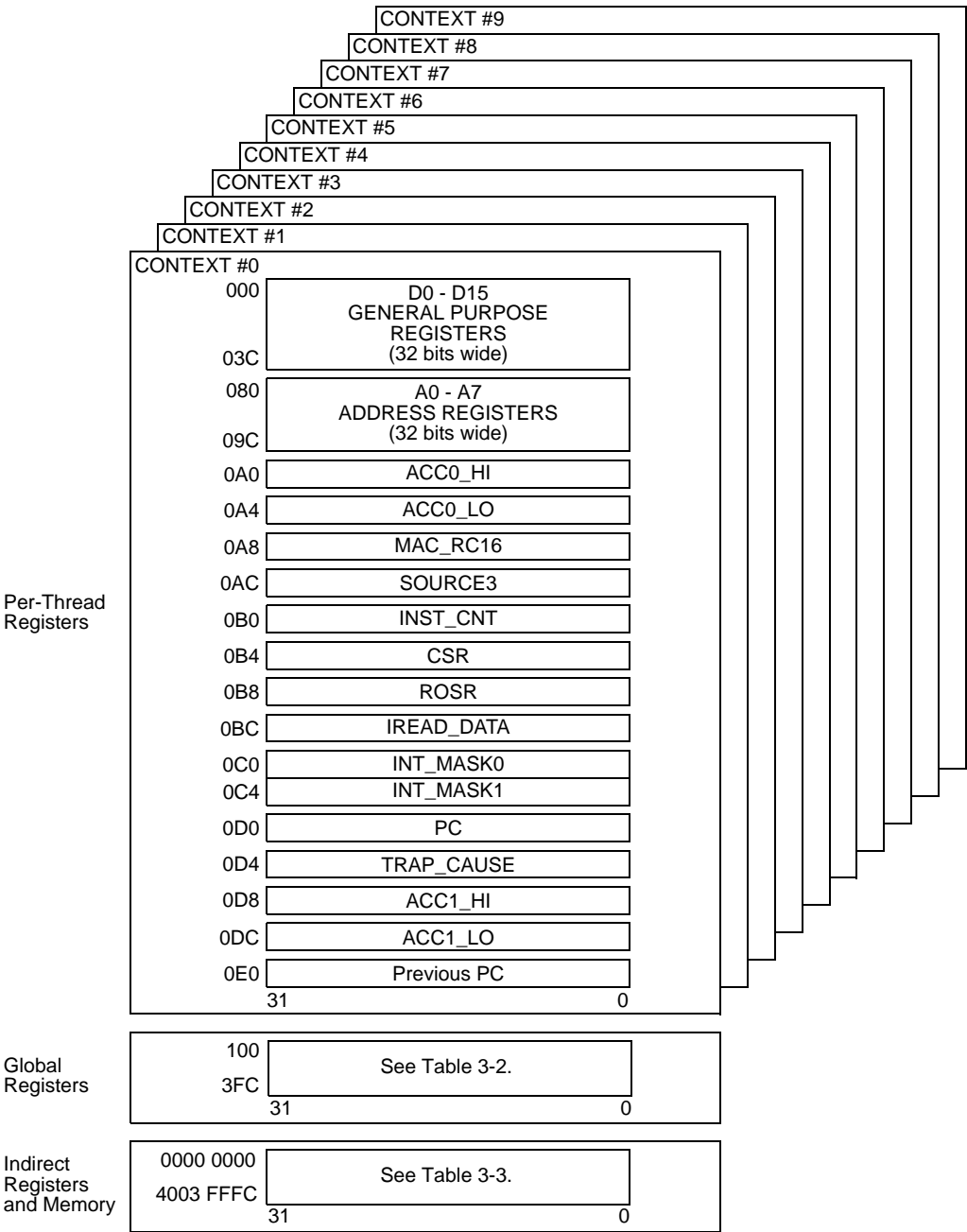
Figure 3-1 Per-Thread, Global, and Indirect Registers, and Indirect Memory

3.1.2   Global Registers

Registers at addresses 0x100 and greater are global; that is, shared among all threads. Table 3-2 shows the addresses of these registers in the register space. Refer also to Section 7.3 for detailed register descriptions.

Registers containing bits that can be set by hardware are generally read-only. To enable software to set or clear bits in these registers, there are associated "write-only" set and clear registers. A value written to a "set" register is atomically ORed, on the next cycle, with the corresponding hardware register. The complement of a value written to a "clear" register is atomically ANDed, on the next cycle, with the corresponding hardware register.

Table 3-2 Global Register Map

Address	Register(s)	Description	Type
100	CHIP_ID	Chip ID	Read Only
104 108	INT_STAT0 INT_STAT1	Interrupt Status	Read Only
10C-110	Reserved		
114 118	INT_SET0 INT_SET1	Set Interrupt Status	Write Only
11C-120	Reserved		
124 128	INT_CLR0 INT_CLR1	Clear Interrupt Status	Write Only
12C-130	Reserved		
134	GLOBAL_CTRL	Processor function control bits	Read/Write
138	MT_ACTIVE	Threads' active/inactive status	Read Only
13C	MT_ACTIVE_SET	Set bits of MT_ACTIVE register	Write Only
140	MT_ACTIVE_CLR	Clear bits of MT_ACTIVE register	Write Only
144	MT_DBG_ACTIVE	Threads' Debug Active status	Read Only
148	MT_DBG_ACTIVE_SET	Set bits of MT_DBG_ACTIVE register	Write Only
14C	MT_EN	Multithreading Enable	Read/Write
150	MT_HPRI	Multithreading High Priority Thread mask for non-real-time (NRT) threads	Read/Write
154	MT_HRT	Multithreading Hard Real Time Thread (HRT) mask	Read/Write
158	MT_BREAK	Multithreading BKPT executed mask	Read Only
15C	MT_BREAK_CLR	Clear bits of MT_BREAK register	Write Only
160	MT_SINGLE_STEP	Multithreading Single Step mask	Read/Write
164	MT_MIN_DELAY_EN	Multithreading Minimum Delay Enable mask	Read/Write
168	MT_BREAK_SET	Set bits of MT_BREAK register	Write Only
16C	Reserved		
170	DCAPT	Write Trap Address	Read/Write
174-178	Reserved		
17C	MT_DBG_ACTIVE_CLR	Clear bits of MT_DBG_ACTIVE register	Write Only
180 184 188 18C	SCRATCHPAD0 SCRATCHPAD1 SCRATCHPAD2 SCRATCHPAD3	Four scratchpad registers	Read/Write
190-19C	Reserved		
1A4	MT_I_BLOCKED	Thread blocked due to instruction fetch (1 bit / thread)	Read Only
1A8	MT_D_BLOCKED	Thread blocked due to data access (1 bit / thread)	Read Only
1AC	MT_I_BLOCKED_SET	Set bits of MT_I_BLOCKED register	Write Only
1B0	MT_D_BLOCKED_SET	Set bits of MT_D_BLOCKED register	Write Only
1B4	MT_BLOCKED_CLR	Clear bits of MT_I_BLOCKED and MT_D_BLOCKED	Write Only
1B8	MT_TRAP_EN	Multithreading Enable Traps mask	Read/Write
1BC	MT_TRAP	Multithreading Trap (set by hardware when an enabled trap occurs, or by writing to MT_TRAP_SET)	Read Only
1C0	MT_TRAP_SET	Set bits of MT_TRAP	Write Only

Table 3-2 Global Register Map (continued)

Address	Register(s)	Description	Type
1C4	MT_TRAP_CLR	Clear bits of MT_TRAP	Write Only
1C8-1FC	Reserved		
200	I_RANGE0_HI	Instruction space memory range 0 high	Read/Write
204	I_RANGE1_HI	Instruction space memory range 1 high	Read/Write
208	I_RANGE2_HI	Instruction space memory range 2 high	Read/Write
20C-21C	Reserved		
220	I_RANGE0_LO	Instruction space memory range 0 low	Read/Write
224	I_RANGE1_LO	Instruction space memory range 1 low	Read/Write
228	I_RANGE2_LO	Instruction space memory range 2 low	Read/Write
22C-23C	Reserved		
240	I_RANGE0_EN	Instruction space memory range 0 thread enables	Read/Write
244	I_RANGE1_EN	Instruction space memory range 1 thread enables	Read/Write
248	I_RANGE2_EN	Instruction space memory range 2 thread enables	Read/Write
24C-25C	Reserved		
260	D_RANGE0_HI	Data space memory range 0 high	Read/Write
264	D_RANGE1_HI	Data space memory range 1 high	Read/Write
268	D_RANGE2_HI	Data space memory range 2 high	Read/Write
26C	D_RANGE3_HI	Data space memory range 3 high	Read/Write
270-27C	Reserved		
280	D_RANGE0_LO	Data space memory range 0 low	Read/Write
284	D_RANGE1_LO	Data space memory range 1 low	Read/Write
288	D_RANGE2_LO	Data space memory range 2 low	Read/Write
28C	D_RANGE3_LO	Data space memory range 3 low	Read/Write
290-29C	Reserved		
2A0	D_RANGE0_EN	Data space memory range 0 thread enables	Read/Write
2A4	D_RANGE1_EN	Data space memory range 1 thread enables	Read/Write
2A8	D_RANGE2_EN	Data space memory range 2 thread enables	Read/Write
2AC	D_RANGE3_EN	Data space memory range 3 thread enables	Read/Write
2B0-3FC	Reserved		

3.2 Addressing Model

The IP51xx has an on-chip 192 KB SRAM memory used for both data and instructions (program). Using separate data and address buses for the data and instruction accesses, instruction fetches and data operand accesses are done concurrently without any contention or waiting. The data access allows up to one 32-bit operand read and one 32-bit operand write in each clock cycle. Instruction memory and data memory have nonoverlapping addresses; so, the data and instruction address space can be treated as a single unified 32-bit space. The actual allocation of the memory ranges for instructions and data is specified in the global register space (Table 3-2) beginning at address 200.

CPU registers belong to an address space separate from data and program memory address spaces.

Table 3-3 shows how the entire indirect and program address space is allocated.

All memories use byte addressing, although all accesses to instruction memory and registers are in 32-bit word multiples, 32-bit word-aligned. Data accesses vary in width, depending on instruction. Operand addressing in data memory is big-endian – i.e., the most significant byte has the lowest address. Bit numbering within registers and instruction and data memory is little-endian, with bit 0 being the least significant bit.

3.3 Instruction Model

Instructions perform memory-memory operations, as well as memory-register, register-memory, and register-register operations. A variety of addressing modes are available. Instructions are 32-bits wide, and execute at the rate of one per cycle.

Table 3-3 Indirect and Program Space Address Map

Address Range	Function
0000 0000–0000 07FC	Reserved
0000 0800–0000 083C	HRT Table 0 See Table 7-4.
0000 0840–0000 08FC	Reserved
0000 0900–0000 093C	HRT Table 1 See Table 7-4.
0000 0940–00FF FFFC	Reserved
0100 0000–0100 0FFC	On-chip peripherals (includes timers and debug port). For data (indirect) space, these on-chip peripherals occupy 4 KB. For instruction space, there is 16 MB available at this block of addresses. See Table 7-5.
0100 1000–01FF FFFC	Reserved
0200 0000–0200 FFFC	I/O ports (64 KB). See Table 7-16
0201 0000–02FF FFFC	Reserved
0300 0000–0302 FFFC	On-chip SRAM (192 KB) *
0303 0000–3FFF FFFC	Reserved
4000 0000–47FF FFFC	External DDR SDRAM (128 MB)
4800 0000–5FFF FFFC	Reserved
6000 0000–60FF FFFC	External Flash (16 MB) **
6100 0000–FFFF FFFC	Reserved

* Ubicom software aliases On-chip SRAM to 3FFC 0000 – 3FFE FFFC.

**Flash memory is aliased to several different ranges of addresses. Ubicom software uses the range 6000 0000 – 60FF FFFC. At start-up, the CPU begins executing at address 6000 0000.

3.4   Fast Multithreading Context Switch

A context is all of the state information for a given task or thread – all the information must be saved when the flow of program execution for a given thread is interrupted, so that the thread can restart as if no interrupt had occurred. The context consists of the following pieces:

- Per-thread registers.
- Data memory area used by the given thread.
- Control and status registers of peripheral support logic that is used by the given thread. For example, if Port A is used by a thread, then its setting and status are part of the context.

The IP51xx and its programming environment support fast context switching in the following ways:

- Per-thread register file with 10 sets of the context-dependent registers, one set of registers for each thread.
- Indexed addressing for data memory. The index registers are themselves part of the per-thread register file.
- Compilation, linking, loading tool chain that provides unique base addresses for each task. All addressing modes for the data memory are address-register-based, so it is possible to have multiple instances of a software thread executing on different data sets.

With this hardware support for context switching, each virtual peripheral has a unique view of memory and the programming model that is unaffected by and mostly unaware of other virtual peripherals that may exist.

Furthermore, because the important registers are duplicated for each context, there is no need to save or restore any registers when switching between different threads. Therefore, a context switch can occur in zero-time between instructions.

3.5   Instruction Level Multithreading

Each set of per-thread registers defines a thread. Each thread is identified by an integer in the range 0–9 which corresponds to its entry in per-thread register file.

Several global registers contain information that the CPU uses to schedule execution of the threads:

- MT_ACTIVE, and the corresponding MT_ACTIVE_SET, and MT_ACTIVE_CLR
- MT_DBG_ACTIVE, and the corresponding MT_DBG_ACTIVE_SET, and MT_DBG_ACTIVE_CLR
- MT_EN
- MT_HPRI
- MT_HRT

- MT_BREAK, and the corresponding MT_BREAK_SET, and MT_BREAK_CLR
- MT_SINGLE_STEP
- MT_MIN_DELAY_EN
- MT_I_BLOCKED, and the corresponding MT_I_BLOCKED_SET
- MT_D_BLOCKED, and the corresponding MT_D_BLOCKED_SET
- MT_BLOCKED_CLR
- MT_TRAP_EN
- MT_TRAP, and the corresponding MT_TRAP_SET, and MT_TRAP_CLR

All of the above registers are structured as bit maps, where each bit position corresponds to a thread; for example, bit 0 corresponds to thread 0, bit 1 corresponds to thread 1, etc. Bits 31:10 are reserved.

3.5.1   Scheduling Table (HRT)

The IP51xx uses two Hard-Real-Time (HRT) tables to control thread scheduling. The HRT tables are located at fixed memory addresses (shown in Table 3-3). One of the two HRT tables is active and being used by the CPU; the other is available for updates. The HRT Table Select bit in the GLOBAL_CTRL register determines which is the active table.

Each of the 64 HRT table entries is 8 bits wide. Table entries are contiguous in memory, one entry per byte. An HRT entry has the format shown in Table 3-4.

Table 3-4 HRT Entry

Bit Field	Description
7	End of Table.
	1 = The next entry executed will be entry zero of the table indicated by the HRT Table Select bit in the GLOBAL_CTRL register.
	0 = Not end of table.
6	Unoccupied Entry.
	1 = This time slot is available for a Non-Real-Time (NRT) thread.
	0 = The thread indicated by Thread Number should be scheduled if it is schedulable.
5:4	Reserved.
3:0	Thread Number.

Each entry in the table represents an available instruction cycle and specifies the thread (if any) to which that cycle is allocated. Software controls how many of the 64 table entries are actually used by setting bit 7 in the last used

entry. The HRT table must have at least one element with bit 7 set.

At each cycle, the CPU steps to the next entry in the current HRT table and determines which thread to execute based on the information in that entry. After it has processed the last entry of the table, it checks the global HRT Table Select bit and goes to entry zero of the currently selected HRT table.

At power-up or reset, bit 6 is cleared and bit 7 is set in all entries (each entry is an unoccupied end-of-table entry). Software must ensure that the active HRT table has at least one entry with the End of Table (bit 7) set. If no entry has bit 7 set, the result is undefined.

3.5.2   Scheduling Policies

For scheduling purposes, threads are defined as:

*   HRT – A thread whose bit in the MT_HRT register is 1. An HRT thread can only be scheduled in time slots allocated to it by the current HRT Table.
*   NRT – A thread whose bit in the MT_HRT register is 0. NRT threads can be scheduled both in the HRT table and by the round-robin scheduler.

The IP51xx implements three scheduling policies:

1. Hard-Real-Time (Time-Division Multiplexing) – An HRT thread is guaranteed to receive CPU cycles in proportion to the number of its HRT Table entries.
2. Round-Robin – NRT threads are scheduled on a round-robin basis (in rotation) during the CPU cycles when either no HRT thread is allocated or the allocated HRT thread is not ready.
3. Priority – Among the NRT threads, those threads whose bit in the MT_HPRI register is set to 1 have high priority; others have low priority. No low-priority NRT threads receive a CPU allocation as long as there are active high-priority NRT threads.

Note that it is possible for an NRT thread to have time slots allocated to it in the HRT table. Such a thread participates in round-robin scheduling but is also guaranteed to receive a minimum level of service from the CPU in proportion to the number of its entries in the HRT Table.

3.5.3   Schedulable Threads

For a thread to be considered for scheduling in the instruction pipe, the following conditions must be true for that thread. A thread is considered to be "schedulable" when these conditions are satisfied:

*   The debug interface has not halted the processor (dbg_mp_halt is deasserted).
*   The thread must be enabled (MT_EN is set).
*   The thread must be active (MT_ACTIVE is set).
*   The thread must not be halted due to a debug condition (MT_DBG_ACTIVE is set).
*   The thread must not be blocked on the instruction port (MT_I_BLOCKED is cleared).
*   The thread must not be blocked on the data port (MT_D BLOCKED is cleared).
*   Minimum instruction spacing enforcement is turned off for this thread (MT_MIN_DELAY_EN is not set), or minimum instruction spacing enforcement is turned on for this thread (MT_MIN_DELAY_EN is set), but the minimum spacing requirement has been met.

The MT_xx values in the list above are individual bits defined for each thread in the corresponding MT_xx global registers. See Table 7-3 for details about these registers.

3.5.4   Hard Real-Time (HRT) Scheduling

The static schedule for HRT threads is specified by the HRT Table.

Figure 3-2 shows an HRT example with three threads in a table that is eight entries long. Thread 1 is scheduled 50% of the time, thread 2 is scheduled 25% of the time and thread 3 is scheduled 12.5% of the time. With the IP51xx clocked at 270 MHz, this would equate to 135, 67.5, and 33.75 MIPS, respectively. The vacant slot in the last entry of the table guarantees that at least 33.75 MIPS remain available for NRT thread execution.

Each HRT thread is guaranteed to be allocated the instruction slots specified in the table, when it is ready to use them, provided it is schedulable. Thus each HRT thread has guaranteed deterministic performance.

The interrupt latency for each HRT thread is deterministic within the resolution of its static allocation. The pipeline length determines the latency and the time until the thread is next scheduled. The added scheduling jitter can be considered to be the same as an asynchronous interrupt synchronizing with a synchronous clock. For example, a thread with 25% allocation will have deterministic interrupt latency with respect to a clock running at 25% of the system clock.
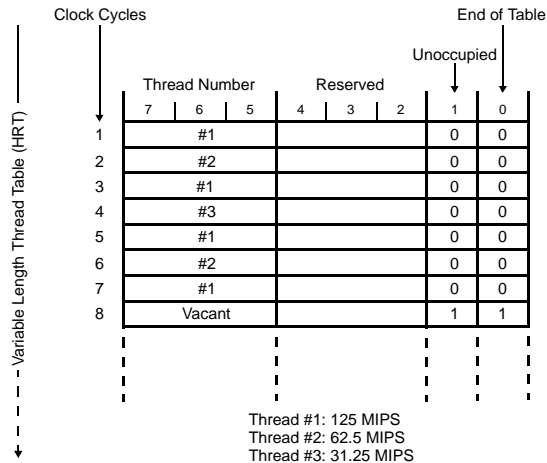
Clock Cycles   End of Table

Unoccupied

Variable Length Thread Table (HRT)

	7	6	5	4	3	2	1	0
	Thread Number			Reserved				
1	#1						0	0
2	#2						0	0
3	#1						0	0
4	#3						0	0
5	#1						0	0
6	#2						0	0
7	#1						0	0
8	Vacant						1	1

Thread #1: 125 MIPS
Thread #2: 62.5 MIPS
Thread #3: 31.25 MIPS

Figure 3-2 HRT Thread Table Example

Although the HRT Table reserves the instruction slots for the hard real-time threads this does not mean that other threads cannot sometimes execute in that instruction slot. For example a UART in thread 3 will actually be idle most of the time. It only needs deterministic performance when it is sending or receiving, and there is no need for it to be scheduled when it is not active. All vacant instruction slots and all slots that are allocated to threads that are not schedulable are used by the scheduler for dynamically schedulable (round-robin) threads.

3.5.5   Round-Robin (NRT) Scheduling

As the name suggests, round-robin threads are scheduled in turn, with one instruction initiated from each schedulable thread. Round-robin threads are scheduled in the vacant slots in the HRT table and in slots where the HRT or NRT thread specified by the table is not schedulable.

Two levels of priority are supported for NRT threads: low and high. Priority is controlled by the thread's bit in the global MT_HPRI register. If any high priority thread has its MT_EN and MT_ACTIVE bits set (regardless of its MT_DBG_ACTIVE bit), no low priority thread will be scheduled. This is true even if the high priority thread is not ready to execute.

3.5.6   Suspend

A thread can temporarily remove itself from scheduling activity with the SUSPEND instruction. SUSPEND clears the MT_ACTIVE bit for the current thread, so that the thread will not be scheduled. An interrupt condition for that thread asserts the MT_ACTIVE and re-enables normal scheduling of the thread.

3.5.7   Startup

At startup or after reset, thread 0 is active, debug active, and enabled, and all other threads are disabled. Thread 0 begins execution at the fixed flash ROM address 6000 0000, and is responsible for initialization; for example:

- Load instructions into the on-chip SRAM;
- Load an HRT table and all thread control registers (including PC);
- Initialize global semaphores and shared memory.

When the initialization is complete, thread 0 enables the other initialized threads, which then are free to execute.

3.6   Programming and Debugging Support

The IP51xx has advanced in-system programming and debug support on-chip. This unobtrusive capability is provided through a dedicated Debug Interface. There is no need for a bond-out chip for software development. This eliminates concerns about differences in electrical characteristics between a bond-out chip and the actual chip used in the target application. Designers can test and revise code on the same part used in the actual application.

Ubicom provides the complete Red Hat GNUPro tools, including C compiler, assembler, linker, utilities, and GNU debugger. In addition, Ubicom offers an integrated graphical development environment which includes an editor, project manager, graphical user interface for the GNU debugger, device programmer, and ipModule™ configuration tool, and profiler.

The IP51xx's external flash memory can be reprogrammed through the debug port (in-system) regardless of the contents of the flash. It is not required to have specific supporting software within the flash to allow in-system reprogramming to take place.

3.7   Debugging Features

The IP51xx has a number of mechanisms that are intended for use by Ubicom's debug kernel, and that support an off-chip debugging system. These mechanisms include:

- MT_DBG_ACTIVE active register
- MT_SINGLE_STEP register
- Breakpoint instruction BKPT and breakpoint interrupt
- SUSPEND instruction
- MT_BREAK Register
- Debug mailboxes and the Debug Mailbox Interrupt
- DCAPT register and Trap

• Minimum Instruction Delay

3.7.1   Single-Step

The MT_SINGLE_STEP register bit allows a controlling thread to single-step threads that are being debugged. This feature is enabled (on a per-thread basis) by setting the MT_SINGLE_STEP bit that corresponds to the thread being single-stepped. When this bit is set, the thread being debugged is executed as scheduled by the multithreading features. Simultaneously, the CPU clears that thread's MT_DBG_ACTIVE bit, so that the thread will not be activated until software sets the MT_DBG_ACTIVE bit again.

3.7.2   Breakpoints

Debugging breakpoints are supported by the Program Breakpoint interrupt, the BKPT instruction, and the MT_DBG_ACTIVE register.

The BKPT instruction suspends the thread that executes it and clears its MT_DBG_ACTIVE bit. In addition, the BKPT instruction can suspend additional threads and clear their MT_DBG_ACTIVE bits. The source operand is a bit mask that specifies which additional contexts to suspend.

The BKPT instruction asserts the MT_BREAK bit of the current thread (so that the debug kernel knows which thread executed the BKPT instruction) and asserts the Program Breakpoint interrupt (if enabled).

3.7.3   Write Address Trap

The DCAPT register can be loaded with a register or data address. Any write to the address in the DCAPT register triggers a Trap. The DCAPT register can't be disabled, but can be loaded with a value that can never match. A value that can never match is one with bit 0 equal to 1 and bits 31:13 not equal to all 0's; for example: 0x8000 0001.

A trap is also asserted upon detection of out-of-range or operand misalignment errors.

Whenever a Trap is asserted, the PC and TRAP_CAUSE registers for the affected thread capture the program counter value and the reason for the interrupt.

3.7.4   Debug Mailboxes

The Debug Mailboxes are the software visible portion of the debug port. The debug kernel uses the Debug Mailboxes to receive requests from an external debugging system and to return results. The Debug Mailbox Interrupt signals to the debug kernel regarding arrival or departure of mailbox messages.

3.7.5   Execution Control

Some of the CPU's instruction execution parameters can be modified by Ubicom's debug kernel to aid in debugging.

When a thread's bit in the MT_MIN_DELAY_EN register is set, the Minimum Instruction Delay value of the GLOBAL_CTRL register (refer to Section 7.3.4) is applied. This value is the minimum number of clocks between instructions of the same thread.

3.8   Interrupts and Exceptions

Interrupts are signaled by setting bits in the global Interrupt Status register. This is a 64-bit register (in two 32-bit parts, INT_STAT0 and INT_STAT1), with each bit representing a potential interrupt source. When a bit is set to 1, it asserts the associated interrupt condition.

Bits in the Interrupt Status register corresponding to I/O interrupts give the state of the corresponding I/O interrupt(s). Other bits can be set by hardware or by software (via the Interrupt Set register). If there is no hardware source associated with a particular bit, that bit represents a software interrupt. However, even if there is a hardware source (other than I/O) associated with an interrupt status bit, the bit can still be set by software. This makes it possible to simulate interrupts for software testing.

Once an interrupt status bit is set, it remains set until explicitly cleared by software (via the Interrupt Clear Register). No automatic interrupt acknowledge signal is sent to an originating peripheral device – neither when the interrupt status bit is set, nor when an interrupt handler responds to it. If an acknowledgement is needed, it is the responsibility of the interrupt handling software to send it, by writing to the appropriate peripheral register.

Note: I/O interrupts must be cleared by writing to the appropriate I/O control register, not by directly clearing the INT_STAT bit.

Section 7.3.2 and Section 7.3.3 show the mapping of interrupt status bits to specific interrupt sources.

3.8.1   INT_STAT[0-1] Registers

The INT_STAT[0-1] registers are dedicated to interrupts:

- Asynchronous Error Interrupt. Used to report any serious error that is asynchronous to the instruction stream.
- Real-Time Compare Register Interrupt.
- Program Breakpoint / Trap Interrupt. This interrupt indicates that one or more threads has been halted, as a result of execution of the BKPT instruction, or that one or more of 13 possible trap interrupts has occurred. When a trap occurs, cause(s) of the trap are indicated by bits set in the TRAP_CAUSE register. Refer to Section 7.2.3 for details.
- Debug Port Interrupt. Indicates that a debug message is waiting or has been successfully sent.
- I/O Interrupts. Most I/O functions can generate a variety of interrupts, which are visible in INT_STAT0 and INT_STAT1. Each I/O function has a Interrupt Status register giving the cause of the interrupt and the interrupt bit itself. Several I/O interrupts might be shared by a single INT_STAT bit.
- A block of ten fine-grained timer interrupts is associated with a corresponding block of ten 32-bit timer registers. When the value held in a given timer register matches the value of the global cycle count register, the corresponding interrupt is asserted.

3.8.2   Interrupt Set and Clear Registers

INT_STAT0 and INT_STAT1 can be read by software, but they cannot be written directly. Instead, special Interrupt Set (INT_SET0 and INT_SET1) and Interrupt Clear (INT_CLR0 and INT_CLR1) registers are defined (see Table 7-3). These are 64-bit registers that parallel INT_STAT0 and INT_STAT1. Writing a 1 to a bit position in INT_SET0 and INT_SET1 sets the corresponding bit in the INT_STAT0 and INT_STAT1. Likewise, writing a 1 to a bit position in INT_CLR0 and INT_CLR1 clears the corresponding bit in INT_STAT0 and INT_STAT1. Other bits in INT_STAT0 and INT_STAT1 are not affected.

3.8.3   Thread Interrupt Mask

Each hardware context has a pair of 32-bit Interrupt Mask registers, INT_MASK[0-1], that determine the interrupts to which it responds (see Table 7-2). The masks are logically ANDed with the contents of the corresponding Interrupt Status registers; if the result is non-zero, an interrupt condition is signaled to the associated hardware thread, setting the INTERRUPT CONDITION bit of its ROSR register. If the thread is currently suspended, it is made active. If it is currently active, it remains active, continuing normal execution. However, if it executes a SUSPEND instruction, the presence of the pending interrupt will immediately reactivate it.

The Interrupt Mask register is a per-context read-write register. It is normally written only at start-up, however, to configure the assignment of interrupts to hardware threads.

3.8.4   Breakpoint and Trap Registers

When the breakpoint / trap interrupt is triggered, software should look at the MT_BREAK and MT_TRAP registers to determine whether the cause was a breakpoint or a trap (or both).

The MT_BREAK global read-only register has one bit per hardware thread. If the bit for a given thread is set, it indicates that the thread is halted for a break condition. The interrupt handler for the breakpoint / trap interrupt can read this register to determine which thread is halted for a break condition.

Bits in the MT_BREAK register are cleared by software, by writing to the Multithreading Break Clear (MT_BREAK_CLR) register. Clearing one of these bits does not restart the corresponding thread; setting the MT_DEBUG_ACTIVE bit accomplishes that.

The MT_TRAP global read-only register also has one bit per hardware thread. If the bit for a given thread is set, it indicates that the thread is halted because of a trap. The interrupt handler for the breakpoint / trap interrupt can read this register to determine which thread is halted for a trap. It can then read the TRAP_CAUSE register for that thread to determine the cause of the trap.

Bits in the MT_TRAP register are cleared by software, by writing to the Multithreading TRAP Clear (MT_TRAP_CLR) register. Clearing one of these bits does not restart the corresponding thread; setting the MT_DEBUG_ACTIVE bit accomplishes that.

3.8.5   Forcing an Interrupt

As mentioned in connection with the Interrupt Mask register, the presence of an interrupt condition signaled to a thread serves merely to reawaken the thread, if it is suspended, or to cancel its suspension, if it is running and executes a SUSPEND instruction. For high priority interrupts with dedicated handler threads, the system design requirement for the interrupt handling time to be less than the inter-arrival time of the interrupt guarantees that the handler will be suspended when the interrupt arrives. Interrupt response, in that case, is immediate.

When independent interrupts share a common interrupt handler thread, it is possible for the handler to be active, responding to a previous interrupt, when a new interrupt arrives. Handling of the new interrupt will then be delayed

until handling of the previous interrupt is completed, and the interrupt handler thread issues a SUSPEND.

In order to minimize interrupt latency for interrupts handled by a common handling thread, the handling functions should be kept short. In some cases, that means using the common interrupt handling thread as a "front end", to force a vectored interrupt to an extended ISR in a thread running a lower priority background process. The instruction sequence to accomplish this is:

1. Halt the target thread by clearing its bit in the Multithreading Enable (MT_EN) register. Note that this does not force the cancellation of any instructions for that thread that are already in the pipe; it merely keeps the thread scheduler from allocating any more cycles to the target thread, until it is re-enabled.
2. Wait until all instructions for that thread have cleared the pipeline.
3. After setting the source thread select field in the CSR to the target thread number, copy its PC and CSR values to control memory, where they can be accessed later.
4. After setting the destination thread select field in the CSR to the target thread number, write the desired ISR address and appropriate CSR value to its PC and CSR.
5. Re-enable the thread by setting its bit in the Multithreading Enable (MT_EN) register.
6. Use the SETCSR instruction to recover the control thread's own destination context.

3.9   Crystal Oscillator

Figure 3-3 shows the connections for attaching a crystal to the OSC oscillator. The crystal is connected across the OSC_IN and OSC_OUT pins. There is about 4pf of capacitance on each of OSC_IN and OSC_OUT pins to VSS. A parallel resonant crystal is recommended that has a maximum ESR of 60 ohms at 12 MHz. A feedback resistor (Rf) of 2 megohms must be connected between OSC_IN and OSC_OUT for reliable crystal startup.

The crystal manufacturer's load capacitance rating (CL) should be equal to (C1 x C2) / (C1 + C2), where C1 = capacitance on OSC_IN (4pF + stray board capacitance + added capacitance), and C2 = capacitance on OSC_OUT (4pF + stray board capacitance + added capacitance). It is recommended that C1 = C2 = 22pF. The trace length between the OSC pins and the crystal should be as short as possible, to avoid noise coupling.
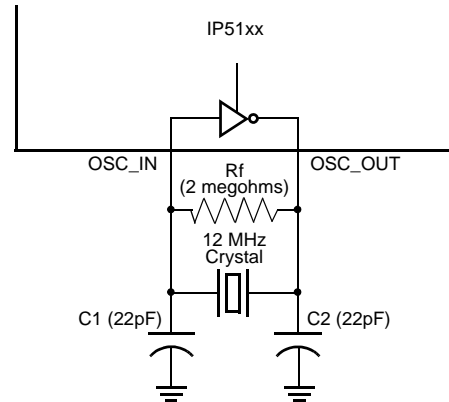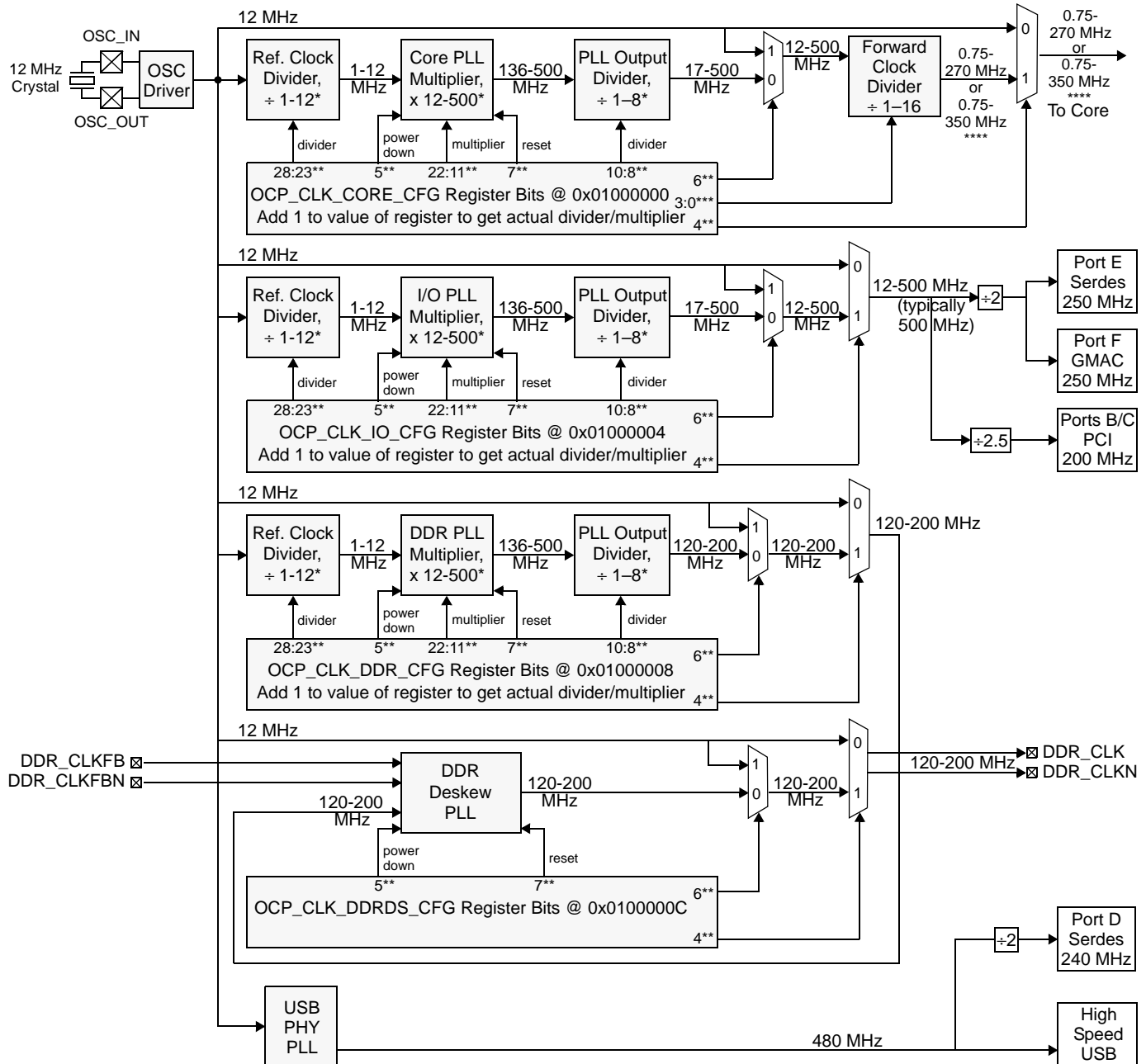
External clocks into OSC_IN are not supported.

Figure 3-3 Crystal Connection

3.10   Clock Circuitry

The clock block supports six independent clock sources for control and operation of the IP51xx. The clock sources are the core clock, the I/O clock, the 200 MHz fixed clock, the DDR SDRAM clock, the real-time clock and the internal R-C clock. The core clock supplies timing for the main processor as well as the subsystems accessed by the main processor such as the system timers and the I/O blocks. The I/O clock provides a clock for the GMAC and Port E Serdes subsystems. It is a 250-MHz fixed frequency clock. The 200-MHz fixed frequency clock provides a clock for the PCI subsystem. The real-time clock provides a precise continuous timing reference, independent of any system configuration (such as low power mode). The R-C clock provides an imprecise continuous timing reference used for tasks such as system reset and initialization. No phase relationship is guaranteed between any of the clocks, even if derived from the same reference.

Figure 3-4 shows the logic for producing core, I/O, and DDR clocks. The clock registers location and layout are specified in Table 7-5.

An external crystal must be connected between OSC_IN and OSC_OUT, as discussed in Section 3.9. External clocks into OSC_IN are not supported.

12 MHz

OSC_IN

12 MHz Crystal

OSC_OUT

OSC Driver

Ref. Clock Divider, ÷ 1-12*

1-12 MHz

Core PLL Multiplier, x 12-500*

136-500 MHz

PLL Output Divider, ÷ 1–8*

17-500 MHz

12-500 MHz

Forward Clock Divider ÷ 1–16

0.75-270 MHz or 0.75-350 MHz ****

0.75-270 MHz or 0.75-350 MHz ****
To Core

divider

power down

multiplier

reset

divider

28:23** 5** 22:11** 7** 10:8** 6**
OCP_CLK_CORE_CFG Register Bits @ 0x01000000 3:0***
Add 1 to value of register to get actual divider/multiplier 4**

12 MHz

Ref. Clock Divider, ÷ 1-12*

1-12 MHz

I/O PLL Multiplier, x 12-500*

136-500 MHz

PLL Output Divider, ÷ 1–8*

17-500 MHz

12-500 MHz

12-500 MHz (typically 500 MHz)

÷2

Port E Serdes 250 MHz

Port F GMAC 250 MHz

÷2.5

Ports B/C PCI 200 MHz

divider

power down

multiplier

reset

divider

28:23** 5** 22:11** 7** 10:8** 6**
OCP_CLK_IO_CFG Register Bits @ 0x01000004
Add 1 to value of register to get actual divider/multiplier 4**

12 MHz

Ref. Clock Divider, ÷ 1-12*

1-12 MHz

DDR PLL Multiplier, x 12-500*

136-500 MHz

PLL Output Divider, ÷ 1–8*

120-200 MHz

120-200 MHz

120-200 MHz

divider

power down

multiplier

reset

divider

28:23** 5** 22:11** 7** 10:8** 6**
OCP_CLK_DDR_CFG Register Bits @ 0x01000008
Add 1 to value of register to get actual divider/multiplier 4**

12 MHz

DDR_CLKFB

DDR_CLKFBN

DDR Deskew PLL

120-200 MHz

120-200 MHz

120-200 MHz

DDR_CLK
DDR_CLKN

power down

reset

5** 7** 6**
OCP_CLK_DDRDS_CFG Register Bits @ 0x0100000C

4**

÷2

Port D Serdes 240 MHz

USB PHY PLL

480 MHz

High Speed USB

* The capability exists to program to any integer value in this range, but, for reliable operation, a value must be used which results in the output frequencies shown in this figure. **Bit field values of N cause divide or multiply by N+1.** For example, if OCP_CLK_CORE_CFG has bits 28:23 = 000000, then the Reference Clock Divider will divide by 1.

** Bits 28 through 5 must not be changed while bit 4 = 1 (this will cause clock glitches).

*** To reduce the core clock to the minimum possible frequency (OSC_IN frequency divided by 16), clear bit 4, set bits 6 and 3:0 (and set bits 5 and 7 if the lowest power is desired), then set bit 4.

**** Only the 350 MHz version is rated for 350 MHz. The 270 MHz version is rated for a max of 270 MHz.

Figure 3-4 Clock Circuitry Diagram

3.11   Reset

The following sources are capable of causing a chip reset:

- Power-on
- External reset (RSTN pin)
- Watchdog module
- Debug port
- Processor trap
- Software initiated reset

When a reset occurs, the reasons for the reset are reflected in the Reset Reason register. For details see Table 7-5.

4.0   Instruction Set

4.1   Operand Addressing

The IP51xx has data types of three principle sizes: 8-bit byte, 16-bit short word and 32-bit long word, as shown in Figure 4-1. There is also a 48-bit data type, used only for accumulator results in the ACC0 or ACC1 register. The byte ordering for operands in memory is big-endian, although bit numbering within registers is little-endian (as shown in Figure 4-1). Big-endian format means that the address of the operand refers to the byte address of the most-significant byte, and bytes are in memory in the order of most to least significant. For example, storing the 16-bit operand 0x1234 at address 0x1000 means that 0x12 is stored at address 0x1000 and 0x34 is stored at address 0x1001.

Both the program and data spaces are byte addressed, and operand addresses must be naturally aligned – that is, they must be integer multiples of the operand size. A misaligned operand address causes a trap. If the misaligned address is a target, the target may or may not be modified, and additional bytes near the target may be modified. When a reference is made to a misaligned data space operand, as a source or target, a TRAP bit is set. The TRAP_CAUSE register for the affected thread captures the reason for the trap event, and the PC register for that thread points to the instruction that caused the trap event.

SHORT WORD ADDRESS
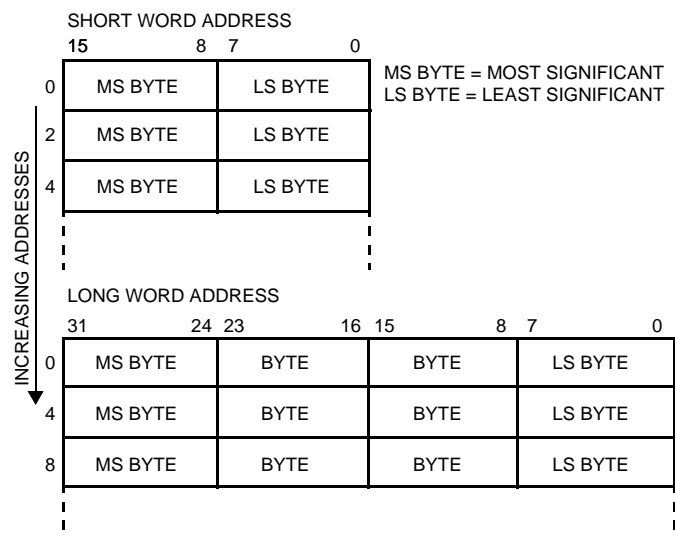
Figure 4-1 Big-Endian Data Formats of IP51xx.

4.2   Addressing Modes

Most of the IP51xx instruction formats (as shown in Table 4-2) use a small number of common fields, which are aligned for ease of decoding. The most important of these are the 11-bit source-1 and 11-bit destination operand specifiers, and the 5-bit source-2 field. The latter may contain either a 5-bit unsigned immediate value, or a data register number (in the right-most 4 bits), depending on the specific instruction. A five-bit immediate value is normally a bit number or a shift count, again depending on the specific instruction. DSP instructions allow a data register, an accumulator, or a 5-bit immediate in this field.

The 11-bit source-1 and the 11-bit destination fields are defined the same. This 11-bit field is used to select one of the following addressing modes (refer to Table 4-1 for more detail):

- Direct Addressing of the register address space
- Register Indirect with 7-bit unsigned offset
- Register Indirect with pre or post increment or decrement
- Register Indirect with indexing
- Immediate: 8-bit value, sign extended to size of operand type

An immediate value in the destination specifier is unusual, but it can be used to prevent write back of the instruction result value to any real destination. The instruction is then executed 'for side effects only' – i.e., setting the condition codes.

To encode all of the above in only 11-bits, variable length encoding is used. The minimum length encoding of a 1-bit is used for the register indirect with offset mode, allowing the maximum number of bits for offset. This results in the following two formats:

Register Indirect With

Offset: `1 i i A A A i i i i i`

Other Addressing Modes: `0 x x A A A n n n n n`

...where:

"i" : Indicates immediate value bits;

"A" : Indicates the three address bits that select one of the 8 address registers;

" x" and "n" : Other fields used by other addressing modes.

Details of this encoding scheme are shown in Figure 4-5.

In the Harvard architecture model, addressing modes for data and program memory space have to be considered

separately. For data address space, the supported addressing modes are described above. For program space, both register indirect and PC relative addressing modes are supported. Addressing modes for both spaces are summarized in Table 4-1.

4.2.1   The Register Address Space

Register addressing mode is used to address the general-purpose registers, D0-D15, as well as the address registers A0-A7, accumulators ACC0/ACC1, SOURCE3, and all on-chip control registers. It is the only mode that can access the core's control registers, because the address space in which these registers reside is not a subset of the general memory address space. A register in the register addressing space cannot be accessed through a regular memory addressing mode that happens to resolve to the same numerical value.

The register addressing space is 256 registers in length, or 1024 bytes. The byte address specified in an assembler statement is right shifted two bits by the assembler, to generate the 8-bit register address offset. Although the assembler syntax requires a byte address, what is addressed in this mode is not bytes, but a space of 256 32-bit register locations. The register addressing space covers access to the following registers:

- All Programmers' Model registers described in Section 3.1.
- On-chip control and status registers for overall chip and timer control.

Because the register addressing space is absolute, an assembler include file of EQU statements can be used to define symbolic names for all the registers.   Then instructions can access these registers directly using their symbolic name as a source or destination operand.

Table 4-1 Addressing Modes

Space	Type	ASM Syntax	Effective Address (EA)
Operand	Register	$xx or Register Mnemonic	No EA. Register address is 10 bits: (8-bit register number) || 00
	Indirect	(An)	EA = An
	Indirect with Offset	offset(An)	EA = An + offset; PDEC only: EA = An - offset (in range 4 to 512); Assembly Syntax: Offset specified in bytes; Opcode Coding: Byte Operand: Offset = 7-bit unsigned immediate value; 16-Bit Operand: Offset = 7-bit unsigned immediate value || 0; 32-Bit Operand: Offset = 7-bit unsigned immediate value || 00; PDEC Operand: Offset = 1111111111111111111111 || 7 bit immediate || 00
	Indirect with Post-Increment	(An)delta++	Step 1: EA = An ; Step 2: An ←An + delta Assembly Syntax: delta specified in bytes; Opcode Coding: Byte Operand: delta = 4-bit signed immediate value; 16-Bit Operand: delta = 4-bit signed immediate value || 0; 32-Bit Operand: delta = 4-bit signed immediate value || 00.
	Indirect with Pre-Increment	delta(An)++	Step 1: An ←An + delta Step 2: EA = An ; Assembly Syntax: delta specified in bytes; Opcode Coding: Byte Operand: delta = 4-bit signed immediate value; 16-Bit Operand: delta = 4-bit signed immediate value || 0; 32-Bit Operand: delta = 4-bit signed immediate value || 00.
	Indirect with Index	(An,Dn)	EA = An + (Dn << $\log_2$(operand size in bytes))
	Immediate	#xxxx #xx	Operand is 16-bit or 8-bit immediate value taken from instruction. Value is sign-extended to 32-bits before use. For 8 bit immediate in general source-1, the EA is the 32-bit sign extended immediate.
CALLI Instruction	Indirect	offset(An)	PEA = An + (sign-extended coded offset <<2) Assembly Syntax: Offset is specified in bytes as a signed 18-bit number. This number is right shifted by two bits for instruction coding. Coded Offset = Assembly Offset[17:0] >> 2
CALL Instruction	Relative	offset(PC)	PEA = PC + (coded offset<<2) Assembly Syntax: Offset is specified in bytes as a signed 26-bit number. This number is right shifted by two bits for instruction coding. Coded Offset = Assembly Offset[25:0] >> 2
JMPcc Instruction	Relative	offset(PC)	PEA = PC + (Opcode Offset)<<2 Assembly Syntax: Offset is specified in bytes as a signed 23-bit number. This number is right shifted by two bits for instruction coding. Opcode Offset = Assembly Offset[23:0] >> 2
Notation: EA: Data Effective Address; PEA: Program Space Effective Address			

4.3   Instruction Set Summary

The instruction set has a fixed-length 32-bit instruction word, and the internal data path is 32 bits wide.

This section discusses the instructions in logical groups, as follows:

- Arithmetic and Logical Operations
- DSP Operations
- Shift and Bit-Field Operations
- Single Bit Operations
- Data Movement And Extension Operations
- Program Control Operations

Detailed descriptions of the individual instructions are presented in Section 4.5.

4.3.1   Arithmetic and Logical Operations

Integer arithmetic support consists of the basic operations of add, subtract, multiply, and multiply-accumulate (MAC). Logical operations include the four basic operations of AND, OR, XOR, and NOT. They perform bit-wise Boolean operations on operands.

The following paragraphs discuss selected instructions in more detail.

ADDC and SUBC

ADDC and SUBC use the C-bit in the 32-bit condition code to implement extended precision arithmetic operations. The C-bit is used for any carry and borrow between different 32-bit words of an extended operand (there is no ADDC.2 or SUBC.2). For SUBC, the complement of the C-bit on input is the "borrow" value for the operation. The borrow is effectively added to the right-hand operand (the subtrahend) before it is subtracted from the left hand operand. (In practice, what that means is that, whereas normal subtraction is implemented by adding the logical complement of the right hand operand to the left hand operand, with a forced '1' as carry in, SUBC uses the input value of the C-bit as the carry in.)

The Z bits are treated differently for ADDC and SUBC than for other instructions. If the result is nonzero, the Z bit is cleared, but if the result is zero, the Z bit is not changed. When adding multiprecision numbers, first an ADD instruction will set or clear the Z bit for the least significant 32 bits. Subsequent ADDC instructions can only clear the Z bit. After the sequence of ADD and ADDC, the Z bit will be set if the multiprecision result is zero.

There is limited scope for inserting instructions between the instruction that sets the carry flag value and the ADDC or SUBC instruction intended to use it. MOVE, and other instructions that don't affect the C flag are safe, but loop end tests are problematic. A loop end test will normally affect the flag value. Therefore, it is desirable to use in-line expansions, rather than loops, for extended precision arithmetic. Since the normal IP51xx arithmetic operations are 32 bits wide, it only takes a single ADD, followed by one ADDC, to perform a 64-bit extended precision add.

In rare cases, where very long extended precision operations must be implemented, it is possible to use the LEA instruction to decrement a loop counter register, and use an EXT instruction to set the Z and N bits of the condition codes, without affecting the C and V flags. That permits a conditional branch on non-zero for the loop end test, at the cost of the extra EXT.

MAC_RC16

The MAC_RC16 register is set by any DSP Instruction that targets ACC0. It is not modified by any other instruction that explicitly targets ACC0. The result being placed in ACC0 is considered to be a number in S16.31 format. This is a two's complement 48-bit fractional number with 31 bits after the decimal point. To create the value in the MAC_R16 register, this 48-bit number is first rounded to have 15 bits after the decimal point. To round, the value of the 16th bit after the decimal point is added to the 15th bit after the decimal. After rounding, the result is clamped to be in the range 0.111111111111111 (binary) to 1.000000000000000 (binary). The sign of the clamped value is the same as the sign of the original number, before rounding, to prevent the highest positive number from being rounded to the lowest negative number. After rounding and clamping, the result is sign extended to 32 bits and stored in the MAC_RC16 register. The resulting format has 17 copies of the sign bit, a decimal point, and 15 fractional bits, in two's complement.

CRCGEN

This is a special purpose instruction for efficient calculation of CRC values in message protocols, such as Ethernet, and bit-stream scrambling. It models the operation of a Linear Feedback Shift Register (LFSR), for any generating polynomial of order 32 or less. It processes eight bits of input at a time. It is defined as follows:

Syntax: CRCGEN s1, s2

Inputs: s1 – next data byte, B (general source, typically from memory)

s2 – generating polynomial, P (Dn register or 5 bit immediate)

ACC0_LO – current CRC value, C

Outputs: ACC0_LO – new CRC value, C'

ACC0_HI – scrambled output byte, S, shifted into the most significant byte

MAC_RC16 - S16.15 image of ACC0_HI, ACC0_LO

Operation:

X = (C ^ B) & 0xFF;

F = X;

for (i=0; i<8; i++) F = (F>>1)^(F&1 ? P : 0);

S = C & 0xFF;

ACC0_LO = F ^ (C >> 8);

ACC0_HI = (ACC0_HI >> 8) | (S << 24);

MAC_RC16 ←S16.15(ACC0_HI, ACC0_LO)

4.3.2   DSP Operations

Digital Signal Processing (DSP) operations perform multiply, multiply-accumulate, add, and subtract, with results going to one of the two accumulators. The accumulators are 48 bits each, split between two 32-bit registers (ACC0_HI / ACC0_LO, ACC1_HI / ACC1_LO). ACC0 can also be referred to using the names MAC_HI / MAC_LO. The 16 most significant bits of the HI accumulator register are set to zero by each operation. The result is not sign extended beyond 48 bits. The A bit in the DSP control field specifies the destination accumulator (0: ACC0. 1: ACC1).

Source-2 for DSP instructions can be an unsigned immediate, a Dn register or an accumulator. Instruction format 10a (in Table 4-2) uses an unsigned 5 bit immediate for the source-2 operand, and the DSP S and T bits are ignored.

For instruction format 10b (in Table 4-2), if the DSP control S-bit is set, the source-2 field in the instruction specifies an accumulator. (0: ACC0, 1: ACC1). MADD and MSUB use the entire 48-bit accumulator as the source. Other instructions use the upper or lower 16-bits of the ACC_LO part of the accumulator, depending on the T bit. If the S-bit is clear, the source-2 field specifies a Dn register.

For instruction format 10b, the DSP control T bit, when set, specifies that a 16-bit source-2 input should be taken from the 16 most significant bits of the source-2 register (Dn or ACC_LO).

The DSP control C bit, when set, causes the result to be clipped. For MULF.C and MACF.C, there is one possible multiply overflow condition that causes clipping. The result of -1.0 * -1.0 is clipped to the largest positive number 0.9999… For MACS.C, MACF.C, MSUF.C, MADD.4.C, and MSUB.4.C, the C bit causes the 48-bit add/subtract result to be clipped to signed 32 bits before being sign extended to 48 bits and stored in the accumulator. For MACU.C and MACUS.C, the C bit causes the 48-bit add result to be clipped to unsigned 32-bits, before being zero extended to 48 bits and stored in the accumulator. For MADD.2.C and MSUB.2.C, the C bit causes the 48-bit add/subtract result to be clipped to signed 16 bits before being sign extended to 48 bits and stored in the accumulator. Overflow of the 48-bit accumulator does not set the O bit in the CSR register.

Whether the C bit is set or not, any operation that would have caused clipping to change the result, will set the O bit in the CSR register.

MSUB and MADD do 48-bit arithmetic. Both source operands are sign extended to 48 bits before the operation if they are not accumulators.

MACUS supports efficient 32x32 bit multiplies with either 32 or 64 bit results.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

4.3.3   Shift and Bit-Field Operations

This category of instructions provides the ability to directly manipulate variable-width fields. Most have two source and one destination operand. The source operand encoded in the source-1 field of the instruction format is a general memory or register operand. The second source operand is invariably a bit number, a shift count, or, in the case of the bit field extract instruction, a two-element bit-field specifier (see below).

All of the instructions in this class, with the exceptions of **bset** and **bclr**, use the restricted 3-operand instruction formats (formats 4a and 4b in Table 4-2 on page 34). Those formats restrict the destination operand to be a data register, but they allow the second source operand to be specified by either a 5-bit immediate field (format 4a) or by the contents of a data register (format 4b).

The **bset** and bclr instructions have their own dedicated format (format 2 in Table 4-2 on page 34). This format only allows for a 5-bit immediate specifier for the second source operand (the bit number), but it allows the destination operand to be a general memory or register operand. This allows **bset** and **bclr** to be used to update peripheral control registers, or to manage bit semaphores in data memory.

The following paragraphs discuss specific shift and bit field instructions in more detail.

4.3.3.1   Bit Field Extract

For **bfextu**, the Unsigned Bit Field Extract instruction, a bit-field is defined by the following parameters in the source-2 operand:

```
Bits 4:0:   Bit-Field Length;
Bits 12:8:  Bit-Field Start.
```

The source-2 operand can be either a data register, or 5-bit immediate field. If the latter is used, it is zero-extended to 32 bits, making the bit field start location automatically zero. The bit field itself resides in the source-1 operand, which is a general EA operand.
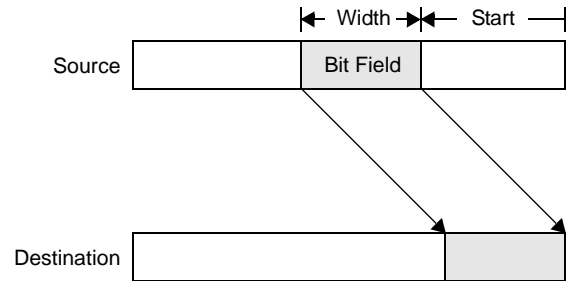

Figure 4-2 Illustration of Bit-Field Extract

4.3.3.2   Bit Merge Instruction

Bits or bit fields from two source operands can also be merged, under control of a selection mask, and saved in a destination operand. This is illustrated in Figure 4-3.

The **merge** instruction takes a general source-1 operand and a source-2 data register, and merges their bits into a general destination operand. Selection of bits is controlled by a mask that must be loaded into the special source-3 operand register prior to the operation. Where there is a 1 in the mask, that bit from source-1 is merged and replaces the corresponding bit from source-2.

The merge instruction performs the merge operation atomically, with respect to other threads that may be running. If the source-1 operand and the destination are the same memory address, or the same control register in the register address space, the merge becomes an atomic read-modify-write operation — again with respect to other threads. However, if the destination is a peripheral register, the operation is *not* atomic, with respect to the peripheral hardware. The latter will see first a read, followed a short time later by a write.
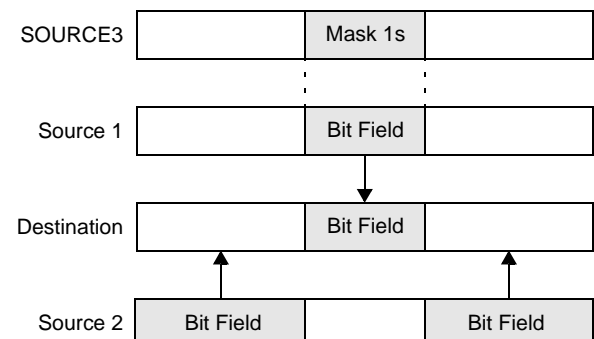

Figure 4-3 Merge Operation

4.3.3.3   Shift and Merge Instructions

Since the IP3000 requires memory operands to be aligned in accordance with the operand size, there is an issue of how to access unaligned data fields within byte streams or external data structures. Two shift and merge instructions, **shmrg.1** and **shmrg.2**, support assembly of larger operands from byte or short word data streams, respectively.

The streams are assumed to have big-endian ordering. **shmrg.1** shifts its 32-bit source-2 operand left by 8 bits, merges the upper 24 bits with the lower 8 bits of its first operand, and places the results in its destination. The second source operand may be a data register or a 5-bit immediate value (zero-extended to 32 bits). The destination operand must be a data register.

shmrg.2 is similar, but the shift is 16 bits and the lower 16 bits of the first operand are merged.

4.3.3.4   Shift Double Instruction

Sometimes a bit field will span word boundaries. The Shift Double instruction, **shftd**, facilitates access to such fields. It uses a 64-bit funnel shifter to right shift two 32-bit source operands that are concatenated together. The shift count comes from the source-2 operand and one of the shifter inputs from the SOURCE3 register. The operation is illustrated in Figure 4-4.
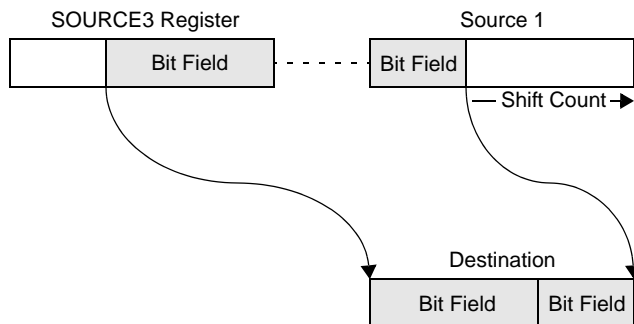

Figure 4-4 Shift Double (shftd) Instruction

4.3.3.5   Other Shift Instructions

The remaining instructions in this category are conventional arithmetic and logical shift operations. In all shift instructions, variant forms are supported for either static or dynamic shift counts.

4.3.4   Single Bit Operations

The single bit instructions handle test, set, and clear of a single bit of a 32-bit operand. For **bset** and **bclr**, both Z condition bits are set according to the state of the specified bit before the set or clear operation. This is different than for all other instructions, which set the condition codes according to the result of their operation. There are for two reasons for the difference:

First, for control of conditional branching, it does not make sense to set the Z bit, when the user knows whether the result will be set or cleared.

Second, and more importantly, setting the Z bit before the operation allows the user to use **bset** as an atomic test and set semaphore instruction.

The **bset** instruction can be used to implement binary semaphores directly. Other types of semaphores can then be implemented in software, using the binary semaphore around them for mutual exclusion.

Examples:

```
bset  (A0), (A0), #8
jmpeq resource_available
```

4.3.5 Data Movement And Extension Operations

This group of instructions performs simple address and data transfers. They are used to move bytes, words and long words from a source address to a destination address.

There are two forms of immediate moves: **movei** (move immediate) and **moveai** (move address immediate). The latter moves a 24-bit immediate value, left-shifted by 7

bits, into an address register. When that register is used as a base register in an addressing mode with a 7-bit offset, it provides immediate addressability to any data location in two gigabytes of data address space.

A special bypass mechanism in the pipeline enables an address register that has been loaded via **moveai** to be used immediately by the following instruction, without the 4-cycle address register load-to-use delay that applies for other address register loads.

Example:

```
move.4  D1,D2
move.1  D1,(A0)
move.4  D9,PORTA_DATA      ; Register Addressing mode
move.2  (A1)2++,(A2)2++
move.2  D4,COEF64(A6)
move.4  (SP)4++,A0         ; Saves address register to stack
move.4  A0,D0              ; Moves D0 →A0
move.4  D0,PARAMETER(SP)   ; PARAMETER = Pos. mod-4 number
movei   D9,#0xffe9         ; D9 = 0xfffffe9
```

4.3.6 Program Control Operations

In program flow control operations, the PC is always incremented by four (when not branching) since all instructions are four bytes long. In terms of the semantic model of the program, the increment takes place after the instruction is executed. Hence, in the descriptions below, "PC" refers to the address of the instruction itself; that is, a PC-relative branch with an offset value of zero branches to itself.

Instruction addressing modes are different from data addressing modes, since they address program space. This is shown with "pea" or Program Effective Address.

For the register-indirect branches, **calli** and **ret**, the two least-significant bits of the register contents are ignored.

Note that **call** and **calli** can be used as ordinary jumps, simply by ignoring the saved return address. **calli** can be used to return if the return address is in an A*n* register. It is typically used this way in leaf routines since it is faster than **ret** on the IP51xx implementation.

4.4   Instruction Formats and Encoding

Table 4-2 shows the formats of instructions. The format codes of the first column provide cross-references for the instruction encodings in Table 4-4.

Table 4-2 Instruction Formats

Format	31 ... 27	26 25 ... 21	20 19 ... 16	15 14 ... 11	10 ... 8	7 ... 5	4 ... 0	Class
1a	Opcode 5 Bits	Unused = 0 11 Bits		Opcode Extension 5 Bits	Unused = 0 11 Bits			No Operand
1b	Opcode 5 Bits	Unused = 0 11 Bits		Opcode Extension 5 Bits	Source-1 11 Bits			1-Operand Source
1c	Opcode 5 Bits	Destination 11 Bits		Opcode Extension 5 Bits	Unused = 0 11 Bits			1-Operand Destination
1d	Opcode 5 Bits	Destination 11 Bits		Opcode Extension 5 Bits	Source-1 11 Bits			2-Operand
2	Opcode 5 Bits	Destination 11 Bits		Bit Number 5Bits	Source-1 11 Bits			BSET, BCLR
3	Opcode 5 Bits	Destination 11 Bits		0 Source-2 Register 4 Bits	Source-1 11 Bits			3-Operand General
4a	Opcode 5 Bits	0 Opcode Extension 5 Bits	0 Dn 4 Bits	Bit Number, Count 5 Bits	Source-1 11 Bits			3-Operand Restricted Dd=G1.op.imm
4b	Opcode 5 Bits	1 Opcode Extension 5 Bits	0 Dn 4 Bits	0 Source-2 Register 4 Bits	Source-1 11 Bits			3-Operand Restricted Dd=G1.op.D2
5	Opcode 5 Bits	Immediate 16 Bits			Source-1 11 Bits			CMPI
6	Opcode 5 Bits	Destination 11 Bits		Immediate 16 Bits				Move Immediate
7	Opcode 5 Bits	Condition 4 Bits	P C	Signed PC-Relative Offset O[20:0] 21 Bits				Conditional Branch P: Prediction Bit C: 16/32 CC select
8	Opcode 5 Bits	O[23:21] 3 Bits	An 3 Bits	Signed PC-Relative Offset O[20:0] 21 Bits				CALL, MOVEAI O: Offset Field
9	Opcode 5 Bits	O[15:13] 3 Bits	An 3 Bits	Offset O[12:8] 5 Bits	Opcode Extension 5 Bits	O[7:5] 3 Bits	Am 3 Bits	O[4:0] 5 Bits	CALLI O: Offset Field
10a	Opcode 5 Bits	0 Opcode Extension 5 Bits	DSP Control 5 Bits	Immediate 5 Bits	Source-1 11 Bits			DSP Imm src2
10b	Opcode 5 Bits	1 Opcode Extension 5 Bits	DSP Control 5 Bits	0 Source-2 Register 4 Bits	Source-1 11 Bits			DSP Reg src2

Table 4-3 defines the DSP Control bits.

Table 4-3 DSP Control Bit Definitions

Bit #	Symbol	Definition
20	C	When set, causes the result to be clipped.
19	T	When set, specifies that a 16-bit Source-2 input should be taken from the 16 most significant bits of the Source-2 register (Dn or ACC_LO).
18	S	Source 2 Select 1: Accumulator: (s2=0:ACC0, s2=1:ACC1) 0: A Dn register
17	Reserved: Set to zero.	
16	A	Destination accumulator: 1: ACC1 0: ACC0

When Source 2 is an immediate value (Format 10a), the S and T bits are ignored.

For a more detailed discussion of these DSP control bits, see Section 4.3.2.

Figure 4-5 shows the 11-bit encodings used in the 11 bit source1 and source2 instruction fields for the various addressing modes.

1	I_6	I_5				I_4	I_3	I_2	I_1	I_0
0	**1**	**1**	A_2	A_1	A_0	0	R_3	R_2	R_1	R_0
0	**1**	**0**				m	I_3	I_2	I_1	I_0
0	**0**	**1**	8-BIT DIRECT ADDRESS							
0	**0**	**0**	8-BIT SIGNED IMMEDIATE							

I[6:0] – 7-bit unsigned immediate value that is left shifted by 0, 1, or 2 depending on operand size.
I[3:0] – 4-bit signed immediate value that is left shifted by 0, 1, or 2 depending on operand size.
M (Mode) – Selects post- (0), or pre- (1) addition of increment mode.
A[2:0] – 3-bit Address Register Selection.
R[3:0] – 4-bit Data Register Selection.
Direct Address – 8-bit Direct Address specifier, left shifted two bits to generate the byte address of a 32-bit register in the direct address space.
Signed Immediate – 8-bit immediate value that is sign-extended to 32 bits for use as an immediate operand.

Figure 4-5 Coding of Addressing Modes.

Table 4-4 specifies instruction formats and opcode assignments for all the instructions described in the preceding sections. Entries in the table are grouped by format, not by functionality. Refer back to Table 4-2 for instruction formats. Opcode and opcode extension values are given in hexadecimal.

Undefined instruction encodings include unassigned opcodes and subops, and instruction bits set to 1 that are defined to be zero. The result of executing an undefined instruction is undefined.

Table 4-4 CPU Instruction Encodings

Type	Mnemonic	Opcodes$_{16}$		Notes
		Primary	**Extension**	
Format 1: 2-operand				
No-Operand Program Control	<reserved>	00	00	
	SUSPEND		01	
One-Operand Source	RET		04	
	BKPT		07	
Two-Operand Data Movement and Unary Operations	NOT.4		0A	
	NOT.2		0B	
	MOVE.4		0C	
	MOVE.2		0D	
	MOVEA		0E	
	MOVE.1		0F	
	SETCSR		12	5
	EXT.2		15	
	EXT.1		17	
	LEA.4		1C	
	LEA.2		1D	
	LEA.1		1F	
	PDEC		1E	
	<reserved>	01	-	
Format 2: BSET, BCLR				
	BSET	04	-	
	BCLR	05	-	
	<reserved>	06, 07	-	
Format 3: 3-Operand, General				
	AND.2	08	-	
	AND.4	09	-	
	OR.2	0A	-	
	OR.4	0B	-	
	XOR.2	0C	-	
	XOR.4	0D	-	
	ADD.2	0E	-	
	ADD.4	0F	-	
	ADDC	10	-	
	SUB.2	11	-	
	SUB.4	12	-	
	SUBC	13	-	
	<reserved>	14-17	-	

Table 4-4 CPU Instruction Encodings (continued)

Type	Mnemonic	Opcodes$_{16}$		Notes
		Primary	**Extension**	
Format 4: 3-Operand, Restricted				
	BTST	02	06	3
	CRCGEN		08	2
	LSL.4		10	
	LSL.2		11	
	LSR.4		12	
	LSR.2		13	
	ASR.4		14	
	ASR.2		15	
	BFEXTU		16	
	BFRVRS		18	
	SHFTD		1A	
	MERGE		1C	
	SHMRG.2		1E	
	SHMRG.1		1F	
	<reserved>	03	-	
Format 5: Compare Immediate				
	CMPI	18	-	3
Format 6: Move Immediate				
	MOVEI	19	-	4
Format 7: Conditional Branch				
	JMP<cc>	1A	-	
Format 8: CALL, MOVEAI				
	CALL	1B	-	
	MOVEAI	1C	-	
	<reserved>	1D	-	
Format 9: Call Indirect				
	CALLI	1E	-	
	<reserved>	1F	-	

Table 4-4 CPU Instruction Encodings (continued)

Type	Mnemonic	Opcodes$_{16}$		Notes
		Primary	**Extension**	
Format 10: DSP				
Src-2 can be Dn or ACC0 or ACC1.	MULS	06	00	
	MACS		01	
Destination can be ACC0 or ACC1.	MULU		02	
	MACU		03	
.C: clip result.	MULF		04	
	MACF		05	
.T: use top 16 bits of src-2.	<reserved>		06	
	MACUS		07	
Multiplies are 16-bit.	<reserved>		08	
	MSUF		09	
Add/sub are 48-bit.	<reserved>		0A-0F	
	MADD.4		10	
	MADD.2		11	
	MSUB.4		12	
	MSUB.2		13	
	<reserved>		14-1F	
Notes: 1. Destination is implicit ACC0_HI, ACC0_LO, MAC_RC16 registers. Destination field in instruction is coded as zero. 2. Destination is implicit CSR register (CCs). 3. If destination is data memory address, operand size is 16 bits. 4. Destination is coded with the direct register address of the CSR register.				

4.5   Detailed Instruction Reference

This section provides detailed information about each instruction. These instructions are presented in alphabetical order.

Table 4-5 shows the functional groups and lists the instructions included in each group. For descriptions applying to these instruction groups, see Section 4.3.

The following points apply to all instructions, in the instruction description tables that follow:

- The ".size" field in instruction syntax refers to the data memory width of operands, and not the instruction width or the ALU result width; both the latter are always 32-bits.
- With arithmetic, logical, and shift operations, if the source is 16-bits, it is always sign extended to 32-bits before the operation to match the width of the internal data path. Only MOVE.1 and MOVE.2 zero-extend a smaller source operand, when the destination is a 32-bit register (i.e., any register in the register address space – most frequently a data register). The 16-bit condition codes allow operation with 16-bit unsigned integers in memory.
- All arithmetic and logical operations are performed in 32-bit resolution. If the destination is in memory and the instruction's operand size is only 16 bits, it is the lower 16 bits of the 32-bit result that is written to memory.
- The source-2 operand, for instructions with more than one input operand, is always either a 32-bit data register, or (in the case of shift and bit field instructions) a 5-bit zero-extended immediate value.

- Data registers, address registers, and other registers in the register address space, used as source or destination operands, are always 32-bits wide.
- An immediate 8-bit value in the source-1 operand specifier is always sign-extended to 32-bits before use.
- The detailed instruction descriptions indicate which condition flags are set by each instruction. In general, any instruction which computes a 32-bit result will set both the N and Z flag bits. Only instructions that can generate a carry or overflow will set the C or V bits. BTST, BSET, and BCLR set only the Z bit. MOVE instructions do not affect the flags.
- The indicated condition flag bits are always set independently in both the 16- and 32-bit condition codes. Thus, if a 32-bit result were all 0's in the lower 16 bits, but had some non-zero bits in the upper 16 bits, the 16-bit condition result would be 1 in the Z bit and 0 in the N bit. The 32-bit result would be 0 in the Z bit and a copy of bit 31 in the N bit.

Example: As the following two instructions demonstrate, the size refers to data memory access size. The arithmetic and logical operations are always performed at 32-bit resolution.

```
; op      dest, s1, s2
ADD.2   (A1),(A0),D2   ; Read 16-Bits →
                       ; Add 32-bits →Store 16-bits
ADD.2   D1,(A0),D2     ; Read 16-Bits →
                       ; Add 32-bits →Store 32-bits
```

Table 4-5 Instructions Included in Each Functional Group

Functional Group	Instructions Included
Arithmetic and Logical Instructions	ADD.2, ADD.4, ADDC, AND.2, AND.4, CMPI, CRCGEN, LEA.1, LEA.2, LEA.4, PDEC, NOT.2, NOT.4, OR.2, OR.4, SUB.2, SUB.4, SUBC, XOR.2, XOR.4
DSP Instructions	MULS[.T], MACS[.C][.T], MULU[.T], MACU[.C][.T], MULF[.C][.T], MACF[.C][.T], MACUS[.C][.T], MSUF[.C][.T], MADD.2[.C][.T], MADD.4[.C], MSUB.2[.C][.T], MSUB.4[.C]
Shift and Bit-Field Instructions	ASR.2, ASR.4, BFEXTU, BFRVRS, LSL.2, LSL.4, LSR.2, LSR.4, MERGE, SHFTD, SHMRG.1, SHMRG.2
Single Bit Instructions	BTST, BSET, BCLR
Data Movement and Extension Instructions	MOVEI, MOVEAI, EXT.1, EXT.2, SETCSR, MOVE.1, MOVE.2, MOVE.4, MOVEA,
Program Control Instructions	JMP<cc>.C.T/F, CALL, CALLI, RET, SUSPEND, BKPT

Table 4-6 lists the abbreviations and symbols used in the detailed instruction descriptions.

Table 4-6 Abbreviations and Symbols Used in the Detailed Instruction Descriptions

Symbol	Descriptions
C	Carry condition code bit (CSR register bit 0 for 16 bits or bit 4 for 32 bits). The C bit is set to the value of the carry out for arithmetic operations where a carry out is a meaningful possibility (add, subtract, and compare). For subtract and compare operations, the C bit contains the complement of the borrow. For other operations, the bit remains unchanged from its value before the operations.
N	Negative condition code bit (CSR register bit 3 for 16 bits or bit 7 for 32 bits). Set if result is negative, cleared otherwise.
V	Overflow condition code bit (CSR register bit 1 for 16 bits or bit 5 for 32 bits). For arithmetic operations where overflow is a meaningful possibility, the V bit is set if overflow occurs and cleared if it does not. For other operations, the bit remains unchanged from its value before the operations.
Z	Zero condition code bit (CSR register bit 2 for 16 bits or bit 6 for 32 bits). Set if result is zero, cleared otherwise. For **addc** and **subc** the Z bit is cleared if the result is nonzero, and left unchanged if the result is zero. This bit does not include any testing of carry bit.
O	DSP Overflow Status (sticky) condition code bit (CSR register bit 20). DSP instructions set this bit if the result is not exact, but never clear it. When a DSP instruction sets this bit, the change is visible several clocks after the DSP instruction.
SE48	Sign extension to 48 bits.
SE32	Sign extension to 32 bits.
ZE48	Zero extension to 48 bits.
S16.15	A 16-bit fractional value (S.15 format) that is sign-extended to 32 bits (S16.15 format).
X^n	n repetitions of bit X.
||	Concatenation of bits.
|	Logical OR.
&	Logical AND.
!	Logical negation.
>>N	Right shift by N bits.
<<N	Left shift by N bits.
d	General-purpose, 11-bit destination operand used to select a register, memory location, or an immediate value.
s1	General-purpose, 11-bit source operand used to select a register, memory location, or an immediate value.
s2	Restricted 4- or 5-bit source operand used to select a data register or an immediate value.
[.T] (top)	In DSP instructions, indicates that the s2 input is from the 16 most significant bits of the s2 register.
[.C] (clip)	In DSP instructions, indicates that the result will be clipped before being stored in the target accumulator.
Dn, Dm	One of 16 data registers.
An, Am	One of 8 address registers.
#<val>	Immediate value.
Rn	Any register.

add.2 d, s1, D*n*   **16/32-Bit Add**
add.4 d, s1, D*n*

31 27	26 16	15	14 11	10 0
OPCODE	DESTINATION		SOURCE-2	SOURCE-1
add.2 0 1 1 1 0	d	0	D*n*	s1
add.4 0 1 1 1 1	d	0	D*n*	s1

Operand size (bytes): 2, 4

Flags affected: C, Z, N, V

Description: d ← s1 + D*n*

For **Add.2**, the 16-bit value in **s1** is added to the 16-bit value in **D*n***. The 16-bit result is stored in **d**.

For **Add.4**, the 32-bit value in **s1** is added to the 32-bit value in **D*n***. The 32-bit result that is stored in **d**.

addc d, s1, D*n*   **32-Bit Add with Carry**

31 27	26 16	15	14 11	10 0
OPCODE	DESTINATION		SOURCE-2	SOURCE-1
1 0 0 0 0	d	0	D*n*	s1

Operand size (bytes): 4

Flag affected: C, Z, N, V

Description: d ← s1 + D*n* + C(32-bit C flag)

The 32-bit value in **s1** is added to the 32-bit value in **D*n***. The 32-bit result is stored in **d**. If a carry occurs, it is stored in **C** (bit 4 of the CSR register).

addc uses **C** to implement extended-precision arithmetic operations. The **C** flag is used for any carry and borrow between 32-bit words of an extended operand.

The **Z** flag is treated differently for **addc** than for other instructions. If the result is nonzero, **Z** is cleared. If the result is zero, **Z** is not changed. When adding multiprecision numbers, an **add** instruction will set or clear **Z** for the least-significant 32 bits. Subsequent **addc** instructions can only clear **Z**. After the sequence of **add** and **addc** instructions, **Z** will be set if the multiprecision result is zero.

and.2 d, s1, D*n*   **Logical AND**
and.4 d, s1, D*n*

31				27	26		16	15	14		11	10		0
	OPCODE					DESTINATION			SOURCE-2				SOURCE-1	
and.2	0	1	0	0	0	d		0	D*n*				s1	
and.4	0	1	0	0	1	d		0	D*n*				s1	

Operand size: 2, 4

Flags affected: N, Z

Description: d ←s1 AND D*n*

The contents of **s1** are combined with the contents of **D*n*** in a bitwise logical AND operation. The result is placed in **d**.

asr.4 D*m,* s1, #cnt   **Arithmetic Shift Right**
asr.2 D*m,* s1, #cnt
asr.4 D*m,* s1, D*n*
asr.2 D*m,* s1, D*n*

31 27	26	25 21	20	19 16	15 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION	SOURCE-2	SOURCE-1

	31 27	26	25 21	20	19 16	15 11	10 0
asr.4	0 0 0 1 0	0	0 1 0 1 0 0	0	D*m*	#cnt	s1
asr.2	0 0 0 1 0	0	0 1 0 1 0 1	0	D*m*	#cnt	s1

31 27	26	25 21	20	19 16	15	14 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION		SOURCE-2	SOURCE-1

	31 27	26	25 21	20	19 16	15	14 11	10 0
asr.4	0 0 0 1 0	1	1 0 1 0 0 0	0	D*m*	0	D*n*	s1
asr.2	0 0 0 1 0	1	1 0 1 0 1 0	0	D*m*	0	D*n*	s1

Operand size (bytes): 2, 4

Flags affected: N, Z

Description: shct ←D*n*, or shct ←#cnt
                     D*m* ←s1[31]$^{\text{shct}}$ || s1[31:shct]

The contents of **s1** are shifted right arithmetically. The number of bits to shift is specified by **Dn** or **#cnt**.

If **s1** is a 16-bit operand, it is sign-extended to 32 bits before the operation.

bclr d, s1, #bit_num   **Bit Clear**

31 27	26 16	15 11	10 0
OPCODE	DESTINATION	SOURCE-2	SOURCE-1
0 0 1 0 1	d	#bit_num	s1

Operand size (bytes): 4

Flag affected: Z

Description: Step 1: Test the bit in **s1** indicated by **#bit_num** and set **Z** accordingly
(Z ←!bit).
Step 2: Clear the selected bit in **d**.

Note: Source and destination operands are typically the same, but this is not
required.

bfextu D*m*, s1, #cnt
bfextu D*m*, s1, D*n*   **Bit Field Extract Unsigned**

31 27	26	25 21	20	19 16	15 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION	SOURCE-2	SOURCE-1
0 0 0 1 0	0	1 0 1 1 0	0	D*m*	#cnt	s1

31 27	26	25 21	20	19 16	15	14 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION		SOURCE-2	SOURCE-1
0 0 0 1 0	1	1 0 1 1 0	0	D*m*	0	D*n*	s1

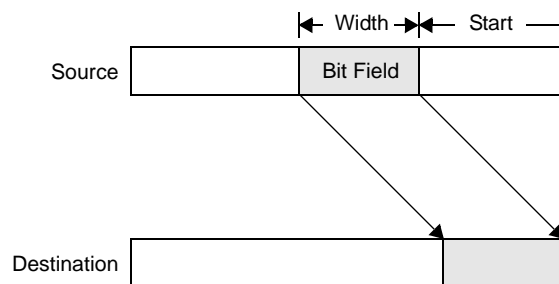**Operand size (bytes):** 4

Flag affected: N, Z

Description:

Extracts a bit-field from the 32-bit **s1** operand and places it in the least-significant bits of **D*m*. D*n*** or **#cnt** contains the parameters:

   Bits 4:0: bit-field width;

   Bits 12:8: bit-field start position (low-order bit);
   (for the **#cnt** format, this field is always five zeros)

The result is zero-extended to 32 bits. If Length equals zero, then result equals zero.
If start + width $\geq$ 32, bit positions $\geq$ 32 are filled with zeros. See Figure 4-6.

This instruction is a combination of an **lsl** by "bit-field start", followed an **and.4** with "bit-field width" bits.

Figure 4-6 Bit-Field Extract

bkpt s1   **Breakpoint**

31 27	26 16	15 11	10 0
OPCODE		OPCODE EXTENSION	SOURCE-1
0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1	s1

Operand size (bytes): n/a

Flags affected: none

Description: PC ←Address of **bkpt** instruction
                  MT_DBG_ACTIVE[*thread*] ←0
                  For each thread *T* specified by s1,
                      MT_DBG_ACTIVE[*7*] ←0
                  MT_BREAK[*thread*] ←1

The **s1** is a bit mask that indicates which additional contexts to suspend (by clearing their MT_DBG_ACTIVE bits). The number of a bit set in the mask is the thread number that is to be suspended. The thread that executes the **bkpt** instruction is always suspended.

bset d, s1, #bit_num   **Bit Set**

31 27	26 16	15 11	10 0
OPCODE	DESTINATION	SOURCE-2	SOURCE-1
0 0 1 0 0	d	#bit_num	s1

Operand size (bytes): 4

Flags affected: Z

Description: Step 1: Test the bit in **s1** indicated by **#bit_num** and set **Z** accordingly
(Z ←!bit).
Step 2: Set the selected bit in **d**.

Note: Source and destination operands are typically the same, but this is not
required.

btst s1, #bit_num   **Bit Test**
btst s1, D*n*

31 27	26	25 21	20 16	15 11	10 0
OPCODE		OPCODE EXTENSION		SOURCE-2	SOURCE-1
0 0 0 1 0	0	0 0 0 1 1 0	0 0 0 0 0	#bit_num	s1

31 27	26	25 21	20 15	14 11	10 0
OPCODE		OPCODE EXTENSION		SOURCE-2	SOURCE-1
0 0 0 1 0	1	0 0 1 1 0	0 0 0 0 0 0	D*n*	s1

Operand size (bytes): 4

Flags affected: Z

Description: $Z \leftarrow !s1[bit]$

The **btst** instruction tests a specified bit of the 32-bit **s1** operand and sets the **Z** accordingly. The bit is specified by the contents of **D*n*** or **#bit_num**.

Note: The **btst** instruction has no destination operand; it is used only for its effect on **Z** in CSR.

call A*n*, offset   **PC-Relative Call to a Subroutine**

31 27	26 24	23 21	20 0
OPCODE	O[23:21]	ADDRESS REGISTER	SIGNED PC-RELATIVE OFFSET O[20:0]
1 1 0 1 1	offset	A*n*	offset

Operand size (bytes):n/a

Flags affected: none

Description: Step 1: A*n* ←PC+4
                     Step 2: PC ←PC + signed offset

offset is two's complement, 26 bits (byte address) in assembly syntax. Only the most-significant 24 bits (word address) are stored for the instruction. The processor shifts this 24-bit **offset** value left by two bits before using it. The return address is saved in the **A*n*** register selected.

Example:

```
CALL  A0, CalcSpeed   ; Jump PC-Relative to Subroutine and
                      ; save return address in A0.
```

calli A*n*, offset(A*m*)   **Address Indirect Call to a Subroutine**

31 27	26 24	23 21	20 16	15 11	10 8	7 5	4 0
OPCODE	O[15:13]	ADDRESS REGISTER	OFFSET[12:8]	OPCODE EXTENSION	O[7:5]	ADDRESS REGISTER	OFFSET[4:0]
1 1 1 1 0	offset	A*n*	offset	0 0 0 0 0	offset	A*m*	offset

Operand size (bytes): n/a

Flags affected: none

Description: Step 1: Target ←A*m* + signed 18-bit offset
Step 2: A*n* ←PC + 4
Step 3: PC ←Target

offset is two's complement, 18 bits (byte address) in assembly syntax. Only the most-significant 16 bits (word address) are stored for the instruction. The processor shifts this 16-bit **offset** value left by two bits before adding it to the contents of **A*m***. The return address is saved into **A*n***.

Example:

```
CALLI  A5, CheckParms(A6) ; Jump indirectly to Subroutine and
                          ; save return address in A5.
```

cmpi s1, #imm-16   **Compare with Immediate Value**

31 27	26 11	10 0
OPCODE	SOURCE-2	SOURCE-1
1 1 0 0 0	#imm-16	s1

Operand size (bytes): 2

Flags affected: C, Z, N, V

Description: s1 − (sign-extended #imm-16)

The **C**, **Z**, **N**, and **V** flags are set according to the results of the operation.

Note: There is no destination operand; this instruction is used only for its effect on the condition codes in CSR.

crcgen s1, #poly
crcgen s1, D*n*

32-bit Incremental CRC Generation

31			27	26	25				21	20				16	15				11	10					0
OPCODE					**OPCODE EXTENSION**										**SOURCE-2**					**SOURCE-1**					
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	#poly					s1					

31			27	26	25				21	20				15	14			11	10					0
OPCODE					**OPCODE EXTENSION**										**SOURCE-2**				**SOURCE-1**					
0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	D*n*			s1					

Operand size (bytes): 1

Flags affected: none

Description: ACC0_LO holds current CRC
s1 specifies the next input byte
D*n* or #poly specifies the generating polynomial
ACC0_LO ←CRC(ACC0_LO, s1, s2)
ACC0_HI[23:0]←ACC0_HI[31:8]
ACC0_HI[31:24]←scrambled output byte
MAC_RC16 ←S16.15(ACC0_HI, ACC0_LO)

This is a special-purpose instruction for efficient calculation of CRC values in message protocols, such as Ethernet, and bit-stream scrambling. It models the operation of a Linear Feedback Shift Register (LFSR) for any generating polynomial of order 32 or less. It processes eight bits of input at a time.

ext.1 d, s1   **Sign-Extend Byte to 32 Bits**

31 27	26 16	15 11	10 0
OPCODE	DESTINATION	OPCODE EXTENSION	SOURCE-1
0 0 0 0 0	d	1 0 1 1 1	s1

Operand size (bytes): 1

Flags affected: N, Z

Description: $d \leftarrow (s1[7])^{24} \,||\, s1[7:0]$

Sign-extend byte from source to 32-bits and store result to destination.

The sign extension is effective only if the destination is a register. If it is a memory destination, the destination size, being the same as the source size, makes the operation equivalent to **move.1**. If the source is a direct register, 8 bits is extracted from that register and sign-extended.

ext.2 d, s1   **Sign-Extend 16 Bits to 32 Bits**

31 27	26 16	15 11	10 0
OPCODE	DESTINATION	OPCODE EXTENSION	SOURCE-1
0 0 0 0 0	d	1 0 1 0 1	s1

Operand size (bytes): 2

Flags affected: N, Z

Description: $d \leftarrow (s1[15])^{16} \parallel s1[15:0]$

Sign-extend 16-bit **s1** to 32-bits and store result in **d**.

The sign extension is effective only if the destination is a register. If it is a memory destination, the destination size, being the same as the source size, makes the operation equivalent to **move.2**. If the source is a direct register, 16 bits is extracted from that register and sign-extended.

jmp<*cc*>.C.P offset   **PC-Relative Conditional Jump**

31	27	26	23	22	21	20	0
OPCODE		CONDITION CODE		P	C	SIGNED PC-RELATIVE OFFSET	
1 1 0 1 0		CC		P	C	offset	

Operand size (bytes): n/a

Flags affected: none

Description: If specified condition function evaluates to TRUE,
then: PC ←PC + signed 23-bit offset
else: PC ←PC + 4
cc: refer to Table 4-7 for definition of the condition codes
C: Condition code set select:
    C = 0: Select 16-bit Condition Codes (S).
    C = 1: Select 32-bit Condition Codes (W).
P: Branch prediction:
    P = 0: Continue (F).
    P = 1: Take branch (T).

offset is two's complement 23 bits (byte address) in assembly syntax. Only the most-significant 21 bits (word address) are stored. The processor shifts this 21-bit offset value left by two bits before using it.

There are two sets of condition codes: 16-bit condition codes and 32-bit condition codes. It is up to the programmer to choose the right set for conditional jump instructions, through the use of the size suffix appended to the jump instruction. **.S** indicates use of the "short," or 16-bit conditions codes, while **.W** indicates use of the "word," or 32-bit condition codes. If no size suffix is used, the assembler uses a default of **.W**.

The IP51xx has static branch prediction, indicated by instruction suffixes **.T** and **.F**. The **T** indicates true or "predict taken," and **F** indicates false or "predict not taken."

There is no branch delay slot; the instruction after the branch is not automatically executed. The branch prediction indicator controls whether the next instruction fetched after the branch is on the "taken" or "not taken'" path[1]. If the branch prediction is subsequently found to be wrong, the fetched instruction is annulled, and an instruction fetch for the correct path is issued.

---

1. In some cases, a fetch for the instruction on the "not taken" path may already have been issued, by default, before the branch instruction is recognized. If the prediction is for "taken," that fetch will be annulled, and a new fetch for the "taken" path will be issued. That fetch, in turn, could be annulled, if the branch, contrary to the prediction, resolves to "not taken." Also note that if the actual branch resolution is available before the next instruction fetch for the thread is issued, the actual resolution overrides any prediction.

Examples:

```
JMPNE.S.T  LOOP_BEGIN  ; Jump Not-Equal using 16-bit CCs
                       ; assuming branch is likely to occur.
JMPCS.W.F  ERROR_COND  ; Use 32-bit CCs, and assume no jump.
```

Table 4-7 Condition Codes

Code$_{10}$	cc	Condition	Test	Signed/ Unsigned
0	F	False	0	Both
1	CC (LO)	Carry Clear (Lower)	!C	Unsigned
2	CS (HS)	Carry Set (Higher or Same)	C	Unsigned
3	EQ	Equal	Z	Both
4	GE	Greater or Equal	(N&V) + (!N&!V)	Signed
5	GT	Greater Than	(N&V&!Z) + (!N&!V&!Z)	Signed
6	HI	Higher Than	C&!Z	Unsigned
7	LE	Less or Equal	Z + (N&!V) + (!N&V)	Signed
8	LS	Lower or Same	!C + Z	Unsigned
9	LT	Less Than	(N&!V) + (!N&V)	Signed
10	MI	Minus	N	Signed
11	NE	Not Equal	!Z	Both
12	PL	Plus	!N	Signed
13	T	True	1	Both
14	VC	Overflow Clear	!V	Signed
15	VS	Overflow Set	V	Signed

Notation:

&   Indicates logical AND operation.

!   Indicates logical negation.

+   Indicates logical OR.

lea.4 d, s1   **Load Effective Address**
lea.2 d, s1
lea.1 d, s1

	31 27	26 16	15 11	10 0
	OPCODE	**DESTINATION**	**OPCODE EXTENSION**	**SOURCE-1**
lea.4	0 0 0 0 0	d	1 1 1 0 0	s1
lea.2	0 0 0 0 0	d	1 1 1 0 1	s1
lea.1	0 0 0 0 0	d	1 1 1 1 1	s1

Operand size (bytes): 1, 2, 4

Flags affected: none

Restriction: **s1** must not be a register.

Description: d ←EA(s1)

Calculates the effective address (EA) for **s1** and stores the address in the **d**.

The destination operand is always 32 bits. If the destination operand is an address register (**A***n*) and the Destination Thread Select bit in CSR is not set, then a fast path eliminates hazards with a later **A***n* use.

Examples:

```
moveai  A0, #%hi(symbol)
lea.4   A1, %lo(symbol)(A0)  ; Word address of symbol into A1
lea.2   A2, %lo(symbol)(A0) ; Half word address of symbol into A2
lea.1   A3, %lo(symbol)(A0)  ; Byte address of symbol into A3

lea.4   A5, (A0,D4)          ; A5 = A0 + (4*D4)
lea.2   A6, (A0,D4)          ; A6 = A0 + (2*D4)
lea.1   A7, (A0,D4)          ; A7 = A0 + (1*D4)

lea.2   A0, 4(A0)            ; A0 = A0 + 4 with no bypass hazard
```

lsl.4 D*m*, s1, #cnt   **Logical Shift Left**
lsl.2 D*m*, s1, #cnt
lsl.4 D*m*, s1, D*n*
lsl.2 D*m*, s1, D*n*

31 27	26	25 21	20	19 16	15 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION	SOURCE-2	SOURCE-1

	31 27	26	25 21	20	19 16	15 11	10 0
lsl.4	0 0 0 1 0	0	1 0 0 0 0	0	D*m*	#cnt	s1
lsl.2	0 0 0 1 0	0	1 0 0 0 1	0	D*m*	#cnt	s1

31 27	26	25 21	20	19 16	15	14 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION		SOURCE-2	SOURCE-1

	31 27	26	25 21	20	19 16	15	14 11	10 0
lsl.4	0 0 0 1 0	1	1 0 0 0 0	0	D*m*	0	D*n*	s1
lsl.2	0 0 0 1 0	1	1 0 0 0 1	0	D*m*	0	D*n*	s1

Operand size (bytes): 2, 4

Flags affected: N, Z

Description: $Dm \leftarrow s1[(31-Dn):0] \parallel 0^{Dn}$
                         $Dm \leftarrow s1[(31-\#cnt):0] \parallel 0^{\#cnt}$

The contents of **s1** are shifted left, inserting zeros into the emptied bits. The result is placed in **D*m***. The number of bits to shift is specified by **D*n*** or **#cnt**.

If **s1** is a 16-bit operand, it is sign-extended to 32 bits before the operation.

lsr.4 D*m*, s1, #cnt   **Logical Shift Right**
lsr.2 D*m*, s1, #cnt
lsr.4 D*m*, s1, D*n*
lsr.2 D*m*, s1, D*n*

31	27	26	25	21	20	19	16	15	11	10	0
OPCODE			OPCODE EXTENSION			DESTINATION		SOURCE-2		SOURCE-1	

	31		27	26	25		21	20	19	16	15	11	10	0
lsr.4	0 0 0 1 0 0			0	1 0 0 1 0		0	Dm		#cnt		s1		
lsr.2	0 0 0 1 0 0			0	1 0 0 1 1		0	Dm		#cnt		s1		

31	27	26	25	21	20	19	16	15	14	11	10	0
OPCODE			OPCODE EXTENSION			DESTINATION		SOURCE-2			SOURCE-1	

	31		27	26	25		21	20	19	16	15	14	11	10	0
lsr.4	0 0 0 1 0 1			1	1 0 0 1 0		0	Dm		0	Dn		s1	
lsr.2	0 0 0 1 0 1			1	1 0 0 1 1		0	Dm		0	Dn		s1	

Operand size (bytes): 2, 4

Flags affected: N, Z

Description: $Dm \leftarrow 0^{Dn} \,||\, s1[31:Dn]$
$Dm \leftarrow 0^{\#cnt} \,||\, s1[31:\#cnt]$

The contents **s1** are shifted right, inserting zeros into the emptied bits. The result is placed in **D*m*.** The number of bits to shift is specified by **D*n*** or **#cnt**.

If **s1** is a 16-bit operand, it is sign-extended to 32 bits before the operation.

macf[.C][.T] acc, s1, #imm
macf[.C][.T] acc, s1, s2

16 x 16-bit Signed Fractional Multiply-Accumulate

31	27	26	25	21	20	16	15	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		0	0 0 1 0 1		C T S 0 A		#imm		s1	

31	27	26	25	21	20	15	14	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		1	0 0 1 0 1		C T S 0 A	0	s2		s1	

Operand size (bytes): 2

Flags affected: O

Description: {ACCn_HI[15:0]:ACCn_LO} ←{ACCn_HI[15:0]:ACC0_LO} +
                                       SE48((s1 * #imm)<<1)

ACCn_HI[31:16] = 0

{ACCn_HI[15:0]:ACCn_LO} ←{ACCn_HI[15:0]:ACCn_LO} +
                                       SE48((s1 * s2)<<1)

ACC0_HI[31:16] = 0

Fractional multiply has an implicit left-shift by one; otherwise, it would end up with two sign bits.

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **macf** instruction performs a multiply-accumulate function on 16-bit fixed point fractional data types (S.15), most commonly used by Digital Signal Processing algorithms. **macf** performs an implicit **mulf** operation and adds the 32-bit result to the 48-bit accumulator value. The format of the accumulator is S16.31.

macs[.C][.T] acc, s1, #imm   **16 x 16-bit Signed Integer Multiply-Accumulate**
macs[.C][.T] acc, s1, s2

31 27	26	25 21	20 16	15 11	10 0
OPCODE		OPCODE EXTENSION	DSP CONTROL	SOURCE-2	SOURCE-1
0 0 1 1 0	0	0 0 0 0 1	C T S 0 A	#imm	s1

31 27	26	25 21	20 15	14 11	10 0
OPCODE		OPCODE EXTENSION	DSP CONTROL	SOURCE-2	SOURCE-1
0 0 1 1 0	1	0 0 0 0 1	C T S 0 A 0	s2	s1

Operand size (bytes): 2

Flags affected: O

Description: {ACCn_HI[15:0]:ACCn_LO} ←{ACCn_HI[15:0]:ACC0_LO} +
                                             SE48(s1 * #imm)
                   ACCn_HI[31:16] = 0

                   {ACCn_HI[15:0]:ACCn_LO} ←{ACCn_HI[15:0]:ACCn_LO} +
                                             SE48(s1 * s2)
                   ACC0_HI[31:16] = 0

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **macs** instruction performs a multiply-accumulate function on 16-bit fixed point signed data types (S.15), most commonly used by Digital Signal Processing algorithms. **macs** performs an implicit **muls** operation and adds the 32-bit result to the 48-bit accumulator value. The format of the accumulator is S16.31.

macu[.C][.T] acc, s1, #imm   16 x 16-bit Unsigned Integer Multiply-Accumulate
macu[.C][.T] acc, s1, s2

31				27	26	25				21	20				16	15				11	10										0
OPCODE						OPCODE EXTENSION					DSP CONTROL					SOURCE-2					SOURCE-1										
0	0	1	1	0	0	0	0	0	1	1	C	T	S	0	A	#imm					s1										

31				27	26	25				21	20				15	14			11	10											0
OPCODE						OPCODE EXTENSION					DSP CONTROL						SOURCE-2			SOURCE-1										
0	0	1	1	0	1	0	0	0	1	1	C	T	S	0	A	0	s2			s1										

Operand size (bytes): 2

Flags affected: O

Description: $\{ACCn\_HI[15:0]:ACCn\_LO\} \leftarrow \{ACCn\_HI[15:0]:ACC0\_LO\} +$
$ZE48(s1 * \#imm)$

$ACCn\_HI[31:16] = 0$

$\{ACCn\_HI[15:0]:ACCn\_LO\} \leftarrow \{ACCn\_HI[15:0]:ACCn\_LO\} +$
$ZE48(s1 * s2)$

$ACC0\_HI[31:16] = 0$

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **macu** instruction performs a multiply-accumulate function on 16-bit fixed point unsigned data types (S.15), most commonly used by Digital Signal Processing algorithms. **macu** performs an implicit **mulu** operation and adds the 32-bit result to the 48-bit accumulator value. The format of the accumulator is S16.31.

macus[.C][.T] acc, s1, #imm
macus[.C][.T] acc, s1, s2

16 x 16-bit Unsigned Integer Multiply-Shift-Accumulate

31 27	26	25 21	20 16	15 11	10 0
OPCODE		OPCODE EXTENSION	DSP CONTROL	SOURCE-2	SOURCE-1
0 0 1 1 0	0	0 0 1 1 1	C T S 0 A	#imm	s1

31 27	26	25 21	20 15	14 11	10 0
OPCODE		OPCODE EXTENSION	DSP CONTROL	SOURCE-2	SOURCE-1
0 0 1 1 0	1	0 0 1 1 1	C T S 0 A 0	s2	s1

Operand size (bytes): 2

Flags affected: O

Description:

$\{ACCn\_HI[15:0]:ACCn\_LO\} \leftarrow \{ACCn\_HI[15:0]:ACC0\_LO\} + ((s1 * \#imm) << 16)$

$ACCn\_HI[31:16] = 0$

$\{ACCn\_HI[15:0]:ACCn\_LO\} \leftarrow \{ACCn\_HI[15:0]:ACCn\_LO\} + ((s1 * s2) << 16)$

$ACC0\_HI[31:16] = 0$

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **macus** instruction performs a multiply-accumulate function on 16-bit fixed point unsigned data types (S.15), most commonly used by Digital Signal Processing algorithms. **macus** performs an implicit **mulu** operation and shifts the 32-bit result left by 16 bits before adding it to the 48-bit accumulator. This instruction supports fast multi-precision multiplies.

madd.4[.C] acc, s1, s2   **48-bit Add**
madd.2[.C][.T] acc, s1, s2

31				27	26	25					21	20					15	14			11	10										0
OPCODE						OPCODE EXTENSION						DSP CONTROL						SOURCE-2				SOURCE-1										

madd.4	0	0	1	1	0	1	1	0	0	0	0	C	T	S	0	A	0		s2						s1							
madd.2	0	0	1	1	0	1	1	0	0	0	1	C	T	S	0	A	0		s2						s1							

Operand size (bytes): 2, 4

Flags affected: O

Description: {ACCn_HI[15:0]:ACCn_LO} ←SE48(s1) + SE48(s2)
                     ACCn_HI[31:16] = 0

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If s2 specifies an accumulator, the entire 48-bit accumulator is used, and the T bit must be zero.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **madd** instruction performs an addition function on signed 48-bit integers. The result is sign-extended to 48 bits and stored in ACC0_HI and ACC0_LO. The upper 16 bits of ACC0_HI are cleared.

msub.4[.C] acc, s1, s2   48-bit Subtract
msub.2[.C][.T] acc, s1, s2

31	27	26	25	21	20	15	14	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	

msub.4	0	0	1	1	0	1	1	1	0	0	1	0	C	T	S	0	A	0	s2	s1
msub2	0	0	1	1	0	1	1	1	0	0	1	1	C	T	S	0	A	0	s2	s1

Operand size (bytes): 2, 4

Flags affected: O

Description: {ACCn_HI[15:0]:ACCn_LO} ←SE48(s1) - SE48(s2)
ACCn_HI[31:16] = 0

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If s2 specifies an accumulator, the entire 48-bit accumulator is used, and the T bit must be zero.

For MSUB.2, s2 is always 16 bits. If s1 is a register, it is 32 bits. Where s1 and s2 are equal, the difference (for example, MSUB.2 acc, Dx, Dx) is not necessarily zero.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **msub** instruction performs a subtraction function on signed 48-bit integers. The result is sign-extended to 48 bits and stored in ACC0_HI and ACC0_LO. The upper 16 bits of ACC0_HI are cleared.

msuf[.C][.T] acc, s1, #imm
msuf[.C][.T] acc, s1, s2

16 x 16-bit Signed Fractional Multiply-Subtract

31			27	26	25			21	20			16	15			11	10			0
OPCODE					OPCODE EXTENSION				DSP CONTROL				SOURCE-2				SOURCE-1			
0	0	1	1	0	0	0	1	0	0	1	C	T	S	0	A	#imm		s1		

31			27	26	25			21	20			15	14			11	10			0
OPCODE					OPCODE EXTENSION				DSP CONTROL				SOURCE-2				SOURCE-1			
0	0	1	1	0	1	0	1	0	0	1	C	T	S	0	A	0	s2		s1	

Operand size (bytes): 2

Flags affected: O

Description: {ACCn_HI[15:0]:ACCn_LO} ←{ACCn_HI[15:0]:ACC0_LO} -
$$SE48((s1 * \#imm) <<1)$$
ACCn_HI[31:16] = 0

{ACCn_HI[15:0]:ACCn_LO} ←{ACCn_HI[15:0]:ACCn_LO} -
$$SE48((s1 * s2) <<1)$$
ACC0_HI[31:16] = 0

Fractional multiply has an implicit left-shift by one; otherwise, it would end up with two sign bits.

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **msuf** instruction performs a multiply-subtract function on 16-bit fixed point fractional data types (S.15), most commonly used by Digital Signal Processing algorithms. **msuf** performs an implicit **mulf** operation and subtracts the 32-bit result from the 48-bit accumulator value. The format of the accumulator is S16.31.

merge D*m*, s1, #imm   **Bitwise Merge**
merge D*m*, s1, D*n*

31 27	26	25 21	20	19 16	15 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION	SOURCE-2	SOURCE-1
0 0 0 1 0	0	1 1 1 0 0 0		D*m*	#imm	s1

31 27	26	25 21	20	19 16	15	14 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION		SOURCE-2	SOURCE-1
0 0 0 1 0	1	1 1 1 0 0 0		D*m*	0	D*n*	s1

Operand size (bytes): 4

Flags affected: N, Z

Restriction: SOURCE3 register must be loaded with a 32-bit selection control mask (**msk**) before the operation.

Description: This is a 32-bit merge that operates as follows:
D*m* ←(s1 & msk) | (#imm & !msk)
D*m* ←(s1 & msk) | (D*n* & !msk)

The **merge** instruction takes **s1** and either **D*n*** or **#imm**, merges their bits, and places the result in **D*m***. Selection of bits is controlled by **msk**. Where there is a 1 in the **msk**, that bit from **s1** is merged and replaces the corresponding bit from **D*n***. See Figure 4-7.

Note: **#imm** is zero-extended to 32 bits for this instruction.
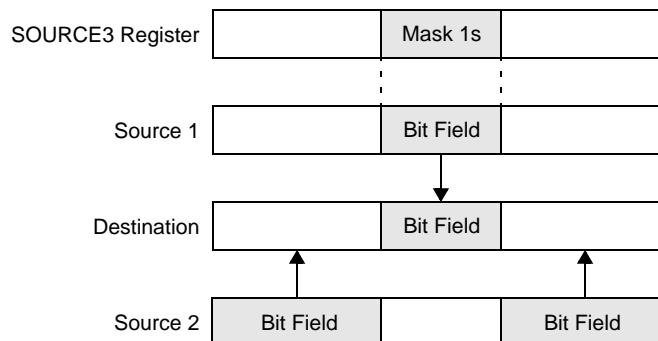
Figure 4-7 Merge Operation

The **merge** instruction performs the merge operation atomically, with respect to other threads that may be running. If **s1** and **D*m*** are the same register, the merge becomes an atomic read-modify-write operation-again with respect to other threads. However, if the destination is a peripheral register, the operation is not atomic, with respect to the peripheral hardware. The latter will first see a read followed shortly by a write.

move.4 d, s1   Move
move.2 d, s1
movea d, s1
move.1 d, s1

	31 27	26 16	15 11	10 0
	OPCODE	DESTINATION	OPCODE EXTENSION	SOURCE-1
move.4	0 0 0 0 0	d	0 1 1 0 0	s1
move.2	0 0 0 0 0	d	0 1 1 0 1	s1
moveA	0 0 0 0 0	d	0 1 1 1 0	s1
move.1	0 0 0 0 0	d	0 1 1 1 1	s1

Operand size (bytes): 1, 2, 4

Flags affected: none

Description: d ←s1

8-Bit Move:

If **d** is a directly addressed register, then the upper 24 bits (31:8) are cleared to zero.

16-Bit Move:

If **d** is a directly addressed register, then the upper 16 bits (31:16) are cleared to zero.

Both the **s1** and the **d** address must be two-byte aligned, The least-significant bit of the addresses must be zero; otherwise, the result is architecturally undefined.

32-Bit Move:

Moves a long word from source to destination.

Both the **s1** and the **d** address must be quad-byte aligned, The two least-significant bits of addresses must be zero; otherwise, the result is architecturally undefined.

For **movea**, if the destination causes a cache miss, and the destination address is a multiple of 32, the cache line is not read from memory, and the other 28 bytes on that cache line are undefined.

Examples:

```
move.4   D1,D2
move.1   D1,(A0)
move.4   D9,PORTA_DATA      ; Register addressing mode
move.2   (A1)2++,(A2)2++
move.2   D4,COEF64(A6)
move.4   (SP)4++,A0         ; Saves address register to stack
move.4   A0,D0              ; Moves D0 →A0
move.4   D0,PARAMETER(SP) ; PARAMETER = Pos. mod-4 number
```

moveai A*n*, #imm-24   Move 24-bit Address Immediate Value

31 27	26 21	20	0
OPCODE	O[23:21]	ADDRESS REGISTER	SIGNED PC-RELATIVE OFFSET O[20:0]
1 1 1 0 0	#imm-24	A*n*	#imm-24

Operand size (bytes): 4

Flags affected: none

Description: Moves the value in #imm-24 into bits [30:7] of A*n*. Bit 31 and bits [6:0] of A*n* are cleared.

When the destination address register is used as a base register in an addressing mode with a 7-bit offset, it provides immediate addressability to any data location in two gigabytes of data address space.

A special bypass mechanism in the pipeline enables an address register that has been loaded via **moveai** to be used immediately by the following instruction, without the 3-cycle address register load-to-use delay that applies for other address register loads.

movei d, #imm-16   **Move 16-Bit Immediate Value**

31 27	26 16	15 0
OPCODE	DESTINATION	SOURCE-1
1 1 0 0 1	d	#imm-16

Operand size (bytes): 4, 2

Flags affected: none

Description: if (destination is a directly addressed register)
then
d ←SE32(#imm16)
else destination is a data memory address, so
d ←#imm16   ; 2-byte destination size

mulf[.C][.T] acc, s1, #imm
mulf[.C][.T] acc, s1, s2

16-bit Signed Fractional Multiply

31	27	26	25	21	20	16	15	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		0	0 0 1 0 0		C T S 0 A		#imm		s1	

31	27	26	25	21	20	15	14	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		1	0 0 1 0 0		C T S 0 A	0	s2		s1	

Operand size (bytes): 2

Flags affected: O

Description: {ACCn_HI[15:0]:ACCn_LO} ←SE48((s1 * #imm)<<1)
ACC0_HI[31:16] = 0

{ACCn_HI[15:0]:ACCn_LO} ←SE48((s1 * s2)<<1)
ACC0_HI[31:16] = 0

Fractional multiply has an implicit left-shift by one; otherwise, it would end up with two sign bits.

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

Sticky O bit is only set, never cleared by these instructions.

The **mulf** instruction performs a multiply function on 16-bit fractional data types (S.15) most commonly used by Digital Signal Processing algorithms. This instruction generates a 32-bit number in the S.31 format, sign-extends the 32-bit result to 48 bits, and stores the result to the 48 least-significant bits of ACC0_HI and ACC0_LO. The upper 16 bits of ACC0_HI are cleared. **#imm** represents a number in the range 0 to $31*2^{-15}$.

muls[.T] acc, s1, #imm
muls[.T] acc, s1, s2

16-bit Signed Integer Multiply

31	27	26	25	21	20	16	15	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		0	0 0 0 0 0		C T S 0 A		#imm		s1	

31	27	26	25	21	20	15	14	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		1	0 0 0 0 0		C T S 0 A	0	s2		s1	

Operand size (bytes): 2

Flags affected: none

Description: {ACCn_HI[15:0]:ACCn_LO} ←SE48(s1 * #imm)
ACCn_HI[31:16] = 0

{ACCn_HI[15:0]:ACCn_LO} ←SE48(s1 * s2)
ACCn_HI[31:16] = 0

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

The **muls** instruction performs a multiply function on signed 16-bit integers. The result is sign-extended to 48 bits and stored in ACC0_HI and ACC0_LO. The upper 16 bits of ACC0_HI are cleared.

mulu[.T] acc, s1, #imm　　　　　　　　　　　　　　　**16-bit Unsigned Integer Multiply**
mulu[.T] acc, s1, s2

31	27	26	25	21	20	16	15	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		0	0 0 0 1 0		C T S 0 A		#imm		s1	

31	27	26	25	21	20	15	14	11	10	0
OPCODE			OPCODE EXTENSION		DSP CONTROL		SOURCE-2		SOURCE-1	
0 0 1 1 0		1	0 0 0 1 0		C T S 0 A	0	s2		s1	

Operand size (bytes): 2

Flags affected: none

Description:　　{ACCn_HI[15:0]:ACCn_LO} ←ZE48(s1 * #imm)
　　　　　　　　　ACCn_HI[31:16] = 0

　　　　　　　　　{ACCn0_HI[15:0]:ACCn_LO} ←ZE48(s1 * s2)
　　　　　　　　　ACC0_HI[31:16] = 0

s2 can be either Dn or acc; acc can be either ACC0 or ACC1.

If the destination is ACC0, the MAC_RC16 register will get the rounded and clipped result.

For definitions of the DSP control bits, see Table 4-3.

The **mulu** instruction mulu performs a multiply function on unsigned 16-bit integers. The result is zero-extended to 48 bits and stored in the ACC0_HI and ACC0_LO registers.

not.4 d, s1   **Logical Negation**
not.2 d, s1

31	27	26	16	15	11	10	0
OPCODE		DESTINATION		OPCODE EXTENSION		SOURCE-1	

	31 27	26 16	15 11	10 0
not.4	0 0 0 0 0	d	0 1 0 1 0	s1
not.2	0 0 0 0 0	d	0 1 0 1 1	s1

Operand size (bytes): 2, 4

Flags affected: N, Z

Description: d ←NOT s1

Performs a bitwise logical NOT operation on the contents of **s1**. The result is placed in **d**.

or.2 d, s1, D*n*   **Logical OR**
or.4 d, s1, D*n*

31	27	26	16	15	14	11	10	0
OPCODE		DESTINATION			SOURCE-2		SOURCE-1	

or.2	0 1 0 1 0	d	0	D*n*		s1
or.4	0 1 0 1 1	d	0	D*n*		s1

Operand size (bytes): 2, 4

Flags affected: N, Z

Description: d ←s1 OR D*n*

Combines the contents of **s1** with the contents of **D*n*** in a bitwise logical OR operation. The result is placed in **d**.

pdec d, s1   **Indirect with Offset**

31	27	26	16	15	11	10	0
OPCODE		DESTINATION		OPCODE EXTENSION		SOURCE-1	
0 0 0 0 0		d		1 1 1 1 0		s1	

Operand size (bytes): 4

Flags affected: none

Description: d ←EA(s1)

Identical to **lea.4**, except that the 7-bit offset in base+offset addressing mode is extended to 32 bits by adding 21 one-bits (1^{21} || 7-bit immediate || 00).

ret s1   **Return from Subroutine**

31	27	26	16	15	11	10	0
OPCODE				OPCODE EXTENSION		SOURCE-1	
0 0 0 0 0		0 0 0 0 0 0 0 0 0 0 0		0 0 1 0 0		s1	

Operand size (bytes): n/a

Flags affected: none

Description: s1 →PC

The **s1** operand is conventionally specified as SP or A7, but this is not a requirement.

setcsr s1   **32-Bit Move to CSR**

31 27	26 16	15 11	10 0
OPCODE	DESTINATION	OPCODE EXTENSION	SOURCE-1
0 0 0 0 0	CSR	1 0 0 1 0	s1

Operand size (bytes):4

Flags affected: none

Description: CSR of current context ←s1

This instruction is used to switch context back, since changing the destination context field of CSR is otherwise not possible to undo.

The assembler encodes the destination field with the direct register address of the CSR register.

shftd D*m*, s1, #imm   **Shift Double**
shftd D*m*, s1, D*n*

31	27	26	25	21	20	19	16	15	11	10	0
OPCODE			OPCODE EXTENSION			DESTINATION		SOURCE-2		SOURCE-1	
0 0 0 1 0		0	1 1 0 1 0		0	D*m*		#imm		s1	

31	27	26	25	21	20	19	16	15	14	11	10	0
OPCODE			OPCODE EXTENSION			DESTINATION			SOURCE-2		SOURCE-1	
0 0 0 1 0		1	1 1 0 1 0		0	D*m*		0	D*n*		s1	

Operand size (bytes): 4

Flags affected: N, Z

Restriction: SOURCE3 register must be loaded with a 32-bit value to be shifted before
                     the operation.
                     s1 is a 32-bit operand.

Description: 64-bit funnel shift that operates as follows:
                     D*m* ←((SOURCE3 || s1) >> #imm)[31:0]

                     D*m* ←((SOURCE3 || s1) >> D*n*[4:0])[31:0]

Sometimes a bit field spans word boundaries. The shift-double instruction, **shftd**, facilitates access to
such fields. It uses a 64-bit funnel shifter to right-shift two 32-bit source operands that are concatenated.
The shift count comes from **D*n*** or **#imm** and one of the shifter inputs from the SOURCE3 register. The
operation is illustrated in Figure 4-8.
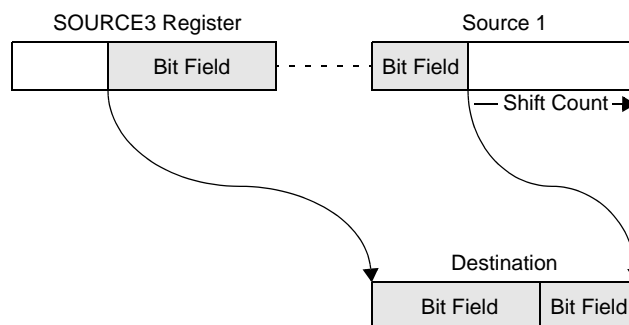
Figure 4-8 Shift-Double (shftd**) Instruction**

shmrg.2 D*m*, s1, #imm   **Shift and Merge**
shmrg.1 D*m*, s1, #imm
shmrg.2 D*m*, s1, D*n*
shmrg.1 D*m*, s1, D*n*

31 27	26	25 21	20	19 16	15 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION	SOURCE-2	SOURCE-1

shmrg.2	0 0 0 1 0	0	1 1 1 1 0	0	D*m*	#imm	s1
shmrg.1	0 0 0 1 0	0	1 1 1 1 1	0	D*m*	#imm	s1

31 27	26	25 21	20	19 16	15	14 11	10 0
OPCODE		OPCODE EXTENSION		DESTINATION		SOURCE-2	SOURCE-1

shmrg.2	0 0 0 1 0	1	1 1 1 1 0	0	D*m*	0	D*n*	s1
shmrg.1	0 0 0 1 0	1	1 1 1 1 1	0	D*m*	0	D*n*	s1

Operand size (bytes): 1, 2

Flags affected: N, Z

Description: Shift and merge 1 byte:
D*m* ←(D*n* << 8) | (s1 & 0x000000FF)
D*m* ←(#imm << 8) | (s1 & 0x000000FF)

Shift and merge 2 bytes:
D*m* ←(D*n* << 16) | (s1 & 0x0000FFFF)
D*m* ←(#imm << 16) | (s1 & 0x0000FFFF)

Because the CPU requires memory operands to be aligned in accordance with the operand size, there is an issue of how to access unaligned data fields within byte streams or external data structures. The shift and merge instructions, **shmrg.1** and **shmrg.2**, support assembly of larger operands from byte or short word data streams, respectively.

sub.2 d, s1, D*n*   **16/32-Bit Subtract**
sub.4 d, s1, D*n*

	31				27	26					16	15	14		11	10			0
	OPCODE					DESTINATION							SOURCE-2			SOURCE-1			
sub.2	1	0	0	0	1	d						0	D*n*			s1			
sub.4	1	0	0	1	0	d						0	D*n*			s1			

Operand size (bytes): 2, 4

Flags affected: C, Z, N, V

Description: d $\leftarrow$ s1 - D*n*

The contents of **D*n*** are subtracted from the contents of **S1**. The result is placed in **d**.

subc d, s1, D*n* **32-Bit Subtract with Carry**

31	27	26	16	15	14	11	10	0
OPCODE		DESTINATION			SOURCE-2		SOURCE-1	
1 0 0 1 1		d		0	D*n*		s1	

Operand size (bytes): 4

Flags affected: C, Z, N, V

Description: d ←s1 – D*n* – !C (32-bit C Flag)
where: C = 1 if there is no borrow.

The contents of **D*n*** are subtracted from the contents of **s1**. The result is placed in **d**. If a carry occurs, **C** is set to 1.

subc uses the **C** flag in the 32-bit condition code to implement extended-precision arithmetic operations. The **C** is used for any carry and borrow between different 32-bit words of an extended operand. (There is no **subc.2**.) The complement of the **C** on input is the borrow value for the operation. The borrow is effectively added to the right-hand operand (the subtrahend) before it is subtracted from the left-hand operand. (In practice, what that means is that, whereas normal subtraction is implemented by adding the logical complement of the right-hand operand to the left-hand operand, with a forced 1 as carry in, **subc** uses the input value of the **C** as the carry in.)

Z is treated differently for **subc** than for other instructions. If the result is nonzero, **Z** is cleared, but if the result is zero, **Z** is not changed.

suspend   Suspend

31	27	26	16	15	11	10	0
OPCODE				OPCODE EXTENSION			
0 0 0 0 0		0 0 0 0 0 0 0 0 0 0 0		0 0 0 0 1		0 0 0 0 0 0 0 0 0 0 0	

Operand size (bytes): n/a

Flags affected: none

Description: PC ←Address of next instruction
MT_ACTIVE[thread] ←0

Suspends the current thread until an interrupt condition for that thread occurs.

xor.2 d, s1, D*n*
xor.4 d, s1, D*n*

Logical Exclusive-OR

	31 27	26 16	15	14 11	10 0
	OPCODE	DESTINATION		SOURCE-2	SOURCE-1
xor.2	0 1 1 0 0	d	0	D*n*	s1
xor.4	0 1 1 0 1	d	0	D*n*	s1

Operand size (bytes): 2, 4

Flags affected: Z, N

Description: d $\leftarrow$ s1 XOR D*n*

Combines the contents of **s1** with the contents of **D*n*** in a bitwise logical Exclusive-OR operation. The result is placed in **d**.

5.0   Programmer's Reference

5.1   IP51xx Startup and Initialization

At startup, or after reset, thread 0 is active and schedulable (bit 0 in MT_EN, MT_ACTIVE, and MT_DEBUG_ACTIVE are set to 1). All other threads are disabled. Thread 0 begins execution at a fixed flash ROM address (6000 0000) and is responsible for these initialization steps:

1. Load the instruction SRAM.
2. Load the HRT table and all thread control registers (including PC).
3. Initialize global semaphores and shared memory.

When the initialization is complete, thread 0 enables the other initialized threads, which then are free to execute.

5.2   Interrupt Handling

The multithreaded nature of the Instruction Set Architecture enables a much more efficient alternative to a traditional vectored interrupt system. This section describes how that alternative works, and how applications can be structured to take advantage of it.

5.2.1   Context Switching

There are two types of context switching: hardware and software.

Hardware context switching is the switching between different physical contexts maintained by the multithreading hardware. The overhead for this type of context switch is zero. The processor can switch from execution in thread A to thread B with no cycles lost to any switching overhead.

Software context switching is what an RTOS does when it preemptively suspends the execution of the currently running task in order to reassign the processor resource that the task was using to a newly enabled task of a higher priority. This is also what has to happen in systems with vectored interrupts. Because the transfer is asynchronous, the ISR has no way of knowing what registers the interrupted task was using or whether the RTOS will even return to that task after the interrupt has been handled. The ISR must save the full context of the interrupted task before it begins to do its own work.

Unlike the zero-overhead hardware context switch, a software context switch is no less expensive on the IP51xx processor than it is on most other processor architectures. If anything, it can be more expensive because of the larger number of physical registers that must be saved and restored, and the inability for software to save or restore more than one register per cycle. As a result, software context switching should be avoided as much as possible.

5.2.2   Avoiding Software Context Switching

The multithreading architecture makes it easy to avoid most software context switching. Rather than preempting a running task in order to use its thread resources, interrupts can be handled in one or more other threads dedicated to that purpose.

When there are no interrupts that need to be handled, the interrupt handling threads are suspended and consume no processor cycles. When an interrupt is asserted, it enables the execution of its handler thread. The handler thread, having higher priority, preempts processor cycles that would otherwise have been allocated to some other thread, but it does not use any of the context resources of the threads it displaces. There is no interrupt context that it needs to save, so the interrupt handler can begin immediately to service the interrupt.

The two 32-bit interrupt mask registers (INT_MASK0 and INT_MASK1) allow the thread-scheduling logic fine-grained control as to which interrupt signals are seen by which threads. A thread can be:

- Shielded from all interrupts.
- Set up to respond to one particular interrupt or a particular set of related interrupts.
- Set up to respond to all interrupts that do not have their own dedicated handler threads.

The most time-critical interrupts will have dedicated, HRT threads waiting to respond to them the instant they are asserted. Less critical interrupts can be grouped for handling by a common handler thread. In that case, the handler does not automatically know which interrupt will occur next, and a small overhead is paid at the start of the handler's execution to identify which interrupt has awakened it. This overhead can be on the order of ten cycles (including the branch to the appropriate sub-handler), which is still an order of magnitude less than the overhead incurred for a software context switch.

5.2.3　Minimizing Interrupt Latency

When one thread handles multiple independent interrupts, it is not guaranteed that the thread will be suspended and waiting for an interrupt when the next interrupt arrives. In order to avoid the cost of a forced software context switch, the new interrupt does not preempt the handling of a previous interrupt. Instead, it waits (pending) until the handler has completed its response to the prior interrupt. At that point, the handler issues a suspend operation to wait for the next interrupt that it is configured to handle. The presence of the pending interrupt causes the thread to immediately re-awaken so that the suspend effectively becomes a no-op, even though pipeline timing may not be identical.

The latency incurred while an interrupt waits for a common interrupt handler to finish responding to a previous interrupt is conceptually no different from the latency incurred, in a vectored priority interrupt system, when a lower priority interrupt must wait while a higher priority interrupt is handled. For the IP51xx's interrupt system, the latency tends to be lower, because interrupt handlers are not saddled with the overhead of software context switching. And, of course, the latency can be avoided altogether, for critical interrupts, simply by assigning the interrupt its own handling thread. It does, however, suggest that as a matter of design policy, interrupt handlers should be kept short. If an interrupt requires an extended response, the handler should simply 1) read and save any critical information needed for interrupt response, and 2) signal the RTOS to activate a task that does the extended processing required to fully respond to the interrupt.

5.2.4　Creating a Traditional ISR Structure

The inclusion of multithreading features in the IP51xx core does not preclude the use of a traditional Interrupt Service Routine (ISR) structure. In most processor architectures there are two threads: the mainline code that executes most of the time, and the ISR that executes periodically and mutually exclusively with the mainline code.

By using two threads, a traditional ISR can be created.

The mainline code is executed in one or more low priority NRT threads. The ISR is a thread that is high priority NRT. By using this set of thread priorities each mainline thread will be completely excluded by the higher priority ISR thread, creating the same scheduling mechanism as a traditional ISR.

5.3　Using the Debug Port

The Debug Port provides a test interface through which an external host can access internal resources within the IP51xx for debug purposes. The signals on the Debug Port are described in Section 6.10.

Section 5.3.1 explains how the host communicates with the debug port and lists the available host commands.

Section 5.3.2 lists the debug port registers, and shows how the various debug commands access these registers.

Section 5.3.3 lists a variety of debug operations that can be performed, and details the command sequences that can accomplish each of these operations.

5.3.1　Debug Commands

The host communicates with the debug port by issuing 40-bit packets. A packet consists of an 8-bit opcode, and a 32-bit payload. The 8-bit opcode consists of a 0 followed by a 7-bit command.

Registers in the debug port which are accessible by the host are addressed by the command. Each Host access consists of two packets — one packet is sent from the host to the debug port and one packet is sent to the host from the debug port. The latter, which is also 40 bits, consists of a 0 followed by the previous opcode followed by the data which is expected by that (previous) command. This provides the host with a mechanism to read back the content of one or more specified registers. Some host commands set or clear registers. Others write the content of packet's payload to a register, or read back a register.

Table 5-1 shows the available host commands.

Table 5-1 Debug Interface Host Commands

Command Code	Command Name	Description
7'h00	NOP	Perform no operation.
7'h01	OPEN	Activate the debug port.
7'h02	CLOSE	Close the debug port. Future commands will be ignored.
7'h03-04	Reserved	
7'h05	MAIL_STATUS	Read the mailbox status. Note: Always wait more than 3 core clock cycles after reading the mail box (RD_MAIL_BOX command) before issuing the MAIL_STATUS command. This allows time for proper updating of the mail status to reflect the current state of MP_OUT_MBOX[31:0].
7'h06	DBG_RST_REQ	Request a core reset.
7'h07	WR_MAIL_BOX	Write a 32-bit word into the incoming mailbox. The debug port returns the status of the mailbox buffers.
7'h08	RD_MAIL_BOX	Read a 32-bit word from the outgoing mailbox buffer.
7'h09	CLEAN_MAIL	Reset the incoming mailbox's write pointer and the outgoing mailbox's read pointer such that the two mailboxes are empty.
7'h0A	TEST_MODE	Write to the Test Register with data.
7'h0B-12	Reserved	
7'h13	HALT_MP	Stop all threads (by setting the DBG_MP_HALT register to 1).
7'h14	REL_MP	Release all threads (by clearing the DBG_MP_HALT register to 0).
7'h15	FORCE_MP	Force a thread to execute (through the dbg_mp_force vector). The DBG_MP_FORCE ['MP_TNUM-1:0] register is set by Data bits ['MP_TNUM-1:0]. Only one thread can be forced at any given time.
7'h16	RD_MP_REG	Perform a read of processor registers.
7'h17	SET_MP_REG_ADR	Set the processor register address used for register write. Data contains the processor register address.
7'h18	WR_MP_REG	Write data to the processor register whose address is set by the last SET_MP_REG_ADR command.
7'h19	RD_IPORT_STAT	Read the status of the instruction port.
7'h1A	WR_IBUF	Write a 32-bit instruction to a 1-entry instruction buffer (IB) targeting a thread.
7'h1B	WR_RST_HALT_MP_EN	Write to the DBG_RST_HALT_MP_EN register. When set, this register enables the halt of the processor when internally generated reset conditions occur. When clear, internally generated reset conditions cause a chip reset.
7'h1C	RD_RST_HALT_MP	Read the status of the DBG_MP_HALT and DBG_RST_CAUSE_HALT registers.
7'h1D-7F	Reserved	

5.3.2   Debug Registers

Table 5-2 Debug Interface Host Accessible Registers

Register Name	Read By	Written By	Reset	Description
MP_IN_MBOX_FULL	MAIL_STATUS, WR_MAIL_BOX, CLEAN_MAIL: {mp_in_mbox_full, mp_in_mbox_empty, mpout_mbox_full, mp_out_mbox_empty, 28'h0}	CLEAN_MAIL: 0	0 (1)	MP Input MBox full
MP_IN_MBOX_EMPTY		CLEAN_MAIL: 1 WR_MAIL_BOX: 0	1 (1)	MP Input MBox empty
MP_OUT_MBOX_FULL		CLEAN_MAIL: 0 RD_MAIL_BOX: 0	0 (1)	MP Output MBox full
MP_OUT_MBOX_EMPTY		CLEAN_MAIL: 1	1 (1)	MP Output MBox empty
MP_IN_MBOX[31:0]*		WR_MAIL_BOX: cur_pkt_data[31:0]		MP Input MBox FIFO input
MP_OUT_MBOX[31:0]*	RD_MAIL_BOX			MP Output MBox FIFO output
DBG_INT_SET		WR_MAIL_BOX set to '1' for one cycle.		Interrupt request to MP
DBG_MP_HALT	HALT_MP, REL_MP: (3) {31'h0,dbg_mp_halt} WR_RST_HALT_MP_EN, RD_RST_HALT_MP: {29'h0,dbg_rst_cause_halt, dbg_mp_halt, dbg_rst_halt_mp_en}	HALT_MP: 1 REL_MP: 0	0 (2)	Halt MP Blocks all instruction execution.
FORCE_MP ['MP_TNUM-1:0]	FORCE_MP: (3) {(32-'MP_TNUM)'h0, dbg_mp_force ['MP_TNUM-1:0]}	FORCE_MP: cur_pkt_data [MP_TNUM-1:0]	0 (1)	Force MP Thread Force chosen thread to run.
DBG_MP_RREQ		RD_MP_REG: set to '1' for one cycle.		Request to read specified MP register
MP_DBG_RDATA[31:0]	RD_MP_REG: mp_dbg_rdata[31:0]			Content of MP register (signal mp_dbg_rdata), the address of which is specified in the payload of the RD_MP_REG command and is stored in DBG_MP_ADDR_ WDATA
DBG_MP_WR_ADR[31:0]	SET_MP_REG_ADR: (3) dbg_mp_wr_adr[31:0]	SET_MP_REG_ADR: cur_pkt_data[31:0]		MP register address to be used in command WR_MP_REG
DBG_MP_WREQ		WR_MP_REG: set to '1' for one cycle.		Request to write to specified MP register
DBG_MP_ADDR_ WDATA[31:0]		RD_MP_REG: cur_pkt_data[31:0] WR_MP_REG: dbg_mp_wr_adr[31:0] – (1st cycle) cur_pkt_data[31:0] – (2nd cycle)		Contains address of an MP register and data to be written to that MP register.

Table 5-2 Debug Interface Host Accessible Registers (continued)

Register Name	Read By	Written By	Reset	Description
IFETCH_REQ_TNUM[3:0]	RD_IPORT_STAT, WR_IBUF: {ifetch_req_tnum[3:0], ifetch_req_addr[27:2], ifetch_req_vld, ibuf_empty}			Contains the thread number of a thread that is fetching from the Debug Port. Resides in bits [31:28] of the response word.
IFETCH_REQ_ ADDR[25:0]	RD_IPORT_STAT, WR_IBUF: {ifetch_req_tnum[3:0], ifetch_req_addr[27:2], ifetch_req_vld, ibuf_empty}			Contains bits [27:2] of the address of the instruction that is fetching from the Debug Port. Resides in bits [27:2] of the response word.
IFETCH_REQ_VLD	RD_IPORT_STAT, WR_IBUF: {ifetch_req_tnum[3:0], ifetch_req_addr[27:2], ifetch_req_vld, ibuf_empty}		0 (1)	Set to '1' if the instruction fetch request to the Debug Port is valid. Resides in bit [1] of the response word.
IBUF_EMPTY	RD_IPORT_STAT, WR_IBUF: {ifetch_req_tnum[3:0], ifetch_req_addr[27:2], ifetch_req_vld, ibuf_empty}	WR_IBUF: 0	1 (1)	Instruction Buffer Empty. Set to '0' when the host puts a word to IBUF; set to '1' when the CPU reads from it. Resides in bit [0] of the response word.
INST_BUF[31:0]		WR_IBUF: cur_pkt_data[31:0]		Instruction Buffer
DBG_RST_HALT_MP_EN	WR_RST_HALT_MP_EN, RD_RST_HALT_MP: {29'h0, dbg_rst_cause_halt, dbg_mp_halt, dbg_rst_halt_mp_en}	WR_RST_HALT_MP_EN: cur_pkt_data[0]	0 (2)	If set to '1' it enables halt of the CPU on any internal reset cause. Resides in bit [0] of the response word.
DBG_RST_CAUSE_HALT	WR_RST_HALT_MP_EN, RD_RST_HALT_MP: {29'h0, dbg_rst_cause_halt, dbg_mp_halt, dbg_rst_halt_mp_en}		0 (2)	It is set to '1' if DBG_RST_HALT_MP_EN=1, and an internal reset cause happens. It is set to '0' when HALT_MP is cleared. Resides in bit [2] of the response word.

Notes:
1. MP = CPU = processor.
2. (1) specifies a core reset.
3. (2) specifies a chip reset.
4. (3) returns the new value of the register that is set or written by this same command.
5. * specifies that this is actually a FIFO, rather than a register.

5.3.3   Debug Operations

Ubicom uses GDB as the front-end for its debugger implementation. The backend for GDB utilizes the Debug Port on the IP51xx to retrieve and set state in the processor.

The host can perform the following functions through the Debug Port:

- Starting a debugging session
- Ending a debugging session
- Stopping all threads
- Restarting all threads
- Setting up the debugger assistant thread
- Restoring the debugger assistant thread to its original state
- Reading register data
- Writing register data
- Reading memory
- Writing memory
- Flushing the d-cache for a range of addresses and invalidating the i-cache for the address range
- Setting a breakpoint
- Removing a breakpoint
- Single-stepping code
- Switching threads to inspect the state of a different thread
- Waiting for breakpoint events
- Resetting the CPU
- Erasing and writing flash

The rest of this section describes the Debug Port command sequences and processor instruction sequences that are injected via the debug space to implement each of the above operations.

In the following discussions, CPU refers to the processor, and IP5K refers to the IP51xx.

5.3.3.1   Starting a Debugging Session

Any external debugger must first activate the Debug Port to accept Debug Port commands. This is done by issuing **OPEN** via the debug port. Until the IP5K sees this command, it will ignore any commands issued to the Debug Port. On reset the processor is in a state where it ignores all commands. The IP5K backend will issue the **OPEN** command when the user issues the **target ip5k** command from the debugger command line.

5.3.3.2   Ending a Debugging Session

When a user decides to close down a debugging session by either issuing **detach** or **quit** commands, the Ubicom debugger will restore the state of the current thread back to what it was at the point to attachment. It will restart all the threads that were running at the point of attachment. The backend will then issue a **CLOSE** command to shutdown the debugger interface.

All subsequent description assumes that the **OPEN** command has been issued and the debug port is actively accepting commands.

5.3.3.3   Stopping All Threads

To stop all threads issue a **HALT_MP** command with a payload data of 1. This will freeze all threads.

5.3.3.4   Restarting All Threads

To restart all the threads issue **REL_MP** with a payload data of 0. The will allow all threads to resume execution.

5.3.3.5   Setting Up the Debugger Assistant Thread

All operations described from this point on need the assistance of code running on the IP5K. The Ubicom debugger running on the host will first stop all the threads by issuing a **HALT_MP** command via the backend. The debugger backend then commandeers one of the **IP5K** hardware threads to act as a debugger assistant. Before a thread can be used as a debugger assistant, some of its register state has to be preserved. The debugger preserves **previous_pc, pc, d0, a0, a1, d0**. During the processor restart process these registers will be restored.

The sequence of commands listed below is issued to set up a debugger assistant thread. In this example, the debugger assistant thread is hardware thread number 2.

1. Retrieve the current PC for thread 2: Issue **RD_MP_REG** with payload (0xd0 | 2 <<10). Issue **NOP** to retrieve the PC from the debugger port.
2. Change thread 2 PC to 0x1000000 (point to debug space): Issue **SET_MP_REG_ADDR** with payload (0xd0 | 2<<10). Issue **WR_MP_REG** with payload 0x1000000.
3. Force thread 2 to start running: Issue **FORCE_MP** with payload (1 << 2).
4. Issue **RD_PORT_STAT**, **RD_PORT_STAT**. Port status tells us whether we can issue commands via the Debug Port.
5. Get the cpu to execute a **move.4 scratchpad0, previous_pc** instruction by sending it through the Debug Port: Issue **WR_IBUF** with payload (move.4 scratchpad0, previous_pc).
6. Read content of **scrathcpad0** register: Issue **RD_MP_REG** with payload 0x180. This is to retrieve **previous_pc** from **scratchpad0**.

7. Get the cpu to execute a **move.4 scratchpad3, a1** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scratchpad3, a1).

8. Get the cpu to execute a **move.4 scratchpad2, a0** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scratchpad2, a0).

9. Get the cpu to execute a **move.4 scratchpad1, csr** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scratchpad1, csr).

10. Get the cpu to execute a **move.4 scratchpad0, d0** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scratchpad0, d0).

11. Get the cpu to execute a **movei a0, #%hi(0x01000300)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (movei a0, %hi(0x01000300). Load **a0** with base address of mailbox. The mailbox is used in other data transfer operations.

12. Read content of the **scrathcpad0** register: Issue **RD_MP_REG** with payload 0x180. This is to retrieve **d0** from **scratchpad0**.

13. Read content of the **scrathcpad1** register: Issue **RD_MP_REG** with payload 0x184. This is to retrieve **csr** from **scratchpad1**.

14. Read content of the **scrathcpad2** register: Issue **RD_MP_REG** with payload 0x188. This is to retrieve **a0** from **scratchpad2**.

15. Read content of the **scrathcpad3** register: Issue **RD_MP_REG** with payload 0x18c. This is to retrieve **a1** from **scratchpad3**.

16. Issue **NOP** to finish off the command sequence.

5.3.3.6 Restoring the Debugger Assistant Thread to its Original State

The sequence of commands listed below is used to restore the state of a debugger assistant thread back to its original state. It should be noted that **previous_pc** is a read only register, and it cannot be restored to its original state. Registers **d0, a1, a2, csr** can be restored as follows:

1. Load the saved copy of **d0** to the mailbox: Issue **WRITE_MAILBOX** with payload copy of **d0**.

2. Get the cpu to execute **move.4 d0, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 d0, (a0)).

3. Load saved copy of **csr** to the mailbox: Issue **WRITE_MAILBOX** with payload copy of **csr**.

4. Get the cpu to execute a **move.4 csr, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 csr, (a0)).

5. Load saved copy of a2 to the mailbox: Issue **WRITE_MAILBOX** with payload copy of **a1**.

6. Get the cpu to execute a **move.4 a1, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 a1, (a0)).

7. Load saved copy of **a0** to the mailbox: Issue **WRITE_MAILBOX** with payload copy of a0.

8. Get the cpu to execute **move.4 a0, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 a0, (a0)).

9. Stop the debug assistant thread: Issue **FORCE_MP** with payload 0.

10. Change thread 2 PC to the original address: Issue **SET_MP_REG_ADDR** with payload (0xd0 | 2<<10). Issue **WR_MP_REG** with payload saved value of **pc**.

5.3.3.7 Reading Register Data

To read the register contents of thread 5 (for example) we have to make thread 5 the debugger assistant thread using the sequence described in Section 5.3.3.5. If the current debugger assistant thread is not 5 and happens to be thread 2, then we first restore the state of thread 2 by using the sequences listed in Section 5.3.3.6. We then make thread 5 the debugger assistant using the sequences in Section 5.3.3.5. Once thread 5 has been set up as the debugger assistant, we then do the following to retrieve the contents of register **a5**.

1. Get the cpu to execute a **move.4 scratchpad0, a5** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scratchpad0, a5).

2. Read content of the **scrathcpad0** register: Issue **RD_MP_REG** with payload 0x180. Issue **NOP**. This is to retrieve **a5** from **scratchpad0**.

If the request is to recover contents of **d0, a0, a1, pc, previous_pc**, then the debugger backend routines serve up the data that was saved while setting up the debugger assistant thread.

5.3.3.8 Writing Register Data

To write the register contents of thread 5 we have to make thread 5 the debugger assistant thread, as mentioned in Section 5.3.3.7. Once that is done, then issue the following to change the contents of register **a5**:

1. Load the new value of **a5** to the mailbox. Issue **WRITE_MAILBOX** with payload the new value of **a5**.

2. Get the cpu to execute a **move.4 a5, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 a5, (a0)).

If the destination register is **d0, a0, a1, csr, pc**, then the debugger backend routine changes these values in the internal saved state that is it maintaining for the debugger assistant thread. The new values will take effect as soon as this thread is restored.

5.3.3.9   Reading Memory

The Ubicom debugger backend is usually passed a source address and a length (in bytes) to read. If the read address is not word aligned the backend will round the address down to the lower word aligned address. If the end address (start address + length) is not word aligned then the backend will increase the length until the end address is also word aligned. The aligned end address minus the aligned start address gives the length in bytes that will be read via the Debug Port interface. The debugger backend then supplies only the data requested by the front end. The (aligned length)/4 gives the number of words that have to be read from the IP5K. The sequence for reading bytes of data is as follows:

1. Calculate the end address of the transfer. End addr = start addr+length
2. Round down the start address.
3. Round up the end address.
4. Number of words = (End addr - Start addr)/4
5. Load start address to the mailbox: Issue **WRITE_MAILBOX** with payload start address of the transfer.
6. Get the cpu to execute a **move.4 a1, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 a1, (a0)). Register **a1** now holds the base address of the transfer block.
7. While (num words –)
8. Get the cpu to execute a **move.4 scrathpad0, (a1)4++** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scrathpad0, (a1)4++).
9. Read the content of **scratchpad0** register: Issue **RD_MP_REG** with payload 0x180. This is to retrieve memory from **scratchpad0**.
10. Issue **NOP**
11. Go back to Step 7.

5.3.3.10   Writing Memory

As in the case of read, we limit writes to be word aligned. To do that, a given data transfer can be separated into 3 parts. There may be leading section that is 1, 2, or 3 bytes long because the start is misaligned. There is a center section where everything is aligned. There may also be a trailing section that is 1, 2, or 3 bytes long because the end address is misaligned. If there is a leading misaligned section, we round down the address and read 4 bytes from that address. We then change the extra leading bytes in this block of 4 bytes. For the trailing bytes we read the 4 bytes that contain those and change the appropriate bytes. Now we can have a write buffer where we can start writing from an aligned boundary and end the writing at an aligned boundary. Once we are done with the leading and trailing manipulation we will have an aligned buffer of data to write and we know the start address of this aligned block and its length in words. Now the sequence to write this data is as follows:

1. Load the start address to the mailbox: Issue **WRITE_MAILBOX** with payload the start address of the transfer.
2. Get the cpu to execute a **move.4 a1, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 a1, (a0)). Register **a1** now holds the base address of the transfer block.
3. While (num words –)
4. Load the transfer data to the mailbox: Issue **WRITE_MAILBOX** with the transfer data. Bump up the buffer pointer.
5. Get the cpu to execute a **move.4 (a1)4++, (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 (a1)4++, (a0)).
6. Go back to Step 3.

5.3.3.11   Flushing the D-Cache and Invalidating the I-Cache

The backend is passed the start address and length in bytes for the data block. Compute the end address of the block. Each cache line is 32 bytes long. Use the following sequence to accomplish this:

1. Compute the end address (Start address + length).
2. Set the number of cache lines = 0.
3. If end address is not cache aligned, then set the number of cache lines = 1.
4. End_address &= ~0x1f.
5. Start_address &= ~0x1f
6. Number of cachelines += (Start address - end address)/32
7. Get the cpu to execute a **move.4 scratchpad0, a2** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 scratchpad0, a2). This saves **a2**, which will be restored at the end.

8. Get the cpu to execute a **movei a1, #%hi(0x1000600)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (movei a1, #%hi(0x1000600). This loads the base address of the D-Cache control block into **a1**.
9. Get the cpu to execute a **movei a2, #%hi(0x1000500)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (movei a2, #%hi(0x1000500). This loads the base address of the I-Cache control block into **a2**.
10. Load 0x90 to the mailbox: Issue **WRITE_MAILBOX** with payload 0x90.
11. Get the cpu to execute a **move.4 16(a1), (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 16(a1), (a0)). This loads the D-cache control register with a "Flush D-cache by address" operation.
12. Load 0x60 to the mailbox: Issue **WRITE_MAILBOX** with payload 0x60.
13. Get the cpu to execute a **move.4 16(a2), (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 16(a2), (a0)). This loads the I-cache control register with an "Invalidate I-cache by address" operation.
14. While (number of cache lines –)
15. Load the start address to the mailbox: Issue **WRITE_MAILBOX** with the start address as the payload.
16. Get the cpu to execute a **move.4 (a1), (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 (a1), (a0)). This loads the D-cache address register with the cache line start address.
17. Get the cpu to execute a **bset 16(a1), 16(a1), #0x3** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (bset 16(a1), 16(a1), #0x3). Activate the D- cache flush operation by setting the start bit in the cache control register.
18. Load the start address to the mailbox: Issue **WRITE_MAILBOX** with the start address as the payload.
19. Get the cpu to execute a **move.4 (a2), (a0)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (move.4 (a2), (a0)). This loads the I-cache address register with the cache line start address.
20. Get the cpu to execute a **bset 16(a2), 16(a2), #0x3** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (bset 16(a2), 16(a2), #0x3). Activate the I- cache invalidation operation by setting the start bit in the cache control register.
21. Start Address += 32
22. Go back to Step 14.

5.3.3.12 Setting a Breakpoint

Setting a breakpoint is just a special case of reading and writing memory. The Ubicom backend does not support writing breakpoints to flash. To insert a break point you have to read back the instruction that is currently present at the address and save it in the host and then write a **bkpt, #-1** instruction to cause all threads to stop when it gets executed. If the destination of the breakpoint is in the DDR SDRAM, then flush the D-Cache and invalidate the I-cache for this address, using the sequence described in Section 5.3.3.11.

5.3.3.13 Removing a Breakpoint

Removing a breakpoint is just a special case of reading and writing memory. In the Ubicom backend we will read the memory at the given location and make sure that it is a **bkpt, #-1** instruction. If it is, then we replace it with the instruction data passed to us by the GDB front end. If the destination of the breakpoint is in the DDR SDRAM, then we flush the D-Cache and invalidate the I-cache for this address, using the sequence described in Section 5.3.3.11.

5.3.3.14 Single-Stepping Code

In this example we single step thread 4. Currently thread 4 is being used as the debugger assistant thread. The sequence is as follows:

1. If the debugger assistant thread is the same as the single stepping thread, then restore the state of the single stepping thread. Now pick some other thread and make it the debugger assistant thread. Refer to Section 5.3.3.5 and Section 5.3.3.6 for details.
2. Get the cpu to execute a **movei mt_dbg_active_clr, #-1** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (movei mt_dbg_active_clr, #-1). This will suspend all threads and keep them from executing.
3. Get the cpu to execute a **movei mt_dbg_active_set, #(1<<4)** instruction by sending it through the debug port: Issue **WR_IBUF** with payload (movei mt_dbg_active_set, #(1<<4). This makes only thread 4 active and runable.
4. Get the cpu to execute a **movei mt_single_step, #(1<<4)** instruction by sending it through the debug port. Issue **WR_IBUF** with payload (movei mt_single_step, #(1<<4). This enables thread 4 to run for only one instruction.
5. Restore the debugger assistant thread.
6. Restart all threads. Issue **REL_MP**.
7. While(1)

8. Read the content of the **mp_dbg_active** register: Issue **RD_MP_REG** with payload 0x144. Issue **NOP**.
9. If **mp_dbg_active** is zero, break; else go to Step 7.
10. Halt all threads: Issue **HALT_MP**.

5.3.3.15   Switching Threads

Restore the state for the current debugger assistant thread. Set up the new thread as the debugger assistant thread. Now read the register contents.

5.3.3.16   Waiting for Breakpoint Events

The state of the cpu can be determined by reading the contents of the **mt_dbg_active**, **mt_trap**, **mt_break** and **RST_HALT_MP** registers, and then analyzing the results. Under normal running conditions **mt_dbg_active** will be non-zero, **mt_trap** should be 0, **mt_break** should be 0, and **RST_HALT_MP** bit 1 should be 0. **mt_dbg_active** will go to zero if a single stepping sequence is being run. It can also go to zero if a **bkpt, #-1** instruction has been executed by IP5K code. **mt_break** will be non-zero if any thread executes a **bkpt** instruction. **mt_trap** will be non-zero if trap events happen in the system. **RST_HALT_MP** bit 1 is set if a software issued reset happens in the system. If any of these events take place, control is transferred back to the GDB front end.

5.3.3.17   Resetting the CPU

To reset the cpu, issue **DBG_RST_REQ** with payload 0 via the debug port. This will only reset the cpu. If **WR_RST_HALT_MP_EN** with a non-zero payload was issued prior to **DBG_RST_REQ**, then the cpu will reset and stop at 0x60000000. If **WR_RST_HALT_MP_EN** was never issued, then the cpu will start running immediately. An **OPEN** command will have to be issued to the debug port to make the debug port start accepting debug commands.

5.3.3.18   Effect of Internal Resets

Most internally generated reset signals would not reset the chip, even if DBG_RST_HALT_MP_EN is set. Instead, a rst_dbg_halt_mp signal would be asserted for one core clock cycle to indicate to the Debug Module that one or more internally generated reset conditions have occurred.

The Debug Module would then halt the processor. The reasons for the reset would be available in the Reset Reasons register (see Section 5.18.5 and Table 7-5).

The rst_dbg_halt_mp signal is generated by the following equation:

```
rst_dbg_halt_mp <=
internal_resets & dbg_reset_halt_mp_en
```

The three resets that would cause normal chip reset regardless the state of the DBG_RST_HALT_MP_EN register are the power-on reset, the external reset, and debug reset.

5.3.3.19   Erasing and Writing Flash

Ubicom creates a standalone piece of executable code called **Loader Kernel** that can fit and run from the **Off Chip Memory**. The **Loader Kernel** consists of the following pieces of code:

1. PLL initialization code.
2. Other initialization code.
3. Main dispatch loop.
4. Code to calculate CRC for a given block of flash memory.
5. Flash driver code.

The **Loader Kernel** is part of the Ubicom SDK and it built for every project. The **Loader Kernel** is linked into the project elf file as a section called **.downloader**.

The **Loader Kernel** consists of 2 threads: thread 1 moves data in and out of the mailbox, and thread 2 is the flash write and erase thread. The **Loader Kernel** uses a double buffer scheme to speed up the download process. Thread 1 receives command requests and data via the mailbox interface.

For a flash erase operation the host sends down an Erase request command, start address, and length in bytes to erase. Thread 1 receives this data, copies it into buffer 1, wakes up thread 2, and then suspends itself, waiting for thread 2 to wake it up. Thread 2 upon wake up picks up the command from buffer 1, switches the command receive buffer to buffer 2 and then wakes up thread 1. This allows thread 1 to receive the next command while the flash erase is in progress.

For flash write operations, the host sends a Flash write command, start address, and number of bytes to write followed by that many bytes of data. Thread 1 will receive all this and copy it to the current write buffer. It then wakes up thread 2 and suspends itself until thread 2 wakes it up. Thread 2 will wake up and then switch the current write buffer to the next available buffer and wake up thread 1 to receive more requests. Thread 2 will then start moving the data from the buffer to the proper location in flash.

5.4   Data Cache

The Data Cache in the IP51xx has the size of 8 Kbytes. It is 4-way associative and has a 32-byte cache line, which means that there are four 2-Kbyte banks, each containing 64 lines of 32 bytes.

In the following discussions, CPU refers to the processor, and IP5K refers to the IP51xx.

5.4.1   Data Cache Policies

CPU source and destination operands that reference off-chip DDR memory use the D-Cache. The D-cache supports simultaneous read and write operations with a Write Queue. If a source operand is found in the cache (hits the cache), data is returned immediately to the CPU. If a target operation hits the cache, the data is written to the cache, and that cache line is marked dirty. There is no write-through to memory.

When a read request does not find data in the cache, the thread that made the request is blocked while 32 bytes of data is fetched from memory. A blocked thread does not schedule any instructions. While the cache is fetching data from off chip memory, it can still satisfy read requests from other threads. If there are additional cache misses, the cache will queue one miss per thread. While the cache is fetching data from off-chip memory, writes are stored in the write queue. If the write queue fills up, any thread that writes to the cache will be blocked.

When there is a cache miss, the cache selects a 32-byte cache line to replace using a pseudo-random policy among the 4 lines associated with the set defined by the address of the request. If the contents of that line are marked dirty, the line is written back to memory before being replaced.

5.4.2   Resetting the Data Cache

There are two sources that can reset (initialize) the D-Cache:

- Hardware reset – a signal from the Reset block
- Software reset – caused by software setting a bit in the D-Cache Control Register (DCCR)

Both types of reset cause the following:

- Invalidation of the entire D-Cache: Valid bits in all entries of the Tag Array are cleared to 0.
- Initialization of all state machines
- Clearing of the MCB error flag

5.4.3   Data Cache Control Registers

Data Cache Control Registers (DCCR) are used to provide direct access, read and write, to Data and Tag arrays in the D-Cache. They also allow some specific Cache operations, such as Invalidation and Flushing.

Registers and their relative addresses inside the OCP + DCCR address space are listed in Table 5-3. All registers are 32 bit. Not implemented bits (in DCSTAT and DCCTRL) return 0's when read.

For more detail, see Section 7.5.7.

Table 5-3 Data Cache Control Registers (DCCR)

Address	Register Name	Description	Read/ Write	32-Bit Reset Value
Ox00	DCADDR	D-Cache Address. Can be used as a direct address / index, or as a logical address.	R/W	xxxx xxxx
Ox04	DCRDD	D-Cache Read Data. Contains data from Data or Tag arrays, returned by executing a direct read command.	R/W	xxxx xxxx
Ox08	DCWRD	D-Cache Write Data. Contains data to be written to Data or Tag arrays by a direct write command.	R/W	xxxx xxxx
Ox0C	DCSTAT	D-Cache Status.	R/W	0000 0000
Ox10	DCCTRL	D-Cache Control.	R/W	0000 0000

5.4.3.1 DCCTRL Register

The operation of the D-Cache is controlled by DCCTRL register settings. The structure of the DCCTRL register is shown below.

DCCTRL

0		OP	V	R	0	D
31	8	7 4	3	2	1	0

The functions of the bits in DCCTRL are as follows:

DCCTRL[31:8]

Reserved. Returns 0 when read.

DCCTRL[7:4]

OP: DCCR operation code. See Table 5-4.

These bits are cleared to 0 by core reset, or by DCCTRL[2] = 1. Software can write any value to DCCTRL[7:4], provided V is set to 0.

DCCTRL[3]

V: DCCR operation is Valid.

It is cleared to 0 by core reset, or by DCCTRL[2] = 1, or upon completion of an operation specified in DCCTRL[7:4] (OP), provided it was set to 1. Software can write to DCCTRL[3] (V), provided V is set to 0.

DCCTRL[2]

R: D-Cache Reset.

If active (set to 1) it resets the D-Cache, as well as the DCSTAT register and all bits in the DCCTRL register, except itself. This bit is set to 0 by core reset. It can be set to any value by software, regardless of the state of the V-bit.

DCCTRL[1]

Reserved. Returns 0 when read.

DCCTRL[0]

D: DCCR operation Done.

This bit cannot be set by software; it is a read only bit. It is cleared to 0 by core reset, or by DCCTRL[2] = 1, or on a software write of 1 to DCCTRL[3] (V), provided V was set to 0 prior to that. Bit D is set to 1 upon completion of an operation specified in DCCTRL[7:4] (OP), provided that V is set to 1.

5.4.3.2 DCSTAT Register

The DCSTAT register stores states associated with D-Cache functionality, so they can be analyzed by software. The structure of the DCSTAT register is shown below:

DCSTAT

0		WIDLE	MCBE
31	2	1	0

DCSTAT[31:2]

Reserved. Returns 0 when read.

DCSTAT[1]

WIDLE: D-Cache write queue is idle.

This bit is cleared to 0 by core reset, or by DCCTRL[2] = 1. It can be written by software (any value). When there is no software write to DCSTAT, then it reflects the state of fullness (or emptiness) of the D-Cache write queue, which resides in the CPU2DC block of D-Cache.

DCSTAT[0]

MCBE: MCB bus Error during a transaction involving D-Cache.

This bit is cleared to 0 by core reset, or by DCCTRL[2] = 1.

It can be written by software (any value). When there is no software write to DCSTAT, then it reflects the state of MCB error detection logic in the DC2MEM block of D-Cache.

5.4.3.3   DCCTRL Operation Codes

Table 5-4 describes the operation codes that are defined
for DCCTRL[7:4].

Table 5-4 DCCTRL Operation Codes

Operation	Opcode	Description
Direct Read DATA	0001	Read word of DATA Array from address specified in DCADDR[10:2] and bank (way) - in DCADDR[12:11]. Put it into DCRDD.
Direct Read TAG	0010	Read content of TAG Array from address specified in DCADDR[10:5] and bank (way) - in DCADDR[12:11]. Put it into DCRRD. See Note1.
Direct Write DATA	0011	Write word from DCWRD to DATA Array to address specified in DCADDR[10:2] and bank (way) - in DCADDR[12:11].
Direct Write TAG	0100	Write data from DCWRD to TAG Array to address specified in DCADDR[10:5] and bank (way) - in DCADDR[12:11]. See Note2.
Invalidate by Index	0101	Clear V-bit in entry of TAG Array, directly addressed using DCADDR[10:5] and DCADDR[12:11].
Invalidate by Address	0110	Clear V-bit in entry of TAG Array, directly addressed by DCADDR[10:5] and associatively by DCADDR[31:11] (to choose the way (bank)).
Flush by Index	0111	If D-bit in entry of TAG Array, directly addressed using DCADDR[10:5] and DCADDR[12:11] is set to 1, write corresponding line back to DRAM if V-bit is set to 1. Clear D-bit.
Flush and Invalidate by Index	1000	If D-bit in entry of TAG Array, directly addressed using DCADDR[10:5] and DCADDR[12:11] is set to 1, write corresponding line back to DRAM if V-bit is set to 1. Clear D-bit, clear V-bit.
Flush by Address	1001	If D-bit in entry of TAG Array, directly addressed by DCADDR[10:5] and associatively by DCADDR[31:11] (to choose the way) is set to 1, write corresponding line back to DRAM if V-bit is set to 1. Clear D-bit.
Flush and Invalidate by Address	1010	If D-bit in entry of TAG Array, directly addressed by DCADDR[10:5] and associatively by DCADDR[31:11] (to choose the way) is set to 1, write corresponding line back to DRAM if V-bit is set to 1. Clear D-bit, clear V-bit.

Note 1: The content of the DCRDD register after
executing command (operation) Direct Read Tag is
shown below:

DCRDD

TAGADDR		INDEX		0	DIR_WAY		TAGD	TAGV
31	11	10	5	4	3	2	1	0

TAGADDR, TAGD, and TAGV reflect the content of the
addressed entry in TAG Array. INDEX = DCADDR[10:5]
and DIR_WAY = DCADDR[12:11] are stored back into
DCRDD from D-Cache just for information / confirmation.

Note 2: For command (operation) Direct Write Tag the
content of the DCWRD register is used as follows:

- Bits DCWRD[31:11] are written to the Address portion
  of the addressed entry in the TAG Array.

- Bit DCWRD[1] is written to bit D, and bit DCWRD[0] is
  written to bit V of the addressed entry in the TAG
  Array.

Any code can be written from the CPU to DCADDR,
DCRDD, and DCWRD if DCCTRL[3] (V) is set to 0. If
DCCTRL[3] = 1, only the content of the DATA Array or the
TAG Array is allowed to be stored in DCRDD upon
completion of a Direct Read operation, but any software
update of DCRDD, as well as DCADDR and DCWRD is
blocked.

Software can read any of the DCCR registers at any time.

5.4.4   Notes to the D-Cache Programmer

- Since a D-Cache operation is initiated by setting DCCTRL[3] to 1 (and thus blocking subsequent update of any register until DCCTRL[3] = 0), software should first write an address to DCADDR and data (if needed) to DCWRD, and only then write to DCCTRL. While DCCTRL[3] = 1, the DCCR block initiates the request to D-Cache. If the operation was a read, read data is put into the DCRDD register. When the D-Cache finishes the operation, it indicates it with the Done bit, which is used to set DCCTRL[0] to 1, and clear DCCTRL[3] to 0. Then the request is de-asserted and DCCR considers the operation complete.
- It is the responsibility of software to provide a sole allocation of DCCR to a thread until an operation it has initiated is complete.

5.4.5   Tracking D-Cache Performance

A variety of statistics related to D-Cache performance can be monitored using the programmable statistics counters provided in the IP51xx. For details, see Section 5.6.

5.5   Instruction Cache

The Instruction Cache in the IP51xx has the size of 16 Kbytes. It is 4-way associative and has a 32-byte cache line, which means that there are four 4-Kbytes banks, each containing 128 sets of 32 bytes.

In the following discussions, CPU refers to the processor, and IP5K refers to the IP51xx.

5.5.1   Instruction Cache Policies

CPU instruction fetches that reference off-chip DDR memory use the I-Cache. If an instruction is found in the cache (hits the cache), the instruction is returned immediately to the CPU.

When a read request does not find the instruction in the cache, the thread that made the request is blocked while 32 bytes of instruction is fetched from memory. A blocked thread does not schedule any instructions. While the cache is fetching an instruction from off-chip memory, it can still satisfy read requests from other threads. If there are additional cache misses, the cache will queue one miss per thread.

When there is a cache miss, the cache selects a 32-byte cache line to replace using a pseudo-random policy among the 4 lines associated with the set defined by the address of the request.

5.5.2   ICCR Requests and Invalidation

Instruction Cache Control Registers (ICCR) can cause six different types of operations in I-Cache. Those include directly addressed read and write from/to the Data Array and the Tag Array, as well as invalidating a cache line using its index or address.

An ICCR operation is encoded in a 4-bit opcode that accompanies the ICCR request and address, as well as write data (which is used in the case of direct writes).

An ICCR request is considered for arbitration only if the I-Cache is not in a miss state, and in that case it has the highest priority. While processing ICCR requests, the I-Cache is considered busy for all other requests.

If there is a direct read, the I-Cache reads the word of the Data Array or the entry of the Tag Array and returns it to the ICCR together with read valid. If there is a direct write type operation, I-Cache writes the data coming from ICCR to the Data Array or the Tag Array. In both cases the address from ICCR is used to directly address these arrays (this is also called indexing).

Invalidation of a cache line is accomplished by clearing the valid bit in the entry of the Tag Array. This entry can by either directly addressed (invalidate by index operation), or associatively addressed (invalidate by address operation).

ICCR request remains active until the requested operation is finished. The I-Cache indicates the end of an ICCR operation by asserting the ICCR done flag, which causes de-assertion of the ICCR request.

5.5.3   Resetting the Instruction Cache

There are two sources that can reset (initialize) the I-Cache:

- Hardware reset – a signal from the Reset block
- Software reset – caused by software setting a bit in the I-Cache Control Register (ICCR)

Both types of reset cause the following:

- Invalidation of the entire I-Cache: Valid bits in all entries of the Tag Array are cleared to 0.
- Initialization of all state machines
- Clearing of the MCB error flag

5.5.4 Instruction Cache Control Registers

Instruction Cache Control Registers (ICCR) are used to provide direct access, read and write, to Data and Tag arrays in the I-Cache. They also allow some specific Cache operations, such as Invalidation.

Registers and their relative addresses inside the OCP + ICCR address space are listed in Table 5-5. All registers are 32 bit. Not implemented bits (in ICSTAT and ICCTRL) return 0's when read.

For more detail, see Section 7.5.6.

Table 5-5 Instruction Cache Control Registers (ICCR)

Address	Register Name	Description	Read/ Write	32-Bit Reset Value
Ox00	ICADDR	I-Cache Address. Can be used as a direct address / index, or as a logical address.	R/W	xxxx xxxx
Ox04	ICRDD	I-Cache Read Data. Contains data from Data or Tag arrays, returned by executing a direct read command.	R/W	xxxx xxxx
Ox08	ICWRD	I-Cache Write Data. Contains data to be written to Data or Tag arrays by a direct write command.	R/W	xxxx xxxx
Ox0C	ICSTAT	I-Cache Status.	R/W	0000 0000
Ox10	ICCTRL	I-Cache Control.	R/W	0000 0000

5.5.4.1 ICCTRL Register

The operation of the I-Cache is controlled by ICCTRL register settings. The structure of the ICCTRL register is shown below.

ICCTRL

0		OP		V	R	0	D
31	8	7	4	3	2	1	0

The functions of the bits in ICCTRL are as follows:

ICCTRL[31:8]

Reserved. Returns 0 when read.

ICCTRL[7:4]

OP: ICCR operation code. See Table 5-6.

These bits are cleared to 0 by core reset, or by ICCTRL[2] = 1. Software can write any value to ICCTRL[7:4], provided V is set to 0.

ICCTRL[3]

V: ICCR operation is Valid.

It is cleared to 0 by core reset, or by ICCTRL[2] = 1, or upon completion of an operation specified in ICCTRL[7:4] (OP), provided it was set to 1. Software can write to ICCTRL[3] (V), provided V is set to 0.

ICCTRL[2]

R: I-Cache Reset.

If active (set to 1) it resets the I-Cache, as well as the ICSTAT register and all bits in the ICCTRL register, except itself. This bit is set to 0 by core reset. It can be set to any value by software, regardless of the state of the V-bit.

ICCTRL[1]

Reserved. Returns 0 when read.

ICCTRL[0]

D: ICCR operation Done.

This bit cannot be set by software; it is a read only bit. It is cleared to 0 by core reset, or by ICCTRL[2] = 1, or on a software write of 1 to ICCTRL[3] (V), provided V was set to 0 prior to that. Bit D is set to 1 upon completion of an operation specified in ICCTRL[7:4] (OP), provided that V is set to 1.

5.5.4.2   ICSTAT Register

The ICSTAT register stores states associated with I-Cache functionality, so they can be analyzed by software. The structure of the ICSTAT register is shown below:

DCSTAT

0		MCBE
31	1	0

DCSTAT[31:1]

Reserved. Returns 0 when read.

DCSTAT[0]

MCBE: MCB bus Error during a transaction involving I-Cache.

This bit is cleared to 0 by core reset, or by ICCTRL[2] = 1.

It can be written by software (any value). When there is no software write to ICSTAT, then it reflects the state of MCB error detection logic in the IC2MEM block of I-Cache.

5.5.4.3   ICCTRL Operation Codes

Table 5-6 describes the operation codes that are defined for ICCTRL[7:4].

Table 5-6 ICCTRL Operation Codes

Operation	Opcode	Description
Direct Read DATA	0001	Read word of DATA Array from address specified in ICADDR[11:2] and bank (way) - in ICADDR[13:12]. Put it into ICRDD.
Direct Read TAG	0010	Read content of TAG Array from address specified in ICADDR[11:5] and bank (way) - in ICADDR[13:12]. Put it into ICRRD. See Note1.
Direct Write DATA	0011	Write word from ICWRD to DATA Array to address specified in ICADDR[11:2] and bank (way) - in DCADDR[13:12].
Direct Write TAG	0100	Write data from ICWRD to TAG Array to address specified in ICADDR[11:5] and bank (way) - in ICADDR[13:12]. See Note2.
Invalidate by Index	0101	Clear V-bit in entry of TAG Array, directly addressed using ICADDR[11:5] and ICADDR[13:12].
Invalidate by Address	0110	Clear V-bit in entry of TAG Array, directly addressed by ICADDR[11:5] and associatively by ICADDR[31:12] (to choose the way (bank)).

Note 1: The content of the ICRDD register after executing command (operation) Direct Read Tag is shown below:

ICRDD

TAGADDR		INDEX		0	DIR_WAY		0	TAGV
31	12	11	5	4	3	2	1	0

TAGADDR and TAGV reflect the content of the addressed entry in the TAG Array. INDEX = ICADDR[11:5] and DIR_WAY = ICADDR[13:12] are stored back into ICRDD from I-Cache just for information / confirmation.

Note 2: For command (operation) Direct Write Tag the content of the ICWRD register is used as follows:

• Bits ICWRD[31:12] are written to the Address portion of the addressed entry in the TAG Array.

• Bit ICWRD[0] is written to bit V of the addressed entry in the TAG Array.

Any code can be written from the CPU to ICADDR, ICRDD, and ICWRD if ICCTRL[3] (V) is set to 0. If ICCTRL[3] = 1, only the content of the DATA Array or the TAG Array is allowed to be stored in ICRDD upon completion of a Direct Read operation, but any software update of ICRDD, as well as ICADDR and ICWRD is blocked.

Software can read any of the ICCR registers at any time.

5.5.5 Notes to the I-Cache Programmer

- Since an I-Cache operation is initiated by setting ICCTRL[3] to 1 (and thus blocking subsequent update of any register until ICCTRL[3] = 0), software should first write an address to ICADDR and data (if needed) to ICWRD, and only then write to ICCTRL. While ICCTRL[3] = 1, the ICCR block initiates the request to I-Cache. If the operation was a read, read data is put into the ICRDD register. When the I-Cache finishes the operation, it indicates it with the Done bit, which is used to set ICCTRL[0] to 1, and clear ICCTRL[3] to 0. Then the request is de-asserted and ICCR considers the operation complete.

It is the responsibility of software to provide a sole allocation of ICCR to a thread until an operation it has initiated is complete.

5.5.6 Tracking I-Cache Performance

A variety of statistics related to I-Cache performance can be monitored using the programmable statistics counters provided in the IP51xx. For details, see Section 5.6.

5.6 Statistics Counters

There are four 32-bit statistics counters that can be used to monitor a variety of performance-related events within the IP51xx. Each of the four counters can be individually configured to monitor a specific kind of event. Table 7-13 shows the four counters and their corresponding configuration registers. Table 7-14 list the events that can be selected for tracking by any of these four counters. Once a counter is set up to monitor a specific event, the counter is incremented once for every clock during which the chosen event occurs. The counters are 32-bit wide.

5.6.1 Notes to the Statistics Counters Programmer

- Since counters cannot be reset by software (they are read only), the starting value needs to be stored prior to the measurement session.
- At 350-MHz CPU clock frequency, a 32-bit register wraps in about 12 seconds. It is recommended that the sampling interval of the counters be less then this number – for example, 4 seconds.
- If there is a write to a configuration register followed by a read from the corresponding counter, there should be at least 11 cycles separating the two instructions. This gap is necessary for the new configuration to take effect.

5.7 Flash Controller Programming Model

When reading this section, please refer to Section 6.3 for a general discussion of the flash controller and its external interface signals. Refer to Section 7.7.1 for the register definitions for the flash controller (FC) on Port A. Note that the flash controller does not use non-blocking registers for configuration of the flash device. Instead, it uses SPI transactions to configure the flash device directly through the SPI interface on Port A.

The port registers provide user control for the flash controller itself, and a secondary transaction interface for the processor to communicate with the external serial flash device directly. See Table 5-7 to get an overview of how the port registers map to the control and transaction interfaces for the flash controller (for a detailed mapping, see Section 7.7.1).

Table 5-7 Port Register Overview

Port Register	Mapping to FC Control / Status
Interrupt Status	FC transaction done
Interrupt Set	FC start transaction
Function Control 0	Configuration for FC
Function Control 1 Function Control 2	Transaction descriptor for a transaction to external flash device
Function Status 0	Processor / Cache transaction activity status
Function Status 1	Read data from external flash (for processor transactions)

5.7.1 FC Initialization

The FC requires no initialization at boot time to read cache lines from a supported external FLASH device. This is a requirement, as the external FLASH device is the boot device for the chip.

Specifically, the FC_CLK_WIDTH parameter defaults to 40 core clock cycles. This results in the SPI clock being driven at a low enough frequency to execute the basic read command for all supported devices. The CACHE_RD_CMD parameter defaults to 0x03, which is the common basic read command for all supported FLASH devices.

5.7.2 Cache Reads

This is a read-only interface. Writes to the external flash device must be done by the processor through the port register transaction interface. Cache misses to the flash address space result in the flash controller automatically fetching the corresponding cache line from the external flash and returning it to the requesting cache.

The flash controller fetches an entire 32-byte cache line before forwarding it to the appropriate Cache.

By default, the SPI Read command used is the basic Read supported by all devices. For devices that support an expedited read command and format, the processor must program the corresponding read command CACHE_RD_CMD and the clock divider value FC_CLK_WIDTH (to be used to generate the SPI clock from the core clock) into Function Control 0.

It is up to software to program these fields to enable high speed reads from the cache interface.

The processor code to set this up might look something like this:

1. Software sets FC_CACHE_LOCKOUT = 1 in Function Control 0.
2. FC clears CACHE_ACTIVE in Function Status 0 at the end of a pending cache transfer.
3. Software re-programs the parameters:
   a. CACHE_RD_CMD = FAST READ
   b. CACHE_DMY_CT = 1
   c. FC_CLK_WIDTH = SPI clock divider value
4. Software clears FC_CACHE_LOCKOUT.

5.7.3   Processor Read/Write/Erase Interface to External Flash

The processor uses the port register interface to Read/Write/Erase the external flash device. Reads through this interface are limited to 4 bytes.

The external flash SPI bus takes several flavors of command formats, which are transmitted serially on the SPI bus. These are shown in Table 5-8.

In order to support these variants, the flash controller must be aware of the command format. It is up to software to provide this information, along with the address and command to be transmitted to the flash device by the controller.

Table 5-8 Command Formats for External Flash

SPI Command Format	FCX_INST	Encoding
(command)	FC_CMD	2'b00
(command)(address)(write data) (command)(write data)	FC_WRITE	2'b01
(command)(read data) (command)(address)(read data) (command)(address)(dummy byte)(read data)	FC_READ	2'b10

The processor writes the SPI bus command, address, data, and some control information to the flash controller data registers. Function Control 1 and 2 define the type and structure of the transaction to be transmitted across the SPI bus. These two registers are essentially a transaction descriptor for communication with an external flash device over the SPI interface. Any data that is read back from the external flash device is available in Function Status 1. It is up to software to know how many bytes are valid, based on the FCX_DATA_CT field specified in Function Control 1. Any data that is to be written to the external flash device is written to the TX FIFO.

When transactions are completed, the flash controller sets an interrupt bit in the Port A Flash Interrupt Status register.

5.7.4   FC_CMD Process

FC_CMD issues a command only to the SPI interface. (e.g., flash device WREN command).

The processor sends a command only instruction to the flash device as follows:

1. Software must ensure that the flash controller function is selected for this port (default is flash controller selected).
2. Software must ensure that the flash controller is enabled by setting the FC_EN bit in Function Control 0 (default behavior).
3. Software writes the Function Control registers as follows:
   a. Function Control 1: FCX_INST = FC_CMD
   b. Function Control 1: FCX_DATA_CT = 0
   c. Function Control 1: FCX_DMY_CT= 0
   d. Function Control 1: FCX_ADDR_EXISTS = 0
   e. Function Control 2: FCX_CMD = <command>
   f. Function Control 2: FCX_ADDR = 0
4. Software initiates the transaction by setting FC_START in the Interrupt Set register.
5. The FC processes the transaction and puts the command on the SPI bus.
6. FC sets FC_DONE in the Interrupt Status register to indicate transaction completion.

5.7.5   FC_READ Process

FC_READ issues a command and expects read data on the SPI data input pin (SI).

The FC_READ instruction can read 1 to 4 bytes of data. The processor specifies the number of bytes in the FCX_DATA_CT field in Function Control 1.

The processor reads data from the flash device as follows:

1. Software must ensure that the flash controller function is selected for this port (default is flash controller selected).
2. Software must ensure that the flash controller is enabled by setting the FC_EN bit in Function Control 0 (default behavior).
3. Software must set up the Function Control registers as follows:
   a. Function Control 1: FCX_INST = FC_READ
   b. Function Control 1: FCX_DATA_CT
       = <byte count>
   c. Function Control 1: FCX_DMY_CT
       = <dummy byte count>
   d. Function Control 1: FCX_ADDR_EXISTS = 1, if an address field exists.
   e. Function Control 2: FCX_CMD
       = <read command>
   f. Function Control 2: FCX_ADDR = <addr>
4. Software initiates the transaction by setting FC_START in the Interrupt Set register.
5. FC clears the read data (FCX_RDATA in Function Status 1) from the previous read transaction.
6. FC processes the transaction and puts the command, address, and dummy bytes onto the SPI bus.
7. FC shifts read data into FCX_RDATA in Function Status 1 (until the transaction data byte count is reached) with the last data bit in bit 0 (right justified).
8. FC sets FC_DONE in the Interrupt Status register to indicate transaction completion.
9. Software reads data from FCX_RDATA in Function Status 1.

5.7.6   FC_WRITE Process

FC_WRITE issues a command and puts write data on the SPI data output pin (SO).

The FC_WRITE instruction can write 1 to 512 bytes of data to the flash device (although current devices only support up to 256). The external flash device must support continuous writes of more than 1 byte. It is up to software to ensure that this is true.

Note: FC writes greater than 4 bytes must be done on word boundaries (4 bytes at a time). FC writes of fewer than four bytes are all right: 1, 2, 3, and 4 bytes are all valid data count sizes.

The processor writes data to the flash device as follows:

1. Software must ensure that the flash controller function is selected for this port (default is flash controller selected).
2. Software must ensure that the flash controller is enabled by setting the FC_EN bit in Function Control 0 (default behavior).
3. Software must set the FC_CACHE_LOCKOUT bit in Function Control 0 before any write operation to the external flash device.
4. FC will clear CACHE_ACTIVE (in Function Status 0) at the end of a pending cache transfer.
5. Software must set up the Function Control registers as follows:
   a. Function Control 1: FCX_INST = FC_WRITE
   b. Function Control 1: FCX_DATA_CT
       = <byte count>
   c. Function Control 1: FCX_DMY_CT =0
   d. Function Control 1: FCX_ADDR_EXISTS = 1, if an address field exists.
   e. Function Control 2: FCX_CMD
       = <write command>
   f. Function Control 2: FCX_ADDR = <addr>
6. Software pushes write data, 4 bytes at a time, into the TX FIFO. It is up to software to keep track of the FIFO underflow.
7. Software initiates the transaction by setting FC_START in the Interrupt Set register.
8. FC processes the transaction and puts the command, address, and write data onto the SPI bus. The data byte count dictates how many bytes of write data are written to the flash device. It is up to software to make sure that the TX FIFO does not underflow for the specified number of bytes in the FCX_DATA_CT field.
9. FC sets FC_DONE in the Interrupt Status register to indicate transaction completion.
   Note: At this point, the data has been written to the flash device over the SPI bus. The data has not been written into the flash array yet. FC_DONE means only that the flash controller transaction is complete. It does not mean that the data is successfully written into the flash device. It is up to software to poll the flash device status register periodically to tell when the write data has been successfully written into the device.
10. Software must clear FC_CACHE_LOCKOUT when the device status register shows that the flash array has been successfully written.

Transaction-Level View of Software Writes to Flash

1. Software must ensure that the Flash controller function is selected for this port.
2. Software must enable the low-level interface by ensuring that FC_EN is set.
3. Software must set the FC_CACHE_LOCKOUT bit to ensure that the port-register interface owns the device.
4. Software issues FC_CMD transaction – WREN
5. Software issues FC_WRITE transaction – writes data to device.
6. Software periodically issues FC_READ transaction – RDSR
7. Software checks read data register to see if write has been successful.
8. If (successful write), done!, else repeat from #6.
9. Software must clear the FC_CACHE_LOCKOUT bit to relinquish the interface.

5.7.7   SPI Interface Clocking

The clock divider generates the External SPI interface clock signal (SCK) for the external flash device (core clock edges are counted and used to generate the SPI clock).

SCK is guaranteed not to change frequency in the middle of an SPI transaction. The default frequency is set to 40 core clock cycles, but is programmed some time after reset by writing a value to FC_CLK_WIDTH in Function Control 0. This value is the total number of core clock cycles per SPI clock cycle. The programmed number must be even. SCK is guaranteed to have a 50% duty cycle.

5.7.8   System / Function Reset

After system reset (rst_core) the FC will come up with the following default parameters in Function Control 0:

- CACHE_RD_CMD = 0x03 (Common read command for all supported flashes)
- CACHE_DMY_CT = 0 (0x0)
- FC_CLK_WIDTH = 40 core clock cycles (0x28)
- FE_CE_WAIT = 40 core clock cycles (0x28)

After reset is deasserted, the first I-Cache miss triggers an FC request to get the first cache line from the flash. It will read the data and respond to the cache request. The initial SPI frequency is guaranteed to be less than 10 MHz (the slowest read frequency for all supported devices).

At some point later, the PLL will be started, and the core clock frequency will go up to speed, resulting in the SPI frequency being increased. It must be guaranteed that the SPI clock frequency does not exceed the minimum common read frequency for supported flash devices (10

MHz) during this process, until the FC is reprogrammed to use the high-speed read command and format particular to the flash device being used. The software driver is responsible for programming these parameters into the Function Control registers.

5.7.9   FC Function De-Selection

De-asserting FC_EN in Function Control 0 will simply mean that upon completion of any current transaction on the SPI bus, neither the cache nor the processor will be able to get access to the SPI interface. Arbitration is inhibited. However, the flash controller will continue to queue up read requests from the caches, to be processed when the FC is re-enabled.

The I/O Port function select is tied to the FC_EN bit. If the function is de-selected, the enable bit driven to the FC module will go to its inactive state.

It is a requirement that if the FC is de-selected, the SPI chip select signal (CE_N) should remain driven to a state that ensures that the external flash device remains de-selected. CE_N must be driven constantly HIGH (de-selecting the flash). This pin can NOT be shared with any other device connected to the port.

The procedure for de-selecting the FC is as follows:

1. Clear the FC_EN bit in Function Control 0 (prohibiting arbitration).

2. Poll the Port A Flash Interrupt Status register to check that any pending transactions are done on the SPI interface.

3. Set the port-register GPIO enable and output register to drive the CE_N pin HIGH on the SPI interface.

4. De-select the function by changing the Function Select register to GPIO.

5.8   DDR SDRAM Programming Model

When reading this section, please refer to Section 6.4 for a general discussion of the DDR SDRAM controller and its external interface signals. Refer to Section 7.12.1 for the register definitions for the DDR SDRAM controller on Port G.

In the following sections, DDR refers to DDR SDRAM.

This section details high level programming information required to enable the DDR block. The DDR block itself is composed of a number of subsystems, specifically:

- **DDR Clock Subsystem**
  This subsystem consists of a pair of PLLs that are responsible for the generation of a DDR clock for both

the DDR associated hardware and its external interface. A dedicated PLL generates the DDR clock. This clock is used to clock the DDR controller as well as the external DDR interface. Additionally, a Deskew PLL deskews the generated clock on the external DDR bus in relation to the internal DDR clock. Both PLLs are configured via registers in the general configuration section of the On-chip Peripheral (OCP) register space (indirect register space).

- **DDR I/O Calibrator Module**
  The DDR I/Os contain circuitry that control output impedance and on-die termination. The circuitry is enabled only when the I/Os are configured in the DDR2 mode of operation. The Calibrator subsystem is a module that dynamically calibrates the termination setting of these I/Os in order to achieve an optimal impedance characteristic on the DDR bus. The configuration registers for the DDR I/O Calibrator are located in the general configuration section of the On-chip Peripheral (OCP) register space (indirect register space).

- **DDR Port I/Os**
  The DDR I/Os are situated across Port G and Port H. These ports must be configured, in order to get the DDR hardware to utilize these I/Os. Otherwise all DDR transactions will be dropped. The corresponding registers to program the ports are located in the Port G and Port H I/O register space.

- **DDR Controller Module**
  The DDR controller subsystem manages all DDR operations. The configuration registers are located in the Port G I/O register space.

The sequence for enabling the DDR subsystem is as follows:

1. Enable the DDR clock subsystem:
   (a) Configure and enable the DDR clock PLL in order to generate a clock with the desired frequency.
   (b) Configure and enable the DDR deskew PLL.
2. Set up the DDR I/O Calibrator to execute a calibration sequence.
   This is done only when the interface is configured for DDR2 mode.
3. Configure Port G and/or Port H for DDR operation.
4. Configure the DDR controller for a specific memory device and activate it.

Once these programming sequences are accomplished, the DDR subsystem is active and the external DDR memory is accessible.

5.8.1 Enabling the DDR clock

The DDR clock must be enabled and stable before the rest of the DDR subsystem is enabled. This consists of configuring and setting up the DDR clock PLL and the DDR deskew PLL.

Both PLLs are configured via registers in the general configuration section of the On-chip Peripheral (OCP) register space (indirect register space). Refer to Section 5.16 section for a recommended method of enabling the DDR clock.

The maximum value of the DDR clock for IP51xx is 200 MHz. The minimum value of the DDR clock for IP51xx is 120 MHz.

5.8.2 Disabling the DDR clock

When the DDR is not being used, the DDR clock can be disabled in order to lower power consumption. This can be accomplished by putting both the DDR PLL and DDR Deskew PLL into power down mode. Once this is done, all data in the DDR memory device will be lost. Re-enabling the DDR subsystem requires the entire programming sequence to be re-initiated.

5.8.3 Calibrating the DDR I/Os

A dynamic calibration scheme is used to calibrate the output impedance as well as the on-die input termination when the DDR I/O is configured for DDR2 mode. Calibration is not necessary, and has no effect, when Port G and Port H are configured for DDR1 operation. The IP51xx has two Calibration controllers. CAL Low is connected to Port G, and CAL High is connected to Port H. CAL Low is the master calibration controller and is connected to an external calibration resistor. The resistor value for DDR2 is specified at 240 Ohms.

The configuration registers for the DDR I/O Calibrator are located in the general configuration section of the On-chip Peripheral (OCP) register space (indirect register space).

The calibration sequence will be executed on the CAL Low controller. The calibration value generated by the calibration sequence will be written to the CAL Low and CAL High when the calibration sequence is complete. The DDR clock is used as the source clock for the calibrator, and must operate at 300 MHz or less to avoid errors due to the speed of the circuit.

The proposed calibration sequence is as follows:

1. Reset and initialize CAL Low and CAL High. After power-on reset the calibration circuit should already be initialized, but it is recommended to repeat the sequence. To reset and initialize the controller,

write a 1 to the CAL_RESET_LOW bit and the CAL_RESET_HIGH bit of the CALCTRL register for the duration of 4 clk_ddr clock cycles. This will guarantee that the CAL_RESET signal in the clk_ddr domain is high for the specified 2 core clock cycles.

2. Read the CAL_DONE_LOW and CAL_FAULT_ LOW values of the CALSTATUS register and verify that they are 0. It will take a minimum of 3 clk_ddr clock cycles and 2 clk_core cycles for the CAL_DONE_LOW and CAL_FAULT_LOW bits to be cleared.

3. Write a 1 to the CAL_START_LOW bit of the CALCTRL register. This will create a toggle and a one clock pulse of the CAL_START input of the calibration controller.

4. Poll the CAL_DONE_LOW bit for the calibration process to complete. The maximum calibration time is 5000 clk_ddr cycles. The minimum time is 1500 clk_ddr cycles.

5. Verify that the calibration circuit was able to match the external resister. After the CAL_DONE_LOW bit is set check the status of the CALFAULT_LOW bit. If the CAL_FAULT_LOW bit is set, the calibration controller failed to calibrate the I/O.

6. Set IMP_SET_UPDATE_LOW to 0 and IMP_SET_ UPDATE_HIGH to 1.

7. To update Port H, copy the value of IMP_PU_OUT_ LOW to IMP_SET_PU_HIGH and IMP_PD_OUT_ LOW to IMP_SET_PD_HIGH.

8. Write a 1 to the IMP_UPDATE_LOW and IMP_UPDATE_HIGH register field. This will update the I/Os with the calibration value generated by the state machine. Do not write IMP_UPDATE_HIGH if you do not want to calibrate Port H.

9. Do not change the value of IMP_SET_UPDATE_ LOW and IMP_SET_UPDATE_HIGH for a minimum of 5 clk_ddr cycles.

Once the pull-down cycle is complete the CAL_DONE signal will be asserted for 200 core clock cycles.

The following sequence can be used to write a calibration value to Port G:

1. Make sure that a calibration sequence has completed and that the CAL_RESET register bit is a 0.

2. Set the IMP_SET_UPDATE_LOW register bit to 1, and the values that you wish to program into the I/Os in the IMP_SET_PU_LOW and IMP_SET_PD_ LOW register fields.

3. Write a 1 to the IMP_UPDATE_LOW register field.

4. Wait a minimum of 5 clk_ddr cycles before changing IMP_SET_UPDATE_LOW, IMP_SET_PU_LOW, and

IMP_SET_PD_LOW. The Calibration will have been written to the Port G DDR2 I/Os.

For Port H follow the same sequence, but set the HIGH register fields in the CALCTRL register.

5.8.4   Configuring the I/O Port for DDR Operation

In order to be able to configure the DDR controller, Port G function select has to be set for DDR operation. This action also configures Port G I/O's to be used as the external DDR interface. Port G implements only the DDR address and command, as well as the lower byte of the DDR bus. Port H implements the upper byte of the DDR bus. Consequently, Port H function select also must be set for DDR operation if the IP51xx is interfacing to a memory device with a 16 bit bus.

5.8.5   Initializing the DDR Controller

Registers that control the DDR controller are mapped to the Port G I/O map when it's function select is set for DDR operation.

The non-blocking region of Port G consists of the registers that control some aspects of the DDR controller hardware, such as a watermark level for the read response FIFO that allows software to optimize for read response latency. The initial level of interrupt management registers are also available there. Additionally, the whole DDR controller hardware can be reset via the Port G function register.

The blocking region of Port G maps most of the configuration registers of the DDR controller. These configuration registers are used for setting up the controller in order to support the type (DDR1 or DDR2), address mapping, and timing of a DDR memory device to which the IP51xx is interfacing.

The following programming sequence is a recommended method for initializing the controller:

1. Configure Port G function select for DDR operation. Additionally, configure Port H function select for DDR operation as well, if interfacing to a DDR memory with a 16-bit bus. Wait at least 4 core clocks for these changes to actually take effect.

2. Assert DDR controller reset via Port G function reset.

3. De-assert DDR controller reset via Port G function reset. Wait at least 4 core clocks for the reset to actually take effect.

4. If desired, configure the DDR read response watermark via the Port G function control register.

5. Configure the DDR controller via the Port G blocking register space in order to support the width of the external DDR bus, the type (DDR1 or DDR2), timing, and address map of the memory device.

6. Signal the DDR controller to become active.

7. Wait for the DDR controller master DLL to lock. This can be done by either setting up the appropriate interrupt or polling on the interrupt status bit.

Once this is done, the DDR controller will initiate the proper bus initialization sequence required to get the external DDR memory into an operational mode. After this, the controller can now access the memory. It will also ensure that the memory remains refreshed accordingly.

5.8.6   Training the DDR DLLs (Delay Locked Loops)

The DDR controller implements a pair of DLLs that control the timing of the DDR data bus for write transactions. Specifically, one DLL controls the placement of the data signals (DQ) and the other controls the placement of the data strobes (DQS). In both cases, they are controlled in relation to the DDR clock. In addition to this, the controller also implements another pair of DLLs that manage read response data capture. This pair skews the incoming data strobes (DQS) in relation to the data signals (DQ) in order to correctly latch the read responses off the bus. One DLL is used on the upper byte and the other is used on the lower byte.

These DLLs need to be programmed correctly in order to facilitate functional transactions on the DDR bus. Ideally, a set of DLL settings should result in the optimal timing margins on the DDR interface for both read and write operations. In the case of DDR1, there is sufficient timing margin on the DDR interface, that one set of DLL settings is probably sufficient to support any DDR memory across all legal process, voltage, and temperature (PVT) settings. However, this is not the case for DDR2. The timing characteristics differ radically enough between all possible PVT conditions, that a single DLL setting will not function reliably in some cases. Process corner and voltage are considered to have the largest effect on the timing margins. Temperature has a minimal effect in relation to either process corner or voltage.

As mentioned before, it is possible to locate a set of DDR1 DLL settings that will work across all process corners, voltage, and temperature (PVT) settings. This is not possible for DDR2. It is possible, however, to find a set of DLL settings that will work fine across some PVT conditions and function marginally across others. The settings are probably sub-optimal for some conditions, and the interface is functional to the degree that it will fail after repeated usage over a sustained period of time. This

characteristic is important, because it allows software to implement a simple DLL training algorithm that will refine that initial set of DLL configurations to one that is tuned for the existing PVT condition.

The DLLs will have to be trained independently in pairs — one pair for read operations and the other pair for write operations. In either case, DDR memory accesses are utilized to discover the possible working range of the DLL pairs. Starting with a setting that at least works marginally, the parameters are increased and decreased until a DDR transaction can no longer function. The optimal setting for each DLL is located in the middle of the this working range. For read operations, the DLL pairs are completely independent of one another and can therefore be trained independently. Care must be taken however to mask out the non-relevant data bytes in the read response when training one of these DLLs. For example, when training the DLL associated with read responses for the lower byte, data that corresponds with the upper byte should be masked out. For write operations, the DLL pairs are dependent and cannot be trained independently. Although each DLL independently controls the placement of either the data signals (DQ) or the data strobe (DQS), the combination of these two placements affects the overall write interface timing. The proposed algorithm can be simplified by assuming a fixed phase relationship between the placement of the data signals (DQ) and the data strobes (DQS). Then the working range of this DLL pair can be determined for that fixed phase relationship. The IP51xx characterization data suggest that a phase relationship of 30% should be used.

In regard to the DDR accesses, all transactions can be addressed to the same DDR location. This is because the DLLs do not determine the timing of the address signals. Consider also that in order to qualify any particular set of DLL settings, minimally a pair of DDR transactions (such as a write followed by a read of the same address) is required. Obviously the algorithm would tune more effectively if the number of transaction pairs were increased (either by testing a larger segment of memory and/or testing a particular location multiple times). Unfortunately, this will happen at the expense of timeliness. Characterization data suggest that the transaction size can be limited to a single word and that one transactions pair is sufficient to qualify a set of DLL settings. Different data patterns present the DDR interface with a different level of electrical stress. The algorithm, which is described in more detail below, assumes that a pattern that is randomized is good enough.

The following example describes the write and read DLL training algorithm using pseudo code:

```
//
// Write DLL training algorithm
//

// Set the initial write_dqs_shift DLL parameter
// Controls the data signals (DQ) placement within the write data eye
int wdll_dq_val = x1 ; // initial parameter for outbound write DQ placement
write_ddr_reg ( WRITE_DQS_SHIFT, wdll_dq_val );

// Set the initial dqs_out_shift DLL parameter
// Controls the data strobes (DQS) placement within the write data eye
int wdll_dqs_val = x1+(0.30*128) ; // initial parameter for outbound write DQS placement
write_ddr_reg ( DQS_OUT_SHIFT, wdll_dqs_val );

// Initialize DDR memory
int ddr_data = rnd();
write_ddr_mem ( ADDR, ddr_data );
invalidate_flush_dcache ( ADDR );

// Locate the maximum range of the DLL
while ( read_ddr_mem (ADDR) == ddr_data ) {
       write_ddr_reg ( WRITE_DQS_SHIFT, wdll_dq_val++ );
       write_ddr_reg ( DQS_OUT_SHIFT, wdll_dqs_val++ );
       ddr_data = rnd();
       write_ddr_mem ( ADDR, ddr_data );
       invalidate_flush_dcache ( ADDR );
}
int max_wdll_dq_val = wdll_dq_val;
int max_wdll_dqs_val = wdll_dqs_val;

// Set the initial write_dqs_shift DLL parameter
// Controls the data signals (DQ) placement within the write data eye
wdll_dq_val = x1 ; // initial parameter for outbound write DQ placement
write_ddr_reg ( WRITE_DQS_SHIFT, wdll_dq_val );

// Set the initial dqs_out_shift DLL parameter
// Controls the data strobes (DQS) placement within the write data eye
wdll_dqs_val = x1+(0.30*128) ; // initial parameter for outbound write DQS placement
write_ddr_reg ( DQS_OUT_SHIFT, wdll_dqs_val );

// Initialize DDR memory
int ddr_data = rnd();
write_ddr_mem ( ADDR, ddr_data );
invalidate_flush_dcache ( ADDR );

// Locate the minimum range of the DLL
while ( read_ddr_mem (ADDR) == ddr_data ){
       write_ddr_reg ( WRITE_DQS_SHIFT, wdll_dq_val-- );
       write_ddr_reg ( DQS_OUT_SHIFT, wdll_dqs_val-- );
       ddr_data = rnd();
       write_ddr_mem ( ADDR, ddr_data );
       invalidate_flush_dcache ( ADDR );
}
int min_wdll_dq_val = wdll_dq_val;
int min_wdll_dqs_val = wdll_dqs_val;

int optimal_wdll_dq_val = ( max_wdll_dq_val + min_wdll_dq_val ) / 2;
int optimal_wdll_dqs_val = ( max_wdll_dqs_val + min_wdll_dqs_val ) / 2;
write_ddr_reg (WRITE_DQS_SHIFT, optimal_wdll_dq_val );
write_ddr_reg (DQS_OUT_SHIFT, optimal_wdll_dq_val );
```

```
//
// Read DLL training algorithm
//

DDR_LO_BYTE_MASK = 0x00ff00ff
DDR_HI_BYTE_MASK = 0xff00ff00

// Set the initial read_dqs0_delay DLL parameter
// Controls the data strobes (DQS) placement within the read data eye
int rdll_dqs0_val = x1 ; // initial parameter for inbound read DQS_0 (lower byte) placement
write_ddr_reg ( DQS_DELAY_0, rdll_dqs0_val );

// Initialize DDR memory
int ddr_data = rnd();
write_ddr_mem ( ADDR, ddr_data );
invalidate_flush_dcache ( ADDR );

// Locate the maximum range of the DLL_0 DLL
while ( (read_ddr_mem (ADDR) && DDR_LO_BYTE_MASK ) == ddr_data ) {
        write_ddr_reg ( DQS_DELAY_0, rdll_dqs0_val++ );
        ddr_data = rnd();
        write_ddr_mem ( ADDR, ddr_data );
        invalidate_flush_dcache ( ADDR );
}
int max_rdll_dqs0_val = rdll_dqs0_val;

// Set the initial read_dqs0_delay DLL parameter
// Controls the data strobes (DQS) placement within the read data eye
rdll_dqs0_val = x1 ; // initial parameter for inbound read DQS_0 (lower byte) placement
write_ddr_reg ( DQS_DELAY_0, rdll_dqs0_val );

// Initialize DDR memory
int ddr_data = rnd();
write_ddr_mem ( ADDR, ddr_data );
invalidate_flush_dcache ( ADDR );

// Locate the minimum range of the DLL_0 DLL
while ( (read_ddr_mem (ADDR) && DDR_LO_BYTE_MASK ) == ddr_data ) {
        write_ddr_reg (DQS_DELAY_0, rdll_dqs0_val-- );
        ddr_data = rnd();
        write_ddr_mem ( ADDR, ddr_data );
        invalidate_flush_dcache ( ADDR );
}
int min_rdll_dqs0_val = rdll_dqs0_val;

int optimal_rdll_dqs0_val = ( max_rdll_dqs0_val + min_rdll_dqs0_val ) / 2;
write_ddr_reg ( DQS_DELAY_0, optimal_rdll_dqs0_val );

// Repeat for DQS_DELAY 1; the DLL parameter for the upper inbound byte
int rdll_dqs1_val = y1 ; // initial parameter for inbound read DQS_1 (upper byte) placement
.
.
.
```

Strictly speaking, the algorithm can be applied anytime once the DDR block is operational. Software will be required to guarantee the available DDR memory resources for the training utility. Also the DDR memory should not be accessed while the interface is being trained. At minimum, the interface should be trained

whenever the DDR block becomes operational. The more difficult question is whether is should be applied after that. Can changes in the environment (voltage and temperature) within the specifications render a trained DDR interface useless? Ideally if software could detect such an event, it could retrain the interface. However there is no practical way for software to determine this, short of a catastrophic failure that triggers a watchdog reset.

Certainly software can periodically and blindly retrain the interface. But this is hardly ideal. One alternative is for software to track changes in the DDR controller's master DLL lock value and the I/O Calibration values. The controller's DLL lock value changes in relation to the core voltage, and the DDR I/O calibration values changes in relation to the DDR I/O voltage. The problem is that software needs to initiate a calibration cycle in order to get new I/O calibration values. In such a scenario software would need to periodically poll these values in order to

determine when to retrain the interface (based on some established criteria). This is obviously a very disruptive exercise, especially considering that the I/Os need to be recalibrated. There is also the option of just tracking the DLL lock value.

The DDR controller itself has some small ability to dynamically adapt to changes in the PVT conditions in order to maintain established timing margins. Currently the assumption is that the interface will only need to be trained once and that the DLL settings at that point are good enough to prevail across any variations in the environment. IP51xx characterization data suggests that this is indeed the case.

Table 5-9 provides the DLL programming parameters for the IP51xx based on bring-up and characterization data. In the case of DDR1, this represents the final values. In the case of DDR2, this represents the starting values which need to be tuned by training the interface.

Table 5-9 DDR DLL Programming Parameter Values

DDR Type	WR_DQS_SHIFT	DQS_OUT_SHIFT	DQS_DELAY_1	DQS_DELAY_0
DDR1	0x33	0x53	0x13	0x13
DDR2	0x30	0x56	0x15	0x15

5.8.7 Configuring the DDR Read Response Watermark

The DDR subsystem contains a read response FIFO with a programmable watermark setting. This read response path functions under the requirement that when the cache fetches a read response, it can do so by reading out the entire cacheline in one uninterrupted burst. The reset value of this watermark is a full cacheline. With this

setting, the read response is only transferred to the cache after a whole cacheline has been received. Lowering the watermark will reduce the latency of a cache miss. However, setting it too low will result in data corruption as the FIFO underflows.

The optimal setting can be calculated based on the rate of data transfer into and out of the FIFO. The calculation is as follows:

x16   DDR1 or DDR2   CORE_CLK_FREQ > DDR_CLK_FREQ
Watermark = ((CL/4) - ((CL/4) * (DDR_CLK_FREQ / CORE_CLK_FREQ)

x8   DDR1 or DDR2   CORE_CLK_FREQ > DDR_CLK_FREQ / 2
Watermark = ((CL/4) - ((CL/4) * (1/2) * (DDR_CLK_FREQ / CORE_CLK_FREQ)

x16   DDR1 or DDR2   CORE_CLK_FREQ $\leq$ DDR_CLK_FREQ
Watermark = 1

x8   DDR1 or DDR2   CORE CLK FREQ $\leq$ DDR_CLK_FREQ / 2
Watermark = 1

CL = Cacheline size in bytes

Fractional watermark values should be rounded up.

Additionally, whenever the watermark setting is configured to anything less than a cacheline, the DDR controller should also be configured for AUTO_REFRESH mode and AUTO_PRECHARGE mode.

AUTO_REFRESH mode prevents the controller from interrupting an ongoing read transaction to service an auto refresh request. AUTO_PRECHARGE mode closes the row after each access, and therefore prevents the

controller from interrupting an ongoing read transaction to close a row whenever Tras_max (the maximum period of time a row can be kept opened) expires.

Without these controller settings, a read transaction may be interrupted. If the FIFO contains at least as much data as the watermark setting, this can lead to the FIFO underflowing when the cache fetches the entire cacheline before it is readily available.

In all circumstances, a watermark setting of 0 is illegal and will result in read response data corruption.

5.9   MII Controller Programming Model

When reading this section, please refer to Section 6.6 for a general discussion of the MII controller and its external interface signals. Refer to Section 7.10.2 for the register definitions for the MII controller on Port E.

5.9.1   MII Controller Initialization

The following sample code shows how to initialize the MII Controller.

```
;=======================================================================
; initialization and start up
;   void ipEthernet_thread_start_@INST@(void* NULL)
    .global _ipEthernet_thread_start_@INST@
    .type _ipEthernet_thread_start_@INST@ @function
;=======================================================================

    .section .text.ipEthernet_thread_start_@INST@,"ax",@progbits


_ipEthernet_thread_start_@INST@:
    ; The following address register are reserved
    ;   A0 -> RAM variable base address
    ;   A1 -> SerDes/MII register base address
    ;   A2 -> RX buffer base address
    ;   A3 -> TX buffer base address
    ;   A7 -> Timer block register base address
    moveai A0, #%hi(eth_isr_var_base_@INST@)
    lea.4  A0, %lo(eth_isr_var_base_@INST@)(A0)
    movei  D0, #ETH_RX_PKT_INFO
    lea.1  A2, (A0,D0)
    movei  D0, #ETH_TX_PKT_INFO
    lea.1  A3, (A0,D0)
    moveai A7, #%hi(TIMER_BASE)

    ; The following data register are reserved
    ;   IFG and slot time are default to 10Base-T value
    ;   SFD differs from preamble pattern by just 1 bit
    movei  D_one, #1
    movei  Difg, #ETH_10BASE_T_IPG
    movei  Dslot, #ETH_10BASE_T_SLOT
    movei  Dsfd, #0x5555
    shmrg.2Dsfd, Dsfd, Dsfd
    movei  Dtmreg, #TIMER_SYSCOM(INT_BIT(HRT_INT_TIME_OUT))>>2
    bset   Dtmint, #0, #INT_BIT(HRT_INT_TIME_OUT)
    bset   Dtxint, #0, #INT_BIT(HRT_INT_TX_REQ)
    bset   Ddsrint, #0, #INT_BIT(HRT_INT_DSR)

    ; Initial suspend to wait for global interrupt enable
    bset   INT_SET(HRT_INT_TIME_OUT), #0, #INT_BIT(HRT_INT_TIME_OUT)
    bset   INT_MASK(HRT_INT_TIME_OUT), INT_MASK(HRT_INT_TIME_OUT), #INT_BIT(HRT_INT_TIME_OUT)
    suspend


    ; Initialize MII I/O HW block for Ethernet operation
    ;   Initialize A1 -> SerDes/MII register base address
    ;   Reset I/O block before doing anything
    ;   Set I/O mode to MII Ethernet (mode setting ?)
```

```
    ;   Set TX water mark = 8 and RX FIFO water mark = 9
    ;   Setup FIFO selection according to the MII FIFO choice
    ;   A6 -> Optional extended port base address
    moveai A1, #%hi(MII_REG_BASE)
    move.4 MII_INT_MASK(A1), #0
    movei  MII_FUNCTION_REG(A1), #(1<<5)|@INST@IPETHERNET_THREAD_NUM
    movei  MII_FUNC_MODE_SEL(A1), #MII_FUNC_RESET_MII|MII_FUNC_CODE
#if defined(MII_2nd_REG_BASE)
    moveai A6, #%hi(MII_2nd_REG_BASE)
    movei  MII_FUNCTION_REG(A6), #(1<<5)|@INST@IPETHERNET_THREAD_NUM
    movei  MII_FUNC_MODE_SEL(A6), #MII_FUNC_RESET_MII|MII_FUNC_CODE
#endif
    cycles 4
#if defined(@INST@USE_MII_RE_FOR_ETHERNET)
  #if defined(@INST@MII_RE_USE_RMII)
    movei  MII_TRX_CONTROL+0(A1),
#%hi(MII_TRX_CONFIG_VAL|(1<<MII_TRXCNTL_TXCRC_EN)|(1<<MII_TRXCNTL_RMII_MODE))
    movei  MII_TRX_CONTROL+2(A1),
#%lo(MII_TRX_CONFIG_VAL|(1<<MII_TRXCNTL_TXCRC_EN)|(1<<MII_TRXCNTL_RMII_MODE))
  #elif defined(@INST@MII_RE_USE_RB_FOR_TX)
    movei  MII_TRX_CONTROL+0(A1), #%hi(MII_TRX_CONFIG_VAL|(1<<MII_TRXCNTL_TXCRC_EN))
    movei  MII_TRX_CONTROL+2(A1), #%lo(MII_TRX_CONFIG_VAL|(1<<MII_TRXCNTL_TXCRC_EN))
  #elif defined(@INST@MII_RE_USE_RI_FOR_TX)
    movei  MII_TRX_CONTROL+0(A1),
#%hi(MII_TRX_CONFIG_VAL|(1<<MII_TRXCNTL_TXCRC_EN)|(1<<MII_TRXCNTL_MII_PORT_RI))
    movei  MII_TRX_CONTROL+2(A1),
#%lo(MII_TRX_CONFIG_VAL|(1<<MII_TRXCNTL_TXCRC_EN)|(1<<MII_TRXCNTL_MII_PORT_RI))
  #else
  #error "Invalid MII mode option for port RE"
  #endif
#else
#error "Ubicom MII must use port RE"
#endif
    movei  MII_FUNC_MODE_SEL(A1), #MII_FUNC_SEL_MII|MII_FUNC_CODE
#if defined(MII_2nd_REG_BASE)
    movei  MII_FUNC_MODE_SEL(A6), #MII_FUNC_SEL_MII|MII_FUNC_CODE
#endif
    move.1 MII_FUNC_TX_LEVEL(A1), #0x0008
    move.1 MII_FUNC_RX_LEVEL(A1), #0x0009

    ;   Configure pin directions
#if defined(MII_2nd_REG_BASE)
    movei  D0, #MII_PORT2_PIN_DIR
    or.4   ETH_GPIO_CONTROL(A6), ETH_GPIO_CONTROL(A6), D0
#endif
    movei  D0, #MII_PORT_PIN_DIR
    or.4   ETH_GPIO_CONTROL(A1), ETH_GPIO_CONTROL(A1), D0

    ;   reset RX & TX FIFO
    ;   Default to work with RX FIFO_0
    bset   Dfifosw, #0, #MII_FUNC_FIFO_SEL
    bset   MII_INT_SET(A1), #0, #MII_SET_RX_FIFO_RESET
    bset   MII_FUNCTION_REG(A1), MII_FUNCTION_REG(A1), #MII_FUNC_FIFO_SEL
    bset   MII_INT_SET(A1), #0, #MII_SET_TX_FIFO_RESET
    bset   MII_INT_SET(A1), #0, #MII_SET_RX_FIFO_RESET
    bclr   MII_FUNCTION_REG(A1), MII_FUNCTION_REG(A1), #MII_FUNC_FIFO_SEL

    ; Prepare MII interrupts and states
    ;   Clear all pending interrupts
    ;   Set INT mask to enble RX interrupts only
```

```
    ;   Enable RX operation (default to 10Base-T half duplex)
    move.4 MII_INT_CLEAR(A1), #-1
    movei  MII_INT_MASK_LOW(A1),
#(1<<MII_INT_RX_1st_DATA)|(1<<MII_INT_RX_FIFO)|(1<<MII_INT_RX_EOP)|(1<<MII_INT_RX_CRS)
    movei  D0, #%hi(HRT_INT_IO_MASK)
    movei  D1, #%lo(HRT_INT_IO_MASK)
    shmrg.2D0, D1, D0
    move.4 INT_MASK1, D0; enable HW Ethernet INTs
    bset   MII_TRX_CONTROL(A1), MII_TRX_CONTROL(A1), #MII_TRXCNTL_RX_ENABLE
    move.4 ETH_RANDOM_MASK(A0), #1
    bset   ETH_RX_SPEED_TEST(A0), #0, #ETH_SPEED_TEST_INVALID

    ; Set thread interrupt mask and kick off the first IFG timer
    thread_timer_set_macro Difg
    bset   INT_CLR(HRT_INT_TIME_OUT), #0, #INT_BIT(HRT_INT_TIME_OUT)
    bset   INT_MASK(HRT_INT_TIME_OUT), INT_MASK(HRT_INT_TIME_OUT), #INT_BIT(HRT_INT_TIME_OUT)
    bset   INT_CLR(HRT_INT_TX_REQ), #0, #INT_BIT(HRT_INT_TX_REQ)
    bset   INT_MASK(HRT_INT_TX_REQ), INT_MASK(HRT_INT_TX_REQ), #INT_BIT(HRT_INT_TX_REQ)

    moveai RP, #%hi(__mii_isr_eth_trx)
    calli  RP, %lo(__mii_isr_eth_trx)(RP)
```

5.10   PCI Controller Programming Model

When reading this section, please refer to Section 6.7 for a general discussion of the PCI controller and its external interface signals. Refer to Section 7.8.1 for the register definitions for the PCI controller on Port B.

The PCI interface is implemented on Port B and Port C. Port C is a slave of Port B. After being put into PCI mode, the Port C interface is no longer used. All PCI transactions are conducted through Port B.

A dedicated hard real-time thread (HRT) is needed to service the PCI hardware.

5.10.1   PCI Startup

The first thing to be done after chip reset is to put Port B and Port C into PCI mode. This is done by first writing Port B's Function register (fields FN_SEL[2:0] = 1 (PCI), RX_FIFO_SEL[3] = 0, FN_RESET[7:4] = 1 (RESET), BR_TNUM[12:8] = thread number of HRT that will service PCI, RX_FIFO_TNUM[20:16] = thread number of HRT that will service PCI, and RX_FIFO_TNUM_EN[21] = 1). Next Port C's Function register is set (field FN_SEL[2:0] = 1 (PCI)). Finally the PCI block is taken out of reset by writing Port B's Function register again (field FN_RESET[7:4] = 0 (RUN), leaving all other fields unchanged).

At this point the PCI block is selected and reset. Note that the PCI Bus is still in an undefined state at this point. The HRT that is supporting PCI may now be started and will continue with the reset process.

5.10.2   PCI Bus Reset

Resetting the PCI bus is accomplished by putting the PCI arbiter into reset, thus preventing any device from using the bus, then setting up the PCI bus clock, and resetting the PCI bus according to the PCI 2.2 specification.

To reset the PCI bus, write the Port B Function Control 0 register (fields ARB_SM_RST_N[31] = 0 (RESET), ARB_SM_SEL[30] = desired arbitration policy, PCI_RST_N[29] = 0 (RESET), PCI_CLK_OUT_ENA[28] = 1 (ENABLED), PCI_CLK_DIV[27:24] = desired PCI_CLK frequency, and the rest of the fields to the desired PCI configuration).

Wait 100 ms per the PCI 2.2 specification.

Come out of PCI bus reset by writing to Port B Function Control 0 register (field PCI_RST_N[29] = 1 (RUN), leaving all the other fields unchanged).

Program all of the PCI configuration registers through the PCI BR register interface. These registers are described in Section 7.8.1. Note that these registers are **byte** addressable only.

Wait 1 second per the PCI specification.

Fill out the watermark levels, flush all buffers, clear and select all Port B interrupts. Also select Port B interrupts 0 and 2, but not 1. Select Port C interrupt 2. Make sure the PCI core is in Target mode (Port B Function Control 0 MASTER_TARGET_IF_SEL[23] = 0 (TARGET)).

Finally release the PCI Host arbiter by writing Function Control 0 ARB_SM_RST_N[31] = 1 (RUN), leaving all other fields unchanged.

The PCI is now in the Idle state.

5.10.3   PCI Idle State

The PCI Idle state occurs when the PCI controller has finished all outstanding transactions or come out of reset, and has nothing to do. It will suspend waiting for an interrupt indicating one or more of the following conditions:

- INTA has occurred on the PCI bus (Interrupt Port B Interrupt 2, Port B PCI Interrupt Status bit 11).
- IP51xx software wants to start a master transaction (appropriate software interrupt).
- A PCI Target transaction has arrived on the PCI bus (Interrupt Port B Interrupt 2, Port B PCI Interrupt Status bit 0).
- An error has occurred (Interrupt Port B Interrupt 2, Port B PCI Interrupt Status bits 10:4).

In any case handle the condition as described below.

5.10.4   PCI Target Transactions

5.10.4.1   PCI Target Read

Target read is detected by a Port B Interrupt 2 interrupt with Port B PCI Interrupt Status bit 0 set (TAR_NEW_CMD). Bit 5 in Port B PCI Function Status 1 (TAR_READ) indicates that a read is desired.

In this case the HRT must fetch the appropriate data as quickly as possible. The PCI core can be configured (Port B Function Control 0 register, bit 16) to retry the read until data is available, or just to insert delays.

The target address is obtained from the Port B Function Status 0 register. This PCI bus address must be translated into an IP51xx Protocol C bus address.

Turn on the Port B Interrupt 1, the Port B Tx FIFO watermark.

Write byte enables into the Port B Transmit FIFO HI once. This FIFO never underflows, it just uses the last value over and over. So it only needs to be written once, not once per data. Note that it needs to be written after each FIFO flush.

Next, fill the Tx FIFO with data and set Port B Interrupt Set register bit 16 (TAR_PCI_ACK) to tell the PCI core to send the data out on the PCI bus. The FIFO is 32 deep and 32 bits wide. Thus each entry corresponds to a data cycle on the PCI bus. Data is written to the FIFO via the Port B Transmit FIFO LO register.

With a 270 MHz IP51xx part a 1/8 HRT gets one clock every 29.6 ns. A PCI running at 33 MHz gets a clock every 30 ns. Thus a 1/8 HRT can keep up with the PCI bus (1/8 HRT PCI driver can write 2 words of data, and then set TAR_PCI_ACK). Writing the additional 30 words of data guarantees that the IP51xx will keep up with PCI for at least 32 words, and still overlap the IP51xx moving of data with data going out on the PCI bus. This calculation must be done for your particular configuration.

Next, suspend and wait for something to happen. Either the read will drain the FIFO below the watermark or the transaction will end. In the former case, put however many words are free into the FIFO and re-suspend. In the latter case, you will get a TAR_NEW_CMD. At this point (if you are trying to do streaming reads) check to see whether the data in the FIFO is still valid and whether the new transaction is a read to the address of the data at the front of the FIFO. If so, simply do a TAR_PCI_ACK to start the data flowing. Otherwise you must clean up by disabling the Port B Interrupt 1, the Port B Tx FIFO watermark and flushing the Tx FIFO.

Then proceed as if the PCI were in the Idle state.

5.10.4.2   PCI Target Write

Target write is detected by a Port B Interrupt 2 interrupt with Port B PCI Interrupt Status bit 0 set (TAR_NEW_CMD). Bit 4 in Port B PCI Function Status 1 (TAR_WRITE) indicates that a write is desired.

The target address is obtained from the Port B Function Status 0 register. This PCI bus address must be translated into a IP51xx Protocol C bus address.

The Port B Rx FIFO watermark interrupt is given by the Port B Interrupt 2.

Now the HRT immediately suspends. It is then expecting either the write will fill the Rx FIFO above the watermark or the transaction will end.

When the FIFO fills above the watermark, read out the watermark number of data words from the Port B Transmit FIFO LO register and write them to the appropriate place. You can use write byte enables from the Port B Transmit FIFO HI register if desired or just write the entire word regardless.

When the transaction finishes an interrupt on Port B PCI Interrupt Status bit 1 (TAR_WRITE_OP_DONE) occurs. In this case check the number of words in the Rx FIFO by reading RX_FIFO_LEVEL. If there are 32 words in the FIFO (it is full) there may be additional words in the PCI core. In that case you must read eight words out and again read RX_FIFO_LEVEL. If there were originally fewer than 32 words in the Rx FIFO you do not have to read out eight words.

At this point, you have a completed write transaction, and know the number of words left to read in the Rx FIFO. Set TAR_PCI_ACK, and then empty the number of words left in the Rx FIFO. This gives the PCI bus a chance to overlap with the emptying of the Rx FIFO.

When you are done emptying the FIFO, proceed as if in the PCI Idle state.

5.10.5   PCI Master Transactions

5.10.5.1   Leaving the PCI Idle State

After receiving a software interrupt and finding out what master transaction the software desires, you must ensure that no target transaction sneaks in and gets clobbered while you turn the PCI interface about. Recall that in the PCI Idle state the PCI core is left in Target mode so that target transactions can start immediately.

First set TAR_FORCE_RETRY in the Port B Function Control 0 register.

Then the driver should process the requested transaction, calculating the address for getting or storing data, and filling out Port B PCI Function Control 1 (MAS_PCI_ADDR).

If the processing after setting TAR_FORCE_RETRY takes more than six PCI clocks, you are good to go. If not stall for the remaining PCI clocks.

Now check TAR_NEW_CMD. If a target command has arrived you must process that command. After you are done, proceed to doing the Master transaction.

5.10.5.2   Doing the Master Transaction

Write Port B PCI Function Control 0 (fields MAS_TAR_IF_SEL, MAS_WCOUNT, MAS_RCV_BE, and MAS_PCI_CMD).

If the transaction is a write transaction, then write all of the data to Port B PCI Transmit FIFO LO and the corresponding byte enables to Port B PCI Transmit FIFO HI.

Then start the master transaction by setting Port B PCI Interrupt Set bit 20 (MAS_PCI_REQ).

Suspend, waiting for the MAS_COMPLETE interrupt.

If the transaction is a read transaction, then read all of the data from Port B PCI Receive FIFO HI.

The Master transaction is now finished; return to the PCI Idle state.

5.10.5.3   Returning to PCI Idle State

Release TAR_FORCE_RETRY and return to the PCI Idle state.

5.10.6   PCI Interrupts and Errors

When you receive an INTA or any of the error interrupts, respond appropriately. For errors, reset the PCI bus unless there is some application specific way to recover.

5.11   GMAC Programming Model

When reading this section, please refer to Section 6.8 for a general discussion of the GMAC and its external interface signals. Refer to Section 7.11.1 for the register definitions for the GMAC on Port F.

5.11.1   GMAC Initialization

The following sample code shows how to initialize the GMAC.

```
;========================================================================
; initialization and start up
;   void ipEthernet_thread_start_@INST@(void* NULL)
    .global _ipEthernet_thread_start_@INST@
    .type _ipEthernet_thread_start_@INST@ @function
;========================================================================

    .section .text.ipEthernet_thread_start_@INST@,"ax",@progbits


_ipEthernet_thread_start_@INST@:
    ; The following address register are reserved
    ;   A0 -> RAM variable base address
    ;   A1 -> SerDes/MII register base address
    ;   A2 -> RX buffer base address
    ;   A3 -> TX buffer base address
    ;   A7 -> Timer block register base address
    moveai A0, #%hi(eth_isr_var_base_@INST@)
    lea.4  A0, %lo(eth_isr_var_base_@INST@)(A0)
    movei  D0, #ETH_RX_PKT_INFO
    lea.1  A2, (A0,D0)
    movei  D0, #ETH_TX_PKT_INFO
    lea.1  A3, (A0,D0)
    moveai A7, #%hi(TIMER_BASE)

    ; The following data register are reserved
    ;   IFG and slot time are default to 10Base-T value
    ;   SFD differs from preamble pattern by just 1 bit
    movei  D_one, #1
    movei  Difg, #ETH_10BASE_T_IPG
    movei  Dslot, #ETH_10BASE_T_SLOT
    movei  Dsfd, #0x5555
    shmrg.2Dsfd, Dsfd, Dsfd
    movei  Dtmreg, #TIMER_SYSCOM(INT_BIT(HRT_INT_TIME_OUT))>>2
    bset   Dtmint, #0, #INT_BIT(HRT_INT_TIME_OUT)
    bset   Dtxint, #0, #INT_BIT(HRT_INT_TX_REQ)
    bset   Ddsrint, #0, #INT_BIT(HRT_INT_DSR)

    ; Initial suspend to wait for global interrupt enable
    bset   INT_SET(HRT_INT_TIME_OUT), #0, #INT_BIT(HRT_INT_TIME_OUT)
    bset   INT_MASK(HRT_INT_TIME_OUT), INT_MASK(HRT_INT_TIME_OUT), #INT_BIT(HRT_INT_TIME_OUT)
    suspend


    ; Initialize MII I/O HW block for Ethernet operation
    ;   Initialize A1 -> GMII register base address
    ;   Reset I/O block before doing anything
    ;   Set I/O mode to RG/MII/RMII Ethernet (mode setting ?)
    ;   Set TX water mark = 16/max and RX FIFO water mark = 17
    ;   Setup FIFO selection according to the MII FIFO choice
    ;   A6 -> Blocking region base address
    moveai A1, #%hi(GMII_REG_BASE)
    moveai A6, #%hi(GMII_REG_BASE + IO_PORT_BR_OFFSET)
    move.4 GMII_INT_MASK(A1), #0
```

```
    movei  GMII_FUNCTION_REG(A1), #(1<<5)|@INST@IPETHERNET_THREAD_NUM
    movei  GMII_FUNC_MODE_SEL(A1),
#(@INST@IPETHERNET_THREAD_NUM<<GMII_FUNC_THREAD)|GMII_FUNC_RESET_MII|GMII_FUNC_CODE
    cycles 4
#if defined(@INST@GMII_RF_USE_GMII)
    movei  GMII_TRX_CONTROL+0(A1), #%hi(GMII_TRXCNTL_RGMII|GMII_TRXCNTL_CLK125MHz)
    movei  GMII_TRX_CONTROL+2(A1), #%lo(GMII_TRXCNTL_RGMII|GMII_TRXCNTL_CLK125MHz)
#elif defined(@INST@GMII_RF_USE_MII)
    movei  GMII_TRX_CONTROL+0(A1), #%hi(GMII_TRXCNTL_MII|GMII_TRXCNTL_CLK2_5MHz)
    movei  GMII_TRX_CONTROL+2(A1), #%lo(GMII_TRXCNTL_MII|GMII_TRXCNTL_CLK2_5MHz)
#elif defined(@INST@GMII_RF_USE_RMII)
    movei  GMII_TRX_CONTROL+0(A1), #%hi(GMII_TRXCNTL_RMII|GMII_TRXCNTL_CLK2_5MHz)
    movei  GMII_TRX_CONTROL+2(A1), #%lo(GMII_TRXCNTL_RMII|GMII_TRXCNTL_CLK2_5MHz)
#else
#error "Invalid MII mode option for port RF"
#endif
    movei  GMII_FUNC_MODE_SEL(A1),
#(@INST@IPETHERNET_THREAD_NUM<<GMII_FUNC_THREAD)|GMII_FUNC_SEL_MII|(1<<GMII_FUNC_FIFO_SEL)|GMII_
FUNC_CODE
    move.1 GMII_FUNC_TX_LEVEL(A1), #0x0000
    move.1 GMII_FUNC_RX_LEVEL(A1), #0x0011


    ;   Configure blocking region
    ;       Reset
    movei  D0, #%lo(GMII_BR_MAC_CONFIG1_RESET)
    movei  D1, #%hi(GMII_BR_MAC_CONFIG1_RESET)
    shmrg.2D0, D0, D1
    move.4 GMII_BR_MAC_CONFIG1(A6), D0
    movei  D0, #%lo(GMII_BR_IF_CNTL_RESET)
    movei  D1, #%hi(GMII_BR_IF_CNTL_RESET)
    shmrg.2D0, D0, D1
    move.4 GMII_BR_IF_CNTL(A6), D0


    ;       Basic configuration
#if defined(@INST@GMII_RF_USE_GMII)
    movei  D0, #%lo(GMII_BR_MAC_CONFIG2_GMII|GMII_BR_MAC_CONFIG2_FD)
    movei  D1, #%hi(GMII_BR_MAC_CONFIG2_GMII|GMII_BR_MAC_CONFIG2_FD)
#elif defined(@INST@GMII_RF_USE_MII) || defined(@INST@GMII_RF_USE_RMII)
    movei  D0, #%lo(GMII_BR_MAC_CONFIG2_MII)
    movei  D1, #%hi(GMII_BR_MAC_CONFIG2_MII)
#endif
    shmrg.2D0, D0, D1
    move.4 GMII_BR_MAC_CONFIG2(A6), D0
    ;move.4GMII_BR_IPG_IFG(A6), #0x40605060; dummy code to repeat default value
    ;move.4GMII_BR_HALF_DUPLEX(A6), #0x00a1f037; dummy code to repeat default value
    ;move.4GMII_BR_MAX_FRAME(A6), #1536; dummy code to repeat default value


    ;       MII management I/F configuration (Disable this unimplemented feature!)
    bset   GMII_BR_MII_CONFIG(A6), #7, #31; Set up MII mgmt I/F (slowest clock)

    ;       Write local MAC address
    move.4 GMII_BR_LOCAL_MAC1(A6), #0; ETH_RX_LOCAL_MAC_BUF(A0)
    move.4 GMII_BR_LOCAL_MAC2(A6), #0; ETH_RX_LOCAL_MAC_BUF+4(A0)

    ;   Configure pin directions
    movei  D0, #GMII_PORT_PIN_DIR
    or.4   ETH_GPIO_CONTROL(A1), ETH_GPIO_CONTROL(A1), D0

    ;   reset RX & TX FIFO
    ;   Default to work with RX FIFO_0
```

```
    bset   Dfifosw, #0, #GMII_FUNC_FIFO_SEL
    bset   GMII_INT_SET(A1), #0, #GMII_SET_RX_FIFO_RESET
    movei  GMII_FUNC_MODE_SEL(A1),
#(@INST@IPETHERNET_THREAD_NUM<<GMII_FUNC_THREAD)|GMII_FUNC_SEL_MII|GMII_FUNC_CODE
    bset   GMII_INT_SET(A1), #0, #GMII_SET_TX_FIFO_RESET
    bset   GMII_INT_SET(A1), #0, #GMII_SET_RX_FIFO_RESET

    ; Prepare MII interrupts and states
    ;   Clear all pending interrupts
    ;   Set INT mask to enble RX interrupts only
    ;   Enable RX operation (default to 10Base-T half duplex)
    move.4 GMII_INT_CLEAR(A1), #-1
    movei  GMII_INT_MASK_LOW(A1),
#(1<<GMII_INT_RX_1st_DATA)|(1<<GMII_INT_RX_FIFO)|(1<<GMII_INT_RX_EOP)|(1<<GMII_INT_TX_EOP)
    movei  D0, #%hi(HRT_INT_IO_MASK)
    movei  D1, #%lo(HRT_INT_IO_MASK)
    shmrg.2D0, D1, D0
    move.4 INT_MASK1, D0; enable HW Ethernet INTs
    move.4 ETH_RANDOM_MASK(A0), #0; initalize ETH_GMII_TX_COUNT
    bset   ETH_RX_SPEED_TEST(A0), #0, #ETH_SPEED_TEST_INVALID

    ; Set thread interrupt mask and kick off the first IFG timer
    ;thread_timer_set_macro Difg
    ;bset   INT_CLR(HRT_INT_TIME_OUT), #0, #INT_BIT(HRT_INT_TIME_OUT)
    ;bset   INT_MASK(HRT_INT_TIME_OUT), INT_MASK(HRT_INT_TIME_OUT), #INT_BIT(HRT_INT_TIME_OUT)
    bset   INT_CLR(HRT_INT_TX_REQ), #0, #INT_BIT(HRT_INT_TX_REQ)
    bset   INT_MASK(HRT_INT_TX_REQ), INT_MASK(HRT_INT_TX_REQ), #INT_BIT(HRT_INT_TX_REQ)

    ; Start operation
    movei  D0, #%lo(GMII_BR_MAC_CONFIG1_START)
    movei  D1, #%hi(GMII_BR_MAC_CONFIG1_START)
    shmrg.2D0, D0, D1
    move.4 GMII_BR_MAC_CONFIG1(A6), D0
#if defined(@INST@GMII_RF_USE_GMII)
    movei  D0, #%lo(GMII_BR_IF_CNTL_GMII_START)
    movei  D1, #%hi(GMII_BR_IF_CNTL_GMII_START)
#elif defined(@INST@GMII_RF_USE_MII)
    movei  D0, #%lo(GMII_BR_IF_CNTL_MII_START)
    movei  D1, #%hi(GMII_BR_IF_CNTL_MII_START)
#elif defined(@INST@GMII_RF_USE_RMII)
    movei  D0, #%lo(GMII_BR_IF_CNTL_RMII_START)
    movei  D1, #%hi(GMII_BR_IF_CNTL_RMII_START)
#else
#error "Invalid MII mode option for port RF"
#endif
    shmrg.2D0, D0, D1
    move.4 GMII_BR_IF_CNTL(A6), D0

    moveai RP, #%hi(__gmii_isr_eth_trx)
    calli  RP, %lo(__gmii_isr_eth_trx)(RP)

#else /* defined(IPETHERNET_USE_GMII) */
#error"ipEthernet instance must use either SerDes or MII"
```

5.12   USB Controller Programming Model

Please refer to Section 6.9 for a general discussion of the high-speed USB controller and its external interface signals. Refer to Section 7.15.1 for the register definitions for the High-Speed USB controller.

The IP51xx supports both USB host mode and USB peripheral mode. Device drivers for Host mode are currently available from Ubicom. To determine the availability of Peripheral mode drivers, contact your Ubicom sales representative.

5.12.1   USB Initialization

The following sample code shows how to initialize the High-Speed USB Controller.

```
/*
    * Configure the port. Set the thread that will access the port, reset the function and
    *  pull PLL out of reset, and deassert function reset
    */
   DEBUG_INFO("Initializing the USB port %d", USB_20_PORT);
   arch_io_port(USB_20_PORT)->function = (IPUSB20_HRT_THREAD_NUM << 8) | (1 << 4) | 1;
   arch_io_port(USB_20_PORT)->ctl0 = (1 << 5)| (1 << 4) | 1;
   usb_timer_blocking_sleep(TICK_RATE/10);
   arch_io_port(USB_20_PORT)->function = (IPUSB20_HRT_THREAD_NUM << 8) | 1;;

   /*
    * Configure interrupts
    */
   DEBUG_INFO("Setting HRT interrupts");
   arch_interrupt_enable(THREAD_INT(IPUSB20_HRT_THREAD_NUM));
   arch_interrupt_enable(PORT_OTHER_INT(USB_20_PORT));
   arch_io_port(USB_20_PORT)->int_clr = (1 << 3);
   arch_io_port(USB_20_PORT)->int_mask = (1 << 3);

   /*
    * Configure USB interrupts. We poll endpoint completion interrupts, so turn off
    * the enable bits for these interrupts
    */
   DEBUG_INFO("Setting USB interrupts");
   u16_t discard = arch_io_port_usb(usbhcdi)->intrTx;
   discard = arch_io_port_usb(usbhcdi)->intrRx;
   discard = arch_io_port_usb(usbhcdi)->intrUsb;
   arch_io_port_usb(usbhcdi)->intrTxE = 0xFFFF;
   arch_io_port_usb(usbhcdi)->intrRxE = 0xFFFF;
   arch_io_port_usb(usbhcdi)->intrUsbE = 0xFF;

   arch_io_port_usb(usbhcdi)->devCtl = 0x01;
```

5.12.2   USB Transactions

Figure 5-1 shows a flow chart for an example bulk IN transaction. Figure 5-2 shows a bulk OUT transaction.

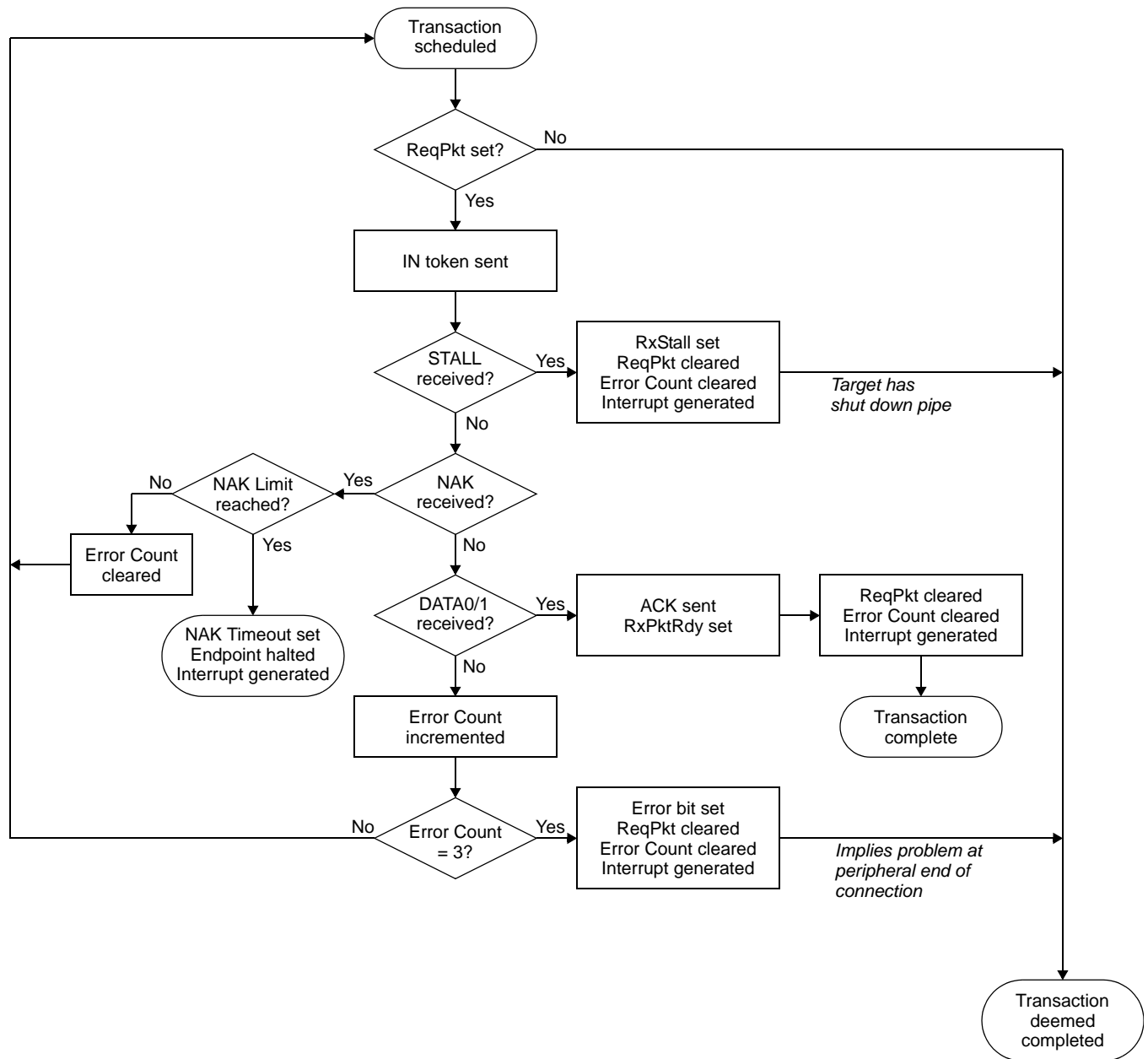Full descriptions of the available USB transactions are included in Ubicom's SDK.

Figure 5-1 USB Bulk IN Transaction

Transaction
scheduled

TxPktRdy set?   No

Yes

OUT token sent

DATA0/1 Packet sent

STALL
received?   Yes   RxStall set
                               TxPktRdy cleared   Target has
                               Error Count cleared   shut down pipe
No   Interrupt generated

ACK
received?   Yes   TxPktRdy cleared
                               Error Count cleared
No   Interrupt generated

Transaction
complete

No   NAK Limit   Yes   NAK
          reached?   received?

          Yes   No

Error Count
cleared

NAK Timeout set   Error Count
Endpoint halted   incremented
Interrupt generated

No   Error Count   Yes   Error bit set
          = 3?   TXPktRdy cleared   Implies problem at
                                    Error Count cleared   peripheral end of
                                    Interrupt generated   connection

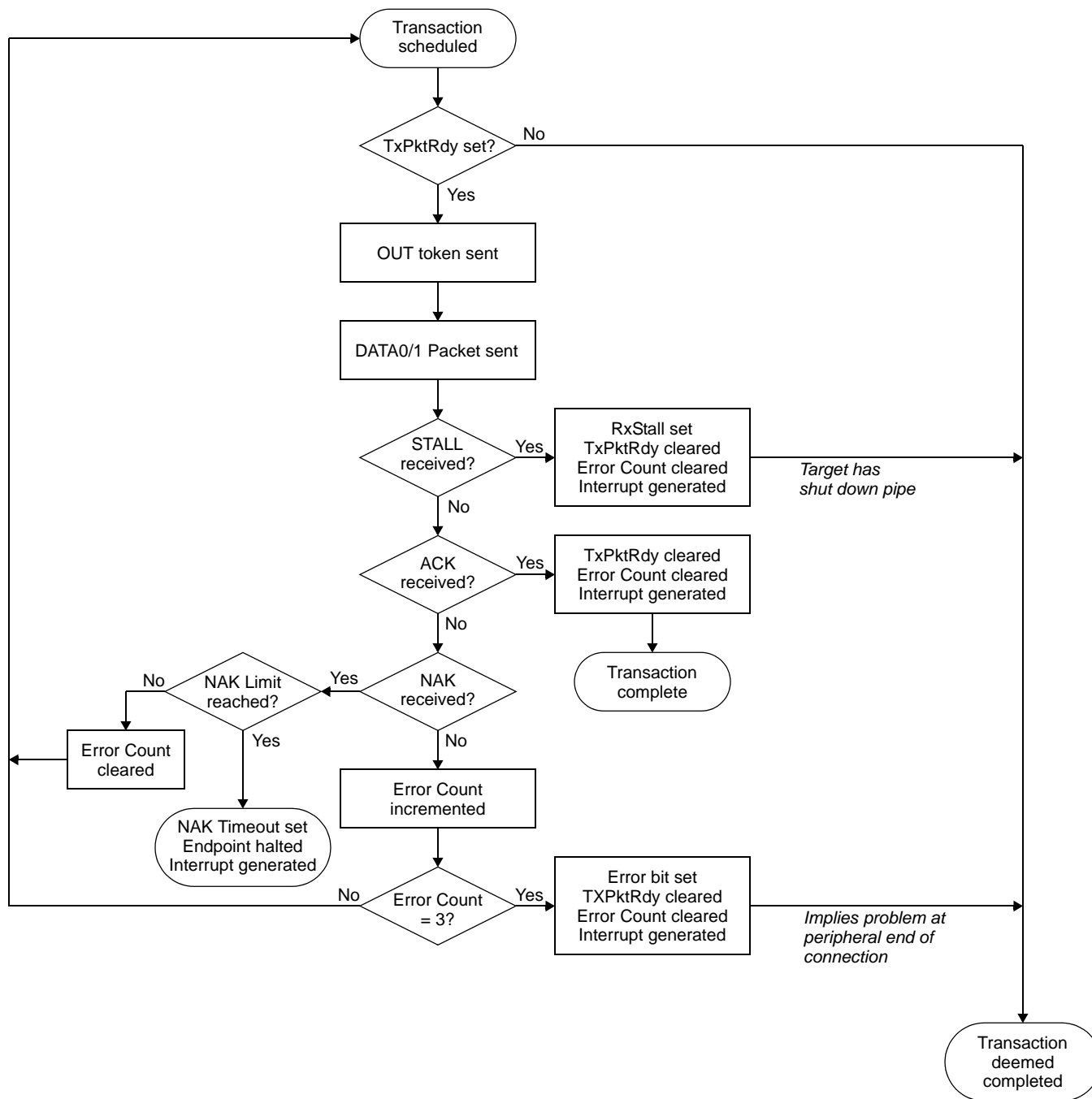Transaction
deemed
completed

Figure 5-2 USB Bulk OUT Transaction

5.13   On-Chip Memory Programming Model

5.13.1   Initialization After System Reset

After a hard chip reset, an initialization sequence needs to be applied to the On-Chip Memory (OCM) in order to get it into a functional state. This is due to system reset being decoupled from the BIST engine. This element allows a BIST soft repair to persist through a system reset but leaves the OCM unusable until the BIST engine is reset by software.

The following code implements a suitable On-Chip Memory initialization sequence:

```
OCM_BIST_CFG_WRCK = 7
OCM_BIST_CFG_WRST = 6
OCM_BIST_CFG_RST_SMS_A = 5

// Initialize pointer to base of OCM Registers
moveai a0, #%hi(OCP_OCMC_ADDR)

// Assert BIST reset signals : WRST and
RST_SMS_A
move.4 OCM_BIST_CFG(a0), #((0x1<<OCM_BIST_
CFG_WRST)│(0x1<<OCM_BIST_CFG_RST_SMSM_A))
move.4 OCM_BIST_CFG(a0), #0x00
// Deassert WRST and RST

// Create a WRCK clock edge
bset OCM_BIST_CFG(a0), OCM_BIST_CFG(a0), #OCM_
BIST_CFG_WRCK
nop        // Allow reset to take effect
```

5.13.2   Executing BIST/BISR to OCM

The following code segment will execute a combined BIST (Built In Self Test) and BISR (Built In Self Repair) sequence to the On-chip Memory. BIST is first executed in order to identify any memory locations that may benefit from soft repair. If BIST fails, thereby indicating that a soft repair is necessary, BISR is then executed. BISR fails if there is insufficient redundant memory resource to repair the location. Otherwise the memory location is succesfully repaired.

```
#define OCMC_BASE (OCP_BASE + OCP_OCMC)
#define OCMC_BANK_MASK 0x00
#define OCMC_BIST_CNTL 0x04
#define OCMC_BIST_STAT 0x08

#define OCMC_BANK_PROG(n) ((1<<(n))-1)

#define OCMC_BIST_WRCK (1<<7)
#define OCMC_BIST_RESET (1<<5)
#define OCMC_BIST_SMART (1<<4)
#define OCMC_BIST_RUN (1<<3)
#define OCMC_BIST_REPAIR (1<<2)
```

```
#define OCMC_BIST_READY (1<<3)
#define OCMC_BIST_FAIL (1<<2)

bclr OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_WRCK)

; Start BIST
bset OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_RUN)
bset OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_REPAIR)
bset OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_SMART)
pipe flush 0

; Wait for BIST to start running
1: bset OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
   #%bit(OCMC_BIST_WRCK)
jmpt.t .+4
bclr OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_WRCK)
btst OCMC_BIST_STAT(A1),
    #%bit(OCMC_BIST_READY)
jmpne.t 1b

1: bset OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_WRCK)
jmpt.t .+4
bclr OCMC_BIST_CNTL(A1), OCMC_BIST_CNTL(A1),
    #%bit(OCMC_BIST_WRCK)
btst OCMC_BIST_STAT(A1),
    #%bit(OCMC_BIST_READY)
jmpeq.t 1b

btst OCM_BIST_STAT(a1),
    #%bit(OCM_BIST_STAT_READY_SMS)
jmpeq fail
btst OCM_BIST_STAT(a1),
    #%bit(OCM_BIST_STAT_FAIL_SMS)
jmpne fail
pass:
```

5.13.3   Applying BIST Soft Repair to OCM

If the BIST soft repair has been applied, its effect will persist even through a system reset. It will remain in effect until the OCM initialization sequence is exercised, BIST/BISR is executed, the OCM Code Request Buffer is flushed, or an interruption to the core power supply causes the chip to reset.

5.13.4   Flushing the OCM Code Request Buffer

All instruction back accesses conducted via the data port are initially posted into a Code Request Buffer. The buffer can store one posted access per thread. Sometimes it is

beneficial to guarantee that any posted accesses are completed. For any particular thread to guarantee that any posted transactions from the same thread are completed, a data port read of a known address from any instruction bank can be applied. The completion of this read guarantees that the entry corresponding to the thread is empty. In order to globally ensure that all posted transactions are complete, software will have to set up some number of idle OCM instruction fetch cycles. The number of cycles is equal to the number of threads.

The OCM code request buffer may need to be flushed if software changes the designation of individual memory banks as either code or data banks. Software should flush the request buffers before changing this code/data bank mapping. This should be done only if software has previously made a data port access of a code bank which it now wants to designate as a data bank.

Additionally, software will need to flush the write data buffer before redesignating a data bank into a code bank. The procedure for flushing the write data buffer is to ensure that there are no write accesses to the on-chip data memory for 5 core clock cycles.

5.14   Processor Programming Model

5.14.1   Handling Serror and Aerror

Error conditions in the Protocol C peripherals may result in an serror (synchronous error) trap or an aerror (asynchronous error) interrupt. The processor simply maintains status to indicate whether an serror or aerror has occurred. It maintains no further information regarding the error. More detailed information regarding the error may be maintained (optionally) by the individual peripherals. When an serror or aerror is detected by the processor, it is intended that the processor read all peripheral specific serror or aerror (as appropriate) status registers to determine the actual cause. Not all peripherals maintain additional status information for serrors. In these cases, the trapped instruction itself must be examined to determine the error cause. When a peripheral does not maintain status information for an aerror, there is little that software can do to determine the error cause. Table 5-10 and Table 5-11 list possible serror causes, as well as a description of the peripheral specific error status (if any).

Table 5-10 Possible Serror Causes

Serror Source	Error Cause	Status Register
OCM	Instruction fetch from data bank	None
I/O	Access to unimplemented memory space in non-blocking region	None
I/O	Illegal access to read-only or write-only registers in nonblocking region	None
I/O	Unauthorized thread access to blocking region	None
I/O	When enabled, unauthorized thread access to the RX FIFO LOW register	None

Table 5-11 Possible Aerror Causes

Aerror Source	Error Cause	Status Register
I-Cache	Illegal transaction on MCB	ICCR_STAT[0]
D-Cache	Illegal transaction on MCB (including write to flash)	DCCR_STAT[0]
PCI	Access to unimplemented memory space	None
PLIO	Access to unimplemented memory space	None

5.14.2   Handling Traps

A processor trap occurs to indicate an error condition for the currently executing instruction. Each thread has a TRAP_CAUSE register with one bit defined for each possible thread cause.

The global register MT_TRAP has one bit for each thread. When that bit is a 1, it indicates that for the corresponding thread, one of the bits in its TRAP_CAUSE register is set.

Software can set or clear bits in the MT_TRAP register by writing to the corresponding bits in MT_TRAP_SET or MT_TRAP_CLR, respectively.

Software cannot write directly to MT_TRAP.

5.14.2.1   How to Enable Traps

You can enable traps for a given thread by writing a 1 in that thread's bit position in the global MT_TRAP_EN register. You can disable traps for a given thread by writing a 0 in that thread's bit position.

5.14.2.2 Determining Trap Causes

When a processor trap occurs, the cause of the trap can be determined by reading the TRAP_CAUSE register for the affected thread. See Section 7.2.3 for general descriptions of the defined trap causes.

5.14.2.3 Simultaneous Trap and Blocking Conditions

There are various types of instruction errors which can cause a trap. Non-acknowledged instruction fetch or data access requests can cause an instruction to be blocked. Instructions can be blocked only if they are free of error conditions. Furthermore, certain error conditions can be reported simultaneously (for the same instruction), while others are precluded by prior errors.

Table 5-12 lists the possible error conditions which can lead to an instruction trap, divided into four groups.

Only errors within the same group will be reported simultaneously. Errors in lower-numbered groups take priority over errors in higher-numbered groups. A src1 serror can only happen if there is no src1_decode_err. Similarly, dst_serror can only happen if there is no dst_decode_err.

Table 5-12 Error Groups

Error Group	Error	Description
1	i_decode_err	PC decode error
2	i_serror	Instruction fetch serror
3	illegal_inst	Illegal instruction
4	src1_decode_err	SRC1 operand address decode error
	dst_decode_err	DST operand address decode error
	src1_misaligned	SRC1 operand misaligned
	dst_misaligned	DST operand misaligned
	src1_serror	SRC1 operand serror
	dst_serror	DST operand serror
	dcapt	Write address trap error
	dst_range_err	DST memory protection error
	src1_range_err	SRC1 memory protection error
	i range err	Instruction memory protection error

5.14.2.4 Trap and Block Actions

The following tables show the trap and block actions which are taken for all combinations of instruction errors, data operand errors, instruction acknowledges, and data acknowledges.

The following abbreviations are used in the tables:

    ierr = any error in groups 1, 2, or 3
    derr = any error in group 4
    iack = acknowledged protocol C instruction fetch
    dack = acknowledged or no protocol C data request
    val = mp8 val
    sval = mp8 sval
    trap = ex trap
    block = ex d block

Table 5-13 applies to error conditions for threads that have traps enabled.

Table 5-13 Actions for Threads with Traps Enabled

Inputs				Outputs				Action	Comment
ierr	derr	iack	dack	val	sval	trap	block		
0	x	0	x	0	1	0	0	blocked by AC	instruction fetch not acknowledged
0	0	1	0	0	1	0	1	blocked by EX	data access not acknowledged
0	0	1	1	1	1	0	0	instruction completes	
1	x	0	x	0	0	1	0	trapped	i_decode_err, i_serror
1	x	1	x	0	0	1	0	trapped	illegal_inst
0	1	1	0	0	0	1	0	trapped	d_serror (plus possibly others)
0	1	1	1	0	0	1	0	trapped	decode_err, misaligned, memory protection, write address

Table 5-14* applies to errors for threads which have traps disabled.

Table 5-14 Actions for Threads with Traps Disabled

Inputs				Outputs				Action	Comment
ierr	derr	iack	dack	val	sval	trap	block		
0	x	0	x	0	1	0	0	blocked by AC	instruction fetch not acknowledged
0	0	1	0	0	1	0	1	blocked by EX	data access not acknowledged
0	0	1	1	1	1	0	0	instruction completes	
1	x	0	x	0	0	0	0	instruction ignored	i_decode_err, i_serror
1	x	1	x	0	0	0	0	instruction ignored	illegal_inst
0	1	1	0	0	0	0	0	instruction ignored	d_serror (plus possibly others)
0	1	1	1	1	1	0	0	instruction completes	decode_err, misaligned, memory protection, write address

5.14.3 Memory Protection

Any thread is permitted to read and write to any direct space register. However, execution from instruction space and reads and writes to the indirect space are permitted only if these operations are enabled for the executing thread and the address being accessed. There are three pairs of address range registers for the instruction space and four pairs for the data space. Additionally, each range pair has a corresponding active-high thread enable mask. If an instruction fetch or indirect space data access falls within the inclusive range defined by one of the appropriate range pairs, and that range is enabled for the executing thread, then the access is permitted. Otherwise, it is trapped, with one (or more) of three possible causes.

The range values and thread enables are received from the register module on the following busses:

```
reg i range hi =
    i range2 hi[31:2],
    i range1 hi[31:2],
    i range0 hi[31:2]

reg i range lo =
    i range2 lo[31:2],
    i range1 lo[31:2],
    i range0 lo[31:2]

reg i range en =
    i range2 en['MP TNUM-1:0],
    i range1 en['MP TNUM-1:0],
    i range0 en['MP TNUM-1:0]

reg d range hi =
    d range3 hi[31:2],
    d range2 hi[31:2],
    d range1 hi[31:2],
    d range0 hi[31:2]

reg d range lo =
    d range3 lo[31:2],
    d range2 lo[31:2],
    d range1 lo[31:2],
    d range0 lo[31:2]

reg d range en =
    d range3 en['MP TNUM-1:0],
    d range2 en['MP TNUM-1:0],
    d range1 en['MP TNUM-1:0],
    d range0 en['MP TNUM-1:0]
```

The following three error indications are asserted in case of access violations for the general destination, source 1, or instruction accesses, respectively:

dst_range_err, src1 range err, i range err

5.15 Security Block Programming Model

5.15.1 Security Block Clock

The clock to the Security Module is controllable by software via the clock core configuration register. This bit controls a logic gate that either enables or disables the clock. The clock is enabled when the chip comes out of reset.

Software must disable the clock when the Security Module is not in use in order to reduce core power consumption.

5.15.2 Enabling the Security Block Clock

When enabling the clock, software must wait for an appropriate duration before it can acess the Security Module.

Software must wait for a period of 6 core clock cycles if the very first security module access is a read operation. This is demonstrated in the following code sequence:

```
bset (a0), (a0), #OCP_EN_CLK_SEC// Instruction
to enable security module clock

// delay for OCP write logic
nop
nop
nop
// delay for clock block logic
nop
nop
nop
```

Software is required to wait for a period of 11 core clock cycles if the very first security module access is a write operation. This is demonstrated in the following code sequence:

```
bset (a0), (a0), #OCP_EN_CLK_SEC// Instruction
to enable security module clock

// delay for OCP write logic
nop
nop
nop
// delay for clock block logic
nop
nop
nop
// distance between read and write in MP
pipeline
nop
nop
nop
nop
nop
```

5.16   Clocks Programming Model

Configuring the PLLs to generate the desired system clocks must be done carefully in order to minimize the impact of any large power surges. When the PLLs are enabled and system clocks are switched over from low to high frequencies, there are instantaneous current spikes across the clock tree. The magnitude and duration of the power surge is dependent on the clock frequency change and the number of clocks being enabled. Therefore, whenever software exercises a critical timing path before core power normalizes, there is a potential for failure. This may be especially evident when switching from a very low frequency to a very high frequency on a slow process part under minimum voltage conditions — for example, switching the core frequency from 12 MHz to 270 MHz.

There are number of things software can do i to alleviate this problem. First, do not enable all the PLLs around the same time. Stage the de-assertion of PLL reset in order to bring each PLL up over a reasonable period of time. Second, use the core forward divider to ramp the core clock up to it's desired frequency. Ramping the core clock up in 2 stages should be sufficient.

The following is a recommended sequence for bringing the system clocks online based on these principles. It assumes that the PLLs are already in reset:

1. Initialize the Core PLL Reference, Feedback, and Output dividers for a desired core clock frequency.
2. Set the Core Forward Divider to divide the target core clock frequency by two.r4
3. Initialize the I/O PLL Reference, Feedback, and Output dividers for a 500 MHz I/O clock frequency.
4. Initialize the DDR PLL Reference, Feedback, and Output dividers for a desired DDR clock frequency.
5. Initialize the DDR Deskew PLL Feedback divider with a value of 0. This is the configuration in order for it to deskew the DDR clock correctly.
6. Wait for a period greater than or equal to 5µs.
7. Take the Core PLL out of reset.
8. Wait 500 system reference clocks for the Core PLL to lock.
9. Configure the Core source select to switch to the Core PLL.
10. Wait 100 core clock cycles for power to normalize.
11. Set the Core Forward Divider for the desired core clock frequency.
12. Wait 100 core clock cycles for power to normalize.
13. Take the I/O PLL out of reset.
14. Wait 500 system reference clocks for the I/O PLL to lock.
15. Configure the I/O source select to switch to the I/O PLL.
16. Take the DDR PLL out of reset.
17. Wait 500 system reference clocks for the DDR PLL to lock.
18. Configure the DDR source select to switch to the DDR PLL.
19. Set the DDR Deskew source select to switch to the Deskew PLL. It is essential that this step be taken before the Deskew PLL is brought out of reset. This ensures that the entire DDR clock system is a closed loop before turning on the Deskew PLL.
20. Take the DDR Deskew PLL out of reset.
21. Wait 500 system reference clocks for the DDR Deskew PLL to lock.

5.17   Random Number Generator

The Random Number Generator (RNG) consists of three oscillators feeding a Linear Feedback Shift Register (LFSR) after synchronization to the core clock.

Each read of this register returns a uniformly distributed random number. Since the register shifts one bit per core clock, if software needs multiple uncorrelated numbers, it should wait at least 32 core clocks between reads. If software is building a strong cryptographic key longer than 32 bits, it should wait much longer between reads to allow true randomness to accumulate.

The oscillators can be disabled. When their enable bit is high, the oscillator is running, and when it is low it is stopped, with the output in a low state. The oscillator enable bit powers up equal to zero. When the oscillators are disabled, the random number generator still generates uniformly distributed random numbers, but the only source of true randomness is in the timing of the register reads.

5.18   Reset

5.18.1   Reset Sources

The possible sources that can cause a chip reset are:

- External chip reset
- Power-on reset
- Reset from the Watchdog module
- Reset from the Debug Port module
- Reset caused by a processor trap
- Software initiated reset

When a chip reset occurs, the reason for the reset can be determined by reading the Reset Reasons register (see Section 5.18.5 and Table 7-5).

5.18.2   Reset Output To External Devices

The IP51xx chip does not have a dedicated reset signal output to cause external devices to reset when it is being reset. If an external device requires such a reset signal, a GPIO pin on the IP51xx could be used for this purpose. All GPIO pins would be configured as input pins after an chip reset. The GPIO pin used as a reset for an external device should have a pullup or pulldown resistor on the board to pull the signal to the correct reset active level for the device. After the chip is initialized, the software can start to drive the GPIO pin to the desired level.

5.18.3   I/O Function Resets

The function reset bits in I/O port Function registers only reset their respective function modules. When resetting an I/O function via its function reset bit, the software must ensure that all requests from the processor have been completed by the I/O function before issuing the reset. For I/O functions that implement only the non-blocking region memory space, all requests are completed when the processor instruction completes. For I/O functions that implement the blocking region memory space, processor write requests to the blocking region are posted. Before resetting the I/O function, software should issue a read request to a valid address in the blocking region of the I/O function to flush any posted write requests.

5.18.4   Warm Reset

If an external device needs to be reset when the IP51xx is reset by internal reasons, the method described in Table 5.18.2 can be used.

One special case is the serial flash device. Serial flash devices do not have a dedicated reset input pin. They can be reset by a special reset command sequence or their internal power-on reset logic. The IP51xx flash controller does not implement the reset command sequence, because reset sequences are slightly different for serial flash devices from different vendors, and some devices do not have a reset command. Therefore, the IP51xx chip depends on the flash device to be reset by its own power-on reset logic.

During the software development stage, if the IP51xx chip fails to reboot after an internally generated reset, the following steps should be taken if the user wants to debug the problem:

1. Connect the debugger to the debug port if it's not already connected.
2. Read the Rest Reason register to get the reset reason information for the last reset.
3. Issue a reset via the Debug Port after the flash device is no longer busy (by waiting for the maximum amount of time needed for the flash command in progress). This will cause the chip to reset and start instruction fetch at the correct location. Do not use the external reset, as it will cause the debug module to lose its configuration, such as halting the processor upon on internal reset. After the second reset, the Reset Reason register would not contain the correct information regarding the cause of the chip reset.

If debugging is not needed, the user can just use the external reset to restart booting after a failed reset.

5.18.5   Using Reset Reason Flags

One or more of the reset cause flags may be asserted after any reset event, and it is the responsibility of software to arbitrate and properly determine the reason for the reset. The reset cause information is captured in the Reset Reasons register (see Table 7-5).

The steps for properly determining the reset cause are:

1. If the POR (Power-on reset) reason bit is asserted, the chip was reset due to a power-on event and no other reset cause bit is valid. If this bit is not asserted, go to Step 2. If the IP51xx is in reset bypass mode, the POR reason bit should be ignored.

2. If the EXT (External reset) reason bit is asserted, the chip was reset due to an external reset event (the external reset pin was asserted) and no other reset cause bit is valid. If this bit is not asserted, go to Step 3.

3. Read the values of the remaining reset cause bits to determine the reset reason. It is possible, and legitimate (but unlikely) that more than one of these bits will be asserted simultaneously. The asserted bits will provide the reason for the most recent chip reset.

5.19   Programming Restrictions

5.19.1   Cancellation Penalties

5.19.1.1   Branch Penalties

When the instruction pipe executes any type of branch, the corresponding thread will not be able to execute instructions for some number of otherwise schedulable instruction slots immediately following such an operation. The number of potentially lost instruction slots varies with the type of branch and the frequency with which the affected thread is scheduled, as shown in Table 5-15.

Note that the penalty is reduced, and eventually eliminated entirely, as the scheduling frequency of the affected thread decreases.

Table 5-15   Branch Penalty as a Function of Scheduling Frequency

Type of Branch	Scheduling Frequency							
	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8
Correctly predicted taken JMP	3	1	1	0	0	0	0	0
Correctly predicted not taken JMP	0	0	0	0	0	0	0	0
Incorrectly predicted JMP	7	3	2	1	1	1	1	0
CALL	3	1	1	0	0	0	0	0
CALLI	4	2	1	1	0	0	0	0
RET	7	3	2	1	1	1	1	0

5.19.1.2   Instruction Cancellation Hazards

There are a number of cases in which requested read data exists in the pipeline in a modified state, but is not available for use via a data bypass path. In these cases, the instruction requesting the read data is cancelled and retried at a later time. The following discussion specifies all such cases, as well as the size of the window in which such cancellations will take place. The hazard window size is specified in instruction slots, and is a function of scheduling frequency.

"Address calculation" hazards occur when an address or data register is used for address calculation purposes following an explicit definition of the registers via the destination operand. Explicit definitions of address registers via the destination operand of an LEA, PDEC, MOVEAI, CALL, or CALLI instruction while DST_SEL_EN is 0 are excluded, since these utilize a data bypass path that eliminates cancellation hazards.

"CALLI" hazards occur when the source address register operand of the CALLI instruction is used following an explicit definition of the register via the destination operand. As with the previous case, definitions via the LEA, PDEC, MOVEAI, CALL, or CALLI instructions are excluded.

"MAC" hazards occur when an ACC0_HI, ACC0_LO, MAC_RC16, ACC1_HI or ACC1_LO register is read explicitly by a source 1 operand following an implicit definition of the same register via the execution of a DSP or CRCGEN instruction. This hazard also includes cases in which an ACC0_LO or ACC1_LO register is read explicitly by the source 2 operand of a DSP instruction following an implicit definition of the same register via the execution of a DSP or CRCGEN instruction.

Table 5-16 shows the hazard window as a function of scheduling frequency.

Table 5-16   Hazard Window as a Function of Scheduling Frequency

Type of Hazard	Scheduling Frequency							
	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8
Address calculation	4	2	1	1	0	0	0	0
CALLI	4	2	1	1	0	0	0	0
MAC	3	1	1	0	0	0	0	0

5.19.1.3   Hazard Cancellation Penalties

When an instruction is cancelled due to one of the hazards described in Table 5-16, the corresponding

thread will not be able to execute instructions for some number of otherwise schedulable instruction slots immediately following such an operation. The number of potentially lost instruction slots varies according to the type of hazard and the frequency with which the affected thread is scheduled, as shown in Table 5-17.

Note that, for scheduling frequencies which are low enough to avoid a particular hazard entirely, the corresponding penalty is shown to be zero, as there is no hazard in these cases.

Table 5-17 Hazard Cancellation Penalty as a Function of Scheduling Frequency

Type of Hazard	Scheduling Frequency							
	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8
Address calculation	5	3	2	2	0	0	0	0
CALLI	5	3	2	2	0	0	0	0
MAC	6	3	2	0	0	0	0	0

5.19.1.4 Suspend, Breakpoint, Trap, and Block Penalties

Whenever a thread executes a SUSPEND instruction, regardless of whether any further outstanding interrupt conditions exist for that thread, any instructions belonging to that thread which are scheduled in the following seven clocks will be cancelled. Similarly, any instructions of the same thread scheduled in the seven clocks following a BKPT instruction or an instruction which is trapped will also be cancelled. The number of instruction slots cancelled due to a protocol C block on the instruction (I-Block) and data (D-Block) ports is also shown. The number of instruction slots cancelled, as a function of scheduling frequency, is as shown in Table 5-18:

Table 5-18 Suspend, Breakpoint, Trap, and Block Penalties as a Function of Scheduling Frequency

Type of Penalty	Scheduling Frequency							
	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8
SUSPEND	7	3	2	1	1	1	1	0
BKPT	7	3	2	1	1	1	1	0
Trap	7	3	2	1	1	1	1	0
I-Block	3	1	1	0	0	0	0	0
D-Block	7	3	2	1	1	1	1	0

5.19.2 Operations with Delayed Effect

5.19.2.1 Reading Registers Affected by SET and CLR Registers

Certain status registers are primarily updated by hardware, but may also be set and cleared by software that writes to special corresponding SET and CLR registers. When an instruction modifies a status register through this mechanism, the result of that modification becomes visible to subsequent instructions performing an explicit read of the register a number of instruction slots later. The number of subsequent slots in which the new result is not yet visible, as a function of scheduling frequency, is as shown in Table 5-19.

Table 5-19 SET and CLR Affected Register Delay as a Function of Scheduling Frequency

Type of Register	Set and/or Cleared By	Scheduling Frequency							
		1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8
INT_STAT0	INT_SET0, INT_CLR0	2	1	0	0	0	0	0	0
INT_STAT1	INT_SET1, INT_CLR1	2	1	0	0	0	0	0	0
MT_BREAK	MT_BREAK_CLR	2	1	0	0	0	0	0	0
MT_TRAP	MT_TRAP_SET, MT_TRAP_CLR	2	1	0	0	0	0	0	0
MT_ACTIVE	MT_ACTIVE_SET, MT_ACTIVE_CLR	4	2	1	1	0	0	0	0
MT_DBG_ACTIVE	MT_DBG_ACTIVE_SET, MT_DBG_ACTIVE_CLR	4	2	1	1	0	0	0	0
MT_I_BLOCKED	MT_I_BLOCKED_SET, MT_BLOCKED_CLR	4	2	1	1	0	0	0	0
MT_D_BLOCKED	MT_D_BLOCKED_SET, MT_BLOCKED_CLR	4	2	1	1	0	0	0	0

5.19.2.2 Reading Registers Affected by BKPT and SUSPEND Instructions

The BKPT instruction affects the MT BREAK and MT DBG ACTIVE registers, while the SUSPEND instruction affects the MT ACTIVE register. When one of these instructions executes, the effect on the corresponding register(s) becomes visible to subsequent instructions performing an explicit read of those registers a number of clocks later. The number of subsequent clocks in which the new result is not yet visible is as shown in Table 5-20.

Table 5-20 BKPT/SUSPEND Affected Register Delay

Instruction	Affected Register	Delay (clocks)
BKPT	MT_BREAK	1
SUSPEND	MT_ACTIVE	2
BKPT	MT_DBG_ACTIVE	2

5.19.2.3 Source and Destination Thread Number Selects

When the source or destination thread number or enables are modified in the CSR (either through the SETCSR instruction or through other means), the effect of this operation on address calculations is not seen until six clocks later. This means that any instructions executed in the five clocks immediately following such a CSR update will not be affected. The number of subsequent instruction slots in which the new source and destination select values are not yet visible to address calculation operations, as a function of scheduling frequency, is as shown in Table 5-21.

Note that the new CSR values become visible to instructions reading them explicitly immediately. There is no delay in that case.

Table 5-21 CSR Modification Delay

Instruction Modifying CSR	Scheduling Frequency							
	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8
SETCSR	5	2	1	1	1	0	0	0
Any other	5	2	1	1	1	0	0	0

5.19.2.4 Operations Affecting Thread Schedulability

The states of the MT_ACTIVE and MT_DBG_ACTIVE registers can affect thread schedulability. These registers, in turn, can be affected by explicit writes to their associated SET and CLR registers, as well as by the execution of BKPT and SUSPEND instructions. MT_ACTIVE can also be affected by writes to INT_MASK0 and INT_MASK1. There is a delay from each of these events until the potential effect on thread schedulability becomes effective. The following specifies the number of clocks following each of these operations in which the corresponding effect on thread schedulability will not yet be effective. Additionally, the "Interrupt conditions" case specifies the number of clocks from the assertion of one of the hardware interrupt set signals entering the processor until a corresponding thread's schedulability can be affected.

Table 5-22 Thread Schedulability Delay

Operation	Delay (clocks)
Write to MT_EN, MT_HPRI, MT_HRT, MT_SINGLE_STEP, MT, MIN_DELAY_EN, MT_MIN_DELAY field, MT_ACTIVE_SET, MT_ACTIVE_CLR, MT_DBG_ACTIVE_SET, MT_DBG_ACTIVE_CLR, MT_I_BLOCKED_SET, MT_D_BLOCKED_SET, MT_BLOCKED_CLR, INT_MASK_0, INT_MASK_1, INT_SET0, INT_SET1, INT_CLR0, or INT_CLR1	10
BKPT or SUSPEND instruction	7
Interrupt conditions	3

5.19.2.5 Reading the DSP Overflow Bit

In the three clocks immediately following a DSP instruction, the state of the DSP overflow bit in the CSR resulting from that instruction is not explicitly readable yet. To read the overflow bit from an NRT thread, at least three non-DSP instructions must be executed after the last DSP instruction, before reading the CSR. If this restriction is not followed, undefined data will be returned for the overflow bit.

5.19.3 Explicitly Writing the CSR of a Running Thread

Writing to the CSR of another running thread is not permitted. However, writing explicitly to one's own CSR is permitted. The following code sequence must be used whenever the CSR is to be modified, in this case, in order to prevent corruption of the condition code bits in the CSR:

```
nop        // Wait for implicit CC writes to
flush
nop        // and be readable explicitly

move.4 DST, csr// Save old CSR value

(insert code to modify CSR here, keeping saved
CC values)

move.4 csr, SRC1// Write new CSR value

nop        // Wait for explicit CC write to flush
nop        // and be readable implicitly
nop        // 3rd NOP is needed to cover one
superfluous
           // CC bypass path
```

This code sequence will function correctly, regardless of scheduling frequency. If the intent is to change the values of the condition code bits explicitly, or if it is not required that the state of these bits be maintained, then the first three instructions of this sequence (nop, nop, move.4) may be omitted.

5.19.4 Instruction Pipe Flushing and Thread Quiescence

5.19.4.1 Instruction Pipe Flushing

Sometimes, it is necessary for a thread to flush the instruction pipe of all of its previously-executed instructions. Doing so ensures that subsequently-executed instructions will see the effect of any state directly affected by those instructions being flushed, including those with delayed effect. On the instruction pipeline, a pipe flush is defined to be a sequence of 11 NOP instructions. It is highly recommended that an assembler macro be defined for the pipe flush operation, rather than hard-coding the sequence, as the definition of pipe flush will change with future pipelines.

5.19.4.2 Thread Quiescence

For the instruction pipeline, a thread is defined to be in the quiescent state if none of its instructions is in the pipeline and none of its instructions will enter the pipeline until after the pipeline has been emptied of all instructions currently in it. The following example shows one technique that a thread may use to place another thread into the quiescent state:

```
PIPE_FLUSH  // Flush thread-disabling instructions
PIPE_FLUSH  // Flush final instructions of other thread

(other thread is in quiescent state here)

PIPE_FLUSH  // Flush all instructions from pipe

(insert code to allow other thread to have instructions
scheduled)
```

Note that there are a number of methods that may be utilized to prevent the other thread from having additional instructions scheduled. Two common techniques are to deassert either the other thread's MT_EN or MT_DBG_ACTIVE bit.

5.19.5 Transmit FIFO Occupancy Status

When writing to the transmit fifo, and subsequently reading the occupancy status of the transmit fifo, at least three cycles must exist between the instruction which executes the write and the instruction which executes the

read (i.e., W-INSTR - nop - nop - nop - R- INSTR). Due to the definition and behavior of the transmit fifo occupancy status, if an instruction that executes a read immediately follows an instruction that executes a write, the read will return the status of the fifo before the write has had the opportunity to update the occupancy status. Since the fifo is normally filled when a known amount of space exists, such as when receiving a watermark interrupt, the need to fill and then immediately check the fifo should cause no problem.

5.19.6 Reading Write-Only Registers and Write-Only Fields

Read operations of purely write-only registers will return undefined data. Likewise, read operations of readable registers which contain write-only fields will return undefined data for the write-only fields.

5.19.7 Writing Read-only Registers and Read-only Fields

Write operations to purely read-only registers are disallowed. If an instruction were to write to a read-only register, a subsequent read of that same register could trigger an internal processor bypass path, which would result in the read operation's falsely returning the data that was written by the previous write instruction, rather than the actual register contents. In a protected architecture, this would be unacceptable. However, the IP51xx is not a protected architecture, and the required software restriction is of no consequence to properly written code. It should also be noted that writes to read-only registers will have only the previously mentioned effect, and will not corrupt the actual state of the register. This restriction affects the following processor registers:

INST_CNT
ROSR
CHIP_ID
INT_STAT0
INT_STAT1
MT_ACTIVE
MT_DBG_ACTIVE
MT_BREAK
MT_I_BLOCKED
MT_D_BLOCKED
MT_TRAP

Additionally, some writable registers contain unused, read-only bit fields. Writing to these fields is unavoidable. If software follows the convention of writing to these fields only values equal to the fields' reset values, then the documented reset values will be returned by the hardware for all read operations to these fields. If this convention is not adhered to, then the values returned by these fields are undefined.

5.19.8 Program Memory Access Instructions (IREAD/IWRITE/IERASE)

These instructions existed in the IP3000 family, but have been removed from the IP51xx. For backwards compatibility, IREAD and IWRITE may be emulated by MOVE.4 instructions. Also, to maintain backwards compatibility, the IREAD DATA register has been retained, and the MEM BUSY bit in the ROSR has been hard-coded to 0 (not busy).

5.19.9 Multiple Modifications of the Same Address Register in One Instruction

A single instruction can generate up to three possibly conflicting modifications of the same address register. These three modifications could originate from an auto-incrementing source addressing mode, an auto-incrementing destination addressing mode, and the destination itself (targeting the same address register). Regardless of any conflict, the address register values actually used in the source and destination address calculations will be as expected. However, in the case of a conflict, a simple priority scheme is used to determine which value is written back to the register itself, as follows:

Priority	Value Written Back to Address Register
1	Destination result
2	Auto-incremented address register used for destination address
3	Auto-incremented address register used for source address

5.19.10 Reading Instruction Counters

The thread-specific instruction counters are updated as instructions complete in the write-back stage. Since read operations are issued much earlier in the pipe, this means that if a thread reads its own instruction counter, it will be reading slightly stale data. Specifically, any instructions executed in the three clocks preceding the read operation, as well as the read operation itself, will not be represented in the returned count value.

5.19.11 Reading INT_STAT and MT_ACTIVE Following Interrupts

When an interrupt occurs, the appropriate INT_STAT bit is set. Simultaneously, the appropriate threads (if any) are

activated. While the new state of INT_STAT becomes visible to software in the clock in which the interrupt occurred, the new state of MT_ACTIVE does not become visible to software until one clock later.

5.19.12 Reading and Writing the PCs

A thread's Program Counter (PC) must not be written until the corresponding thread is disabled (its MT_EN bit is cleared) and clear of the instruction pipe. After a thread's PC has been updated through software, the PC should not be read by software during the following four clocks. Undefined data will be returned during this window.

5.19.13 LEA and PDEC Instructions

LEA and PDEC instructions are handled in two different ways, depending on the destination. If the destination is an address register and the DST_SEL_EN field in the CSR is 0, then the address register being modified may be used immediately in the following instruction for address calculation purposes. If the destination is a data register, or if the destination is an address register and DST_SEL_EN is 1 (even if DST_SEL points to the current thread), a subsequent instruction which uses the destination too soon for address calculation purposes will be cancelled. These cases fall under the "address calculation" hazard category in Instruction Cancellation Hazards (Section 5.19.1.2).

5.19.14 Writing Another Running Thread's Address or Data Registers

In the IP3000 family, writing to a non-quiescent thread's address or data registers is in violation of the IP3000 architecture specification. In the IP51xx, a thread may write to another running thread's address or data registers by setting the DST_SEL field in its CSR appropriately. However, due to possible unresolvable data dependencies in the pipeline between such an operation and the use of these registers by the affected thread in its address calculations, instructions of the affected thread may be cancelled. The functional operation of the affected thread is guaranteed to be correct. However, its determinism may be affected. Note that this also applies to write operations which are executed speculatively (i.e. writes in the instructions following a conditional branch instruction).

5.19.15 Operands Affected by Source Select

The settings of the SRC_SEL and SRC_SEL_EN fields in the CSR are used to override the thread number in thread-specific direct space addresses for all general source 1 operands. This applies to the following instruction

formats: 1b, 1d, 2, 3, 4a, 4b, 5, 10a, and 10b. This is a complete list of affected source operand addresses.

Examples of source addresses not affected by source select include: the source 2 operand, the source 3 operand, address and data registers used by source 1 and destination address calculations, and the source address register used by the CALLI instruction.

5.19.16 Operands Affected by Destination Select

The settings of the DST_SEL and DST_SEL_EN fields in the CSR are used to override the thread number in thread-specific direct space addresses for all explicit general destination and restricted (data register only) destination operands, with the exception of the general destination operand utilized by the SETCSR instruction. This applies to the following instruction formats: 1c, 1d, 2, 3, 4a, 4b, and 6. This is a complete list of affected destination operand addresses.

Examples of destination addresses not affected by destination select include: all implicit destinations (i.e. DSP and CRCGEN instructions), address registers modified by auto-incrementing addressing modes, and restricted address register-only destination operands utilized by CALL, MOVEAI, and CALLI instructions.

5.19.17 Operands Covered by the DCAPT Write Address Trap

The DCAPT write address trap applies to all explicit destination operands in the direct or indirect address spaces, with the following exceptions:

• The destination of MOVEAI
• The destination of LEA and PDEC, when it is an address register and DST SEL EN is 0
• The destination of CALL
• The destination of CALLI

5.19.18 Writing to Another Thread's Registers

A thread may write to another thread's thread-specific registers only if that thread is quiescent. One exception to this rule is the Program Counter (PC). The PC may only be written to when the corresponding thread is disabled (MT_EN is 0) and clear of the instruction pipe (a PIPE_FLUSH should be performed).

5.19.19 Single-Stepping

It is not possible to single-step an instruction that sets the MT_DBG_ACTIVE bit of its own thread.

5.19.20 Writing the MT_TRAP_CAUSE Register

This thread-specific register should be written to by software only when the corresponding thread is quiescent.

5.19.21 Serror Traps

If an instruction which addresses the same peripheral with the source 1 and destination operands encounters any synchronous error (serror), both source 1 and destination serror traps will be signalled. In other words, for these instructions, it is not possible to distinguish source 1 serrors from destination serrors. If, however, the source 1 and destination operands do not address the same peripheral, then source 1 and destination serrors will be distinguishable.

5.19.22 Initialization of Address and Data Registers

The IP51xx processor does not initialize its address and data registers on hardware reset (unlike the IP3000 processor). These registers should be initialized by software. Not doing so may result in speculative read accesses to random addresses. While any such accesses will be cancelled by the processor, they may result in unwanted side-effects in certain peripherals (for example, an unnecessary speculative cache line fill).

5.19.23 Performing a Hard Reset through the Debug Port

While the debug port implements a chip reset command, this command may be ineffective, if the on-chip PLL driving the processor clock has been disabled. In such a case, the debug dongle will have to reset the IP51xx by causing the external reset input signal to be cycled. If it is desired for the processor to be halted after coming out of reset, the external host will need to wait after issuing a hard reset command, before halting the processor. This is necessary, because a hard reset will also reset the debug port logic, which will be unable to register a halt command until after reset is deasserted.

5.19.24 Performing a Soft Reset of Only the External Memory Subsystem

It is possible to apply a soft reset to the entire external memory subsystem. This will entail resetting both the Flash and DDR SDRAM controllers, as the following sequence shows:

1. Stop all threads that access external memory and ensure that the pipeline is quiescent of these instructions.
2. Flush and invalidate the Data Cache by index.
3. Invalidate the Instruction Cache by index.
4. Disable the DDR controller.
   (a) Disable DDR auto refreshes and wait for any in-flight operations to complete.
   (b) Reset the DDR controller via the corresponding I/O Function Reset field.
5. Disable the Flash controller.
   (a) Lock out any in-flight Flash accesses.
   (b) Wait for any pending operations to complete.
   (c) Reset the Flash controller via the corresponding I/O Function Reset field.
6. Deassert reset to the DDR controller and reapply DDR configuration.
7. Deassert reset to the Flash controller and reapply Flash configuration.
8. Restart relevant threads.

The Cache subsystems can be reset along with the controllers, but the above sequence doesn't do this.

5.19.25 Sharing Code and Data within the Same On-Chip Memory Bank

Atomicity of data port dual operand memory accesses to a code bank is not supported by the hardware. Although the atomic properties of such transaction are intrinsic when considering other accesses within the same thread, this is not the case for accesses from other threads. Therefore, read-modify-write operations to a code bank which are initiated on the data port between different threads are not guaranteed to complete in an easily deterministic order.

5.19.26 Writing Self-Modifying Code for the On-Chip Memory

When a non-speculative code bank request is initiated and acknowledged on the data port, the write request is posted into a Code Request Buffer. The request remains pending until the instruction port is able to locate the appropriate cycle to service the write. While the posted write request is pending, any instruction port reads from the same address will return data last written as a

response (rather than the pending write data). That is, the instruction port will not block any instruction fetches to the address of the posted write. Therefore, software that implements self-modifying code will have to apply the appropriate programming procedure in order to guarantee that the instruction fetch of such code occurs after all posted Code Request Buffer writes are completed. Therefore, a thread that is modifying code needs to perform a pipe flush in order to guarantee correct execution of modified code. Any pending code writes will be completed after the pipe flush.

5.19.27 Dynamic Modification of the On-Chip Memory Bank Mask Register

The On-Chip Memory Bank Mask register which is located in the on-chip peripheral control block specifies whether a bank is to be used for instruction or data. The behavior of the On-Chip Memory is undefined if the bank mask register is changed while executing code or accessing data from an affected bank. The behavior is also undefined if the mask is modified while pending code accesses to the affected bank exist in the Code Request Buffer. Software must apply the appropriate procedure in order to guarantee that none of the above conditions are true while modifying the mask register. Any modification of the mask register is guaranteed to be in effect after a pipe flush.

5.20   Programming Errata

Table 5-23 presents known IP51xx problems and their workarounds.

Table 5-23 IP51xx Problems and Workarounds

1	TWR programming with TDAL for DDR1
Problem	The issue is that a timing parameter called "tdal" has been introduced for some memory parts and is active when using Auto-Precharge Mode. For certain DDR1 parts, if TDAL is used, then the TWR timing parameter must be adjusted.
Workaround	Program Twr = Tdal - Trp for DDR1 devices which utilize TDAL timing parameter.
2	Port A: 250 MHz clock divider peripheral on pin PA5 cannot be configured when Function 1 is selected.
Problem	When Port A, Function 1 is selected, the 250 MHz clock divider peripheral output cannot be controlled. The Port A Function Control 1 register field that configures the divide value is also assigned to the Flash Controller FC Instruction field. A clock output will still be available on pin PA5, but should not be used, as it cannot be programmed.
Workaround	None
3	When an instruction reads from locations in an I/O blocking region that has read side-effect (reading from the location alters the state of the I/O function) — for example, FIFOs in the I/O blocking region as in the USB controller — it may return an incorrect result.
Problem	When the processor retries a read transaction to a blocking region, and the retry is canceled later, the IOPCS restores its states back to the states before the retry transaction was received. However, under the condition when the retry is not followed by another request to the blocking region within the protocol C cancellation window, the IOPCS incorrectly restores the retry transaction address back to the previous transaction address. The end effect is that, when the transaction is retried again, the IOPCS will treat it as a new transaction, and pass the retry onto the I/O function. Therefore, the I/O function can receive multiple requests for the same processor instruction. If the target read address has read side-effect, the states in the I/O function will be unintentionally updated.

Workaround	When an instruction reads from locations in an I/O blocking region that has read side-effect, it must not use cache/DDR as its destination. When reading from the same address in the blocking region consecutively, only the first read instruction to that address cannot use cache/DDR destinations. The subsequent read instructions can safely target cache/DDR as the destination. For example, the following code will execute correctly:
	
	``
	    ; a0 points to an IO blocking region address
	    ; with read side effect
	    ; a1 points to DDR address
	    ; move the first data to a data register
	    move.4 d0, (a0)
	    ; move data from the register to DDR
	    move.4 (a1)4++, d0
	    1:
	    ; move directly to DDR
	    move.4 (a1)4++, (a0)
	    .. ; instructions that do not read/write IO blocking
	    .. ; region addresses
	    jmpcc 1b
	``
	
	The performance impact of this problem is believed to be minimal. The locations in the I/O blocking region that have read side-effect are either status registers or FIFOs. Data in status registers are temporary. Software never needs to move them into cache before processing them. For FIFO data, only one additional instruction is needed for moving a block of data.

4	Access to an unimplemented I/O NBR space may not trigger a trap.

Table 5-23 IP51xx Problems and Workarounds (continued)

Problem	The I/O subsystem is designed to cause a trap when a request is targeting at an unimplemented NBR address. However, the part of the unimplemented address space, address between offset 0x80 to 0x800, does not trigger a serror. If a thread issues a validated request to the above address range, no serror nor ack will be asserted for the request. The effect is that the thread will be blocked indefinitely. Speculative instructions that are canceled later do not cause this problem.
Workaround	None
5	High-Speed USB Controller: EP1-5 packet loading may fail when a packet is in the EP0 FIFO
Problem	If a packet has been loaded into the EP0 FIFO and then one of the endpoints EP1-5 loads a packet into its FIFO, an incorrect RAM write address is used, causing data corruption. The problem is seen if the packet size to be loaded into the EP1-5 FIFO is not a multiple of 4 bytes, and if software loads the EP1-5 FIFO before the "transmit confirmation" interrupt occurs.
Workaround	When sending data on endpoint 0, software must wait for the "transmit confirmation" interrupt before loading any other endpoint's FIFO.
6	High-Speed USB Controller: Peripheral mode: The DataEnd bit is not cleared after incomplete status stage and receipt of a new SETUP packet.

Problem	The DataEnd bit is not cleared after an incomplete status stage and when the core receives a new SETUP packet. This will cause a STALL response in a subsequent data stage. The following demonstrates the problem:
	    1. SETUP-ACK
	    2. OUT DATA(RX)-ACK(Set DataEnd)
	    3. IN Status Stage - No response from Host
	    4. SETUP-ACK

Workaround	Software should clear DataEnd on receipt of IRQ with SentStall bit set.
7	High-Speed USB Controller: Split transaction data incorrectly concatenated in RAM
Problem	When acting as a high speed host, and receiving multiple packets of a ISO IN split transaction, to the same endpoint, of a size other than modulo 4, the RAM controller may incorrectly concatenate the data in RAM, causing corruption. Isochronous data from a full-speed device behind a hub may be corrupted.
Workaround	None
8	High-Speed USB Controller: Data corruption when AutoSet is enabled and Tx- PktRdy is set on different EPs at the same time, while USB is accessing the RAM.
Problem	When multiple Endpoints are used, and if the CPU sets the TxPktRdy just after another Endpoint (AutoSet is enabled) completes, and the FIFO load and USB are accessing the FIFO simultaneously, the transferred data will be corrupt.
Workaround	Do not use AutoSet.
9	SPI slave tri-state mis-wired

Problem	The SPI (Serdes) slave tri-state control signal is incorrectly wired. The tri-state control should be wired to the SPI Data-Out signal (TXP / bit 6), but is instead wired to another Serdes port signal (TXPE / bit 7). This bug affects both Serdes ports on the IP51xx. The effect of this bug is:
	    1. When using SPI slave, bit-7 (TXPE) of the port is usable only as a GPIO input.
	    2. When using SPI slave, the IP51xx cannot be used in a multi-drop SPI environment.

Workaround	When the Serdes is configured as an SPI slave, Serdes/SPI can be used only in a point-to-point environment (as opposed to a multi-drop environment). Additionally, when the Serdes is configured for use as a SPI slave, bit 7 of the Serdes port can be used only as a GPIO input and not as a GPIO output.

Table 5-23 IP51xx Problems and Workarounds (continued)

10	PCI TAR_WRITE_OP_DONE interrupt doesn't work.
Problem	The PCI TAR_WRITE_OP_DONE interrupt fires at the beginning instead of the end of a target write transaction.
Workaround	For the IP51xx to handle write transactions as a PCI target efficiently and with good performance, a PCI driver would need both the RX_FIFO_WATERMARK interrupt and the TAR_WRITE_OP_DONE interrupt. Since the latter interrupt is not useful in its current implementation, and since PCI burst lengths are variable, the driver will need to poll the RX_FIFO_LEVEL indicator to determine when it reaches zero. Extra care must be taken when polling to ensure that no data is 'in flight' through the PCI core logic even though the RX_FIFO_LEVEL reports zero. This is especially true for small PCI bursts. To be absolutely certain, additional polling may be necessary once zero is initially detected.

5.21   Writing Assembly Code

This section summarizes some of the most useful features of *gas*, the GNU assembler. It also presents the extensions made to *gas* for the IP5000 family, `ip5k-elf-gas`. Complete documentation of *gas* is available at http://www.gnu.org/manual/.

5.21.1   Comments, Constants, and Symbols

5.21.1.1   Comments

Comments can occur after a semicolon:

```
move.4 D1, #3 ; this is a comment
;this whole line is a comment
```

Comments can also be enclosed in C-style comment delimiters, such as:

```
/* this is a comment */
```

and:

```
/* this is
a multi-line
comment */
```

As with C, comments may not be nested.

5.21.1.2   Constants

Constants may be character constants, string constants, or numeric constants.

Character Constants

A *character constant* is a single character enclosed in single quotes, such as **'f'**, which is a byte with the value 102 (decimal) corresponding to its ASCII code.

To use a character with special meaning, or a character outside of the standard ASCII printing characters, a backslash (****) is used to indicate a representation for the character, as shown in Table 5-24.

Table 5-24 Assembly Special Characters

Representation	Value	Character
\b	0x08	Backspace (control-H).
\f	0x0C	Form Feed (control-L).
\n	0x0A	New line (control-J).
\r	0x0D	Carriage return (control-M).
\t	0x09	Horizontal tab (control-I).
\xNN	0xNN	NN is the ASCII code (in hexadecimal) for the character — for example, \x09 is equivalent to \t
\\	0x5C	Backslash (\).
\"	0x22	Double Quote (").

String Constants

A *string constant* consists of one or more characters enclosed in double quotes, such as **"Ubicom"**.

Numeric Constants

A *numeric constant* is an integer. By default, it is interpreted as a decimal number. To express it in binary, prefix the value with **0b**; for example **0b01101001**. To express it in hexadecimal, prefix the value with **0x**, as in **0x9F**. Hexadecimal digits may be either upper or lower case.

Table 5-25 lists the notation syntax for constants.

Table 5-25 Notation Syntax

Notation	Example
dec	65
bin	0b01000001
hex	0x41 or 0X41
octal	O101
ascii	'A'

5.21.1.3   Symbols

A *symbol* is a name for any nameable object, such as labels and constants. A symbol consists of one or more characters from the set of letters, digits, period (**.**), and underscore (_). A symbol may not begin with a digit.

Symbols are case-sensitive, so **abc** is distinct from **aBc**. Symbols may not be reserved words (see Table 5-29 for a list of the reserved words).

To make a symbol visible outside the file where it is defined, a **.global** directive is required. For example:

```
.global _gl_symbol
_gl_symbol:
    add.4  D3, D1, D0
    calli  RP, 0(RP)
```

The **.func** directive emits debugging information to denote function name, and is ignored unless the file is assembled with debugging enabled. Only the **--gstabs** debugging option is currently supported.

Label is the entry point of the function. If the label is omitted, then the function name prepended with the "leading char" is used. The "leading char" is usually "_" or nothing, depending on the target. All functions are currently defined to have **void** return type. The function section must be terminated with the **.endfunc** directive. For example:

```
.global reset_vector
.func reset_vector,reset_vector
reset_vector:
; Emergency space
    .rept   16
    .word   0xFFFFFFFF
    .endr
; Null check (debug)
    cmpi    SP, #0
    jmpne.f _null_function_pointer
    .endfunc
```

5.21.2   Directives

ip5k-elf-gas has four directives in addition to the standard list:

.word – four bytes.
.long – four bytes.
.half – two bytes.
.short – two bytes.

Please consult the *gas* documentation for details on other directives.

5.21.3   Operators

Operators are arithmetic functions, like **+** or **%**. *Prefix operators* are followed by an argument (see Section 5.21.3.1). *Infix operators* appear between their arguments (see Section 5.21.3.2). Operators may be preceded and/or followed by white space.

5.21.3.1   Prefix Operators

Prefix operators are shown in Table 5-26. **ip5k-elf-gas** has two prefix operators, each taking one absolute argument.

Table 5-26 Prefix Operators

Prefix	Description
–	Complement negation.
~	Bitwise not (complement).

5.21.3.2   Infix Operators

Infix operators take two arguments, one on either side of the operator. Operators have precedence, but operations with equal precedence are performed left to right. Apart from **+** or **–**, both arguments must be absolute, and the result is absolute.

Table 5-27 Infix Operators

Highest Precedence:	
*	Multiplication.
/	Division. Truncation is the same as the / operator of C.
%	Remainder.
<	Less.
<<	Shift Left. Same as the << operator of C.
>	Greater.
>>	Shift Right. Same as the >> operator C.
Intermediate Precedence:	
|	Bitwise Inclusive OR.
&	Bitwise AND.
^	Bitwise Exclusive OR.
!	Bitwise OR NOT.
Lowest Precedence:	
+	Addition. If either argument is absolute, the result has the section of the other argument. Arguments from different sections may not add together.
–	Subtraction. Only meaningful to add or subtract the *offsets* in an address; there can only be a defined section in one of the two arguments.

5.21.4   Assembly to C Calling Conventions

In order to call a C function called *main* from assembly routine. The following instructions must be executed:

```
moveai  A2, #%hi(_main)
lea.4   A2, %lo(_main)(A2)
calli   RP, 0(A2)
```

5.21.5   Operand Qualifiers

lp5k-elf-gas extends the GNU assembler to qualify operands with special syntax. There are a lot of occurrences of operand qualifiers throughout the assembly source files in the Ubicom SDK. Operand qualifiers are interpreted as shown in Table 5-28.

Table 5-28 Operand Qualifiers

Qualifier	Addressing Mode	Description	Example
%hi	Immediate for **moveai** instruction	Extracts bits 30:7 of the operand	`moveai An, #%hi(symbol_name)`
	Immediate for **movei** instruction	Extracts bits 31:16 of the operand	`movei Dn, #%hi(symbol_name)`
%lo	Register Indirect with 7-bit unsigned offset	Extracts bits 6:0 of the operand	`move.4 Dn, %lo(symbol_name)(An)`
	Indirect for **calli** instruction	Extracts bits 6:0 of the operand	`calli An, %lo(symbol_name)(An)`
	Immediate for **movei** instruction	Extracts bits 15:0 of the operand	`movei Dn, #%lo(symbol_name)`
%lo18	Indirect for **calli** instruction	Extracts bits 17:2 of the operand	`calli An, %lo18(symbol_name)(An)`
%bit	Immediate	Log2 (operand). Converts the bit mask position into a bit number that can be used by **bset** or **bclr** instruction.	`bset Dn, Dn, #%bit(0x00000100)`

Note: New operand qualifiers might be added in future releases of the SDK.

5.21.6 IP5K-Specific Reserved Words

Table 5-29 shows all of the instruction mnemonic names and special-purpose register names. Both the uppercase and lowercase versions of these names are reserved words. Reserved words may not be used as symbolic names.

Table 5-29 CPU Reserved Words

add.2	bset	lea.2	madd	muls	shftd
add.4	btst	lea.4	msub	mulu	shmrg.1
addc	call	lsl.2	msuf	nop	shmrg.2
and.2	calli	lsl.4	merge	not.2	sub.2
and.4	cmpi	lsr.2	move.1	not.4	sub.4
asr.2	crcgen	lsr.4	move.2	or.2	subc
asr.4	ext.1	macf	move.4	or.4	suspend
bclr	ext.2	macs	moveai	pdec	xor.2
bfextu	jmpcc.c.p	macu	movei	ret	xor.4
bkpt	lea.1	macus	mulf	setcsr	

6.0   Peripherals

6.1   Overview

The IP51xx supports a number of I/O ports, each of which can be configured as general purpose I/O or assigned to a specific function. The available hardware functions include support and control functions, such as clock generation and interrupts, as well as functions for allowing connections to specific on-chip I/O controller/accelerator units.

6.2   Shared Port Architecture

All ports share a common base architecture. Depending on the specific requirements of the port, each port can be composed of:

• A register set to control and monitor the port functions.
• Data FIFOs for efficient movement of transmit and receive data.
• One or more independently operating function blocks.
• Muxes that control function selection.

All ports share these common features:

• Configurable to operate as GPIO only.
• FIFO interface for data, independent of protocol. Alignment and packing of data within the FIFO word is function specific. Actual size is port specific.
• External clock sources available.

6.2.1   Port Registers

Port registers are used for function selection, setting function parameters, function control, and reporting function and port status. Port specific register definitions are provided beginning in Section 7.6.

6.2.2   Interrupts

Each port has four registers that participate in interrupt control for that port:

• Interrupt Status Register
• Interrupt Mask Register
• Interrupt Set Register
• Interrupt Clear Register

6.2.3   FIFO Management

The FIFOs are initialized by asserting one of the set bits TX FIFO Reset or RX FIFO Reset.

The TX Underflow and RX Overflow Interrupts are provided to inform software of attempts to transmit when the transmit FIFO is empty or to receive when the receive FIFO is full.

At any time, software can determine how full the transmit and receive FIFOs are by examining the FIFO Level register.

To help avoid overflow or underflow conditions, the concept of FIFO watermarks is implemented. Software uses control registers to set the FIFO Watermark level. Hardware uses the FIFO Watermark interrupt to inform software when a watermark level has been reached.

If a transmit FIFO Watermark Interrupt, or a receive FIFO Watermark Interrupt has occurred, and the appropriate action has not been taken to address the interrupt (i.e., fill the transmit FIFO or empty the receive FIFO) the interrupt will persist against any attempt by software to clear the interrupt. (The mask bit must be set to cause an interrupt.)

The transmit FIFO watermark interrupt will clear automatically when the FIFO occupancy level is higher than the transmit FIFO watermark value. The receive FIFO watermark interrupt will clear automatically when the FIFO occupancy level is lower than the receive FIFO watermark value.

6.2.3.1   Receive FIFO Selection, Behavior and Restrictions

A second receive FIFO is installed in those I/O ports that have a function needing a second FIFO. Where only one receive FIFO is installed in an I/O port, the receive FIFO is installed as FIFO 0.

Access to a specific receive FIFO is controlled through the Receive FIFO Select bit (Function Select[3]). Setting this bit to 0 causes all accesses to the receive FIFO (reset, status, data) to reference receive FIFO 0. Setting this bit to 1 causes receive FIFO accesses to reference receive FIFO 1.

An interrupt that was set due to the action of one FIFO will remain set after the Receive FIFO Select bit is changed to select the other FIFO, even if the newly selected FIFO is not asserting this interrupt. If a FIFO is filled beyond the level set by the watermark trigger level and that FIFO is not currently selected, an interrupt will not be asserted until that FIFO has been selected through the Receive FIFO Select bit. When a receive FIFO overflows or a transmit FIFO underflows, all other FIFO status information is undefined, and software must reset the FIFO.

Table 6-1 Port Function Summary

Port	Port Width	TX FIFO Size	RX FIFO Size	Function 0	Function 1	Function 2	Function 3
A	8	8 x 32	N/A	GPIO	Flash / INT / Clock	GPIO / INT / Clock	GPIO / INT
B	20	32 x 36	32 x 36	GPIO	PCI	---	---
C	32	N/A	N/A	GPIO	PCI (I/O only)	Reserved	---
D	12	16 x 32	16 x 32	GPIO	Serdes (240 MHz)	Reserved	---
E	8	16 x 32	2 - 16 x 32	GPIO	Serdes (250 MHz)	Reserved	MII / RMII
F	16	16 x 32	2 - 32 x 32	GPIO	GMAC (MII / RMII / RGMII)	---	---
G	32	N/A	N/A	GPIO	DDR SDRAM	---	---
H	10	N/A	N/A	GPIO	DDR SDRAM	---	---
I	12	N/A	N/A	GPIO	N/A	Reserved	MII (Port E Extension)
USB Port	2	N/A	N/A	N/A	High-Speed USB	N/A	N/A

6.3 External Flash Controller (FC)

The External Flash Controller (FC) manages the interface between the IP51xx and an external serial flash device on Port A. Table 6-2 shows how the external flash interface signals are assigned to the port pins, using a simple SPI interface. Section 7.7.1 gives a detailed description of the port registers used for the FC.

Table 6-2 External Flash Interface Signals

Signal	Port A Pin	Port A I/O	Description
SI	0	I	Serial Data Input
SO	1	O	Serial Data Output
SCK	2	O	SPI Clock
CE_N	3	O	Chip Select

The FC provides shared access to the external serial flash device from the caches and the processor (through the port-registers).

The IP51xx supports the following families of external serial flash devices:

- Atmel 25 series devices
- SST 25 series devices
- AMD 25 series devices
- ST 25 series devices

All of these flash devices are Read compatible for basic read operations. For programming, each device vendor uses its own variant command set with its own peculiarities. It is left to software to appropriately program the specific device through the port-register interface.

The following are restrictions imposed by the design of this flash controller:

- Flash Device Chaining: No chaining of external serial flash devices on the SPI bus is supported.
- Maximum Device Size: The address size for the supported devices is 24 bits. As such, the maximum size device that can be supported using this addressing scheme is 16 Mbytes.

6.3.1 Cache Read Interface

Data accesses from the instruction and data caches are automatically triggered by cache misses to the flash address space. The caches request a cache line, and the FC constructs the data transaction to the external device. The FC collects a full 32-byte cache line from the external device and returns it to the requesting cache. Arbitration between the instruction and data caches for the FC is round-robin.

6.3.2 Port-Register Read / Write / Erase Interface

The port-registers (see Section 7.7.1) provide an interface for the processor to construct arbitrary SPI transactions, to be passed along to the SPI flash device by the FC.

6.3.3 Arbitration

The FC is responsible for arbitration between cache and processor transactions to the external flash device. Priority is always given to processor requests, unless an outstanding request from a cache is being processed at the time of the processor request. In that case, the processor will get the external SPI bus as soon as the cache transaction completes. Through the port-registers, the processor may lock-out the caches from accessing the external flash device. This is desirable during flash erase or write sequences of transactions.

6.4 External DDR SDRAM Controller

The External DDR SDRAM controller manages the interface between the IP51xx and an external DDR SDRAM data memory on Ports G and H.

Table 6-3 shows how the external DDR SDRAM interface signals are assigned to the port pins. Signals ending with "_N" are active low.

Table 6-3   DDR SDRAM Port Signals

Signal	Port Pin	I/O	Description
ADDR[13]	G0	O	
ADDR[12]	G1	O	
ADDR[11]	G2	O	
ADDR[09]	G3	O	
ADDR[08]	G4	O	
ADDR[07]	G5	O	
ADDR[06]	G6	O	
ADDR[05]	G7	O	Address pins for read / write
ADDR[04]	G8	O	
ADDR[03]	G9	O	
ADDR[02]	G10	O	
ADDR[01]	G11	O	
ADDR[00]	G12	O	
ADDR[10]	G13	O	
CS_N	G14	O	Chip Select
CAS_N	G15	O	Column Address Strobe
BA2	G16	O	
BA1	G17	O	Bank Select Address
BA0	G18	O	
RAS_N	G19	O	Row Address Strobe
WE_N	G20	O	Write Enable
CKE	G21	O	Clock Enable
DM / LDM	G22	O	Data Mask / Lower Data Mask
DQ[7]	G23	I/O	
DQ[6]	G24	I/O	Read / write data
DQ[5]	G25	I/O	
DQ[4]	G26	I/O	
LDQS	G27	I/O	Lower Data Strobe
DQ[3]	G28	I/O	
DQ[2]	G29	I/O	Read / write data
DQ[1]	G30	I/O	
DQ[0]	G31	I/O	
UDM	H0	O	Upper Data Mask
DQ[15]	H1	I/O	
DQ[14]	H2	I/O	Read / write data
DQ[13]	H3	I/O	
DQ[12]	H4	I/O	

Table 6-3   DDR SDRAM Port Signals (continued)

Signal	Port Pin	I/O	Description
UDQS	H5	I/O	Upper Data Strobe
DQ[11]	H6	I/O	
DQ[10]	H7	I/O	Read / write data
DQ[09]	H8	I/O	
DQ[08]	H9	I/O	

In addition to the pins on Ports G and H, there are several pins that are dedicated to the DDR SDRAM (not subject to Function Select). These are shown in Table 6-4.

Table 6-4   Dedicated DDR SDRAM Interface Signals

Signal	I/O	Description
DDR_CAL	I/O	Calibrator pin for input and output impedance calibration
DDR_CLK	O	Clock output to DDR SDRAM
DDR_CLKN	O	Clock output to DDR SDRAM (inverted)
DDR_CLKFB	I	Clock input from DDR SDRAM
DDR_CLKFBN	I	Clock input from DDR SDRAM (inverted)
DDR_ODT	O	Output to DDR SDRAM

Registers used for configuration of the DDR SDRAM controller are located in both the Port G non-blocking region and the Port G blocking region.

In order to use an External DDR SDRAM with a 16-bit data bus, Function 1 must be selected on both Port G and Port H. For a device with an 8-bit data bus, only Port G Function 1 need be selected.

Section 7.12.1 gives a detailed description of the port registers used for the DDR SDRAM Controller.

6.4.1 DDR SDRAM Controller Features

The DDR SDRAM Controller provides these features:

- Supports DDR SDRAM sizes from 128 Mbits to 1 Gbit.
- Supports DDR1 and DDR2 device types.
- Supports DDR devices with 8 (x8) or 16 (x16) bit data buses.
- Supports DDR clock frequencies from 120 MHz to 200 MHz.
- Does not support Self Refresh mode.

6.5   Serializer/Deserializer (Serdes)

The IP51xx has two Serdes units, which support a variety of serial communication protocols, including GPSI, SPI, UART, and USB (only USB Low Speed and Full Speed modes are supported on the Serdes). By performing data serialization / deserialization in hardware, the CPU bandwidth needed to support serial communication is greatly reduced, especially at high baud rates. Providing two units allows easy implementation of protocol conversion or bridging functions between the two high-speed serial interfaces.

One Serdes unit is associated with Port D, and one with Port E. The reference clock for the Port D Serdes is 240 MHz, and the reference clock for the Port E Serdes is 250 MHz. Each Serdes unit uses up to 8 external digital signals shown in Table 6-7. Not all signals are used in all protocol modes. Refer to Table 6-6 for details on signal pin usage in various protocol modes. The mapping of these signals onto the port pins is shown in Table 2-8 and Table 2-9.

Serdes Registers and Interrupts

Section 7.9.1 gives detailed descriptions of the registers and interrupts used by the Serdes units.

6.5.1   Serdes TX/RX Buffers

RX FIFO

Received data is placed into the port's RX FIFO. The Serdes asserts the RXBF interrupt to indicate when new data is available from the FIFO (the interrupt mask bit must be set to cause an interrupt). The RX FIFO Overflow Interrupt indicates when the receive FIFO becomes full during receive, and the RX FIFO Watermark Interrupt indicates when the receive FIFO level meets or exceeds the receive FIFO watermark trigger level.

The TXBUF Register Field

The 16-bit TXBUF field of the Function Control 2 register is for loading data to be transmitted. Asserting TXBUF_VALID in the Interrupt Set register signals to the Serdes that the data in TXBUF is ready to be transmitted. The Serdes asserts the TXBE interrupt to indicate when the data has been transmitted and the register is ready to be loaded with new data (the interrupt mask bit must be set to cause an interrupt).

6.5.2   Serdes Configuration

Software prepares a Serdes unit to receive data by programming the receive shift count register field (RXSCNT) and the clock divider (CLKDIV) appropriately

for the selected protocol. When the number of bits received equals the value of RXSCNT, the received data is loaded into the RX FIFO.

In GPSI or USB mode, when an EOP is detected, the RXCTR register field is loaded with the number of bits actually received (with the exception of the last transfer), the RXEOP interrupt is asserted, and the data bits are loaded into the RX FIFO (the interrupt mask bit must be set to cause an interrupt). For details concerning the last transfer, refer to Section 7.9.1.11.

The TXP and TXM signals correspond to the differential outputs of the USB bus. Other serial protocols require only one output pin, which is TXP by default.

For transmitting, software must specify the number of bits to transmit (in the TXSCNT register field), load the data into the TXBUF register, and assert TXBUF_VALID to signal availability of new data. This data is then transferred to an internal register, from which it is serially shifted out to the transmit logic. The TXBE interrupt is asserted when the data has been transferred from the TXBUF register (the interrupt mask bit must be set to cause an interrupt).

When there is a transmit buffer underrun event (i.e. all of the data has been shifted out from the internal register, but the TXBUF register has not been reloaded), an EOP condition is generated on the TXP and TXM outputs. The TXEOP interrupt is asserted when an underrun event occurs (the interrupt mask bit must be set to cause an interrupt).

For protocols other than USB, the EOP generator is bypassed.

6.5.3   Protocol Modes

Table 6-5 shows the protocols selected by the PRS bits in the MODE field of the Function Control 0 register. The selection of protocol affects which registers and register fields are used, for example the RSYNC field of the Function Control 1 register is only used in the USB mode. The protocol mode also affects the signal usage, as shown in Table 6-6. Pins not used for protocols can be used for general I/O. Table 6-7 provides more information about each signal.

Table 6-5 Protocol Selection

PRS	Mode
0001	Not used
0010	USB Bus
0011	UART
0101	SPI
0110	GPSI

Table 6-6 Serdes Protocol Modes And Signal Usage

Signal	Port D or Port E Pin	Serdes Mode USB *		UART		SPI				GPSI			
						Master		Slave		Master		Slave	
RXD	0	RCV	(I)	RXD	(I)	DI	(I)	DI	(I)	RxD	(I)	RxD	(I)
RXM	1	VM	(I)										
RXP	2	VP	(I)					SS	(I)	RxEN	(I)	RxEN	(I)
CLK	3					SCK	(O)	SCK	(I)			RxCLK	(I)
TXME	4											TxBUSY	(I)
TXM	5	VMO	(O)							TxCLK/ RxCLK	(O)	TxCLK	(I)
TXP	6	VPO	(O)	TXD	(O)	DO	(O)	DO	(O)	TxD	(O)	TxD	(O)
TXPE**	7	OE	(O)							TxEN	(O)	TxEN	(O)

I: Input, O: Output
* USB is available only on Port D.
** TXPE in SPI slave mode is available only as a GPIO input.

Table 6-7 Serdes Signals

Pin	Description
RXD	Serial data for USB, UART, SPI and GPSI modes.
RXM	Negative-side differential input (USB only).
RXP	Positive-side differential input (USB only), Slave Select (for SPI Slave), or data valid (GPSI).
CLK	Serial Clock in SPI or GPSI Slave modes.
TXME	TxBUSY in GPSI mode.
TXM	Negative-side differential output (USB mode), transmit clock (GPSI Slave), or transmit and receive clock (GPSI Master).
TXP	Positive-side differential output (USB mode), or serial data (UART, SPI, and GPSI modes).
TXPE	Output enable for external transceiver (USB), or data valid for GPSI mode.

6.5.4   USB

The USB on the Serdes Port supports only Low Speed and Full Speed modes. This is not to be confused with the dedicated USB port on the IP51xx.

Table 6-8 USB Interface Signal Usage

USB Signal Name	Serdes Signal Name	Port D Pin	Direction	Description
VP	RXP	2	Input	Plus-side differential input
VM	RXM	1	Input	Minus-side differential input
VPO	TXP	6	Output	Plus-side differential output
VMO	TXM	5	Output	Minus-side differential output
OE	TXPE	7	Output	Output enable
RCV	RXD	0	Input	Receive data
Clock	CLK	3	Input	External clock input (optional)
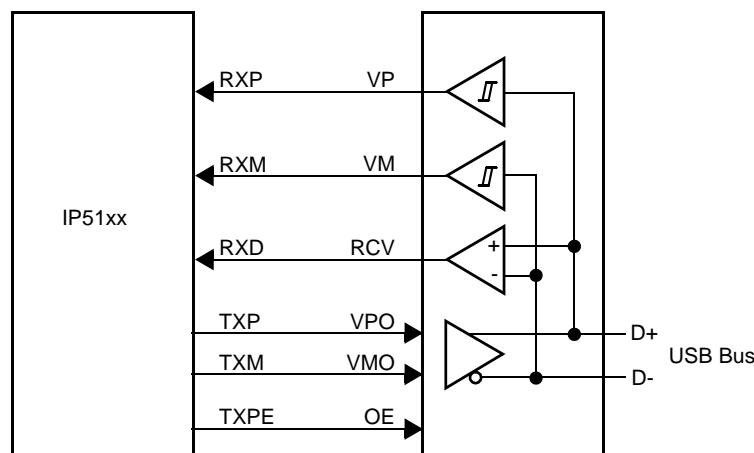
Figure 6-1 USB Interface Example

Compatible USB interface devices include Sipex™ SP5301 and Fairchild™ USB1T20.

The Port D Serdes provides support for USB host and device modes of operation.

Hardware

To set up the Port D Serdes unit for USB mode, the received data output of the USB transceiver should be connected to RXD. The VP and VM pins of the transceiver are connected to the RXP and RXM pins to allow detection of the EOP condition. Figure 6-1 shows the connections required between an external USB transceiver and the IP51xx. Table 6-8 shows the mapping of USB signals to the Serdes pins. For additional hardware configuration information, reference designs are available on the Ubicom technical support portal for

registered development kit users. Please visit the portal for the latest information, or contact Ubicom.

Software

The MODE register field must be programmed with values for the desired USB mode, Full Speed or Low Speed. The serial I/O clock divider CLKDIV also needs to be programmed to generate the appropriate frequency according to the USB submode selection. Table 6-9 shows the serial I/O clock frequencies required for the low and full speed modes of the USB. Since the clock into the Port D Serdes is 240 MHz, the Serdes serial clock can be programmed to 48 MHz for full speed with a divisor of 5 (encoded as 4). A divisor of 40 (encoded as 39) is required for 6 MHz low-speed USB. Table 6-10 shows the submode values for selecting the low- or high-speed modes.

Table 6-9 Required Clock Frequencies for Serial I/O Clock in USB Mode

Protocol	Clock Frequency
USB Full Speed *	48 MHz
USB Low Speed	6 MHz

* On-chip Serdes USB supports maximum data rates up to 12 Mbps.

Table 6-10 Submodes for USB

Name	Description
SUBM1:0	Submode select for USB mode: 01 = Low-speed USB interface 10 = Full-speed USB interface

In USB mode, the Serdes uses two registers, RSYNC and SYNCMASK, to detect the sync pattern marking the beginning of a USB data stream. In order to achieve this, RSYNC must be programmed with 0x80 and SYNCSMASK must be programmed with 0xE0.

Receive behavior is controlled by several register fields. For normal USB operation, USB_SYNC_IGNORE should be 0, REV_POLARITY_EN should be 0, and RXSCNT should be set to the desired number of bits received, usually 8 or 16. BIT_ORDER should be cleared to make sure receive is performed LSB first. Once the Serdes matches the USB SYNC pattern, the receive count is reset to zero and the Serdes receives bits from the line until either the desired count is received or an EOP is encountered, at which point the received data is transferred to the RX FIFO. If more data is coming in, the procedure will be repeated. Software is responsible for reading the data from the RX FIFO fast enough to avoid overflow. When the EOP is received, the Serdes remains idle until the next match of the SYNC pattern.

Transmit behavior is controlled by several register fields. For normal USB operation, LOOP_BACK should be 0, TX_DATA_INV should be 0, and TXSCNT bits should be set to the number of bits to transmit. Transmit is initiated by writing the data to the TXBUF register, then asserting TXBUF_VALID. If the transmit count needs to be changed, it must be changed before setting TXBUF_VALID. For continued transmission, the TXBUF register has to be written before the TXSCNT count is reached. Otherwise, the Serdes automatically inserts the EOP signaling.

While receiving data, the clock/data separation circuit performs NRZI decoding, after which bit unstuffing is performed. This means every bit after a series of six consecutive ones is dropped. On transmit, the Serdes performs bit stuffing, and the clock/data separation circuit NRZI encodes the data.

Note: While configured for USB mode, the Serdes cannot be configured to interrupt on carrier status (RXXCRS).

Software must perform the following functions to implement the USB protocol for a device:

- CRC generation and checking (can be done with the CRCGEN instruction).
- Detecting reset of the device function, which is indicated by 10 milliseconds of a single-ended zero (SE0) condition on the bus.
- Detecting the suspend state, which is indicated by more than 3 milliseconds of idle. Software must make sure that the suspend current of 500 µA will be drawn after 10 milliseconds of bus inactivity.
- Formation of the USB packet by putting the sync, pid, and data into the transmit data registers and setting the proper count.
- Endpoint and device management and other higher level protocol tasks.

Timing Considerations

USB relies on certain timing limitations for error detection and recovery. Response time requirements are specifically harder to meet. The ISR for USB must be carefully structured to satisfy these requirements, and this is possible because of IP51xx's deterministic ISR execution times. The time from the SE0 on bus to the RXEOP indication is about 208 ns. The time from writing to TX data registers and the data put on the bus is about 125 ns. Software tasks like address, error, CRC checking, and determining the endpoint response must be carefully timed and cycle counted to assure that the required timing limitations are satisfied.

6.5.5   UART

For UART operation, two internal divide-by-16 circuits are used. The receive section and the transmit section use two divided-by-16 clocks that potentially can be out of phase. This is due to the nature of the UART bus transfers. The receive logic, based on the 16x bit clock, samples the incoming data for a falling edge. Once the edge is detected, the receive logic counts 8 clock cycles and samples the number of bits specified in the RXSCNT register using the bit clock (which is obtained by dividing the clock source by 16).

Hardware

Figure 6-2 shows an example circuit to connect the Serdes in UART mode. Table 6-11 shows the UART signal to port pin usage.

Software

To set up a Serdes unit for UART mode, select UART mode in the PRS bits of the MODE register field. This causes the data to be clocked in after a valid start bit is detected. Make sure that the polarity selected by the REV_POLARITY_EN and TX_DATA_INV matches the polarity provided by the RS-232 transceiver (most of them are inverted). Make sure the BIT_ORDER is compatible with the data format (RS-232 uses LSB-first bit order). The receiver uses 16X oversampling, so select a serial I/O clock divisor (CLKDIV) that is 16 times the desired baud rate.

To operate in UART mode, depending on the application, either transmit or receive can be performed first. In both cases, the shift count register must be programmed with a bit count that is appropriate for the format. The bit count depends on the number of data bits, stop bits, and parity bits. The start bit is included in the bit count. The receiver does not check for the presence of stop bits. To detect framing errors caused by missing stop bits, increase the receiver's bit count (i.e., the RXSCNT field) and test the trailing bit(s) in software.
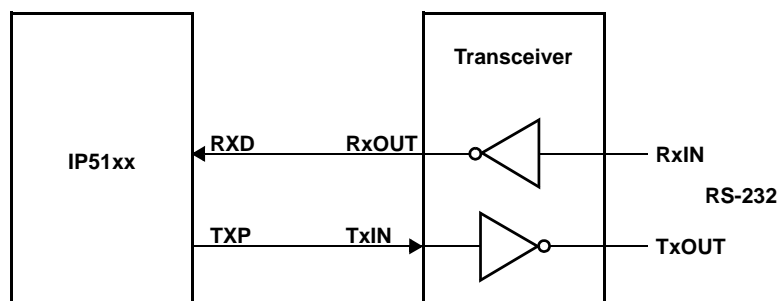
Figure 6-2 UART Interface Example

Table 6-11 UART Interface Signal Usage

UART Signal Name	Serdes Signal Name	Port D or E Pin	Direction	Description
RXD	RXD	0	Input	Receive data
TXD	TXP	6	Output	Transmit data

6.5.6   SPI

Hardware

Figure 6-3 shows example circuits to connect the Serdes in SPI mode. Table 6-13 and Table 6-14 show the SPI signal to port pin usage. Refer to Section 7.9.1 for the Serdes port register interface.

Configuration

The Serdes can be configured for either master or slave mode:

SPI_MASTER_SEL = 1: Master
SPI_MASTER_SEL = 0: Slave

The Serdes SCK idle-level (i.e., when $\overline{SS}$ is deasserted) can be configured by the SUBM field of MODE (refer to Table 6-15).

MODE[3] (CPOL) = 0: idle is low
MODE[3] (CPOL) = 1: idle is high

Finally, the Serdes can be configured for the phase relationship of the DO/DI pins with respect to the SCK edge:

MODE[2] (CPHA) = 0:
   DI will be sampled by this device (the slave) on the first edge (transition).

MODE[2] (CPHA) = 1:
   DI will be sampled by this device (the slave) on the second edge (transition).

Note: The use of the term "edge" in the above paragraphs implies any transition, not a specific type of transition (i.e., rising or falling). Therefore, "first edge" implies a rising edge when CPOL=0, and implies a falling edge when CPOL=1.

In the SPI scheme implemented by Motorola®, which the IP51xx follows, data being emitted on DO and data being sampled on DI always occur on opposing edges of the clock, on either master or slave. Transmitting and sampling on the same edge of the clock is not supported by the Serdes.

CPOL, in conjunction with CPHA, determines which clock edges the Serdes will use to drive and sample data on, as given by Table 6-12.

Table 6-12 Serdes Output and Sample Configuration

CPOL	CPHA	
0	0	drive on falling, sample on rising
0	1	drive on rising, sample on falling
1	0	drive on rising, sample on falling
1	1	drive on falling, sample on rising

When the Serdes is configured as a slave, the state of the DO line when $\overline{SS}$ is deasserted is determined by the value in the RxOUT GPIO register for that pin, which the user can configure.
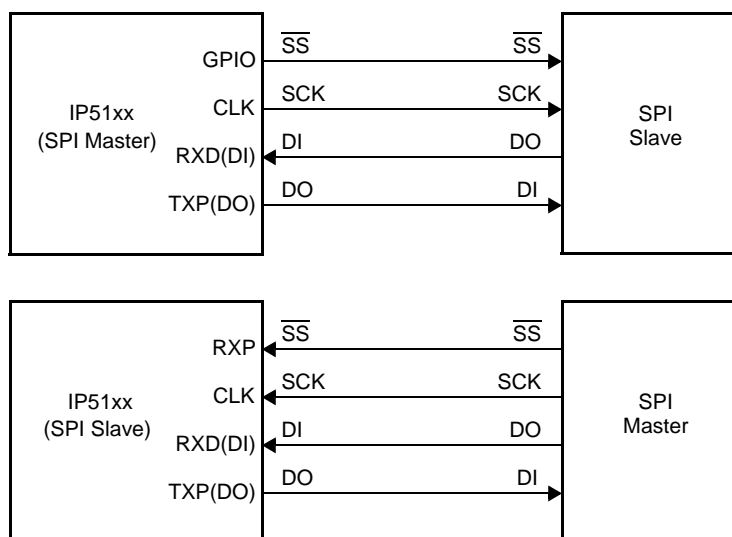
Figure 6-3 SPI Interface Examples

www.ubicom.com

Table 6-13 SPI Master Interface Signal Usage

SPI Device Signal Name	IP51xx SPI Signal Name	Serdes Signal Name	Port D or E Pin	Direction	Description
SCK	SCK	CLK	3	Output	Serial clock output in master mode, input in slave mode
DO	DI	RXD	0	Input	Receive data
DI	DO	TXP	6	Output	Transmit data
SS	SS	GPIO	any other	Output	Slave select pin used in slave mode only (Master select handled by software)

Table 6-14 SPI Slave Signal Usage

SPI Device Signal Name	IP51xx SPI SIgnal Name	Serdes Signal Name	Port D or E Pin	Direction	Description
SCK	SCK	CLK	3	Input	Serial clock output in master mode, input in slave mode
DO	DI	RXD	0	Input	Receive data
DI	DO	TXP	6	Output	Transmit data
SS	SS	RXP	2	Input	Slave select pin used in slave mode only (Master select handled by software)
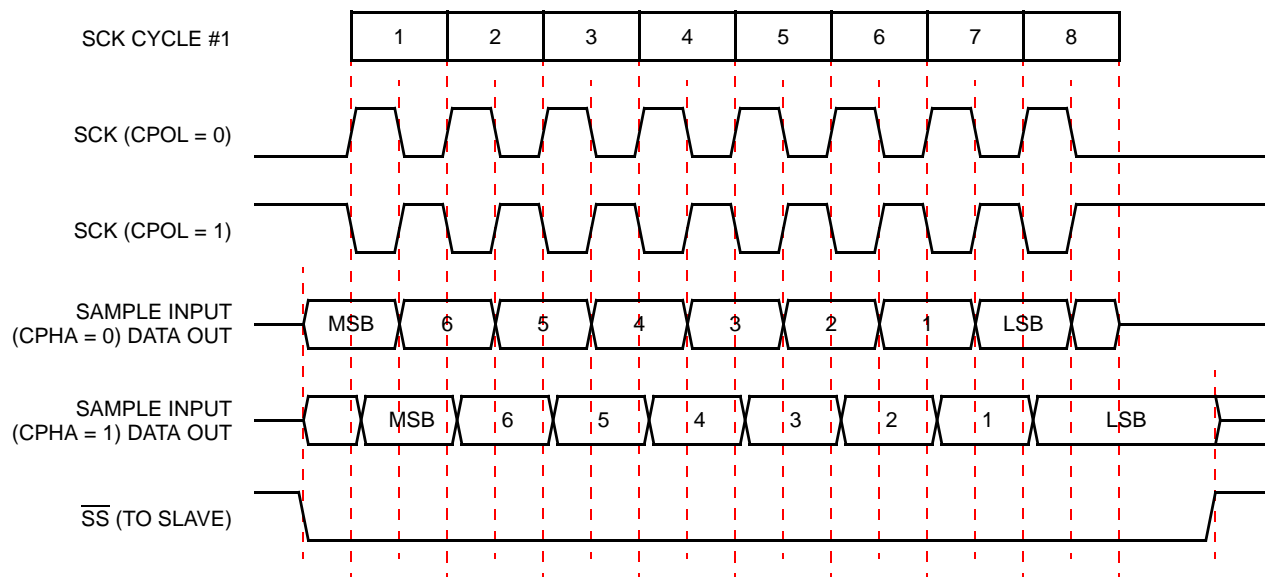
Figure 6-4 SPI Signal Timing

Table 6-15 Submodes for SPI

Name	Description
SUBM1:0	Submode select for SPI mode 00 = Positive clock polarity, receive on rising edge, transmit on falling edge 01 = Positive clock polarity, receive on falling edge, transmit on rising edge 10 = Negative clock polarity, receive on falling edge, transmit on rising edge 11 = Negative clock polarity, receive on rising edge, transmit on falling edge

6.5.7   GPSI

Hardware

Figure 6-5 shows example circuits to connect the Serdes in GPSI (General Purpose Serial Interface) mode. Table 6-16 shows the GPSI signal to port pin mapping in Master mode, and Table 6-17 shows the GPSI signal to port pin mapping in Slave mode.

Software

GPSI is a general-purpose, point-to-point, full-duplex serial bus protocol. Only two devices are allowed to exist on a bus. The GPSI PHY device is responsible for maintaining bus timing by driving two continuously running clocks, TxCLK and RxCLK. The device that does not drive the clocks is the MAC device. The TxEn and TxD signals are synchronized to the TxClk clock. The RxD and RxEn signals are synchronized to the RxClk clock.

The COLLISION and TxBUSY signals do not participate in actual data transfer on the GPSI bus. COLLISION and TxBUSY provide additional flow control capabilities for the software device driver. The COLLISION signal indicates that a PHY device has detected a collision condition. This signal is useful only when the Serdes is connected to a PHY device or acting as a PHY device.

The TxBUSY signal is used by a GPSI device to indicate that the device is currently busy, and that another device should not attempt to start a data transfer.
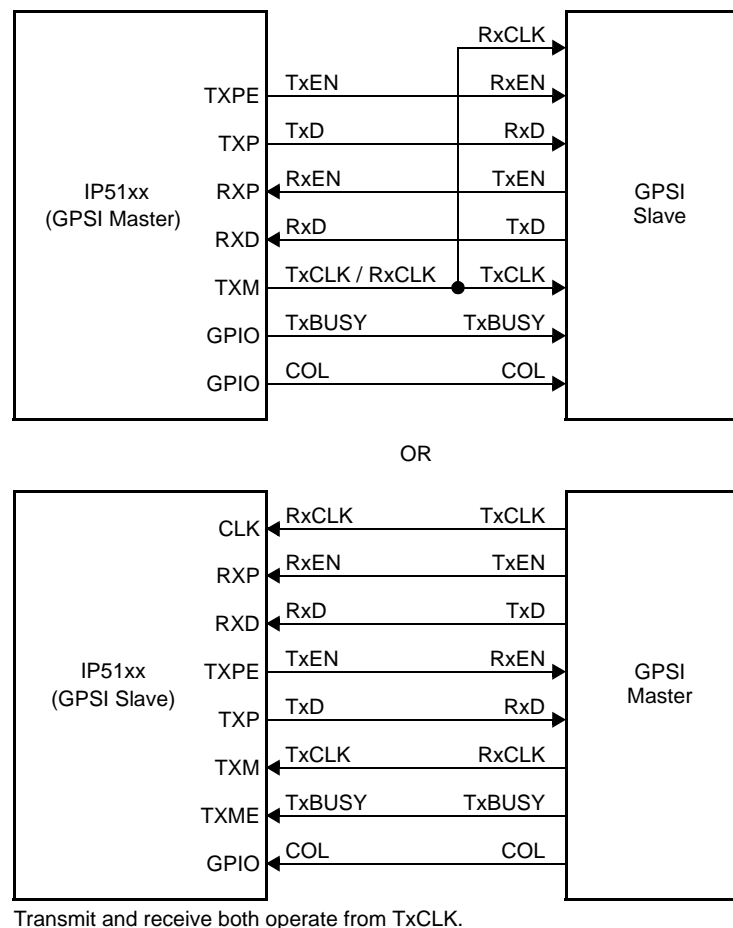
Figure 6-5 GPSI Interface Examples

Table 6-16 IP51xx GPSI Master Interface Signal Usage

GPSI Slave Signal Name	IP51xx GPSI Signal Name	Serdes Signal Name	Port D or E Pin	IP51xx's Direction	Description
TxCLK and RxCLK	TxCLK and RxCLK	TXM	5	Output	Transmit and Receive clock
TxD	RxD	RXD	0	Input	Transmit data
TxEN	RxEN	RXP	2	Input	Transmit data valid
RxD	TxD	TXP	6	Output	Receive data
RxEN	TxEN	TXPE	7	Output	Receive data valid
TxBUSY	TxBUSY	GPIO	-	Output	Indicates a data transfer in progress (handled by software)
COLLISION	COLLISION	GPIO	-	Output	Indicates a collision at PHY layer (handled by software)

Note: In GPSI master mode, the TXM Serdes pin should be used by the GPSI slave for both TxCLK and RxCLK inputs.

Table 6-17 IP51xx GPSI Slave Interface Signal Usage

GPSI Master Signal Name	IP51xx GPSI Signal Name	Serdes Signal Name	Port D or E Pin	IP51xx's Direction	Description
RxCLK	TxCLK	TXM	5	Input	Transmit clock
RxD	TxD	TXP	6	Output	Transmit data
RxEN	TxEN	TXPE	7	Output	Transmit data valid
TxCLK	RxCLK	CLK	3	Input	Receive clock
TxD	RxD	RXD	0	Input	Receive data
TxEN	RxEN	RXP	2	Input	Receive data valid
TxBUSY	TxBUSY	TXME	4	Input	Indicates a data transfer in progress (handled by software)
COLLISION	COLLISION	GPIO	-	Input	Indicates a collision at PHY layer (handled by software)

Table 6-18 Submodes for GPSI

Name	Description
SUBM1:0	Submode select for GPSI mode
	00 = Receive on rising edge, transmit on falling edge
	01 = Receive on falling edge, transmit on falling edge
	10 = Receive on rising edge, transmit on rising edge
	11 = Receive on falling edge, transmit on rising edge

6.6 Media Independent Interface (MII)

The IP51xx MII controller supports signaling and data transfer as defined in IEEE 802.3. This includes data transmit, data receive, and carrier and collision detect. The MII controller, however, does not support the station management functions of MII (no MDIO / MDC signaling). Additionally, this controller implements in hardware most of the functions of an Ethernet MAC, including packet filtering based on frame size, transmit CRC generation and receive CRC checking. Station management is handled through a different function/port.

Table 6-19 shows how the MII interface signals are assigned to the port pins. Table 6-20 shows the pin assignments for Reduced MII (RMII). Section 7.10.2 gives a detailed description of the port registers used for MII.

When using the MII interface, half of the signals are available at Port E, and the remaining signals can be accessed at Port I. Both ports (E and I) must select the MII function, but only the registers of Port E manage the MII controller. For RMII, only the pins of Port E are used.

Table 6-19 MII Interface Signals

Signal	Port E Pin	Port I Pin	I/O
RX_CLK	0		I
RXD [0]	1		I
RXD [1]	2		I
RXD [2]	3		I
RXD [3]	4		I
RX_DV	5		I
RX_ER	6		I
COL	7		I
TX_CLK		0	I
TXD [0]		1	O
TXD [1]		2	O
TXD [2]		3	O
TXD [3]		4	O
TX_EN		5	O
TX_ER		6	O
CRS		7	I

Table 6-20 RMII Interface Signals

Signal	Port E Pin	I/O
REF_CLK	0	I
TXD [0]	1	O
TXD [1]	2	O

Table 6-20 RMII Interface Signals

Signal	Port E Pin	I/O
TX_EN	3	O
RX_ER	4	I
RXD [0]	5	I
RXD [1]	6	I
CRS_DV	7	I

6.6.1 Receive Sequence

The MII receiver moves data from the MII interface to the receive FIFO and signals exceptional conditions.

For both normal data reception and data reception with errors, data is moved into the receive FIFO for access by software. In the case of data reception with errors, a processor interrupt is generated. For normal interframe messages as well as for reserved messages, the MII controller takes no action. For a false carrier message, the MII receiver asserts the FALSE_CARRIER_INT interrupt.

6.6.1.1 Receive Start

1. The RX_EN bit must be set for the MII receiver to become active. If RX_EN is set while the RX_DV signal is active, the MII receiver waits until RX_DV is deasserted before becoming active.
2. Receiver activity is independent of the state of CRS (carrier sense).
3. Upon detection of an SFD (Start-of-Frame Delimiter), the RX_SFD_INT is asserted.
4. At reception of the minimum frame length (32) number of bytes,
   – The RX_PKT_SIZE_THRESHOLD interrupt is asserted.
   – The RX_FIFO_SELECT bit changes to indicate which receive FIFO is being filled with data from the current frame.
   – The CRC_OK bit no longer reflects the CRC check from the previous frame and now reflects the CRC check for those bytes received in the current frame.
   – The RX_BYTE_COUNT field no longer reflects the count of bytes received in the previous frame, and now reflects the count of bytes received in the current frame.

6.6.1.2 Receive

The receiver continues to accept data and move it to the FIFO as long as RX_DV is active. The receiver does not

know about the state of the FIFO; software must monitor the FIFO and remove data to avoid FIFO overflow.

6.6.1.3   Receive End

1. The receiver recognizes deassertion of RX_DV as the end-of-frame delimiter.
2. The RX_EOP_INT interrupt is asserted at the termination of receive data, at which time the receive data is available at the receive FIFO and the count of received bytes is available in RX_BYTE_COUNT.
3. In the case of an odd number of nibbles, the byte count does not include the odd nibble.
4. The CRC is calculated continuously, so the CRC_OK bit reflects the state of the CRC check at the time the signal RX_EOP_INT is asserted.

6.6.1.4   Receive Error – RX FIFO Overflow

During a receive FIFO overflow, the MII receiver continues to receive data and write the data into the receive FIFO without interruption. At the completion of a packet during which there was a receive FIFO overflow, the receive FIFO must be reset (using the RX FIFO Reset bit of the port's Interrupt Set register).

6.6.1.5   Receive Error – RX_ER asserted

RX_ERR_INT is asserted each time the MII signal RX_ER is asserted. The detection of RX_ER does not affect the reception and movement of data to the receive FIFO. The detection of RX_ER does not cause the receive function to terminate.

6.6.2   Transmit Sequence

During transmit, the MII controller moves data to the MII interface from the transmit FIFO and signals exception conditions.

6.6.2.1   Transmit Start

1. The first piece of transmit data must be present in the TX FIFO before starting to transmit.
2. The transmit byte count must be set before starting to transmit. The behavior with a transmit byte count of 0 is undefined.
3. The transmit byte count must be present before TX_START is asserted.
4. Transmission is started when the TX_START bit is set.

5. While the transmitter is active, assertion of TX_START produces undefined behavior.

6.6.2.2   Transmit End

1. The transmitter ceases operation after transmitting the number of bytes in TX_BYTE_COUNT (TX_BYTE_COUNT + 4 if TX_AUTO_CRC is enabled in Function Control 0).
2. The transmitter signals completion by raising TX_EOP_INT.

6.6.2.3   Transmit Error – Collision and Jam

When operating in half duplex mode, if the COL signal is asserted while the MII controller is transmitting, the MII controller:

1. Signals detection of a collision by raising COL_INT.
2. Ceases transmitting the user supplied data and asserts a TX_EOP_INT interrupt.
3. Transmits a jam pattern consisting of the repeating pattern of 1 0 1 0 … (least significant bit first) until a total of 32 bits have been transmitted (8 nibbles of 0x5)
4. Ceases transmission of all data.

The three events, COL_INT, TX_EOP_INT, and jam data transmission, do not occur with deterministic timing, but do obey the following rules:

1. COLLISION and COL_INT are always asserted together and are asserted first.
2. TX_EOP_INT can be asserted as early as one core clock cycle after the assertion of COL_INT and as late as 8 MII tx_clk cycles after the assertion of COL_INT but always synchronous to the core clock.
3. Transmission of the jam sequence is always the last operation to complete.

6.6.2.4   Transmit Error – Premature End of Data

The MII transmitter has no means to validate the contents of the TX FIFO. The transmitter continues to remove data from the FIFO and transmits until it has transmitted TX_BYTE_COUNT bytes. If the FIFO becomes empty during transmission, the transmit FIFO asserts an underflow interrupt.

6.7   PCI Interface

The PCI controller manages the interface between the IP51xx and external PCI devices on Ports B and C.

Table 6-21 shows how the external PCI interface signals are assigned to the port pins. Signals ending with "_N" are active low.

Table 6-21   PCI Port Signals

Signal	Port Pin	I/O	Description
DEVSEL_N	B0	I/O	Device select output. As an input, it indicates that a selected target device has decoded an operation.
PERR_N	B1	I/O	Parity error
STOP_N	B2	I/O	PCI stop signal
SERR_N	B3	I/O	System error
TRDY_N	B4	I/O	Target ready
FRAME_N	B5	I/O	PCI frame signal
IRDY_N	B6	I/O	Initiator ready
PAR	B7	I/O	Parity signal
CBE[0]	B8	I/O	Command / Byte Enable (Byte Enable is active low).
CBE[1]	B9	I/O	
CBE[2]	B10	I/O	
CBE[3]	B11	I/O	
REQ0_N	B12	O	Request PCI bus 0. Applies only when IP51xx is Master.
GNT0_N	B13	I	PCI bus 0 granted. Applies only when IP51xx is Master.
REQ1_N	B14	O	Request PCI bus 1. Applies only when IP51xx is Master.
GNT1_N	B15	I	PCI bus 1 granted. Applies only when IP51xx is Master.
RST_N	B16	I	Reset input
CLK	B17	I	PCI clock input
CLK_OUT	B18	O	PCI clock output
INTA	B19	O	Interrupt line open collector driver enable
AD[00:31]	C[0:31]	I/O	32- bit address / data bus

Registers associated with normal read and write operations from / to the external PCI devices are located in the non-blocking region of Port B. Registers used for configuration of the PCI controller are located in the Port B blocking region.

In order to use the PCI interface, Function 1 must be selected on both Port B and Port C.

Section 7.8.1 gives a detailed description of the port registers used for the PCI Controller.

Note: The PCI interface in the IP51xx operates always as a PCI host, and never as a PCI device. The IP51xx can be either a Master or a Target on the PCI bus.

6.7.1   PCI Interface Features

The PCI interface implements PCI specification 2.2.

General features of the interface are:

* 32-bit data and address bus
* PCI memory space commands
* Master channel FIFO Interface
* Target channel FIFO interface
* PCI configuration registers accessible from the processor

PCI host-specific features are:

* Support for two external devices on PCI bus
* Programmable PCI clock / reset generator
* Arbiter implementing the following selectable schemes:
    – Round-Robin arbitration (default)
    – PCI host priority arbitration

The following features are not supported:

* EEPROM configuration
* Power management interface
* CLKRUN_N protocol
* BIOS ROM interface
* Target channel register interface
* PCI I/O space commands

6.7.2 PCI Bus Commands

Table 6-22 lists the PCI bus commands supported by the IP51xx. The table shows, for both Master and Target modes, whether a command is supported, not supported, or aliased to another command.

Table 6-22   PCI Bus Commands

CBE[3:0]	Command	PCI Master	PCI Target
0000	Interrupt Acknowledge	Yes	Ignored
0001	Special Cycle	Yes	Ignored
0010	I/O Read	Yes	Yes
0011	I/O Write	Yes	Yes
0100	Reserved	No	Ignored
0101	Reserved	No	Ignored
0110	Memory Read	Yes	Yes
0111	Memory Write	Yes	Yes
1000	Reserved	No	Ignored
1001	Reserved	No	Ignored
1010	Configuration Read	Yes	No
1011	Configuration Write	Yes	No
1100	Memory Read Multiple	Yes	Aliased to Memory Read
1101	Dual Address Cycle	No	Ignored
1110	Memory Read Line	Yes	Aliased to Memory Read
1111	Memory Write Invalidate	Yes	Aliased to Memory Write

6.8 GMAC Interface

The GMAC (Gigabit Ethernet Media Access Controller) manages the interface between the IP51xx and an external GMAC PHY device on Port F.

The GMAC supports three types of MAC-PHY interfaces: MII (Media Independent Interface), RMII (Reduced Media Independent Interface), and RGMII (Reduced Gigabit Media Independent Interface).

Table 6-23 shows how the external GMAC interface signals are assigned to the port pins.

Table 6-23   GMAC Port Signals

Port F Pin	Mode			I/O	Description
	MII	RMII	RGMII		
0	TXD[0]	TXD[0]	TXD[0]	O	Transmit Data
1	TXD[1]	TXD[1]	TXD[1]	O	Transmit Data
2	TXD[2]		TXD[2]	O	Transmit Data
3	TXD[3]		TXD[3]	O	Transmit Data
4	TX_ER			O	Transmit Error
5	TX_EN	TX_EN	TX_CTL	O	Transmit Data Enable / Transmit Control (RGMII)
6	TX_CLK	REF_CLK_I		I	Transmit Clock / Reference Clock Return Path (RMII)
7	COL	REF_CLK_O	TX_CLK_O	I/O	Collision / Reference Clock (RMII) / Transmit Clock
8	RXD[0]	RXD[0]	RXD[0]	I	Receive Data
9	RXD[1]	RXD[1]	RXD[1]	I	Receive Data
10	RXD[2]		RXD[2]	I	Receive Data
11	RXD[3]		RXD[3]	I	Receive Data
12	RX_ER	RX_ER		I	Receive Error
13	RX_DV		RX_CTL	I	Receive Data Valid / Receive Control (RGMII)
14	RX_CLK		RX_CLK	I	Receive Clock
15	CRS	CRS_DV		I	Carrier Sense / Data Valid (RMII)

Registers associated with normal read and write operations from / to the external PHY are located in the non-blocking region of Port F. Registers used for configuration of the GMAC are located in the Port F blocking region.

In order to use the GMAC interface, Function 1 must be selected on Port F.

Section 7.11.1 gives a detailed description of the port registers used for the GMAC.

6.9   High-Speed USB Interface

The High-Speed USB controller manages the interface between the IP51xx and external USB devices on the USB Port. The High-Speed USB port supports High Speed, Full Speed, and Low Speed USB Modes.

Table 6-24 shows how the external USB interface signals are assigned to the port pins.

Table 6-24   High-Speed USB Port Signals

Pin Name	I/O	Description
USB2_ID	I	If this input is high, then this port is a USB peripheral port. If this input is low, then this port is a USB host port.
USB2_N	I/O	USB data (negative)
USB2_P	I/O	USB data (positive)
USB2_RBIAS	O	Tied to VSS with a 9.1 Kohm resistor.
USB2_VBUS	I	The IP51xx senses whether this is 5V or floating — if it is floating, then the IP51xx can output on a GPIO to turn on a circuit to supply 5V to the USB2 bus.

USB Port interrupt, function control, and function status registers are located in the USB Port non-blocking region. Registers associated with normal read and write operations from / to the external USB devices and the registers used for configuration of the USB controller are located in the USB Port blocking region.

In order to use the USB interface, Function 1 must be selected on the USB Port.

Section 7.15.1 gives a detailed description of the port registers used for the USB Controller.

6.9.1   USB Controller Features

The USB Controller can function as any of the following:

- The function controller of a high- / full-speed USB peripheral.
- The host controller for a multi-point USB system (when connected to a hub).

The controller complies both with USB standard for High-speed and Full-speed functions. The controller is user-configurable for up to 5 Transmit endpoints and / or up to 5 Receive endpoints in addition to Endpoint 0. The use of these endpoints for IN transactions and OUT transactions depends on whether the controller is being used as a peripheral or as a host. When used as a peripheral, IN transactions are processed through Tx endpoints and OUT transactions are processed through Rx endpoints. When used as a host, IN transactions are processed through Rx endpoints and OUT transactions are processed through Tx endpoints. These additional endpoints can be configured individually in software to handle either Bulk transfers (which also allows them to handle Interrupt transfers), Isochronous transfers, or Control transfers. Further, the endpoints can also be allocated to different target device functions on the fly, maximizing the number of devices that can be simultaneously supported.

Each endpoint has a FIFO associated with it. The FIFO for Endpoint 0 is 64 bytes deep and will buffer one packet. Other endpoint FIFOs may be as follows:

- Endpoint 1 FIFO: up to 16 bytes
- Endpoint 2 FIFO: up to 1024 bytes
- Endpoint 3 FIFO: up to 512 bytes
- Endpoint 4 FIFO: up to 512 bytes
- Endpoint 5 FIFO: up to 64 bytes

A Tx endpoint and the Rx endpoint with the same endpoint number share the same FIFO.

The controller provides all the encoding, decoding, and checking needed for sending and receiving USB packets, interrupting the processor only when endpoint data has been successfully transferred.

The controller offers a range of test modes — primarily the four test modes for High-speed operation described in the USB specification. It also includes options that allow it to be forced into Full-speed mode, High-speed mode, or Host mode. The last of these may be useful in helping to debug PHY problems in hardware.

6.9.2   Operation as Host or Peripheral

The controller may be used in a range of different environments. It can be used as either a High-speed or a Full-speed peripheral attached to a conventional USB host (such as a PC). Additionally, the controller may be used as the host to a range of such peripheral devices in a Multi-point setup.

In all cases, Control, Bulk, Isochronous, or Interrupt transactions are supported between the controller and the devices to which it is attached.

6.9.3   Support for Multiple Devices

The controller has the facility, when operating in Host mode, to act as the host to a range of USB peripheral devices — High-speed or Full-speed — where these devices are connected to the USB interface via a USB

hub. In theory, the controller could support a full USB tree of 128 such devices, although it is likely to be used with very much smaller numbers of devices in practice.

The key feature of the core's support for multiple devices is its facility to allow the functions of the target devices to be individually allocated to the different Rx and Tx endpoints implemented in the controller. Furthermore, this allocation can be made dynamically, allowing the system designer to make full use of these endpoints to service the needs of the different devices that are attached to the core.

Note: The controller's Multi-Point capability is associated with a range of registers recording the allocation of device functions to individual endpoints and device function characteristics such as endpoint number, operating speed and transaction type on an endpoint-by-endpoint basis. Although principally associated with the use of the controller as the host to a number of devices, these registers must also be set when the core is used as the host for a single target device.

6.9.4   Throughput

The throughput allocation for this USB controller module is 120 Mbps. This assumes that only 1/8 of the CPU resources will be allocated to USB devices to meet this target throughput. This USB controller contains a four stage buffer which can hold up to four commands from the USB port.

Throughput can go up to 175 Mbps if double-buffering is used.

6.10   Debug Port

The Debug Port provides a test interface through which an external host can access internal resources within the IP51xx for debug purposes. Table 6-25 shows how the debug port interface signals are assigned to the port pins, using a simple SPI interface.

Table 6-25 Debug Port Interface Signals

Signal	I/O	Description
TSI	I	Serial Data Input — driven by the external host
TSO	O	Serial Data Output — driven by the debug port.
TSCK	O	SPI Clock
TSSN	I	Slave Select

TSSN (Slave Select) is an active low signal. When asserted, the external host is sending data through TSI and receiving data through TSO. The external host is always the master, and the IP51xx is always the slave.

Section 5.3 describes the commands that the host can use to access the internal resources of the IP51xx, and shows how to use the Debug Port to perform a variety of debugging operations.

7.0 Memory Reference

For an overview of IP51xx memory spaces, see Figure 3-1.

The following sections show addresses, reset values, and descriptions of IP51xx registers.

7.1 Alphabetical List of Registers

Table 7-1 lists the IP51xx registers alphabetically. The subsequent sections describe these registers in functional groups – Per-Thread Registers (Section 7.2), Global Registers (Section 7.3), and Indirect Registers, which are composed of HRT Tables (Section 7.4), On-Chip Peripherals Registers (Section 7.5), and Per-Port Registers (Section 7.6).

Note: This alphabetical list does not include any blocking registers, because they do not have unique names that are meaningful out of context. For blocking registers, refer to the per-port register descriptions beginning in Section 7.6.

Table 7-1 Alphabetical List of Registers

Register	Type	Address	Description	Details
A0-A6	Per-Thread	080-098	32-bit address registers	Table 7-2
A7 or SP	Per-Thread	09C	32-bit stack pointer	Table 7-2
ACC0_HI	Per-Thread	0A0	High 32 bits of MAC, DSP, and multiplier result	Table 7-2
ACC0_LO	Per-Thread	0A4	Low 32 bits of MAC, DSP, and multiplier result	Table 7-2
ACC1_HI	Per-Thread	0D8	High 32 bits of DSP result	Table 7-2
ACC1_LO	Per-Thread	0DC	Low 32 bits of DSP result	Table 7-2
CHIP_ID	Global	100	Chip Device ID and Revision Number	Table 7-3, Section 7.3.1
CSR	Per-Thread	0B4	Control and Status Register	Table 7-2, Section 7.2.1
D_RANGE0_EN	Global	2A0	Data space memory range 0 thread enables	Table 7-3
D_RANGE0_HI	Global	260	Data space memory range 0 high	Table 7-3
D_RANGE0_LO	Global	280	Data space memory range 0 low	Table 7-3
D_RANGE1_EN	Global	2A4	Data space memory range 1 thread enables	Table 7-3
D_RANGE1_HI	Global	264	Data space memory range 1 high	Table 7-3
D_RANGE1_LO	Global	284	Data space memory range 1 low	Table 7-3
D_RANGE2_EN	Global	2A8	Data space memory range 2 thread enables	Table 7-3
D_RANGE2_HI	Global	268	Data space memory range 2 high	Table 7-3
D_RANGE2_LO	Global	288	Data space memory range 2 low	Table 7-3
D_RANGE3_EN	Global	2AC	Data space memory range 3 thread enables	Table 7-3
D_RANGE3_HI	Global	26C	Data space memory range 3 high	Table 7-3
D_RANGE3_LO	Global	28C	Data space memory range 3 low	Table 7-3
D0-D15	Per-Thread	000-03C	General-purpose data registers	Table 7-2
DCADDR	Indirect	0100 0600	Data Cache Address	Table 7-11
DCAPT	Global	170	Write Trap address	Table 7-3, Section 7.3.5
DCSTAT	Indirect	0100 060C	Data Cache Status	Table 7-11
DCRDD	Indirect	0100 0604	Data Cache Read Data	Table 7-11
DCWRD	Indirect	0100 0608	Data Cache Write Data	Table 7-11
DDR Cal Ctrl	Indirect	0100 0030	DDR Calibrator control register	Table 7-5
DDR Cal Stat	Indirect	0100 0034	DDR Calibrator status register	Table 7-5

Table 7-1 Alphabetical List of Registers (continued)

Register	Type	Address	Description	Details
FIFO Level	Indirect, Per-Port	4C+base	FIFO current level	Table 7-17
FIFO Watermark	Indirect, Per-Port	48 + base	FIFO watermark trigger level	Table 7-17
Function	Indirect, Per-Port	00 + base	Function select, reset, and FIFO configuration	Table 7-17
Function Ctrl 0	Indirect, Per-Port	30 + base	Function control register 0	Table 7-17
Function Ctrl 1	Indirect, Per-Port	34 + base	Function control register 1	Table 7-17
Function Ctrl 2	Indirect, Per-Port	38 + base	Function control register 2	Table 7-17
Function Status 0	Indirect, Per-Port	3C + base	Function status register 0	Table 7-17
Function Status 1	Indirect, Per-Port	40 + base	Function status register 1	Table 7-17
Function Status 2	Indirect, Per-Port	44 + base	Function status register 2	Table 7-17
GLOBAL_CTRL	Global	134	Global Control Register	Table 7-3
GPIO Ctrl	Indirect, Per-Port	04 + base	GPIO output enable	Table 7-17
GPIO In	Indirect, Per-Port	0C + base	GPIO data input — the current state of the external I/O pins	Table 7-17
GPIO Mask	Indirect, Per-Port	50 + base	GPIO mask	Table 7-17
GPIO Out	Indirect, Per-Port	08 + base	GPIO data output	Table 7-17
HASH_OUT_0	Indirect	0100 0470	Security hash register, bits [159:128]	Table 7-9
HASH_OUT_1	Indirect	0100 0474	Security hash register, bits [127:96]	Table 7-9
HASH_OUT_2	Indirect	0100 0478	Security hash register, bits [95:64]	Table 7-9
HASH_OUT_3	Indirect	0100 047C	Security hash register, bits [63:32]	Table 7-9
HASH_OUT_4	Indirect	0100 0480	Security hash register, bits [31:0]	Table 7-9
HRT0	Indirect, HRT	800-83C	Hard Real-Time Table 0 (Sixteen 32 bit registers)	Table 7-4
HRT1	Indirect, HRT	900-93C	Hard Real-Time Table 1 (Sixteen 32 bit registers)	Table 7-4
I_RANGE0_EN	Global	240	Instruction space memory range 0 thread enables	Table 7-3
I_RANGE0_HI	Global	200	Instruction space memory range 0 high	Table 7-3
I_RANGE0_LO	Global	220	Instruction space memory range 0 low	Table 7-3
I_RANGE1_EN	Global	244	Instruction space memory range 1 thread enables	Table 7-3
I_RANGE1_HI	Global	204	Instruction space memory range 1 high	Table 7-3
I_RANGE1_LO	Global	224	Instruction space memory range 1 low	Table 7-3
I_RANGE2_EN	Global	248	Instruction space memory range 2 thread enables	Table 7-3
I_RANGE2_HI	Global	208	Instruction space memory range 2 high	Table 7-3
I_RANGE2_LO	Global	228	Instruction space memory range 2 low	Table 7-3

Table 7-1 Alphabetical List of Registers (continued)

Register	Type	Address	Description	Details
ICADDR	Indirect	0100 0500	Instruction Cache Address	Table 7-10
ICCTRL	Indirect	0100 0510	Instruction Cache Control	Table 7-10
ICSTAT	Indirect	0100 050C	Instruction Cache Status	Table 7-10
ICRDD	Indirect	0100 0504	Instruction Cache Read Data	Table 7-10
ICWRD	Indirect	0100 0508	Instruction Cache Write Data	Table 7-10
INMAIL	Indirect	0100 0300	Incoming Mailbox	Table 7-8
INST_CNT	Per-Thread	0B0	Executed instruction count	Table 7-2
INT_CLR0	Global	124	Clears bits in INT_STAT0.	Table 7-3
INT_CLR1	Global	128	Clears bits in INT_STAT1.	Table 7-3
INT_MASK0	Per-Thread	0C0	AND'ed with Global INT_STAT0 to mask interrupts	Table 7-2
INT_MASK1	Per-Thread	0C4	AND'ed with Global INT_STAT1 to mask interrupts	Table 7-2
INT_SET0	Global	114	Sets bits in INT_STAT0.	Table 7-3
INT_SET1	Global	118	Sets bits in INT_STAT1.	Table 7-3
INT_STAT0	Global	104	Interrupt Status Register 0	Table 7-3, Section 7.3.2.
INT_STAT1	Global	108	Interrupt Status Register 1	Table 7-3, Section 7.3.3.
Interrupt Clear	Indirect, Per-Port	1C + base	Port interrupt clear	Table 7-17
Interrupt Mask	Indirect, Per-Port	14 + base	Port interrupt mask	Table 7-17,
Interrupt Set	Indirect, Per-Port	18 + base	Port interrupt set	Table 7-17
Interrupt Status	Indirect, Per-Port	10 + base	Port interrupt status from selected function	Table 7-17
IO PU Config	Indirect	0100 008C	I/O driver cell Power Up Enable signal	Table 7-5
IREAD_DATA	Per-Thread	0BC	IREAD result	Table 7-2
MAC_RC16	Per-Thread	0A8	Rounded and clipped multiplier accumulate/result	Table 7-2
MAIL_STAT	Indirect	0100 0308	Mailbox Status	Table 7-8
MPT_VAL	Indirect	0100 0100	Multipurpose Timer value	Table 7-6
MT_ACTIVE	Global	138	Active/inactive status for each thread	Table 7-3
MT_ACTIVE_CLR	Global	140	Clears bits in MT_ACTIVE.	Table 7-3
MT_ACTIVE_SET	Global	13C	Sets bits in MT_ACTIVE.	Table 7-3
MT_BLOCKED_CLR	Global	1B4	Clear bits of MT_I_BLOCKED and MT_D_BLOCKED	Table 7-3
MT_BREAK	Global	158	Multithreading BKPT Executed	Table 7-3
MT_BREAK_CLR	Global	15C	Clears bits in MT_BREAK.	Table 7-3
MT_DBG_ACTIVE	Global	144	AND'ed with MT_ACTIVE for debug control.	Table 7-3
MT_DBG_ACTIVE_CLR	Global	17C	Clears bits in MT_DBG_ACTIVE.	Table 7-3
MT_DBG_ACTIVE_SET	Global	148	Sets bits in MT_DBG_ACTIVE.	Table 7-3
MT_EN	Global	14C	Multithreading Enable	Table 7-3
MT_HPRI	Global	150	Multithreading High-Priority Thread	Table 7-3
MT_HRT	Global	154	Multithreading Hard Real-Time Thread	Table 7-3
MT_D_BLOCKED	Global	1A8	Thread blocked due to data access (1 bit / thread)	Table 7-3

Table 7-1 Alphabetical List of Registers (continued)

Register	Type	Address	Description	Details
MT_D_BLOCKED_SET	Global	1B0	Set bits of MT_D_BLOCKED register	Table 7-3
MT_I_BLOCKED	Global	1A4	Thread blocked due to instruction fetch (1 bit / thread)	Table 7-3
MT_I_BLOCKED_SET	Global	1AC	Set bits of MT_I_BLOCKED register	Table 7-3
MT_MIN_DELAY_EN	Global	164	Multithreading Minimum Delay Enable	Table 7-3
MT_SINGLE_STEP	Global	160	Controls single-step operation for each thread.	Table 7-3
MT_TRAP_EN	Global	1B8	Multithreading Enable Traps mask	Table 7-3
MT_TRAP	Global	1BC	Multithreading Trap (set by hardware when an enabled trap occurs, or by writing to MT_TRAP_SET)	Table 7-3
MT_TRAP_CLR	Global	1C4	Clear bits of MT_TRAP	Table 7-3
MT_TRAP_SET	Global	1C0	Set bits of MT_TRAP	Table 7-3
MTESTADDR	Indirect	0100 0900	Memory Address for memory test	Table 7-15
MTESTCTRL	Indirect	0100 090C	Memory Test Control Register	Table 7-15
MTESTRD	Indirect	0100 0908	Memory Test Read Data	Table 7-15
MTESTWD	Indirect	0100 0904	Memory Test Write Data	Table 7-15
OCM_BIST_CFG	Indirect	0100 0704	On-Chip Memory Built-In Self Test Configuration	Table 7-12
OCM_BIST_STAT	Indirect	0100 0708	On-Chip Memory Built-In Self Test Status	Table 7-12
OCM_CFG	Indirect	0100 0700	On-Chip Memory Configuration	Table 7-12
OCP_CLK_CORE_CFG	Indirect	0100 0000	Core PLL configuration register	Table 7-5
OCP_CLK_DDR_CFG	Indirect	0100 0008	DDR PLL configuration register	Table 7-5
OCP_CLK_DDRDS_CFG	Indirect	0100 000C	DDR Deskew PLL configuration register	Table 7-5
OCP_CLK_IO_CFG	Indirect	0100 0004	I/O PLL configuration register	Table 7-5
OCP_CLK_SLIP_CLR	Indirect	0100 0010	Global PLL slip latch clear register	Table 7-5
OCP_CLK_SLIP_STAT	Indirect	0100 0014	Global PLL slip latch status register	Table 7-5
OUTMAIL	Indirect	0100 0304	Outgoing Mailbox	Table 7-8
PC	Per-Thread	0D0	32-bit program counter	Table 7-2
PREVIOUS_PC	Per-Thread	0E0	Program counter value for last successfully executed instruction for this thread.	Table 7-2
ROSR	Per-Thread	0B8	Read-only Status Register	Table 7-2, Section 7.2.2
Reset Reasons	Indirect	0100 0084	Reasons for the last chip reset	Table 7-5
RT_COM	Indirect	0100 0104	Real-Time Timer Compare register	Table 7-6
Receive FIFO HI	Indirect, Per-Port	2C + base	Receive FIFO high word, bits [63:32]	Table 7-17
Receive FIFO LO	Indirect, Per-Port	28 + base	Receive FIFO low word, bits [31:0]	Table 7-17
SCRATCHPAD0	Global	180	Scratch Pad Register 0	Table 7-3
SCRATCHPAD1	Global	184	Scratch Pad Register 1	Table 7-3
SCRATCHPAD2	Global	188	Scratch Pad Register 2	Table 7-3
SCRATCHPAD3	Global	18C	Scratch Pad Register 3	Table 7-3
SEC_IN_0	Indirect	0100 0430	Security input register, bits [159:128]	Table 7-9
SEC_IN_1	Indirect	0100 0434	Security input register, bits [127:96]	Table 7-9
SEC_IN_2	Indirect	0100 0438	Security input register, bits [95:64]	Table 7-9

Table 7-1 Alphabetical List of Registers (continued)

Register	Type	Address	Description	Details
SEC_IN_3	Indirect	0100 043C	Security input register, bits [63:32]	Table 7-9
SEC_IN_4	Indirect	0100 0440	Security input register, bits [31:0]	Table 7-9
SEC_KEY_0	Indirect	0100 0410	Security key register, bits [255:224]	Table 7-9
SEC_KEY_1	Indirect	0100 0414	Security key register, bits [223:192]	Table 7-9
SEC_KEY_2	Indirect	0100 0418	Security key register, bits [191:160]	Table 7-9
SEC_KEY_3	Indirect	0100 041C	Security key register, bits [159:128]	Table 7-9
SEC_KEY_4	Indirect	0100 0420	Security key register, bits [127:96]	Table 7-9
SEC_KEY_5	Indirect	0100 0424	Security key register, bits [95:64]	Table 7-9
SEC_KEY_6	Indirect	0100 0428	Security key register, bits [63:32]	Table 7-9
SEC_KEY_7	Indirect	0100 042C	Security key register, bits [31:0]	Table 7-9
SEC_OUT_0	Indirect	0100 0450	Security output register, bits [159:128]	Table 7-9
SEC_OUT_1	Indirect	0100 0454	Security output register, bits [127:96]	Table 7-9
SEC_OUT_2	Indirect	0100 0458	Security output register, bits [95:64]	Table 7-9
SEC_OUT_3	Indirect	0100 045C	Security output register, bits [63:32]	Table 7-9
SEC_OUT_4	Indirect	0100 0460	Security output register, bits [31:0]	Table 7-9
Security Control	Indirect	0100 0400	Security Module Control Register	Table 7-9
Security Status	Indirect	0100 0404	Security Module Status Register	Table 7-9
SOURCE3	Per-Thread	0AC	Implicit third source operand	Table 7-2
SP or A7	Per-Thread	09C	32-bit stack pointer	Table 7-2
STS_CFG0	Indirect	0100 0800	Statistics Configuration Register 0	Table 7-13
STS_CFG1	Indirect	0100 0808	Statistics Configuration Register 1	Table 7-13
STS_CFG2	Indirect	0100 0810	Statistics Configuration Register 2	Table 7-13
STS_CFG3	Indirect	0100 0818	Statistics Configuration Register 3	Table 7-13
STS_CNT0	Indirect	0100 0804	Statistics Counter 0	Table 7-13
STS_CNT1	Indirect	0100 080C	Statistics Counter 1	Table 7-13
STS_CNT2	Indirect	0100 0814	Statistics Counter 2	Table 7-13
STS_CNT3	Indirect	0100 081C	Statistics Counter 3	Table 7-13
SW Reset	Indirect	0100 0080	Software reset	Table 7-5
SYS_COM_0	Indirect	0100 0118	System Timer Compare Register 0 (INT_STAT1[0])	Table 7-6
SYS_COM_1	Indirect	0100 011C	System Timer Compare Register 1 (INT_STAT1[1])	Table 7-6
SYS_COM_2	Indirect	0100 0120	System Timer Compare Register 2 (INT_STAT1[2])	Table 7-6
SYS_COM_3	Indirect	0100 0124	System Timer Compare Register 3 (INT_STAT1[3])	Table 7-6
SYS_COM_4	Indirect	0100 0128	System Timer Compare Register 4 (INT_STAT1[4])	Table 7-6
SYS_COM_5	Indirect	0100 012C	System Timer Compare Register 5 (INT_STAT1[5])	Table 7-6
SYS_COM_6	Indirect	0100 0130	System Timer Compare Register 6 (INT_STAT1[6])	Table 7-6
SYS_COM_7	Indirect	0100 0134	System Timer Compare Register 7 (INT_STAT1[7])	Table 7-6
SYS_COM_8	Indirect	0100 0138	System Timer Compare Register 8 (INT_STAT1[8])	Table 7-6
SYS_COM_9	Indirect	0100 013C	System Timer Compare Register 9 (INT_STAT1[9])	Table 7-6
SYS_VAL	Indirect	0100 0114	The value of the System Timer	Table 7-6
TKEY	Indirect	0100 0108	Timer Block's security key code	Table 7-6
TRAP_CAUSE	Per-Thread	0D4	The cause(s) of the last instruction trap for this thread.	Table 7-2

Table 7-1 Alphabetical List of Registers (continued)

Register	Type	Address	Description	Details
TRNG_CFG	Indirect	0100 0200	True Random Number Generator Configuration	Table 7-7
TRNG_VAL	Indirect	0100 0204	32-bit True Random Number Value	Table 7-7
Transmit FIFO HI	Indirect, Per Port	24 + base	Transmit FIFO high word, bits [63:32]	Table 7-17
Transmit FIFO LO	Indirect, Per Port	20 + base	Transmit FIFO low word, bits [31:0]	Table 7-17
USB DFT Ctrl	Indirect	0100 0038	USB DFT control register	Table 7-5
USB DFT Stat	Indirect	0100 003C	USB DFT status register	Table 7-5
WD_CFG	Indirect	0100 0110	Watchdog Configuration register	Table 7-6
WD_COM	Indirect	0100 010C	Watchdog Compare register	Table 7-6

7.2   Per-Thread Registers

Table 7-2 Per-Thread Registers

Address	Register	Read/ Write	Description	32-Bit Reset Value
000-03C	D0–D15	R/W	General-purpose 32-bit data registers. The only restriction is that they cannot be used as address or stack pointer registers. Note: Unlike earlier Ubicom chips, the IP51xx has no power-on reset for D0-D15. They are uninitialized at power on.	xxxx xxxx
040-07C	Reserved			
080-098	A0–A6	R/W	32-bit address registers. These are used as pointers to operands. Note: Unlike earlier Ubicom chips, the IP51xx has no power-on reset for A0-A6. They are uninitialized at power on.	xxxx xxxx
09C	A7 or SP	R/W	32-bit Stack pointer, also referred to as A7. The use of A7 as the stack pointer register is conventional, not "hard-wired" in the architecture. There are no instructions that use SP explicitly. Note: Unlike earlier Ubicom chips, the IP51xx has no power-on reset for A7. It is uninitialized at power on.	xxxx xxxx
0A0	ACC0_HI	R/W	High 32 bits of MAC, DSP, and multiplier result. Also set by CRCGEN.	0000 0000
0A4	ACC0_LO	R/W	Low 32-bits of MAC, DSP, and multiplier result. Also set by CRCGEN.	0000 0000
0A8	MAC_RC16	R/W	Rounded and Clipped S16.15 format of the most recent MAC, multiplier, CRCGEN result. The 48 bit result is interpreted as s16.31 format and rounded/clipped to S.15 format. This is then sign extended to 32 bits.	0000 0000
0AC	SOURCE3	R/W	Used as an implicit third source operand by certain instructions. Since the instruction formats are limited to 2 source and one destination operand, this register is used when a third operand is needed.	0000 0000
0B0	INST_CNT	RO	This register maintains a count of the executed instructions for the associated context. When the execution of a context is suspended, the associated counter also stops. This 32-bit count starts as zero after a reset operation, and otherwise cannot be reset. The count rolls over when it reaches its maximum value of 0xFFFF FFFF. This count, in conjunction with the global timer value, can be used to determine run-time performance distribution statistics, as well as to help in debugging (e.g., did thread #N execute any instructions yet?). Reading INST_CNT for a thread that is not quiescent gives an approximate value.	0000 0000
0B4	CSR (Control & Status Register)	R/W	Contains condition codes, and other status bits, as well as some thread-specific control bits (refer to Section 7.2.1)	0000 0000

Table 7-2 Per-Thread Registers

Address	Register	Read/ Write	Description	32-Bit Reset Value
0B8	ROSR (Read-Only Status Register)	RO	Extension of CSR containing bits or fields that are set by hardware, but read-only by software (refer to Section 7.2.2).	0000 0000
0BC	IREAD_DATA	R/W	The IREAD result is placed in this register. This register is a 32-bit scratchpad, for backwards compatibility with the IREAD instruction.	0000 0000
0C0	INT_MASK0	R/W	A 32-bit mask that, when AND'ed with the global 32-bit INT_STAT0 register, determines whether an interrupt condition is seen by a given thread.	0000 0000
0C4	INT_MASK1	R/W	A 32-bit mask that, when AND'ed with the global 32-bit INT_STAT1 register, determines whether an interrupt condition is seen by a given thread.	0000 0000
0C8-0CC	Reserved			
0D0	PC	R/W	32-bit Program Counter. Only valid for access in a thread that is quiescent (has no instructions in the pipeline). In that case, it points to the next instruction to be executed when that thread resumes. The PC is set by one of the following:· • Set to the power-up address after reset/power-up. • Direct write by another thread, when this thread is inactive. The controlling thread uses the destination thread select field in its CSR to address the target thread's PC register. Do not try to set PC of the current thread to do a jump.	6000 0000
0D4	TRAP_CAUSE	R/W	This register returns the cause(s) of the last instruction trap for this thread (refer to Section 7.2.3).	0000 0000
0D8	ACC0_HI	R/W	High 32 bits of DSP result.	0000 0000
0DC	ACC0_LO	R/W	Low 32-bits of DSP result.	0000 0000
0E0	PREVIOUS_PC	RO	The value of the program counter corresponding to the last successfully executed instruction for this thread. This value may be read while the corresponding thread is executing.	0000 0000
0E4-0FC	Reserved			

For those per-thread registers that have special functions assigned to bits or fields within the register, those bits and fields are described below.

7.2.1   CSR

The Control and Status Register contains condition codes and other status bits, as well as some thread-specific control bits as follows:

Bits	Description
31:21	Reserved
20	DSP_O. DSP Overflow Status (sticky). DSP instructions set this bit if the result is not exact, but never clear it. When a DSP instruction sets this bit, the change is visible several clocks after the DSP instruction.
19	Reserved
18:14	Destination Thread Select. Used to override the default context selection for directly addressed destination operand register. Has no effect for address calculations. Does not affect implicit destinations (such as ACC0_HI). Bits are defined as follows:
	18:15 DST_SEL. Destination context number. Relevant only if bit 14 is set.
	14 DST_SEL_EN: 1 On; 0 Off. Enables bits 18:15.
13	Reserved
12:8	Source Thread Select. Used to override the default context selection for directly addressed source 1 operand register. Has no effect for address calculations. Bits are defined as follows:
	12:9 SRC_SEL. Source 1 context number. Relevant only if bit 8 is set.
	8 SRC_SEL_EN: 1 On; 0 Off. Enables bits 13:9.
7-4	32-bit-Operand Condition Code Bits as {N,Z,V,C}32 ordering. Valid only when the thread has no instructions in the pipeline. Bit 7: N (negative) Bit 6: Z (zero) Bit 5: V (overflow) Bit 4: C (carry)
3-0	16-bit-Operand Condition Code Bits as {N,Z,V,C}16 ordering. Valid only when the thread has no instructions in the pipeline. Bit 3: N (negative) Bit 2: Z (zero) Bit 1: V (overflow) Bit 0: C (carry)

Note: Writing the source or destination thread select fields will also modify NZVC bits of the current thread at an unpredictable time.

7.2.2   ROSR

The Read-Only Status Register is an extension of CSR that contains bits or fields that are set by hardware, but read-only by software.

Bits	Description
31:6	Reserved
5:2	TNUM. Thread Number: Returns the thread number of the current instruction.
1	MEM_BUSY: Memory Busy. This bit is hard-coded to 0, for backwards compatibility with the IREAD/IWRITE instructions.
0	INT: Interrupt Condition. Indicates a pending interrupt condition for the thread. Derived by AND'ing the 64-bit thread-specific interrupt mask with the corresponding bits in the global interrupt status registers

7.2.3   TRAP_CAUSE

The TRAP_CAUSE register returns the cause(s) of the last instruction trap for this thread. Software should write to this register only when the thread is quiescent.

Bits	Description
31:13	Reserved
12	DST_RANGE_ERR. Destination memory protection error. When set, this bit indicates that a destination write access was disallowed by the memory protection mechanism.
11	SRC1_RANGE_ERR. Source 1 memory protection error. When set, this bit indicates that a source 1 read was attempted to a disallowed address.
10	I_RANGE_ERR. Instruction fetch memory protection error. When set, this bit indicates that an instruction fetch was attempted to a disallowed address.
9	DCAPT. Write address trap. When set, this bit indicates that a write address trap was triggered. This means that a write was performed to the address specified by the DCAPT register. This watchpoint applies only to writes performed using the restricted data register-only destination and those using the general destination. Additionally, this watchpoint applies to the general destination of LEA and PDEC instructions, unless the destination is an address register and DST_SEL_EN in the CSR is 0. This watchpoint does not apply to address registers modified through the auto-incrementing addressing modes, or the destination of MOVEAI, CALL, or CALLI.
8	DST_SERROR. Destination synchronous error. When set, this bit indicates that a Protocol C synchronous error was encountered on a write operation.
7	SRC1_SERROR. Source 1 synchronous error. When set, this bit indicates that a Protocol C synchronous error was encountered on a read operation.
6	DST_MISALIGNED. Destination operand alignment error. When set, this bit indicates that a misaligned write was attempted with the destination operand.

Bits	Description
5	SRC1_MISALIGNED. Source 1 operand alignment error. When set, this bit indicates that a misaligned read was attempted with the source 1 operand. Note that this does not apply to LEA and PDEC instructions, since they do not actually read source operands, but merely perform address calculations.
4	DST_DECODE_ERR. Destination address decode error. When set, this bit indicates that a write operation was attempted to an address that was decoded neither by the top-level address decoder in the main processor itself, nor by the Protocol C data bus address decoder.
3	SRC1_DECODE_ERR. Source 1 address decode error. When set, this bit indicates that a read operation was attempted to an address that was decoded neither by the top-level address decoder in the main processor itself, nor by the Protocol C data bus address decoder.
2	ILLEGAL_INST. Illegal instruction. When set, this bit indicates that execution of an undefined (illegal) instruction was attempted.
1	I_SERROR. Instruction synchronous error. When set, this bit indicates that a Protocol C synchronous error was encountered on an instruction fetch.
0	I_DECODE_ERR. Instruction address decode error. When set, this bit indicates that an instruction fetch was attempted to an address that was not decoded by the Protocol C instruction bus address decoder.

7.3 Global Registers

All these registers are 32 bits wide.

Table 7-3 Global Registers

Address	Register(s)	Read/Write	Description	32-Bit Reset Value
100	CHIP_ID	RO	Chip Device ID and Revision Number (refer to Section 7.3.1).	0002 0001
104 108	INT_STAT0 INT_STAT1	RO	These two 32-bit registers contain 64-bits of hardware and software generated interrupt conditions (refer to Section 7.3.2 and Section 7.3.3).	0000 0000
10C–110	Reserved			
114 118	INT_SET0 INT_SET1	WO	When a value is written to one of these 32-bit registers, each bit position containing a 1 causes the corresponding bit in the INT_STAT0 or INT_STAT1 register to be set, unless it is an I/O interrupt bit.	XXXX XXXX
11C–120	Reserved			
124 128	INT_CLR0 INT_CLR1	WO	When a value is written to one of these 32-bit registers, each bit position containing a 1 causes the corresponding bit in the INT_STAT0 or INT_STAT1 register to be cleared, unless it is an I/O interrupt bit.	XXXX XXXX
12C–130	Reserved			
134	GLOBAL_CTRL	R/W	This register contains miscellaneous control bits and values. The control bits act as global enable control for certain functions of the processor. Setting one of these enable bits to one acts as an overall enable for the function, but other registers are still involved in the detailed operation of the functions (refer to Section 7.3.4).	0000 0000

138	MT_ACTIVE	RO	This 32-bit register controls the thread's active / inactive status. The register has one status bit per thread; bit position equals thread number (bits 31:10 are reserved). Bits are cleared by a suspend instruction or write to MT_ACTIVE_CLR; bits are set by an unmasked interrupt or write to MT_ACTIVE_SET.
			
			    1: Active
			    0: Suspended
				0000 0001
				(Thread #0 active, all other threads inactive)

13C	MT_ACTIVE_SET	WO	When a value is written to this register, each bit position containing a 1 causes the corresponding bit in the MT_ACTIVE register to be set (bits 31:10 are reserved).	XXXX XXXX
140	MT_ACTIVE_CLR	WO	When a value is written to this register, each bit position containing a 1 causes the corresponding bit in the MT_ACTIVE register to be cleared (bits 31:10 are reserved).	XXXX XXXX

Table 7-3 Global Registers

Address	Register(s)	Read/Write	Description	32-Bit Reset Value
144	MT_DBG_ACTIVE	RO	This register is ANDed with MT_ACTIVE register, for purposes of thread scheduling. It prevents threads that are halted for debugging from being inadvertently reactivated by the occurrence of an interrupt. Bits in this register are cleared by break conditions or traps, or by single step operations or writing to MT_DBG_ACTIVE_CLR. Once cleared, they can only be set by software, by writing to the MT_DBG_ACTIVE_SET register. (Bits 31:10 are reserved.)	0000 0001 (Thread #0 debug active, all other threads debug inactive)
148	MT_DBG_ACTIVE_SET	WO	Writing a 1 into a given bit position causes the corresponding bit in the MT_DBG_ACTIVE register to be set (bits 31:10 are reserved).	XXXX XXXX
14C	MT_EN (Multithreading Enable)	R/W	Indicates which threads are currently enabled. If a thread is not enabled, then it cannot be allocated any execution slots. If it is enabled, it could either be suspended (inactive) or running. Clearing a thread's enable bit blocks further execution of the thread, but does not clear its bit in the MT_ACTIVE register. Bit position corresponds to thread number (bits 31:10 are reserved). 1: Enabled 0: Disabled	0000 0001 (Thread #0 enabled, all other threads disabled)
150	MT_HPRI (Multithreading High Priority Thread)	R/W	32-bit register that determines the priority of Non-Real-Time (NRT) threads as high or low priority. For HRT threads this bit is ignored. Bit position corresponds to thread number (bits 31:10 are reserved). 1: High Priority 0: Low Priority	0000 0001 (Thread #0 high priority, all other threads low priority)
154	MT_HRT (Multithreading Hard Real-Time Thread)	R/W	32-bit register that determines which threads are HRT and NRT: Bit position corresponds to thread number (bits 31:10 are reserved). 1: HRT 0: NRT	0000 0000
158	MT_BREAK (Multithreading BKPT executed)	RO	32-bit register that indicates which thread or threads are halted for a break condition of some sort (1 = halted). The BKPT instruction can cause an arbitrary number of threads to be halted, but only the bit for the thread executing the instruction is set in this register. It can therefore be read to determine which thread executed the BKPT instruction (bits 31:10 are reserved).	0000 0000
15C	MT_BREAK_CLR	WO	Writing a 1 to a given bit position in this register causes the corresponding bit in the multithreading break register to be cleared (bits 31:10 are reserved).	XXXX XXXX

Table 7-3 Global Registers

Address	Register(s)	Read/Write	Description	32-Bit Reset Value
160	MT_SINGLE_STEP	R/W	A 1 bit in this register causes the corresponding thread to be allocated exactly one pipeline slot the next time it is scheduled. After that, the scheduling hardware immediately clears its enable bit in the MT_DBG_ACTIVE register. This blocks further scheduling of the thread until its bit in the MT_DBG_ACTIVE register is set by writing a 1 to its position in the MT_DBG_ACTIVE_SET register (bits 31:10 are reserved).	0000 0000
164	MT_MIN_DELAY_EN (MT Minimum Delay Enable)	R/W	Specifies threads whose scheduling is constrained by the minimum delay interval specified in the GLOBAL_CTRL register. The minimum delay feature can be used to provide more deterministic behavior to threads whose coding leaves them subject to pipeline hazards. Writing a 1 to a given bit position in this register enables the minimum delay function for the corresponding thread (bits 31:10 are reserved).	0000 0000
168	MT_BREAK_SET	WO	Writing a 1 to any bit in this register causes the corresponding bit in the multithreading break register to be set (bits 31:10 are reserved).	XXXX XXXX
16C	Reserved			
170	DCAPT	R/W	Write Trap Address (refer to Section 7.3.5).	0000 0000
174–178	Reserved			
17C	MT_DBG_ACTIVE_CLR	WO	Writing a 1 into a given bit position causes the corresponding bit in the MT_DBG_ACTIVE register to be cleared.	XXXX XXXX
180	SCRATCHPAD0	R/W	Four scratch-pad registers. Cleared at power-on and by any other reset.	0000 0000
184	SCRATCHPAD1	R/W		
188	SCRATCHPAD2	R/W		
18C	SCRATCHPAD3	R/W		
190–1A0	Reserved			
1A4	MT_I_BLOCKED	RO	A 1 in a given bit position indicates that the corresponding thread is blocked due to an instruction fetch. Bits 31:10 are reserved.	0000 0000
1A8	MT_D_BLOCKED	RO	A 1 in a given bit position indicates that the corresponding thread is blocked due to a data access. Bits 31:10 are reserved.	0000 0000
1AC	MT_I_BLOCKED_SET	WO	Writing a 1 in a given bit position sets the corresponding bit in MT_I_BLOCKED. Writing a 0 has no effect. Bits 31:10 are reserved.	XXXX XXXX
1B0	MT_D_BLOCKED_SET	WO	Writing a 1 in a given bit position sets the corresponding bit in MT_D_BLOCKED. Writing a 0 has no effect. Bits 31:10 are reserved.	XXXX XXXX
1B4	MT_BLOCKED_CLR	WO	Writing a 1 in a given bit position clears the corresponding bits in MT_I_BLOCKED and MT_D_BLOCKED. Writing a 0 has no effect. Bits 31:10 are reserved.	XXXX XXXX

Table 7-3 Global Registers

Address	Register(s)	Read/Write	Description	32-Bit Reset Value
1B8	MT_TRAP_EN	R/W	Writing a 1 in a given bit position enables traps for the corresponding thread. Writing a 0 disables traps for that thread. Bits 31:10 are reserved.	0000 0000
1BC	MT_TRAP	RO	A 1 in a given bit position indicates that the corresponding thread has been trapped. A 0 indicates that the thread has not been trapped. Bits 31:10 are reserved.	0000 0000
1C0	MT_TRAP_SET	WO	Writing a 1 in a given bit position sets the corresponding bit in MT_TRAP. Writing a 0 has no effect. Bits 31:10 are reserved.	XXXX XXXX
1C4	MT_TRAP_CLR	WO	Writing a 1 in a given bit position clears the corresponding bit in MT_TRAP. Writing a 0 has no effect. Bits 31:10 are reserved.	XXXX XXXX
1C8-1FC	Reserved			
200	I_RANGE0_HI	R/W	Instruction space memory range 0 high (see Note 1).	FFFF FFFC
204	I_RANGE1_HI	R/W	Instruction space memory range 1 high (see Note 1).	FFFF FFFC
208	I_RANGE2_HI	R/W	Instruction space memory range 2 high (see Note 1).	FFFF FFFC
20C-21C	Reserved			
220	I_RANGE0_LO	R/W	Instruction space memory range 0 low (see Note 1).	0000 0000
224	I_RANGE1_LO	R/W	Instruction space memory range 1 low (see Note 1).	0000 0000
228	I_RANGE2_LO	R/W	Instruction space memory range 2 low (see Note 1).	0000 0000
22C-23C	Reserved			
240	I_RANGE0_EN	R/W	Writing a 1 in a given bit position enables instruction access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
244	I_RANGE1_EN	R/W	Writing a 1 in a given bit position enables instruction access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
248	I_RANGE2_EN	R/W	Writing a 1 in a given bit position enables instruction access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
24C-25C	Reserved			
260	D_RANGE0_HI	R/W	Data space memory range 0 high (see Note 1).	FFFF FFFC
264	D_RANGE1_HI	R/W	Data space memory range 1 high (see Note 1).	FFFF FFFC
268	D_RANGE2_HI	R/W	Data space memory range 2 high (see Note 1).	FFFF FFFC
26C	D_RANGE3_HI	R/W	Data space memory range 3 high (see Note 1).	FFFF FFFC
270-27C	Reserved			
280	D_RANGE0_LO	R/W	Data space memory range 0 low (see Note 1).	0000 0000
284	D_RANGE1_LO	R/W	Data space memory range 1 low (see Note 1).	0000 0000
288	D_RANGE2_LO	R/W	Data space memory range 2 low (see Note 1).	0000 0000
28C	D_RANGE3_LO	R/W	Data space memory range 3 low (see Note 1).	0000 0000
290-29C	Reserved			

Table 7-3 Global Registers

Address	Register(s)	Read/Write	Description	32-Bit Reset Value
2A0	D_RANGE0_EN	R/W	Writing a 1 in a given bit position enables data access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
2A4	D_RANGE1_EN	R/W	Writing a 1 in a given bit position enables data access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
2A8	D_RANGE2_EN	R/W	Writing a 1 in a given bit position enables data access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
2AC	D_RANGE3_EN	R/W	Writing a 1 in a given bit position enables data access to this address range for the corresponding thread. Writing a 0 disables it. Bits 31:10 are reserved. Refer to Section 7.3.6.	0000 03FF
2B0-3FC	Reserved			

Notes:
1. The address information for these range registers is contained in bits [31:2]. Bits [1:0] are reserved.

For those global registers that have special functions assigned to bits or fields within the register, the definitions of those bits and fields are described below.

7.3.1   CHIP_ID

Bits	Description	Read-Only Value
31:16	Chip ID	0002
15:0	Revision number	0001

7.3.2   INT_STAT0

Bits	Description
31	Asynchronous error interrupt
30	Real-time timer interrupt
29	I/O port C, interrupt 2
28	I/O port B, interrupt 2
27	I/O port B, interrupt 1
26	I/O port B, interrupt 0
25	I/O port A, interrupt 2
24	I/O port A, interrupt 1
23:0	Status for 24 software interrupts

7.3.3 INT_STAT1

Bits	Description
31	Breakpoint / trap interrupt
30	Debug port interrupt
29	I/O port E, interrupt 2
28	I/O port E, interrupt 1
27	I/O port E, interrupt 0
26	I/O port D, interrupt 2
25	I/O port D, interrupt 1
24	I/O port D, interrupt 0
23	I/O Port I, Interrupt 2
22	I/O Port I, Interrupt 1
21	I/O Port I, Interrupt 0
20	I/O Port H, Interrupt 2
19	I/O Port G, Interrupt 2
18	I/O Port F, Interrupt 2
17	I/O Port F, Interrupt 1
16	I/O Port F, Interrupt 0
15	I/O USB Port, Interrupt 2
14:10	Reserved interrupts
9:0	10 system timer interrupts

I/O port interrupt bits in INT_STAT0 and INT_STAT 1 give the status of the associated interrupt bits in I/O blocks and can not be set or cleared directly using INT_SET1 or INT_CLR1. These bits correspond to port interrupt conditions as follows:

- Interrupt 0 – Receive FIFO high watermark condition
- Interrupt 1 – Transmit FIFO low watermark condition
- Interrupt 2 – All other port interrupt conditions

7.3.4 GLOBAL_CTRL

Bits	Description
31:10	Reserved
9	TRAP_RST_EN. Trap Reset Enable. If this bit is set, then the occurrence of any enabled trap will cause a chip reset.
8	AERROR_RST_EN. Asynchronous Error Reset Enable. If this bit is set, then the occurrence of any Protocol C asynchronous error (on the instruction port or the data port) will cause a chip reset.
7	Reserved

Bits	Description
6:3	MT_MIN_DELAY. Minimum Instruction Delay. The minimum number of clock cycles between successive execution slots for a thread. This setting applies to threads whose bit is set in the MT_MIN_DELAY_EN register.
2	HRT_BANK_Select. Selects one of two HRT tables. This bit is sampled only when an end-of-table bit is encountered in the HRT table. This ensures that the current table is always completed before a new one is started. 0 selects Table 0 1 selects Table 1
1	Reserved
0	INT_EN. Interrupt Enable. 0 disables interrupts. 1 enables interrupts.

7.3.5 DCAPT (Trap Address)

The DCAPT register contains the Write Trap Address. This address applies only to general destinations and data register-only restricted destinations.

If a write is attempted to the address specified by DCAPT, a trap is triggered and the DCAPT bit is set in the TRAP_CAUSE register for the participating thread.

The format of the DCAPT register depends on whether the register specifies an address in the direct address space or in the indirect or program memory address space.

For a direct address, the format is as follows:

Bits	Description
31:14	Must be 0.
13:10	Thread number (0 for global registers).
9:2	Register number (8 most significant bits of the register address).
1	Reserved
0	Equal to 1 to specify direct address space.

For an indirect address, the format is as follows:

Bits	Description
31:2	Indirect address
1	Reserved
0	Equal to 0 to specify indirect address space.

The DCAPT register is always enabled. However, it can be loaded with a value that never matches. A value that

can never match is one that has bit 0 equal to 1 and bits 31:14 not equal to all 0's; for example: 0x8000 0001.

7.3.6 Memory Protection

Any thread is permitted to read and write to any direct space register. However, execution from instruction space and reads and writes to the indirect space are permitted only if these operations are enabled for the executing thread and the address being accessed. There are three pairs of address range registers for the instruction space (I_RANGE[0:2]_HI/LO) and four pairs for the data space (D_RANGE[0:3]_HI/LO). Additionally, each range pair has a corresponding active-high thread enable mask (I_RANGE[0:2]_EN) or (D_RANGE[0:3]_EN). If an instruction fetch or indirect space data access falls within the inclusive range defined by one of the appropriate range pairs, and that range is enabled for the executing thread, then the access is permitted. Otherwise, it is trapped, with one (or more) of three possible causes set in the executing thread's TRAP_CAUSE register:

- DST_RANGE_ERR
- SRC1_RANGE_ERR
- I_RANGE_ERR

7.4 HRT Tables

There are two HRT tables:

- HRT0 at addresses 00000800–0000083C
- HRT1 at addresses 00000900–0000093C

At any given time, one of the two HRT tables is active and being used by the CPU; the other is available for updates. The HRT Table Select bit in the GLOBAL_CTRL register determines which is the active table.

Each HRT table is composed of 64 8-bit entries. Four 8-bit entries are packed into a 32-bit word, as shown in Table 7-4.

Table 7-4 HRT Word (Four HRT Table Entries)

Name	Bits	Read/Write	Description	Reset value
HRT_END0	31	RW	1 indicates end of HRT table	C0C0C0C0
HRT_NO_THREAD0	30	RW	1 indicates unoccupied entry	
			0 indicates occupied entry	
	29:28	Reserved		
HRT_TNUM0	27:24	RW	HRT thread number	
HRT_END1	23	RW	1 indicates end of HRT table	
HRT_NO_THREAD1	22	RW	1 indicates unoccupied entry	
			0 indicates occupied entry	
	21:20	Reserved		
HRT_TNUM1	19:16	RW	HRT thread number	
HRT_END2	15	RW	1 indicates end of HRT table	
HRT_NO_THREAD2	14	RW	1 indicates unoccupied entry	
			0 indicates occupied entry	
	13:12	Reserved		
HRT_TNUM2	11:8	RW	HRT thread number	
HRT_END3	7	RW	1 indicates end of HRT table	
HRT_NO_THREAD3	6	RW	1 indicates unoccupied entry	
			0 indicates occupied entry	
	5:4	Reserved		
HRT_TNUM3	3:0	RW	HRT thread number	

7.5   On-Chip Peripherals

The On-Chip Peripherals registers reside in the indirect address space at locations 0100 0000 – 0100 0FFC.

This 4K byte range provides room for 16 peripherals, each having 256 bytes (64 x 4-byte registers) available.

Address ranges for the on-chip peripherals are as follows:

Address Range	Peripheral
0100 0000 – 0100 00FC	General Configuration
0100 0100 – 0100 01FC	Timer Registers
0100 0200 – 0100 02FC	TRNG — True Random Number Generator
0100 0300 – 0100 03FC	Debug Port
0100 0400 – 0100 04FC	Security Module
0100 0500 – 0100 05FC	ICCR — Instruction Cache Control Registers
0100 0600 – 0100 06FC	DCCR — Data Cache Control Registers
0100 0700 – 0100 07FC	OCMC — On-Chip Memory Control
0100 0800 – 0100 08FC	Statistics Counters
0100 0900 – 0100 09FC	MTEST — Memory Test
0100 0A00 – 0100 0AFC	Reserved
0100 0B00 – 0100 0BFC	
0100 0C00 – 0100 0CFC	
0100 0D00 – 0100 0DFC	
0100 0E00 – 0100 0EFC	
0100 0F00 – 0100 0FFC	

Section 7.5.1 through Section 7.5.10 shows details for these groups of registers.

7.5.1   OCP General Configuration

This group of registers control the overall configuration of the on-chip peripheral block. Table 7-5 shows details for each register.

Table 7-5 OCP General Configuration Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0000	OCP_CLK_ CORE_CFG		Core PLL configuration register	R/W	8000 0080
		31	Security Module clock enable		
		30	Core PLL enable saturation.		
		29	Core PLL fast lock enable		
		28:23	Core PLL reference divider		
		22:11	Core PLL multiplier		
		10:8	Core PLL output divider		
		7	Core PLL reset 1: Resets the core PLL 0: Normal operation		
		6	Core PLL bypass 1: Puts the core PLL into bypass mode 0: Normal operation		
		5	Core PLL powerdown 1: Puts the core PLL into powerdown mode 0: Normal operation		
		4	Core clock source select 1: PLL is the source for clk_core 0: sys-refclk is the source for clk_core		
		3:0	Core clock forward divider. The PLL output is divided by (forward divider +1). This allows for divide values from 1 to 16.		

Table 7-5 OCP General Configuration Registers (continued)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0004	OCP_CLK_ IO_CFG		I/O PLL configuration register	R/W	0103 E080
		31	Reserved		
		30	I/O PLL enable saturation.		
		29	I/O PLL fast lock enable		
		28:23	I/O PLL reference divider		
		22:11	I/O PLL multiplier		
		10:8	I/O PLL output divider		
		7	I/O PLL reset 1: Resets the I/O PLL 0: Normal operation		
		6	I/O PLL bypass 1: Puts the I/O PLL into bypass mode 0: Normal operation		
		5	I/O PLL powerdown 1: Puts the I/O PLL into powerdown mode 0: Normal operation		
		4	Core clock source select 1: PLL is the source for clk_io 0: sys-refclk is the source for clk_io		
		3:0	Reserved		
0100 0008	OCP_CLK_ DDR_CFG		DDR PLL configuration register	R/W	0000 0080
		31	Reserved		
		30	DDR PLL enable saturation.		
		29	DDR PLL fast lock enable		
		28:23	DDR PLL reference divider		
		22:11	DDR PLL multiplier		
		10:8	DDR PLL output divider		
		7	DDR PLL reset 1: Resets the DDR PLL 0: Normal operation		
		6	DDR PLL bypass 1: Puts the DDR PLL into bypass mode 0: Normal operation		
		5	DDR PLL powerdown 1: Puts the DDR PLL into powerdown mode 0: Normal operation		
		4	DDR clock source select 1: PLL is the source for clk_ddr 0: sys-refclk is the source for clk_ddr		
		3:0	Reserved		

Table 7-5 OCP General Configuration Registers (continued)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 000C	OCP_CLK_ DDRDS_CFG		DDR Deskew PLL configuration register	R/W	0000 0080
		31:13	Reserved		
		12:11	DDR Deskew PLL multiplier. Use a value of 00 (multiply by 1) for correct operation.		
		10:8	Reserved		
		7	DDR Deskew PLL reset 1: Resets the DDR Deskew PLL 0: Normal operation		
		6	DDR Deskew PLL bypass 1: Puts the DDR Deskew PLL into bypass mode 0: Normal operation		
		5	DDR Deskew PLL powerdown 1: Puts the DDR Deskew PLL into powerdown mode 0: Normal operation		
		4	DDR Deskew clock source select 1: PLL is the source for clk_ddr_out 0: sys-refclk is the source for clk_ddr_out clk_ddr_out is the DDR clock signal on the DDR bus.		
		3:0	Reserved		
0100 0010	OCP_CLK_ SLIP_CLR	31:0	Reserved for test	WO	0000 0000
0100 0014	OCP_CLK_ SLIP_STAT	31:0	Reserved for test	RO	xxxx xxxx
0100 0030	DDR Cal Ctrl	31:0	DDR Calibrator control register	R/W	0000 0000
0100 0034	DDR Cal Stat	31:0	DDR Calibrator status register	RO	xxxx xxxx
0100 0038	USB DFT Ctrl	31:0	Reserved for test	R/W	0000 0000
0100 003C	USB DFT Stat	31:0	Reserved for test	RO	xxxx xxxx
0100 0080	SW Reset	31:1	Reserved	WO	0000 0000
		0	Software reset. Writing a 1 generates a one-cycle pulse to the reset module. Writing a 0 has no effect.		

Table 7-5 OCP General Configuration Registers (continued)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0084	Reset Reasons		Reasons for the last chip reset	RO	xxxx xxxx
		31:19	Reserved		
		18	Destination memory protection error		
		17	Source1 memory protection error		
		16	Write address watchpoint trap		
		15	Destination synchronous error trap		
		14	Source1 synchronous error trap		
		13	Destination operand misalignment error trap		
		12	Source1 operand misalignment error trap		
		11	Destination address decode error trap		
		10	Source1 address decode error trap		
		9	Illegal instruction trap		
		8	Instruction synchronous error trap		
		7	Instruction address decode error trap		
		6	Data port asynchronous error		
		5	Instruction port asynchronous error		
		4	Software reset		
		3	Debug port reset		
		2	Watchdog timer reset		
		1	Power-on reset		
		0	External reset		
0100 0088	Reserved	31:0	Reserved	R/W	0000 0000
0100 008C	IO PU Config		I/O driver cell Power Up Enable signal. When LOW, the receiver circuits are disabled and draw low current. (Note: The pads are held disabled during reset, except in Analog Test Mode / Scan.)	R/W	0000 0007
		31:3	Reserved		
		2	Port H (DDR high group) PU control bit.		
		1	Port G (DDR low group) PU control bit.		
		0	Port F (HSTL Pad type) PU control bit.		
0100 0090	Reserved	31:0	Reserved	R/W	0000 0000

7.5.2   Timer Registers

Table 7-6 Timer Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0100	MPT_VAL	31:0	The value of the Multipurpose Timer	RO	0000 0000
0100 0104	RTCOM	31:0	Real-Time Timer Compare Register	R/W	0000 0000
0100 0108	TKEY	31:0	Timer Block's security key code. If TKEY = 0xa1b2c3d4, then the user can write to WD_COM and WD_CFG.	R/W	0000 0000
0100 010C	WD_COM	31:0	Watchdog Compare Register. This register can be written only if TKEY has the correct value.	R/W	0000 0000
0100 0110	WD_CFG	31:0	Watchdog Configuration Register. This register can be written only if TKEY has the correct value. 0x4d3c2b1a: Disable the Watchdog Register compare. Other value: Enable the Watchdog Register compare.	R/W	4d3c 2b1a
0100 0114	SYSVAL	31:0	The value of the System Timer	RO	0000 0000
0100 0118	SYS_COM_0	31:0	System Timer Compare Register 0 (INT_STAT1[0])	R/W	0000 0000
0100 011C	SYS_COM_1	31:0	System Timer Compare Register 1 (INT_STAT1[1])	R/W	0000 0000
0100 0120	SYS_COM_2	31:0	System Timer Compare Register 2 (INT_STAT1[2])	R/W	0000 0000
0100 0124	SYS_COM_3	31:0	System Timer Compare Register 3 (INT_STAT1[3])	R/W	0000 0000
0100 0128	SYS_COM_4	31:0	System Timer Compare Register 4 (INT_STAT1[4])	R/W	0000 0000
0100 012C	SYS_COM_5	31:0	System Timer Compare Register 5 (INT_STAT1[5])	R/W	0000 0000
0100 0130	SYS_COM_6	31:0	System Timer Compare Register 6 (INT_STAT1[6])	R/W	0000 0000
0100 0134	SYS_COM_7	31:0	System Timer Compare Register 7 (INT_STAT1[7])	R/W	0000 0000
0100 0138	SYS_COM_8	31:0	System Timer Compare Register 8 (INT_STAT1[6])	R/W	0000 0000
0100 013C	SYS_COM_9	31:0	System Timer Compare Register 9 (INT_STAT1[7])	R/W	0000 0000

7.5.3   True Random Number Generator

Table 7-7 True Random Number Generator (TRNG) Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0200	TRNG_CFG		True Random Number Generator (TRNG) configuration	R/W	0000 0000
		31:6	Reserved	RO	
		5	Output of TRNG slow oscillator	RO	
		4	Output of TRNG medium oscillator	RO	
		3	Output of TRNG fast oscillator	RO	
		2	Enable TRNG slow oscillator	R/W	
			1: Enable		
			0: Disable		
		1	Enable TRNG medium oscillator	R/W	
			1: Enable		
			0: Disable		
		0	Enable TRNG fast oscillator	R/W	
			1: Enable		
			0: Disable		
0100 0204	TRNG_VAL	31:0	32-bit True Random Number Generator (TRNG) value	RO	FFFF FFFF

7.5.4   Debug Port

Table 7-8 Debug Port (DBG) Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0300	INMAIL	31:0	Incoming Mailbox. Contains data from the external host.	RO	xxxx xxxx
0100 0304	OUTMAIL	31:0	Outgoing Mailbox. Contains data from the processor.	WO	xxxx xxxx
0100 0308	MAIL_STAT		Mailbox Status	RO	5000 0000
		31	Incoming mailbox full. Resets to 0 (not full).		
		30	Incoming mailbox empty. Resets to 1 (empty).		
		29	Outgoing mailbox full. Resets to 0 (not full).		
		28	Outgoing mailbox empty. Resets to 1 (empty).		
		27:0	Reserved. Return 0s on read.		

7.5.5   Security Module

Table 7-9 Security Module Registers

Address	Register Name	Bits	Description	Read/Write	32-Bit Reset Value
0100 0400	Security Control		Security Module Control Register		0000 0000
		31:24	Security Module Reset Command. Code 0xAB causes "soft reset", which is the same as a hardware reset, except for changing the Security Control register.	R/W	
		23:22	Reserved. Returns 0 when read.	RO	
		21:20	SEC_OUT_OFFSET. Byte offset for output data. Used to reduce alignment overhead in software.	R/W	
		19:18	Reserved. Returns 0 when read.	RO	
		17:16	SEC_IN_OFFSET. Byte offset for input data. Used to reduce alignment overhead in software.	R/W	
		15:10	Reserved. Returns 0 when read.	RO	
		9:8	SEC_KEY_SIZE. Key size options. See Note 1. For AES cypher: 00: 128 bits 01: 192 bits 10: 256 bits 11: Reserved	R/W	
		7:6	Reserved. Returns 0 when read.	RO	
		5:4	SEC_HASH_TYPE: 00: None 01: MD5 10: SHA1 11: Reserved	R/W	
		3	SEC_CBC (supported for AES only): 1: CBC 0: No CBC	R/W	
		2:1	SEC_ALGORITHM: 00: AES 01: None 10: DES 11: Reserved	R/W	
		0	SEC_DIR. Security Direction: 1: Encypher 0: Decypher	R/W	
0100 0404	Security Status		Security Module Status Register	RO	
		31:1	Reserved. Returns 0 when read.	RO	xxxx xxxx
		0	SEC_BUSY: 1: Security Module is busy cyphering. 0: Security Module is idle.	R/W	0x0

Table 7-9 Security Module Registers (continued)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0410	SEC_KEY_0	31:0	Key register, bits [255:224]	R/W	0000 0000
0100 0414	SEC_KEY_1	31:0	Key register, bits [223:192]	R/W	0000 0000
0100 0418	SEC_KEY_2	31:0	Key register, bits [191:160]	R/W	0000 0000
0100 041C	SEC_KEY_3	31:0	Key register, bits [159:128]	R/W	0000 0000
0100 0420	SEC_KEY_4	31:0	Key register, bits [127:96]	R/W	0000 0000
0100 0424	SEC_KEY_5	31:0	Key register, bits [95:64]	R/W	0000 0000
0100 0428	SEC_KEY_6	31:0	Key register, bits [63:32]	R/W	0000 0000
0100 042C	SEC_KEY_7	31:0	Key register, bits [31:0]	R/W	0000 0000
0100 0430	SEC_IN_0	31:0	Input register, bits [159:128]	R/W	0000 0000
0100 0434	SEC_IN_1	31:0	Input register, bits [127:96]	R/W	0000 0000
0100 0438	SEC_IN_2	31:0	Input register, bits [95:64] (Write here to start DES.)	R/W	0000 0000
0100 043C	SEC_IN_3	31:0	Input register, bits [63:32]	R/W	0000 0000
0100 0440	SEC_IN_4	31:0	Input register, bits [31:0] (Write here to start AES.)	R/W	0000 0000
0100 0450	SEC_OUT_0	31:0	Output register, bits [159:128]	RO	xxxx xxxx
0100 0454	SEC_OUT_1	31:0	Output register, bits [127:96]	RO	xxxx xxxx
0100 0458	SEC_OUT_2	31:0	Output register, bits [95:64]	RO	xxxx xxxx
0100 045C	SEC_OUT_3	31:0	Output register, bits [63:32]	RO	xxxx xxxx
0100 0460	SEC_OUT_4	31:0	Output register, bits [31:0]	RO	xxxx xxxx
0100 0470	HASH_OUT_0	31:0	Hash register, bits [159:128]	R/W	0000 0000
0100 0474	HASH_OUT_1	31:0	Hash register, bits [127:96]	R/W	0000 0000
0100 0478	HASH_OUT_2	31:0	Hash register, bits [95:64]	R/W	0000 0000
0100 047C	HASH_OUT_3	31:0	Hash register, bits [63:32]	R/W	0000 0000
0100 0480	HASH_OUT_4	31:0	Hash register, bits [31:0]	R/W	0000 0000
Notes: 1. DES uses a 64-bit key. For AES, the key size is selected by bits [9:8].					

7.5.6   Instruction Cache Control Registers

Table 7-10 Instruction Cache Control Registers (ICCR)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0500	ICADDR	31:0	Instruction Cache Address	R/W	xxxx xxxx
0100 0504	ICRDD	31:0	Instruction Cache Read Data	R/W	xxxx xxxx
0100 0508	ICWRD	31:0	Instruction Cache Write Data	R/W	xxxx xxxx
0100 050C	ICSTAT		Instruction Cache Status		0000 0000
		31:0	Reserved. Return 0s on read.	RO	
		0	Cache Bus Error	R/W	
0100 0510	ICCTRL		Instruction Cache Control		0000 0000
		31:8	Reserved. Return 0s on read.	RO	
		7:4	OP: ICCR Operation	R/W	
		3	V: ICCR Operation Valid	R/W	
		2	R: Instruction Cache Reset (software)	R/W	
		1	Reserved. Returns 0 on read.	RO	
		0	D: ICCR Operation Done	RO	

7.5.7   Data Cache Control Registers

Table 7-11 Data Cache Control Registers (DCCR)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0600	DCADDR	31:0	Data Cache Address	R/W	xxxx xxxx
0100 0604	DCRDD	31:0	Data Cache Read Data	R/W	xxxx xxxx
0100 0608	DCWRD	31:0	Data Cache Write Data	R/W	xxxx xxxx
0100 060C	DCSTAT		Data Cache Status		0000 0000
		31:2	Reserved. Return 0s on read.	RO	
		1	WIDLE: Data Cache Write Queue is idle.	R/W	
		0	Cache Bus Error	R/W	
0100 0610	DCCTRL		Data Cache Control		0000 0000
		31:8	Reserved. Return 0s on read.	RO	
		7:4	OP: DCCR Operation	R/W	
		3	V: DCCR Operation Valid	R/W	
		2	R: Data Cache Reset (software)	R/W	
		1	Reserved. Returns 0 on read.	RO	
		0	D: DCCR Operation Done	RO	

7.5.8   On-Chip Memory Control Registers

Table 7-12 On-Chip Memory Control (OCMC) Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0700	OCM_CFG		On-Chip Memory Configuration		0000 003F
		31:6	Reserved	RO	

		5:0	OCM_BANK_MASK. Code/Data Bank Mask register. Each bit corresponds to an SRAM bank in the OCM. LSB corresponds to the bank mapped to the lower OCM addresses. MSB corresponds to the bank mapped to the upper OCM addresses.
			  1:Code Bank
			  0:Data Bank
				R/W	

0100 0704	OCM_BIST_ CFG		On-Chip Memory Built-In Self Test Configuration		0000 0000
		31:13	Reserved	RO	
		12	WSI. Wrapper serial input. This terminal is used for scan-in of wrapper instructions and data.	R/W	
		11	UpdateWR. Update wrapper register. Asserting this signal while CaptureWR and ShiftWR signals are inactive, enables update operation for the selected wrapper register.	R/W	
		10	SelectWIR. Select wrapper instruction register. Asserting this signal selects the wrapper instruction register (WIR). Deasserting this signal selects a data register.	R/W	
		9	ShiftWR. Shift wrapper register. Asserting this signal while CaptureWR and UpdateWR signals are inactive, enables shift operation for the selected wrapper register.	R/W	
		8	CaptureRW. Capture wrapper register. Asserting this signal while ShiftWR and UpdateWR signals are inactive, enables capture operation for the selected wrapper register.	R/W	
		7	WRCK. Wrapper clock. This is the clock input of P1500 interface. All P1500 interface registers operate at the frequency of this clock. WRCK max frequency is 1/10th of the core clock frequency.	R/W	
		6	WRST. This register bit is connected through an inverter to WRSTN. The signal must be asserted for at least one WRCK cycle. Assertion of this signal will put the P1500 interface into bypass mode. Assertion will also put the memory into normal operation. The reset condition of this bit does not reset the SMS or P1500 interface.	R/W	
		5	reset_sms_a. This pin is connected to reset_sms_a. The signal must be asserted for at least five core clock cycles. Assertion of this signal will reset the SMS controller. Assertion will also put the memory into normal operation. The reset condition of this bit does not reset the SMS controller.	R/W	

Table 7-12 On-Chip Memory Control (OCMC) Registers (continued)

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0704	OCM_BIST_ CFG (continued)	4	smart. Smart mode. This signal is used to put the SMS into smart mode. The smart mode signal needs to be held high for a minimum of five core clock cycles. run_bist and/or soft_repair needs to be asserted before the smart signal is asserted. Before the SMS controller can execute smart mode the SMS controller needs to be reset, or a bypass instruction needs to be loaded into the WIR.	R/W	
		3	run_bist. Execute BIST_RUN instruction. High level of this signal enables execution of BIST_RUN instruction during smart mode. The run_bist signal should be asserted before the smart signal is asserted. The signal can be deasserted after the ready_sms OCM_BIST_STAT register bit has gone high, signaling the end of smart mode.	R/W	
		2	soft_repair. Perform soft repair. High level of this signal enables sequential execution of BIST_RUN and BISR_RUN instructions during smart mode. The soft_repair signal should be asserted before the smart signal is asserted. The signal can be deasserted after the ready_sms status register bit has gone high, signaling the end of smart mode.	R/W	
		1	ext_biste. External BIST enable. This signal is used for switching between memory functional pins and memory test pins. High level of this signal enables memory test pins. This signal is ORed with an internal signal having the same function and coming from the STAR processor.	R/W	
		0	dft_mode. Used during ATPG tests to keep memory safe. High level of this signal sets RM in default value, and TEST1, AWT, RSCEN and BISTE inputs into inactive state.	R/W	
0100 0708	OCM_BIST_ STAT		On-Chip Memory Built-In Self Test Status	RO	0000 000x (see Note 1).
		31:4	Reserved		
		3	ready_sms. This signal indicates that the sms controller or the P1500 controller is ready to take a command. When the signal is asserted ready_sms also indicates that an sms controller or a P1500 command has completed.		
		2	fail_sms. When this signal is asserted and ready_sms is asserted it indicates that the memory failed the sms or P1500 command.		
		1	WSOR. Wrapper serial output. This terminal is used for scan-out of wrapper instructions and data. It is triggered by the rising edge of the wrapper clock WRCK.		
		0	WSO. Wrapper serial output. This terminal is used for scan-out of wrapper instructions and data. It is triggered by the falling edge of the wrapper clock WRCK.		

Notes:
1. OCM_BIST_STAT[3:0] values remain unknown until the OCM BIST controller is explicitly reset by software. Then:
   OCM_BIST_STAT[3] = 1
   OCM_BIST_STAT[2] = 0
   OCM_BIST_STAT[1] = Unknown
   OCM_BIST_STAT[0] = Unknown

7.5.9   Statistics Counters

The statistics counters can be used to observe and analyze the performance of the IP51xx. There are four sets of registers available for collecting statistics. Each set consists of a 32-bit configuration register (STS_CFGn) and an associated 32-bit counter (STS_CNTn). Each configuration register selects one of 32 events to be tracked by its associated counter.

Table 7-13 shows the address map for the four statistics counters. Table 7-14 lists the events that can be selected by a configuration register.

Table 7-13 Statistics Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0800	STS_CFG0		Statistics Configuration Register 0		0000 0000
		31:5	Reserved. Return 0s on read.	RO	
		4:0	Selects an event for STS_CNT0	R/W	
0100 0804	STS_CNT0	31:0	Statistics Counter 0	RO	xxxx xxxx
0100 0808	STS_CFG1		Statistics Configuration Register 1		0000 0000
		31:5	Reserved. Return 0s on read.	RO	
		4:0	Selects an event for STS_CNT1	R/W	
0100 080C	STS_CNT1	31:0	Statistics Counter 1	RO	xxxx xxxx
0100 0810	STS_CFG2		Statistics Configuration Register 2		0000 0000
		31:5	Reserved. Return 0s on read.	RO	
		4:0	Selects an event for STS_CNT2	R/W	
0100 0814	STS_CNT2	31:0	Statistics Counter 2	RO	xxxx xxxx
0100 0818	STS_CFG3		Statistics Configuration Register 3		0000 0000
		31:5	Reserved. Return 0s on read.	RO	
		4:0	Selects an event for STS_CNT3	R/W	
0100 081C	STS_CNT3	31:0	Statistics Counter 3	RO	xxxx xxxx

Table 7-14 Selectable Events for Statistics Counters

Event Number	Event Source	Description
0	Main Processor	Instruction issued
1-3	Main Processor	Reserved
4	On-Chip Memory	On-chip memory access
5	On-Chip Memory	On-chip memory request not acknowledged
6-7	On-Chip Memory	Reserved
8-12		Reserved
13	Instruction Cache	Instruction cache request - not validated
14	Instruction Cache	Instruction cache miss - not validated
15	Instruction Cache	Instruction cache request not acknowledged - not validated
16	Instruction Cache	Instruction cache request - validated
17	Instruction Cache	Instruction cache miss - validated
18	Instruction Cache	Instruction cache request not acknowledged - validated
19	Instruction Cache	Instruction cache miss queue not empty
20	Data Cache	Data cache read request

Table 7-14 Selectable Events for Statistics Counters (continued)

Event Number	Event Source	Description
21	Data Cache	Data cache read miss
22	Data Cache	Data cache write request
23	Data Cache	Data cache write miss
24	Data Cache	Data cache miss queue not empty
25	Data Cache	Data cache write buffer full
26	Data Cache	Data cache request not acknowledged
27	Data Cache	Data cache core request
28	Data Cache	Data cache miss
29	Data Cache	Data cache evict
30		Always active event
31		Null event — never occurs

7.5.10   Memory Test Registers

Table 7-15 Memory Test Registers

Address	Register Name	Bits	Description	Read/ Write	32-Bit Reset Value
0100 0900	MTESTADDR	31:0	Memory Address for memory test. Memory word accurate.	R/W	xxxx xxxx
0100 0904	MTESTWD	31:0	Memory Test Write Data	R/W	xxxx xxxx
0100 0908	MTESTRD	31:0	Memory Test Read Data	R/W	xxxx xxxx
0100 090C	MTESTCTRL		Memory Test Control Register		0000 0000
		31	MTEST. Memory Test mode enable	R/W	

		30	WR_EN. Write Enable. Specifies which operation to perform. Meaningful only when MTESTCTRL[31] = 1.
			    1: Write
			    0: Read
				R/W	

		29:12	Reserved. Return 0s on read.	RO	
		11:8	WORD_SEL. Word Select. Select word in a wide memory.	R/W	
		7:4	GROUP_SEL. Group Select. Select a group of memories.	R/W	
		3:0	MEM_SEL. Memory Select. Select a memory inside a group.	R/W	

7.6   Per-Port Registers

Each port has a set of registers that constitute the primary input, output, and control interfaces for that port.

The I/O port registers reside in locations 0200 0000 to 0200 FFFC in the indirect address space. Table 7-16 shows the base addresses. For each port there is a set of non-blocking registers located at the base address, and a set of blocking registers located at an offset of 800.

The non-blocking registers are used in normal operation. Any instructions from a thread that target a non-blocking region are completed normally, without causing the thread to be blocked.

Some functions on some ports also use blocking registers. The blocking registers have two primary uses:

- To configure an I/O controller for use with a particular thread, as, for example, during initialization.

- To handle situations in which an I/O controller cannot execute a read operation within the required two clock cycles. In that case execution of instructions on a thread can be blocked temporarily, and resumed when the requested data is available.

Within a given port's blocking region, the starting offset is determined by the selected I/O function of that port. The organization of these blocking regions is very specific to the devices involved, and will be discussed in the context of each I/O module.

Table 7-17 shows the set of registers available in the non-blocking region for each port. The address of each register is the sum of the port base address (Table 7-16) and the register address offset (Table 7-17).

The non-blocking registers for the various ports are similar in number, general format, and general usage, but differ in particular respects from port to port and from function to function within each port.

Some functions are common to all or most of the ports; and those functions are controlled by corresponding registers and register fields in all the ports. The common functions are primarily those dealing with function selection and FIFO management.

Table 7-17 shows the basic address map for the per-port registers. Table 7-18 gives more detail, and indicates which registers are generic, and which are I/O port dependent.

Table 7-16 I/O Port Base Addresses

I/O Port	Address Range	Offset	Description
A	0200 0000 – 0200 0FFC	000	I/O Port A non-blocking region
		800	I/O Port A blocking region
B	0200 1000 – 0200 1FFC	000	I/O Port B non-blocking region
		800	I/O Port B blocking region
C	0200 2000 – 0200 2FFC	000	I/O Port C non-blocking region
		800	I/O Port C blocking region
D	0200 3000 – 0200 3FFC	000	I/O Port D non-blocking region
		800	I/O Port D blocking region
E	0200 4000 – 0200 4FFC	000	I/O Port E non-blocking region
		800	I/O Port E blocking region
F	0200 5000 – 0200 5FFC	000	I/O Port F non-blocking region
		800	I/O Port F blocking region
G	0200 6000 – 0200 6FFC	000	I/O Port G non-blocking region
		800	I/O Port G blocking region
H	0200 7000 – 0200 7FFC	000	I/O Port H non-blocking region
		800	I/O Port H blocking region
I	0200 8000 – 0200 8FFC	000	I/O Port I non-blocking region
		800	I/O Port I blocking region
USB	0200 9000 – 0200 9FFC	000	USB Port non-blocking region
		800	USB Port blocking region

The value of bits 2:0 of the Function register (see Table 7-18) determines which of the four functions (Function 0, 1, 2, or 3) is controlled by the set of registers shown in Table 7-17. One set of these registers exists for each I/O port.

Table 7-17 I/O Port Non-Blocking Region Register Map

Offset	Register Name	Description	Read/ Write	32-Bit Reset Value
00	Function	Function select, reset, and FIFO configuration	R/W	0000 0000
04	GPIO Ctrl	GPIO output enable	R/W	0000 0000
08	GPIO Out	GPIO data output	R/W	0000 0000
0C	GPIO In	GPIO data input	RO	xxxx xxxx
10	Interrupt Status	Interrupt status from selected function	RO	xxxx xxxx
14	Interrupt Mask	Interrupt mask	R/W	0000 0000
18	Interrupt Set	Interrupt set and one-cycle pulse generation	WO	xxxx xxxx
1C	Interrupt Clear	Interrupt clear	WO	xxxx xxxx
20	Transmit FIFO LO	Transmit FIFO low word, bits [31:0]	WO	xxxx xxxx
24	Transmit FIFO HI	Transmit FIFO high word, bits [63:32]	WO	xxxx xxxx
28	Receive FIFO LO	Receive FIFO low word, bits [31:0]	RO	xxxx xxxx
2C	Receive FIFO HI	Receive FIFO high word, bits [63:32]	RO	xxxx xxxx
30	Function Ctrl 0	Function control register 0	R/W	0000 0000
34	Function Ctrl 1	Function control register 1	R/W	0000 0000
38	Function Ctrl 2	Function control register 2	R/W	0000 0000
3C	Function Status 0	Function status register 0	RO	xxxx xxxx
40	Function Status 1	Function status register 1	RO	xxxx xxxx
44	Function Status 2	Function status register 2	RO	xxxx xxxx
48	FIFO Watermark	FIFO watermark trigger level	R/W	0000 0000
4C	FIFO Level	FIFO current level	RO	xxxx xxxx
50	GPIO Mask	GPIO mask	RO	0000 0000

Table 7-18 gives more detailed definitions for the registers listed in Table 7-17. Some of these registers are generic — that is, they have the same definition for all I/O ports and functions. For these registers, detailed definitions are given in this table. Some of the other registers in this table depend on the specific port and function. Details for those registers are given in the sections covering the individual ports, beginning in Section 7.7.

Table 7-18 Generic I/O Port Register Definitions (Non-Blocking Region)

Offset	Register Name	Bits	Description	Read/ Write	Reset Value
00	Function		Function select, reset, and FIFO configuration		
		31:22	Reserved	RO	0x000
		21	RX FIFO TNUM EN. Enables Receive FIFO LO register thread number checking. When this bit is set, the value programmed in the RX FIFO TNUM field is checked against the thread number of each request that attempts to access the Receive FIFO LO register. Only requests from the matching thread are processed. Requests from other threads are dropped and would cause a synchronous error.	R/W	0x0
		20:16	RX FIFO TNUM. The number of the thread that is allowed to access the I/O port Receive FIFO LO register.	R/W	0x00
		15:13	Reserved	RO	0x0
		12:8	Blocking region thread number. The number of the thread that is allowed to access the blocking region for this port. If a function does not use blocking registers, then this field is reserved.	R/W	0x00
		7:4	Function reset. Asserting these bits resets the corresponding functions: Bit 7=1 resets Function 4 (not used in IP51xx). Bit 6=1 resets Function 3. Bit 5=1 resets Function 2. Bit 4=1 resets Function 1. There is no reset needed for Function 0. Note: These bits are static and must be deasserted in order to allow the functions to come out of reset.	R/W	0x0
		3	RX FIFO Select. Selects the receive FIFO to access: 1: Selects Receive FIFO 1 0: Selects Receive FIFO 0	R/W	0x0
		2:0	Function select: 0x0 selects Function 0. 0x1 selects Function 1. 0x2 selects Function 2. 0x3 selects Function 3. 0x4 to 0x7 are reserved. The reset value of this field is I/O port dependent.	R/W	See Table 7-19.
04	GPIO Ctrl	31:0	GPIO Output Enable. A 1 is any bit position enables the output buffer corresponding to that specific bit. A 0 disables it.	R/W	0x00000000
08	GPIO Out		GPIO Data Output. With the corresponding GPIO Ctrl bit enabled, the value in this register will be driven by the enabled output buffer.	R/W	0x00000000

Table 7-18 Generic I/O Port Register Definitions (Non-Blocking Region) (continued)

Offset	Register Name	Bits	Description	Read/Write	Reset Value
0C	GPIO In		GPIO Data Input. This register reflects the current state of the external I/O pins. The function of this register is unaffected by any other register settings.	RO	0xXXXXXXXX
10	Interrupt Status	31:16	Reserved	RO	0x00000000
		15	TX FIFO UF. Transmit FIFO Underflow interrupt — set when the transmit FIFO is read from while it is empty.		
		14	TX FIFO WM. Transmit FIFO Watermark interrupt — set when the transmit FIFO contains a number of entries equal to or less than the value set in the TX FIFO WM field of the FIFO Watermark register.		
		13	RX FIFO OF. Receive FIFO Overflow interrupt — set when the receive FIFO is written when filled to its capacity.		
		12	RX FIFO WM. Receive FIFO Watermark interrupt — set when the receive FIFO contains a number of entries equal to or greater than the value set in the RX FIFO WM field of the FIFO Watermark register.		
		11:0	Function interrupts. These interrupts are set when the corresponding input signals `fn_int[11:0]` are driven high. For details, see the individual port / function register descriptions (starting in Section 7.7).		
14	Interrupt Mask	31:16	Reserved	RO	0xXXXX
		15:0	A one in a given bit position enables the corresponding interrupt in the Interrupt Status register.	R/W	0x0000
18	Interrupt Set		Interrupt set and one-cycle pulse generation	WO	0x00000000
		31	TX FIFO Reset. Writing a 1 to this bit resets the Transmit FIFO.		
		30	RX FIFO Reset. Writing a 1 to this bit resets the Receive FIFO selected by the RX FIFO Select bit in the Function register.		
		29	Reserved		
		28:16	Set[12:0]. Writing a 1 to a given bit position produces a one-cycle pulse on the corresponding output signal `set[12:0]` to the function blocks.		
		15:0	Interrupt Set. Writing a 1 to a given bit position sets the corresponding bit in the Interrupt Status register.		
1C	Interrupt Clear	31:16	Reserved	WO	0x0000
		15:0	Interrupt Clear. Writing a 1 to a given bit position clears the corresponding bit in the Interrupt Status register.	WO	0x0000
20	Transmit FIFO LO	31:0	Transmit FIFO data [31:0]	WO	0xXXXXXXXX
24	Transmit FIFO HI	31:0	Transmit FIFO data [63:32]	WO	0xXXXXXXXX
28	Receive FIFO LO	31:0	Receive FIFO data [31:0]	RO	0xXXXXXXXX
2C	Receive FIFO HI	31:0	Receive FIFO data [63:32]	RO	0xXXXXXXXX
30	Function Ctrl 0		Function Control register 0. Control signals to function blocks. For details, see the individual port/function register descriptions (starting in Section 7.7). The reset value is I/O port and function dependent.	R/W	0x00000000

Table 7-18 Generic I/O Port Register Definitions (Non-Blocking Region) (continued)

Offset	Register Name	Bits	Description	Read/Write	Reset Value
34	Function Ctrl 1		Function Control register 1. Control signals to function blocks. For details, see the individual port/function register descriptions (starting in Section 7.7). The reset value is I/O port and function dependent.	R/W	0x00000000
38	Function Ctrl 2		Function Control register 2. Control signals to function blocks. For details, see the individual port/function register descriptions (starting in Section 7.7). The reset value is I/O port and function dependent.	R/W	0x00000000
3C	Function Status 0		Function Status register 0. Status signals from function blocks. For details, see the individual port/function register descriptions (starting in Section 7.7).	RO	0xXXXXXXXX
40	Function Status 1		Function Status register 1. Status signals from selected function. For details, see the individual port/function register descriptions (starting in Section 7.7).	RO	0xXXXXXXXX
44	Function Status 2		Function Status register 2. Status signals from function blocks. For details, see the individual port/function register descriptions (starting in Section 7.7).	RO	0xXXXXXXXX
48	FIFO Watermark	31:16	TX FIFO WM. Triggering level for the Transmit FIFO Watermark interrupt.	R/W	0x00000000
		15:0	RX FIFO WM. Triggering level for the Receive FIFO Watermark interrupt.		
4C	FIFO Level	31:16	TX FIFO WM. Transmit FIFO current level.	RO	0xXXXXXXXX
		15:0	RX FIFO WM. Receive FIFO current level.		
50	GPIO Mask	31:0	GPIO Mask register. When a bit in this register is set to 1, the corresponding bit in the I/O port is allocated as a GPIO pin, regardless of whether the selected function uses that pin or not. The value in this register overrides the effect of function select on a bit-by-bit basis.	R/W	0x00000000

Table 7-19 gives an overview of the possible I/O Port allocations. For each port, the table shows the port width (number of signal pins), which interrupts are used, which functions can be selected, the function select reset value, and the size of the available receive and transmit FIFOs.

Table 7-19 I/O Port and Function Mapping

Port	Port Width	Interrupts Used [a]	Function 0	Function 1	Function 2	Function 3	Function Select Reset Value	RX FIFO Size	TX FIFO Size
A	8	int[2:1]	GPIO	Flash / INT / Clock	GPIO / INT / Clock	GPIO / INT	0x1	N/A	8 x 32
B	20	int[2:0]	GPIO	PCI	---	---	0x0	32 x 36	32 x 36
C	32	int[2]	GPIO	PCI (I/O only)	Reserved	---	0x0	N/A	N/A
D	12	int[2:0]	GPIO	Serdes (240 MHz)	Reserved	---	0x0	16 x 32	16 x 32
E	8	int[2:0]	GPIO	Serdes (250 MHz)	Reserved	MII / RMII	0x0	2 - 16 x 32	16 x 32
F	16	int[2:0]	GPIO	GMAC (MII / RMII / RGMII)	---	---	0x0	2 - 32 x 32	16 x 32
G	32	int[2]	GPIO	DDR SDRAM	---	---	0x0	N/A	N/A
H	10	int[2]	GPIO	DDR SDRAM	---	---	0x0	N/A	N/A
I	12	int[2:0]	GPIO	N/A	Reserved	MII (Port E Extension)	0x0	N/A	N/A
USB Port	2	int[2]	N/A	High-Speed USB	N/A	N/A	0x0	N/A	N/A

a. int[0]: Receive FIFO high watermark condition
    int[1]: Transmit FIFO low watermark condition
    int[2]: All other port interrupt conditions

The following sections describe how the per-port registers are used when specific port / function combinations are selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

7.7   Port A Registers

Port A includes support for the external flash memory controller. Once program code has been downloaded into the on-chip RAM, and when not accessing the flash memory, Port A pins can be used for GPIO functionality.

7.7.1   Port A Function 1 (Flash / INT / Clock)

This section describes the function-specific attributes of the registers used when Port A Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

Function 1 controls the external serial flash memory (using an SPI interface), in addition to certain interrupts and clocks.

The flash controller uses only non-blocking registers.

7.7.1.1   Port A Flash Interrupt Status

Bits	Field Name	Description	Read/Write	Reset Value
31:12	Refer to Table 7-18.		RO	0x00000000
11	P_INT[2]	Peripheral interrupt 2 Port_A[6]	RO	
10	P_INT[1]	Peripheral interrupt 1 Port_A[5]	RO	
9	P_INT[0]	Peripheral interrupt 0 Port_A[4]	RO	
8:1	Reserved		RO	
0	FC_DONE	The flash controller transaction has completed.	RO	

7.7.1.2   Port A Flash Interrupt Set

Bits	Field Name	Description	Read/Write	Reset Value
31	Refer to Table 7-18.		WO	0x00000000
30:17	Reserved		WO	
16	FC_START	Start IOPCS transaction.	WO	
15:0	Refer to Table 7-18.		WO	

7.7.1.3   Port A Flash Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31:24	CACHE_RD_CMD	Cache Read command for external flash. This is the common read command for all supported flash devices.	R/W	0x03
23:21	CACHE_DMY_CT	Cache Dummy Byte Count.	R/W	0x0
20:19	P_INT_CFG[2]	Configuration for Peripheral Interrupt 2	R/W	0x0
18:17	P_INT_CFG[1]	Configuration for Peripheral Interrupt 1	R/W	0x0
16:15	P_INT_CFG[0]	Configuration for Peripheral Interrupt 0	R/W	0x0
14:8	FC_CLK_WIDTH	Number of core clock cycles per SPI clock cycle.	R/W	0x28

Bits	Field Name	Description	Read/Write	Reset Value
7:2	FC_CE_WAIT	Number of SPI clock cycles that SPI CE_N (chip select) must be held de-asserted (HIGH) between consecutive SPI transactions.	R/W	0x28
1	FC_CACHE_ LOCKOUT	Cache lockout bit. When set to 1, cache is inhibited from getting access to the SPI bus. The flash controller will continue to queue cache transactions to be completed when it is deasserted.	R/W	0x0
0	FC_EN	FC enable. This signal is ON by default. Deasserting FC_EN will result in the FC arbiter inhibiting all access to the SPI interface, after any pending transactions are completed.	R/W	0x1

7.7.1.4   Port A Flash Function Control 1

Bits	Field Name	Description	Read/Write	Reset Value
31:30	FCX_INST	FC Instruction for transactions.	R/W	0x0
29:24	Reserved		R/W	0x0
23:16	P_CLK_CFG[1]	Configuration for Interrupt Peripheral Clock Divider 1 (core clock source)	R/W	0x0
15:14	Reserved		R/W	0x0
13:4	FCX_DATA_CT	Transaction data byte count. The number of bytes of data to be read / written to the external flash.	R/W	0x0
3:1	FCX_DMY_CT	Transaction dummy byte count. The number of dummy bytes to be inserted before read / write data.	R/W	0x0
0	FCX_ADDR_ EXISTS	The transaction has an address. If true, the SPIMaster state machine will sequence the contents of the FCX_ADDR to be part of the constructed transaction.	R/W	0x0

7.7.1.5   Port A Flash Function Control 2

Bits	Field Name	Description	Read/Write	Reset Value
31:24	FCX_CMD	Transaction command to be put on the SPI bus.	R/W	0x0
23:0	FCX_ADDR	Transaction address to be put on the SPI bus.	R/W	0x0

7.7.1.6   Port A Flash Function Status 0

Bits	Field Name	Description	Read/Write	Reset Value
31:2	Reserved		RO	0x0
1	IOX_ACTIVE	Processor I/O transaction is currently being processed.	RO	0x0
0	CACHE_ACTIVE	Cache transaction is currently being processed.	RO	0x0

7.7.1.7   Port A Flash Function Status 1

Bits	Field Name	Description	Read/Write	Reset Value
31	FCX_RDATA	Processor I/O transaction read data.	RO	0x0000
30:0	Reserved		RO	0x0000

7.7.1.8 Port A Flash FIFO Watermark

Bits	Field Name	Description	Read/Write	Reset Value
31:16	TX FIFO WM	Triggering level for the Transmit FIFO Watermark interrupt.	R/W	0x0
15:0	Reserved		R/W	0x0

7.7.1.9 Port A Flash FIFO Level

Bits	Field Name	Description	Read/Write	Reset Value
31	TX FIFO Level	Transmit FIFO current level.	RO	0xXXXX
30:0	Reserved		RO	0xXXXX

7.7.2 Port A Function 2 (GPIO / INT / Clock)

This section describes the function-specific attributes of the registers used when Port A Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

Function 2 controls the GPIO mode, in addition to certain interrupts and clocks.

This function uses only non-blocking registers.

7.7.2.1 Port A Function 2 Interrupt Status

Bits	Field Name	Description	Read/Write	Reset Value
31:16	Reserved		RO	0x0000
15:12	FIFO Status. Not used for this function selection.		RO	0x0
11	P_INT[2]	Peripheral interrupt 2 Port_A[6]	RO	0x0
10	P_INT[1]	Peripheral interrupt 1 Port_A[5]	RO	0x0
9	P_INT[0]	Peripheral interrupt 0 Port_A[4]	RO	0x0
8:0	Function interrupts.	These interrupts are set when the corresponding input signals `fn_int[8:0]` are driven high.	RO	0x000

7.7.2.2 Port A Function 2 Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31:21	Reserved		R/W	0x000
20:19	P_INT_CFG[2]	Configuration for peripheral interrupt 2	R/W	0x0
18:17	P_INT_CFG[1]	Configuration for peripheral interrupt 1	R/W	0x0
16:15	P_INT_CFG[0]	Configuration for peripheral interrupt 0	R/W	0x0
14:0	Reserved		R/W	0x0000

7.7.2.3 Port A Function 2 Function Control 1

Bits	Field Name	Description	Read/Write	Reset Value
31:24	P_CLK_CFG[0]	Configuration for interrupt peripheral clock divider 0 (250 MHz clock source)	R/W	0x00
23:16	P_CLK_CFG[1]	Configuration for interrupt peripheral clock divider 1 (core clock source)	R/W	0x00
15:0	Reserved		R/W	0x0000

7.7.3 Port A Function 3 (GPIO / INT)

Function 3 controls the GPIO mode, in addition to certain interrupts.

The registers are the same as for Port A Function 2, except that Function Control 1 (which controls clocks) and Function Control 2 are not used.

This function uses only non-blocking registers.

7.8   Port B Registers

7.8.1   Port B Function 1 (PCI)

This section describes the function-specific attributes of the registers used when Port B Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

The PCI interface uses both non-blocking registers and blocking registers. The non-blocking registers are listed first.

Notes:

1. The Function Control, Interrupt Mask, and FIFO Watermark non-blocking registers return to their reset values during a chip reset only.
2. The FIFO Level non-blocking register returns to its reset value only after a FIFO flush, which is accomplished by writing 1s to the MAX_TX_FLUSH, MAX_RX_FLUSH, TAR_TX_FLUSH, and TAR_RX_FLUSH bits in the Interrupt Set non-blocking register.

7.8.1.1   Port B PCI Function Register

Bits	Field Name	Description	Read/ Write	Reset Value
31:13	Reserved		RO	0x00000000
12:8	BR_TNUM	Blocking region thread number. The number of the thread that is allowed to access the blocking region for this port.	R/W	
7:5	Not applicable to the PCI function.		RO	
4	FN_RESET	Setting this bit will stop the PCI clock generator output and put the entire PCI function into the reset state. It will remain in the reset state until FN_RESET is cleared. The PCI clock generator is activated (in Function Ctrl 0) by setting PCI_CLK_OUT_ENA and writing a non-zero value to PCI_CLK_DIV; finally PCI_RST_N is set (deasserted). Bringing the PCI function out of reset should be performed in this order, while also ensuring the minimum required power-on / reset assertion of 100 ms (PCIv2.2 specification) before deasserting PCI_RST_N].	R/W	
3	RX_FIFO_SEL	There is no RX FIFO 1 in the PCI interface. Setting this bit has no effect. RX FIFO 0 is always selected.	R/W	
2:0	FN_SEL	'3h1 selects PCI. If the PCI function is not selected, it will be held in the reset state and the PCI clock generator will be forced inactive.	R/W	

7.8.1.2   Port B PCI Interrupt Status

Bits	Field Name	Description	Read/ Write	Reset Value
31:16	Reserved		RO	0x0000
15	TX_FIFO_UF	Not applicable to PCI function. IO TX FIFO underflow is prevented.	RO	0
14	TX_FIFO_WM	This interrupt is occurs when there are more than (32-FIFO_Watermark [TX_FIFO_WM]) FIFO entries free. For details see Section 5.10.4.2.	RO	1
13	RX_FIFO_OF	Not applicable to PCI function. I/O RX FIFO overflow is prevented.	RO	0

Bits	Field Name	Description	Read/ Write	Reset Value
12	RX_FIFO_WM	This interrupt is present when the RX FIFO contains a number of entries greater than or equal to the value set in FIFO_Watermark [RX_FIFO_WM].	RO	1
11	IO_INTA	An interrupt has been triggered at the corresponding I/O pin. Configure the behavior of this interrupt in Function_Ctrl_0 [IO_INTA_MODE].	RO	0
10	ARB_GNT_ TIMEOUT	The PCI Arbiter granted a requesting master; however no access was started within 16 PCI clock cycles. ARB_GNT_TIMEOUT_ID in Function Status 1 reports the Master that caused the timeout condition. ARB_GNT_TIMEOUT_CLR must be set in the Interrupt Set register to enable capture of a subsequent misbehaving Master.	RO	0
9	MAS_TRDY_ERR	Master detected that the target has inserted more wait states than the number programmed into the TRDY Count configuration register (located at address 0x49 in the blocking region).	RO	0
8	MAS_RETRY_ ERR	Master detected that the target has responded with 'retry' more times than the number programmed into the Retry Count configuration register (located at address 0x48 in the blocking region).	RO	0
7	MAS_DET_SERR	Master detected SERR# asserted by the addressed target.	RO	0
6	MAS_DET_PERR	Master detected that PERR# was asserted during a write transfer, or master asserted PERR# itself during a read.	RO	0
5	MAS_TAR_ ABORT	Master detected that a target responded with a target abort condition.	RO	0
4	MAS_MAS_ ABORT	Master aborted because no target has responded with DEVSEL#.	RO	0
3	MAS_COMPLETE	Indicates that the MPCI32 core has completed a transaction on the PCI bus. The error flags should be checked (Interrupt Status [9:4]). If no error flag was set, the master has either transmitted all data (write operation) or received all data (read operation). There may still be data in the RX FIFO to be read.	RO	0
2	TAR_DISCARD_ ERROR	A master has tried a delayed read to a non-prefetchable memory target and has failed to retry the transfer with the same address / command code etc. after being told to do so by the target. To prevent bus lockup; data still waiting in the TX FIFO will be flushed after 32768 PCI clock cycles and this error will be signaled.	RO	0
1	TAR_WRITE_OP_ DONE	The current write operation to this PCI target has completed.	RO	0
0	TAR_NEW_CMD	A new PCI command has been claimed by the PCI target interface. Function Status 1 [TAR_NEW_CMD] is set. A new PCI command and address is valid. If there was a read operation to this target in progress; this interrupt also signals the completion of that operation; thus the TX FIFO should be flushed before continuing. The new PCI command and address is waiting to be read and decoded from the Function Status registers.	RO	0

7.8.1.3   Port B PCI Interrupt Set

Bits	Field Name	Description	Read/Write	Reset Value
31	IO_TX_FIFO_RESET	Flush only the IO TX FIFO. It is recommended to use MAS_TX_FLUSH or TAR_TX_FLUSH instead.	WO	0
30	IO_RX_FIFO_RESET	Flush only the IO RX FIFO. It is recommended to use MAS_RX_FLUSH or TAR_RX_FLUSH instead.	WO	0
29:25	Reserved		WO	0x00
24	ARB_GNT_TIMEOUT_CLR	If an ARB_GNT_TIMEOUT interrupt has occurred, a write to this bit must be performed in order to allow a subsequent master to generate the ARB_GNT_TIMEOUT interrupt again.	WO	0
23	MAS_TX_FLUSH	Flushes both the IO TX FIFO and the Master's Async TX FIFO in the PCI core. Use before initiating a Master write transaction.	WO	0
22	MAS_RX_FLUSH	Flushes both the IO RX FIFO and the Master's Async RX FIFO in the PCI core. Use before initiating a Master read transaction.	WO	0
21	MAS_FORCE_TERM	Forces the master to terminate a transaction in progress before the word count has been exhausted. The core will terminate the transaction on the PCI bus in an orderly fashion.	WO	0
20	MAS_PCI_REQ	Initiate a new master request. Write to this bit after the desired command code; starting address; receive byte enables; etc. and any necessary Data / BEs has been written to TX FIFO (for master write commands).	WO	0
19	TAR_TX_FLUSH	Flushes both the IO TX FIFO and the Target's Async TX FIFO in the PCI core. Use at initial response to a new target read command before ACKing to PCI core.	WO	0
18	TAR_RX_FLUSH	Flushes both the IO RX FIFO and the Target's Async TX FIFO in the PCI core.	WO	0
17	TAR_FORCE_ABORT_NEXT	Manually force the PCI core to respond to the next transaction in its address ranges with a target abort.	WO	0
16	TAR_PCI_ACK	Manually force an acknowledge to the PCI core.	WO	0
15:0	Interrupt Set	Writing a 1 to a given bit position sets the corresponding bit in the Interrupt Status register.	WO	0x0000

7.8.1.4   Port B PCI Transmit FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	TX_DATA	Data to TX FIFO. The write pointer in this FIFO will automatically increment after writing data to this address.	WO	0x00000000

7.8.1.5   Port B PCI Transmit FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:4	Reserved		WO	0xXXXXXXXX
3:0	TX_DATA_BE	Byte Enables for data to TX FIFO.	WO	

7.8.1.6   Port B PCI Receive FIFO LO

Bits	Field Name	Description	Read/ Write	Reset Value
31:0	RX_DATA	Data from RX FIFO. The read pointer in this FIFO will automatically increment after reading data from this address.	RO	0xXXXXXXXX

7.8.1.7   Port B PCI Receive FIFO HI

Bits	Field Name	Description	Read/ Write	Reset Value
31:4	Reserved		RO	0xXXXXXXXX
3:0	TX_DATA_BE	Byte Enables for data from RX FIFO.	RO	

7.8.1.8   Port B PCI Function Control 0

Bits	Field Name	Description	Read/ Write	Reset Value
31	ARB_SM_RST_N	PCI Arbiter State Machine reset: 1: The State Machine is allowed to behave normally; bus parks at last requesting device. 0: The State Machine is held in the reset state; no PCI device is granted. Note: If the State Machine is reset, the current grant will be pulled away immediately, synchronous with the PCI BUS clock. The PCI specification states that it is legal to pull a grant away at any time.	R/W	0
30	ARB_SM_SEL	PCI Arbitration method select: 1: Host Priority Arbitration 0: Round Robin Arbitration	R/W	0
29	PCI_RST_N	Clearing this bit will assert reset on the PCI BUS, the PCI core, and the I/O FIFOs. It will not reset the clock generator. Until the PCI clock generator is active this reset control bit will be ignored and all logic in the PCI function (except the clock generator) and the I/O FIFOs will be held in reset.	R/W	0
28	PCI_CLK_OUT_ ENA	PCI_CLK output driver enable: 1: Enables the PCI_CLK output driver 0: Disable the PCI_CLK output driver. This can allow the use of an external PCI_CLK source if necessary.	R/W	0
27:24	PCI_CLK_DIV	Clock divisor value for the PCI Clock Generator (if the output driver is enabled). 0x0: Stops the PCI clock at the end of the current PCI clock cycle. 0x1- Valid clock divisors minus 1. The clock frequency is 200 0xF: MHz divided by the value of this field + 1. For example, 1 produces 200 MHz / 2 (100 MHz), 2 produces 200 MHz / 3 (66 MHz), 3 produces 200 MHz / 4 (50 MHz), etc.	R/W	0x0

Bits	Field Name	Description	Read/ Write	Reset Value
23	MAS_TAR_IF_SEL	Selects the interface from / to which the I/O FIFOs will route data: 　　1: Selects the master interface. 　　0: Selects the target interface. Notes: 　　1. It is recommended to leave the Target interface selected while the Master interface is inactive. 　　2. It is okay to switch to Master mode at any time except when data from a target write still exists in the FIFOs. 　　3. If software wants to do a Master transaction service, it must complete any outstanding Target writes first and should not ACK the Target write until after the Master transaction has been initiated and completed. This will prevent potential data corruption by preventing data from a new Target write from 'sneaking' into the FIFO while switching to Master mode.	R/W	0
22:21	IO_INTA_MODE	Configures the behavior of the interrupt signal generated at Interrupt Status [IO_INTA]: 　　00: Do not generate an interrupt. 　　01: Generate an interrupt due to a rising edge at the corresponding I/O pin. 　　10: Generate an interrupt due to a falling edge at the corresponding I/O pin. 　　11: Generate an interrupt due to either a rising or falling edge at the corresponding I/O pin.	R/W	0x0
20	RX_BYTE_SWAP	PCI Byte Swap RX 　　1: Data in RX FIFO is byte swapped (little endian ⇔ big endian). 　　0: Data in RX FIFO is not byte swapped.	R/W	0
19	TX_BYTE_SWAP	PCI Byte Swap TX 　　1: Data in TX FIFO is byte swapped (little endian ⇔ big endian). 　　0: Data in TX FIFO is not byte swapped.	R/W	0
18	TAR_FORCE_ABORT	Setting this bit causes the PCI target to respond to all PCI transactions in its address ranges with a target abort. This may be useful if the software detects some fatal error and determines that it can no longer function. Clearing this bit causes the PCI target to respond normally.	R/W	0
17	TAR_FORCE_RETRY	Setting this bit causes the PCI target to respond to all PCI transactions in its address ranges with a retry. This may be useful if the software performs some initialization routine at startup. Clearing this bit causes the PCI target to respond normally.	R/W	0
16	TAR_IMMEDIATE_READS	Clearing this bit forces the PCI target to perform delayed read transactions. Setting this bit forces the PCI target to perform immediate read transactions. Delayed reads will cause target to release the PCI bus while data is being fetched. Immediate reads will cause target to hold the PCI bus while data is being fetched. It is not recommend to perform immediate reads unless it can be guaranteed that data will be available within 16 PCI clock cycles of [TAR_NEW_CMD] going high; which may be hard to do in a software driven system.	R/W	0

Bits	Field Name	Description	Read/Write	Reset Value
15:14	Reserved		R/W	0x0
13:8	MAS_WCOUNT	The number of 32-bit words to be transferred by the master. The maximum transfer per master request that can be made is 32 words (128 bytes). This field must be written before the master transfer is started by setting MAS_PCI_REQ in the Interrupt Set register. 6'h00: illegal 6'h01- One 32-word burst 6'h20: 6'h21- illegal 6h3F:	R/W	0x0
7:4	MAS_RCV_BE	Byte enables for a master read command. This should stay the same for each read cycle. This field must be written before the master read transfer is started by setting MAS_PCI_REQ in the Interrupt Set register.	R/W	0x0
3:0	MAS_PCI_CMD	The desired PCI command code is written here. The legal command codes are: I/O Read or Write; Memory Read / Write; Memory Read Line / Read Multiple; Memory Write+Invalidate; and Configuration Read / Write. This field must be written before the master transfer is started by setting MAS_PCI_REQ in the Interrupt Set register.	R/W	0x0

7.8.1.9   Port B PCI Function Control 1

Bits	Field Name	Description	Read/Write	Reset Value
31:0	MAS_PCI_ADDR	Start address for the current PCI transaction when MPCI32 core is a master. The byte enables fed to the core must tally with address bits[1:0] for unaligned transfers. This register must be written before the master transfer is started by setting MAS_PCI_REQ in the Interrupt Set register.	R/W	0x00000000

7.8.1.10   Port B PCI Function Control 2

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Reserved		R/W	0x00000000

7.8.1.11   Port B PCI Function Status 0

Bits	Field Name	Description	Read/Write	Reset Value
31:0	TAR_PCI_ ADDRESS	Start address for the current PCI transaction when MPCI32 core is a target. Valid when TAR_NEW_CMD is set in the Function Status 1 register.	RO	0xXXXXXXXX

7.8.1.12 Port B PCI Function Status 1

Bits	Field Name	Description	Read/ Write	Reset Value
31	ARB_GNT_ TIMEOUT	This bit is set if the PCI Arbiter granted a requesting master; but no access was started within 16 PCI clock cycles.	RO	0
30:29	ARB_GNT_ TIMEOUT_ID	Identifies the PCI master that caused the most recent ARB_GNT_TIMEOUT interrupt. This value has no meaning if ARB_GNT_TIMEOUT (bit 31) is clear. 00: PCI Device 0 01: PCI Device 1 10: PCI Device 2 11: MPCI32 Host	RO	0x0
28:17	Reserved		RO	0x000
16	MAS_PCI_ STARTED	The MPCI32 core has latched the supplied MAS_PCI_ADDR (from Function Ctrl 1), MAS_PCI_CMD (from Function Ctrl 0), etc., so these registers may be cleared if necessary. Transfer will begin as soon as PCI bus conditions allow.	RO	0
15:10	Reserved		RO	0x00
9	TAR_ PREFETCHABLE	Echoes the state of the 'prefetchable' flag (bit 3) in the specified target's base address register.	RO	1
8	TAR_NEW_CMD	Active when a new transaction on the PCI bus has been claimed by this target. When this signal goes active; a TAR_NEW_CMD interrupt is set in the Interrupt Status register. Goes inactive when the command is acknowledged. ACK can be done manually by setting TAR_PCI_ACK in the Interrupt Set register.	RO	0
7:6	Reserved		RO	0x0
5	TAR_READ	A simple hardware decode of TAR_PCI_CMD (bits 3:0) has determined that this command is a target read operation. All memory read commands are aliased to the MEMORY READ command (4'b0110).	RO	0
4	TAR_WRITE	A simple hardware decode of TAR_PCI_CMD (bits 3:0) has determined that this command is a target write operation. All memory write commands are aliased to the MEMORY WRITE command (4'b0111).	RO	0
3:0	TAR_PCI_CMD	PCI command code for the current target transaction. Valid when TAR_NEW_CMD (bit 8) is set.	RO	0x0

7.8.1.13 Port B PCI Function Status 2

Bits	Field Name	Description	Read/ Write	Reset Value
31:16	CORE_TX_ FIFO_ LVL	The PCI core's TX FIFO level indicator. Valid only after receiving a TAR_NEW_CMD interrupt and before writing TAR_PCI_ACK. Use of this indicator is necessary to calculate address linearity if software chooses to implement data prefetching for Target Reads.	RO	0x0000
15:0	Reserved		RO	0x0000

7.8.1.14 Port B PCI Blocking Region Registers

The PCI blocking region registers are used to configure the PCI core as part of the initialization sequence.

Notes:

1. These registers are byte addressable only. All other move instructions (where more than one byte enable is asserted on the protocol C bus) will be ignored.
2. The Configuration Register Interface does not support single instruction read-modify-write access (for example, move.1 from PCI_BR to PCI_BR).
3. The PCI blocking region is Little-Endian!

The PCI Configuration registers reside at offsets 0x00 - 0x4D in the I/O Port B blocking region (see Table 7-16).

Table 7-20 shows the register definitions for the PCI Configuration Registers.

Table 7-20 PCI Configuration Register Definitions

Offset	Register Name	Bits	Description	Read/Write	Reset Value
0x01 – 0x00	Vendor ID	15:0	Unique Manufacturer ID. Reset value factory-configured.	R/W	0xFFFF
0x03 – 0x02	Device ID	15:0	Unique Device ID. Identifies the device as per vendor.	R/W	0xFFFF
0x05 – 0x04	Command		PCI Configuration Command register	R/W	0x0000
		15:10	Reserved. Always read zero.		
		9	Fast Back to Back Enable. Hardwired to zero so that the master cannot perform fast back-to-back transfers; but targets can accept them.		
		8	SERR Enable		
		7	Not implemented. Always reads zero.		
		6	Parity Error Response		
		5	Not implemented. Always reads zero.		
		4	Memory Write+Invalidate Enable		
		3	Not implemented. Always reads zero.		
		2	Bus Master Enable		
		1	Memory Space Enable		
		0	I/O Space Enable		

Table 7-20 PCI Configuration Register Definitions (continued)

Offset	Register Name	Bits	Description	Read/ Write	Reset Value
0x07 – 0x06	Status		PCI Configuration Status register	R/WS	
		15	Detected parity error. Latched status bit: write 1 to reset it.		0x00
		14	Signaled system error. Latched status bit: write 1 to reset it.		
		13	Received master abort. Latched status bit: write 1 to reset it.		
		12	Received target abort. Latched status bit: write 1 to reset it.		
		11	Signaled target abort. Latched status bit: write 1 to reset it.		
		10:9	DEVSEL timing. Always reads 01 = medium.	RO	0x1
		8	Master data parity detected. Latched status bit: write 1 to reset it.		0x080
		7	Fast back-to-back capable. Always reads as 1 because the target is capable of accepting fast back-to-back transfers.		
		6	Reserved. Always reads zero.		
		5	66-MHz capable. Factory-configured. Set to 1 if the core can run at 66 MHz.		
		4	Capabilities list present. Set to 1 if a power management interface is present. Parameterized as pwrman_exist.		
		3:0	Reserved. Always read zero.		
0x08	Revision ID	7:0	Reset value parameterized as def_revID.	R/W	0x00
0x0B – 0x09	Class Code	24:0	Reset value parameterized as def_classCode.	R/W	0xFF0000
0x0C	Cache Line Size	7:0	Cache Line Size. All cache line sizes that are powers of two are supported. Presented only when master is implemented.	R/W	0x00
0x0D	Latency Timer	7:0	Bus Latency Timer. Bits [2:0] are hardwired to zero to give a granularity of 8 clocks. Presented only when master is implemented.	R/W	0x00
0x0E	Header Type	7:0	Defines Type 0 configuration header layout as per PCI spec.	RO	0x00
0x0F	BIST	7:0	Not implemented. Always reads zero.	RO	0x00
0x13 – 0x10	BAR0	31:0	Register-type Target Interface. Not applicable to the IP51xx, Do not write to BAR0.	RO	0x00000000

Table 7-20 PCI Configuration Register Definitions (continued)

Offset	Register Name	Bits	Description	Read/Write	Reset Value
0x17 – 0x14	BAR1		FIFO-type Target Interface	R/W	0x00000000
		0	Memory or I/O space indicator. When bit 0 = 1, the target is mapped to the I/O space, and bits 31:1 are defined as follows: 31:2: MSBs written by the system startup software with the base address of the target address range. The number of active bits is factory configured in line with the amount of PCI Address Space allocated to the target. The remaining bits are hardwired to zero. 1: From PCI side of core: Reserved; always reads zero. From backend interface: As for Memory space option below.		
		0	Memory or I/O space indicator. When bit 0 = 0, the target is mapped to the Memory space, and bits 31:1 are defined as follows: 31:4: MSBs written by the system startup software with the base address of the target address range. The number of active bits is factory configured in line with the amount of PCI Address Space allocated to the target. The remaining bits are hardwired to zero. 3: always reads 1. The memory region to which the target is mapped is prefetchable. 2: always reads 0. 1: always reads 0. Located above 1 MB address boundary.		
0x1B – 0x18	BAR2	31:0	Target channel not implemented. Always reads 0.	RO	0x00000000
0x1F – 0x1C	BAR3	31:0	Target channel not implemented. Always reads 0.	RO	0x00000000
0x23 – 0x20	BAR4	31:0	Target channel not implemented. Always reads 0.	RO	0x00000000
0x27 – 0x24	BAR5	31:0	Target channel not implemented. Always reads 0.	RO	0x00000000
0x2B – 0x28	Cardbus CIS Pointer	31:0	Pointer to Cardbus data structure	R/W	0x00000000
0x2D – 0x2C	Subsystem Vendor ID	15:0	Two-byte subsystem vendor ID	R/W	0x0000
0x2F – 0x2E	Subsystem ID	15:0	Two-byte subsystem ID	R/W	0x0000
0x33 – 0x30	BIOS ROM BAR	31:0	Not implemented. Always reads 0.	R/W	0x00000000
0x34	Capabilities Pointer	7:0	Points to the first capabilities structure, which will either be the power management registers at 0x40 or the vital product data at 0x4C or nothing (0x00). For the IP51xx PCI function, this register always points to 0x00.	R/W	0x00
0x3B – 0x35	Reserved			RO	0

Table 7-20 PCI Configuration Register Definitions (continued)

Offset	Register Name	Bits	Description	Read/Write	Reset Value
0x3C	Interrupt Line	7:0	Programmable in a PC environment to a number between 0 and 15 corresponding to the interrupt channel assigned to this device. For information only.	R/W	0xFF
0x3D	Interrupt Pin	7:0	Indicates that the device uses INTA.	RO	0x01
0x3E	Min_GNT	7:0	Minimum requested bus grant duration.	R/W	0x00
0x3F	Max_LAT	7:0	Maximum requested bus grant latency.	R/W	0x00
0x47 – 0x40	Reserved			RO	0
0x48	Retry Count	7:0	Holds the number of retries the master may receive from a target before signaling an error. Zero disables it.	R/W	0x00
0x49	TRDY Count	7:0	Holds the number of wait states a remote target may insert in a master initiated transfer before an error is signaled. Zero disables it.	R/W	0x00
0x4B – 0x4A	Reserved			RO	0
0x4C	Capability ID	7:0	Indicates that the device has VPD (vital product data) capability. Not applicable to the IP51xx PCI function.	RO	0x03
0x4D	Next Item Pointer	7:0	Always reads zero.	RO	0x00

7.9   Port D Registers

7.9.1   Port D Function 1 (240 MHz Serdes)

This section describes the function-specific attributes of the registers used when Port D Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

This Serdes interface uses only non-blocking registers.

7.9.1.1   Port D Serdes Function Register

Bits	Field Name	Description	Read/Write	Reset Value
31:7	Reserved		RO	0x000000
6	Resets Function 3		RO	0x0
5	Resets Function 2		RO	0x0
4	Resets Function 1	Setting this bit will stop the Serdes clock generator and put the entire Serdes function into the reset state. The Serdes will remain in the reset state until this bit is cleared.	R/W	0x0
3	RX_FIFO_SEL	There is no RX FIFO 1 in the Serdes interface. Setting this bit has no effect. RX FIFO 0 is always selected.	R/W	0x0
2:0	FN_SEL	'3h1 selects Serdes. If the Serdes function is not selected, it will be held in the reset state and the Serdes clock generator will be forced inactive.	R/W	0x0

7.9.1.2   Port D Serdes Interrupt Status

Bits	Field Name	Description	Read/Write	Reset Value
31:16	Reserved		RO	0x0000
15	TX FIFO UF	Not used for Serdes.	RO	0x0
14	TX FIFO WM	Not used for Serdes.	RO	0x0
13	RX FIFO OF	Receive FIFO Overflow interrupt — set when the receive FIFO is written when filled to its capacity.	RO	0x0
12	RX FIFO WM	Receive FIFO Watermark interrupt — set when the receive FIFO contains a number of entries equal to or greater than the value set in the RX FIFO WM field of the FIFO Watermark register.	RO	0x0
11:8	Reserved		RO	0x0
7	RXERR	Receive Error - Used for USB: The Serdes has detected 7 consecutive 1s.	RO	0xX
6	RXEOP	Receive End-Of-Packet - Used for USB (asserted at end of packet) and GPSI (asserted at de-assertion of RxEn).	RO	0xX
5	SYND	Sync Data match - Used for USB: Received data matches sync pattern (RSYNC[7:0] in Function Ctrl 1).	RO	0xX
4	TXBE	Transmit Buffer Empty.	RO	0xX
3	TXEOP	Transmit End-Of-Packet - Used for USB: Serdes has finished transmitting all available data and no new data is available. An EOP is transmitted after the last data.	RO	0xX

Bits	Field Name	Description	Read/Write	Reset Value
2	SX LP	Used for USB: Bus Idle after Serdes stops driving the bus.	RO	0xX
1	RXBF	Receive Buffer data available	RO	0xX
0	RXXCRS	Receive Busy - Used for USB: RxBusy is detected.	RO	0xX

7.9.1.3   Port D Serdes Interrupt Set

Bits	Field Name	Description	Read/Write	Reset Value
31	Not used for Serdes.		WO	0x0
30	RX FIFO Reset	Writing a 1 to this bit resets the Receive FIFO selected by the RX FIFO Select bit in the Function register.	WO	0x0
29:17	Reserved		WO	0x0000
16	TXBUF_VALID	Signals to the Serdes that valid transmit data is available.	WO	0x0
15:0	Interrupt Set	Writing a 1 to a given bit position sets the corresponding bit in the Interrupt Status register.	WO	0x0000

7.9.1.4   Port D Serdes Transmit FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for Serdes.		WO	0x00000000

7.9.1.5   Port D Serdes Transmit FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for Serdes.		WO	0xXXXXXXXX

7.9.1.6   Port D Serdes Receive FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	RX_DATA	Data from RX FIFO. Only bits [15:0] are used.	RO	0xXXXXXXXX

7.9.1.7   Port D Serdes Receive FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for Serdes.		RO	0xXXXXXXXX

7.9.1.8   Port D Serdes Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31	GLOBAL_EN	Serdes output enable 1: Enable. Serdes I/O pins are driven from Serdes as defined by the MODE[7:0] field. 0: Disable (without resetting device). Serdes I/O pins become accessible through GPIO.	R/W	0x0
30	LOOP_BACK	Loopback control 1: Enable loopback 0: Disable loopback	R/W	0x0
29	TX_DATA_INV	Invert all transmit data.	R/W	0x0
28:24	TXSCNT[4:0]	Specifies the number of bits to transmit.	R/W	0x00
23:16	MODE[7:0]	Serdes mode/submode select: 23:20: PRS (see Table 6-5). 19:18: SUBM 17:16: Reserved	R/W	0x00
15:0	CLKDIV[15:0]	Divide value used in generating the Serdes internal clock.	R/W	0x0000

7.9.1.9   Port D Serdes Function Control 1

Bits	Field Name	Description	Read/Write	Reset Value
31:24	SYNCMASK[7:0]	Mask for RSYNC[7:0] - Used for USB	R/W	0x00
23:16	RSYNC[7:0]	Sync pattern - Used for USB	R/W	0x00
15:10	Reserved		R/W	0x00
9	BIT_ORDER	MSB/LSB: 1: MSB first 0: LSB first	R/W	0x0
8	CRS_INT_POLARITY	Carrier sense interrupt polarity: 1: Interrupt on loss of carrier (RXXCRS) 0: Interrupt on gain of carrier (RXXCRS)	R/W	0x0
7	SPI_MASTER_SEL	Master/Slave select - SPI or GPSI only: 1: Master 0: Slave	R/W	0x0
6	USB_SYNC_IGNORE	USB Sync Ignore: 1: Do not detect sync 0: Detect sync	R/W	0x0
5	REV_POLARITY_EN	Reverse Polarity Enable: 1: Invert received data 0: Do not invert received data	R/W	0x0
4:0	RXSCNT[4:0]	Receive count interrupt level. Number of received bits that will cause an interrupt.	R/W	0x00

7.9.1.10 Port D Serdes Function Control 2

Bits	Field Name	Description	Read/Write	Reset Value
31:16	Reserved		R/W	0x0000
15:0	TXBUF[15:0]	Data for transmit operations	R/W	0x0000

7.9.1.11 Port D Serdes Function Status 0

Bits	Field Name	Description	Read/Write	Reset Value
31:10	Reserved		RO	0x000000
9:8	Reserved. Value is undefined when read.		RO	0xX
7:5	Reserved		RO	0xX
4:0	RXCTR[4:0]	Actual number of received bits. However, the following applies for the last transfer only: - RXCTR[4:0] = the number of bits received, if fewer than 8. - RXCTR[4:0] = 8, if >8 and <16 bits are received. - RXCTR[4:0] = 16 if ≥16 bits are received and RXSCNT = 16.	RO	0xX

7.10   Port E Registers

7.10.1   Port E Function 1 (250 MHz Serdes)

The registers for this function are the same as those for Port D Serdes (Section 7.9.1), except for the value of the reference clock frequency (240 MHz for Port D and 250 MHz for Port E) and the fact that USB is supported on Port D Serdes, but not on Port E Serdes.

7.10.2   Port E Function 3 (MII / RMII)

This section describes the function-specific attributes of the registers used when Port E Function 3 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

This Serdes interface uses only non-blocking registers.

7.10.2.1   Port E MII / RMII Interrupt Status

Bits	Field Name	Description	Read/Write	Reset Value
31:12	Refer to Table 7-18, Interrupt Status section.		RO	0x00000
11:9	Reserved		RO	0x0
8	RX_THRESHOLD_INT	Receive Threshold. The frame currently being received has reached the threshold at which it will not be discarded (32 bytes).	RO	0xX
7	RX_EOP_INT	End-of-packet detected during receive.	RO	0xX
6	RX_SFD_INT	Start-of-frame delimiter detected during receive.	RO	0xX
5	RX_ERR_INT	Error detected during receive.	RO	0xX
4	TX_EOP_INT	Transmit end-of-packet.	RO	0xX
3	COL_INT	Collision detected during transmission.	RO	0xX
2	CRS_INT	Carrier sense.	RO	0xX
1	ODD_NIB_ERR_INT	Odd nibble reception error.	RO	0xX
0	FALSE_CARRIER_INT	False carrier message.	RO	0xX

7.10.2.2   Port E MII / RMII Interrupt Set

Bits	Field Name	Description	Read/Write	Reset Value
31	TX FIFO Reset	Writing a 1 to this bit resets the Transmit FIFO.	WO	0x0
30	RX FIFO Reset	Writing a 1 to this bit resets the Receive FIFO selected by the RX FIFO Select bit in the Function register.	WO	0x0
29:19	Reserved		WO	0x000
18	TX_ERR_SEND	Transmit Error Send	WO	0x0
17	Reserved. Only write 0 to this bit.		WO	0x0
16	TX_START	Transmit Start. Writing a 1 to this bit starts transmission.	WO	0x0
15:0	Interrupt Set	Writing a 1 to a given bit position sets the corresponding bit in the Interrupt Status register.	WO	0x0000

7.10.2.3 Port E MII / RMII Transmit FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	TX_DATA	Data to TX FIFO.	WO	0x00000000

7.10.2.4 Port E MII / RMII Transmit FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for MII / RMII.		WO	0xXXXXXXXX

7.10.2.5 Port E MII / RMII Receive FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	RX_DATA	Data from RX FIFO.	RO	0xXXXXXXXX

7.10.2.6 Port E MII / RMII Receive FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for MII / RMII.		RO	0xXXXXXXXX

7.10.2.7 Port E MII / RMII Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31:23	Reserved		R/W	0x000
22	MII_PORT_SELECT	Set this bit to 1 when using the MII interface.	R/W	0x0
21	TX_AUTO_CRC	Automatic Transmit of CRC: 1: Automatically generate TX CRC. 0: Do not automatically generate TX CRC.	R/W	0x0
20	RMII_SPEED	RMII Speed: 1: 100 Mbps 0: 10 Mbps	R/W	0x0
19	RMII_MODE	Interface Mode: 1: RMII Mode 0: MII Mode	R/W	0x0
18	Reserved		R/W	0x000
17	HALF_DUPLEX	Sets the MII controller into half duplex mode. 1: Half duplex 0: Full duplex	R/W	0x0
16	RX_EN	Receive Enable: 1: Enable receive mode. 0: Disable receive mode.	R/W	0x0
15:0	TX_BYTE_COUNT[15:0]	Transmit Byte Count. The number of bytes to be transmitted.	R/W	0x0000

7.10.2.8   Port E MII / RMII Function Control 1

Bits	Field Name	Description	Read/Write	Reset Value
31:2	Reserved		R/W	0x00000000
1:0	TX_BYTE_START[1:0]	Transmit Byte Start. The byte number, within a big-endian ordered word, to be used as the first data byte of a transmit packet: 00: Bits [31:24] 01: Bits [23:16] 10: Bits [15:8] 11: Bits [7:0]	R/W	0x0

7.10.2.9   Port E MII / RMII Function Status 0

Bits	Field Name	Description	Read/Write	Reset Value
31:20	Reserved		RO	0x000
19	CRC_OK	The CRC sent with the most recently received packet matches the CRC calculated on the payload of the same packet.	RO	0x000
18	RX_FIFO_SELECT	Indicates to which FIFO the currently received frame is being written: 1: FIFO 1 0: FIFO 0	R/W	0x0
17	COLLISION	The state of the COL signal after being synchronized to the core clock domain.	RO	0x000
16	CARRIER_SENSE	The state of the CRS signal after being synchronized to the core clock domain.	RO	0x000
15:0	RX_BYTE_COUNT[15:0]	Total number of bytes received, including the CRC, but not including the SFD or any part of the preamble.	RO	0x000

7.11   Port F Registers

7.11.1   Port F Function 1 (GMAC)

This section describes the function-specific attributes of the registers used when Port F Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

The GMAC interface uses both non-blocking registers and blocking registers. The non-blocking registers are listed first.

7.11.1.1   Port F GMAC Interrupt Status

Bits	Field Name	Description	Read/Write	Reset Value
31:12	Refer to Table 7-18.		RO	0x0000
11:6	Reserved		RO	0x00
5	TX_EOF	Transmit packet End-Of-Frame	RO	0
4	TX_PAR	Transmit packet abort	RO	0
3	TX_PRT	Transmit packet retry	RO	0
2	RX_SOF	Receive packet Start-Of-Frame	RO	0
1	RX_EOF	Receive packet End-Of-Frame	RO	0
0	RX_PRT	Receive packet reached minimum packet size threshold.	RO	0

7.11.1.2   Port F GMAC Interrupt Set

Bits	Field Name	Description	Read/Write	Reset Value
31	TX_FIFO _RESET	TX FIFO Reset. Writing a 1 to this bit resets the Transmit FIFO.	WO	0
30	RX_FIFO _RESET	RX FIFO Reset. Writing a 1 to this bit resets the Receive FIFO selected by the RX FIFO Select bit in the Function register.	WO	0
29:19	Reserved		WO	0x000
18	STOP_PAUSE	Writing a 1 to this bit produces a one-cycle pulse that initiates the transmission of a pause control packet with timer value 0x0000.	WO	0
17	START_PAUSE	Writing a 1 to this bit produces a one-cycle pulse that initiates the transmission of a pause control packet with timer value 0xFFFF	WO	0
16	START_XMIT	Writing a 1 to this bit produces a one-cycle pulse that initiates the transmission of the data packet currently in the TX FIFO.	WO	0
15:0	Interrupt Set	Writing a 1 to a given bit position sets the corresponding bit in the Interrupt Status register.	WO	0x0000

7.11.1.3   Port F GMAC Transmit FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	TX_DATA	Data to TX FIFO.	WO	0x00000000

7.11.1.4   Port F GMAC Transmit FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for GMAC.		WO	0xXXXXXXXX

7.11.1.5 Port F GMAC Receive FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	RX_DATA	Data from RX FIFO.	RO	0xXXXXXXXX

7.11.1.6 Port F GMAC Receive FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used for GMAC.		WO	0xXXXXXXXX

7.11.1.7 Port F GMAC Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31:4	Reserved		R/W	0xXX
3:2	CLK_SPD	Selects transmit clock speed: 2'b00: 125 MHz 2'b01: 25 MHz 2'b10: 2.5 MHz 2'b11: Reserved	R/W	0x0
1:0	INTERFACE_MODE_SELECT	Physical Interface Selection: 2'b00: RGMII mode 2'b01: MII mode 2'b10: RMII mode 2'b11: Reserved	R/W	0x0

7.11.1.8 Port F GMAC Function Control 1

Bits	Field Name	Description	Read/Write	Reset Value
31:18	Reserved		R/W	0x0000
17:16	TX_OFFSET	Transmit packet offset. Indicates the byte address of the first valid data in the TX FIFO.	R/W	0x00
15:0	TX_SIZE	Transmit packet size (bytes).	R/W	0x00

7.11.1.9 Port F GMAC Function Status 0

Function Status 0 shows bits from the Transmit Statistics Vector (TSV) as follows:

Bits	Field Name	Description	Read/Write	Reset Value
31	Reserved		RO	0x0
30	TX_VLAN	Transmit VLAN Tagged Frame: Frame's length/type field contained 0x8100, which is the VLAN Protocol Identifier.	RO	0xX
29	TX_BP	Backpressure Applied: Carrier-sense-method backpressure was previously applied.	RO	0xX
28	TX_PAUSE	Transmit PAUSE Control Frame: Frame transmitted was a Control frame with a valid PAUSE opcode.	RO	0xX

Bits	Field Name	Description	Read/Write	Reset Value
27	TX_CF	Transmit Control Frame: Frame transmitted was a Control frame.	RO	0xX
26	TX_UR	Transmit Under-run: Host side failed to transfer complete frame to PETFN.	RO	0xX
25	TX_GIANT	Transmit Giant: Byte count for frame was greater than MAXIMUM FRAME parameter.	RO	0xX
24	TX_LATE_COL	Transmit Late Collision: Collision occurred beyond the collision window (512 bit times).	RO	0xX
23	TX_MAX_COL	Transmit Maximum Collisions: Packet was aborted after number of collisions exceeded RETRANSMISSON MAXIMUM.	RO	0xX
22	TX_ED	Transmit Excessive Defer: Packet was deferred in excess of 6,071 nibble-times in 100 Mbps mode, or 24,287 bit-times in 10 Mbps mode.	RO	0xX
21	TX_PD	Transmit Packet Defer: Packet was deferred for at least one attempt, but less than an excessive defer.	RO	0xX
20	TX_DONE	Transmit Done: Transmission of the packet was completed.	RO	0xX
19:16	TX_COL_CNT	Transmit Collision Count: Number of collisions the current packet incurred during transmission attempts. Note: Bits 19 through 16 are the collision count on any successfully transmitted packet and as such will not show the possible maximum count of 16 collisions.	RO	0xX
15:0	TX_BYTE_CNT	Transmit Byte Count: Total bytes in frame not counting collided bytes.	RO	0xXXXX

7.11.1.10 Port F GMAC Function Status 1

Bits	Field Name	Description	Read/Write	Reset Value
31:16	GMAC_TX_FIFO_LEVEL	Occupancy (in 32-bit words) of the transmit packet buffer.	RO	0xXXXX
15:2	Reserved		RO	0x0000
1	GMAC_RXFIFO_ACTIVE	GMAC RX FIFO active	RO	0x0
0	GMAC_IOPCS_TX_CHAN_RDY	GMAC IOPCS transmit channel ready	RO	0x0

7.11.1.11 Port F GMAC Function Status 2

Function Status 2 shows bits from the Receive Statistics Vector (RSV) as follows:

Bits	Field Name	Description	Read/Write	Reset Value
31	Reserved		RO	0x0
30	RX_TRUNC	Receive frame truncated.	RO	0xX
29	RX_LONG	Receive long event.	RO	0xX
28	RX_VLAN	Receive VLAN Tag Detected: Frame's length/type field contained 0x8100 which is the VLAN Protocol Identifier.	RO	0xX

Bits	Field Name	Description	Read/Write	Reset Value
27	RX_U_OP	Receive Unsupported Opcode: The current frame was recognized as a Control Frame, but it contained an unknown opcode. This can be qualified by verifying RSV[20] = 0, and length = (64 - 1518) to verify that the frame was a valid Control Frame.	RO	0xX
26	RX_PAUSE	Receive PAUSE Control Frame: The current frame was recognized as a Control Frame containing a valid PAUSE Frame opcode and a valid address. This can be qualified by verifying RSV[20] = 0, and length = (64 - 1518) to verify that the frame was a valid Control Frame.	RO	0xX
25	RX_CF	Receive Control Frame: The current frame was recognized as a Control Frame for having a valid Type-Length designation. This can be qualified by verifying RSV[20] = 0, and length = (64 - 1518) to verify that the frame was a valid Control Frame.	RO	0xX
24	RX_DN	Receive Dribble Nibble: Indicates that after the end of the packet an additional 1 to 7 bits were received. A single nibble, called the dribble nibble, is formed but not sent to the system (10/100 Mbps only).	RO	0xX
23	RX_OK	Receive OK: Frame contained a valid CRC and did not have a code error.	RO	0xX
22	RX_OR	Receive Length Out of Range: Indicates that a frame's length was larger than 1518 bytes but smaller than the Host's Maximum Frame Length Value (Type Field).	RO	0xX
21	RX_LCE	Receive Length Check Error: Indicates that the frame length field value in the packet does not match the actual data byte length and is not a Type Field.	RO	0xX
20	RX_CRC_ER	Receive CRC Error: The packet's CRC did not match the internally generated CRC.	RO	0xX
19	RX_ERR	Receive Code Error: One or more nibbles were signaled as errors during the reception of the packet	RO	0xX
18	RX_FALSE_CAR	Receive False Carrier: Indicates that at some time since the last receive statistics vector, a false carrier was detected, noted and reported with the current receive statistics vector. The false carrier is not associated with this packet. A false carrier is activity on the receive channel that does not result in a packet receive attempt being made. A false carrier is detected as RX_ER = 1, RX_DV = 0, RXD[3:0] = 0xE (RXD[7:0] = 0x0E).	RO	0xX
17	RX_DV_ER	Receive RX_DV Event: Indicates that the last receive event seen was not long enough to be a valid packet.	RO	0xX
16	RX_PKT_DRP	Receive Previous Packet Dropped: Indicates that since the last receive statistics vector a packet was dropped (i.e. IFG too small).	RO	0xX
15:0	RX_BYTE_CNT	Receive Byte Count: Total number of bytes in receive frame, not counting collided bytes.	RO	0xXXXX

7.11.1.12 Port F GMAC Blocking Region Registers

The GMAC blocking region registers reside at offsets 0x00 - 0x11 in the I/O Port F blocking region (see Table 7-16).

Table 7-21 lists the registers in this region.

Table 7-21 Organization of the Port F GMAC Blocking Region

Offset	Description	Reset Value
0x00	MAC Configuration #1	0x8000 0000
0x01	MAC Configuration #2	0x0000 7000
0x02	Inter-Packet Gap (IPG) / Inter-Frame Gap (IFG)	0x4060 5060
0x03	Half-Duplex	0x00A1 F037
0x04	Maximum Frame Length	0x0000 0600
0x05 - 0x06	Reserved	0x0000 0000
0x07	Test Register	0x0000 0000
0x08 - 0x0D	Reserved	0x0000 0000
0x0E	Interface Control	0x0000 0000
0x0F	Interface Status	0x0000 0000
0x10	Station Address, Part 1	0x0000 0000
0x11	Station Address, Part 2	0x0000 0000

Details for the registers in Table 7-21 are given below, starting in Table 7-22.

Table 7-22 Port F GMAC Blocking Register Definitions — MAC Configuration #1

Bits	Description	Read/Write	Reset Value
31	SOFT RESET: Setting this bit will put all modules within the GMAC into reset, except the Host Interface. The Host Interface is reset via HRST.	R/W	1
30	Reserved. This bit must only be written as 0.	R/W	0
29:20	Reserved	R/W	0x000
19	RESET RX MAC CONTROL: Setting this bit will put the PERMC Receive MAC Control block in reset. This block detects Control frames and contains the pause timers.	R/W	0
18	RESET TX MAC CONTROL: Setting this bit will put the PETMC Transmit MAC Control block in reset. This block multiplexes data and Control frame transfers. It also responds to XOFF PAUSE Control frames.	R/W	0
17	RESET RX FUNCTION: Setting this bit will put the PERFN Receive Function block in reset. This block performs the receive frame protocol.	R/W	0
16	RESET TX FUNCTION: Setting this bit will put the PETFN Transmit Function block in reset. This block performs the frame transmission protocol.	R/W	0
15:9	Reserved	R/W	0x00
8	LOOP BACK: Setting this bit will cause the PETFN MAC Transmit outputs to be looped back to the MAC Receive inputs. Clearing this bit results in normal operation.	R/W	0
7:6	Reserved	R/W	0x0
5	RECEIVE FLOW CONTROL ENABLE: Setting this bit will cause the PERFN Receive MAC Control to detect and act on PAUSE Flow Control frames. Clearing this bit causes the Receive MAC Control to ignore PAUSE Flow Control frames.	R/W	0

Table 7-22 Port F GMAC Blocking Register Definitions — MAC Configuration #1 (continued)

Bits	Description	Read/Write	Reset Value
4	TRANSMIT FLOW CONTROL ENABLE: Setting this bit will allow the PETMC Transmit MAC Control to send PAUSE Flow Control frames when requested by the system. Clearing this bit prevents the Transmit MAC Control from sending Flow Control frames.	R/W	0
3	SYNCHRONIZED RECEIVE ENABLE: Receive Enable synchronized to the receive stream.	RO	0
2	RECEIVE ENABLE: Setting this bit will allow the MAC to receive frames from the PHY. Clearing this bit will prevent the reception of frames.	R/W	0
1	SYNCHRONIZED TRANSMIT ENABLE: Transmit Enable synchronized to the transmit stream.	RO	0
0	TRANSMIT ENABLE: Setting this bit will allow the MAC to transmit frames from the system. Clearing this bit will prevent the transmission of frames.	R/W	0

Table 7-23 Port F GMAC Blocking Register Definitions — MAC Configuration #2

Bits	Description	Read/Write	Reset Value
31:16	Reserved	R/W	0x0000
15:12	PREAMBLE LENGTH: This field determines the length of the preamble field of the packet, in bytes.	R/W	0x7
11:10	Reserved	R/W	0x0
9:8	INTERFACE MODE: This field determines the type of interface to which the MAC is connected: 2'b00: Reserved 2'b01: MII / RMII 2'b10: RGMII 2'b11: Reserved	R/W	0x0
7:6	Reserved	R/W	0x0
5	HUGE FRAME ENABLE: Set this bit to allow frames longer than the MAXIMUM FRAME LENGTH to be transmitted and received. Clear this bit to have the MAC limit the length of frames at the MAXIMUM FRAME LENGTH value. Maximum Frame Length is set in the separate Maximum Frame Length register.	R/W	0
4	LENGTH FIELD CHECKING: Set this bit to cause the MAC to check the frame's length field to ensure that it matches the actual data field length. Clear this bit if no length field checking is desired.	R/W	0
3	Reserved	R/W	0
2	PAD / CRC ENABLE: Set this bit to have the MAC pad all short frames and append a CRC to every frame, whether or not padding was required. Clear this bit if frames presented to the MAC have a valid length and contain a CRC.	R/W	0
1	CRC ENABLE: Set this bit to have the MAC append a CRC to all frames. Clear this bit if frames presented to the MAC have a valid length and contain a valid CRC. If the PAD/CRC ENABLE configuration bit or the per-packet PAD/CRC ENABLE is set, CRC ENABLE is ignored.	R/W	0
0	FULL-DUPLEX: Setting this bit will configure the GMAC to operate in Full-Duplex mode. Clearing this bit will configure the PE-MCXMAC to operate in Half-Duplex mode only.	R/W	0

Table 7-24 Port F GMAC Blocking Register Definitions — Inter-Packet Gap (IPG) / Inter-Frame Gap (IFG)

Bits	Description	Read/Write	Reset Value
31	Reserved	R/W	0
30:24	NON-BACK-TO-BACK INTER-PACKET GAP PART 1 (IPGR1): This programmable field represents the optional carrierSense window referenced in IEEE 802.3/4.2.3.2.1 'Carrier Deference'. If a carrier is detected during the timing of IPGR1, the MAC will defer to the carrier. If, however, the carrier becomes active after IPGR1, the MAC will continue timing IPGR2 and transmit, knowingly causing a collision. This ensures fair access to the medium. The permitted range of values is 0x0 to IPGR2. Default is 0x40 (64d) which follows the two-thirds/one-thirds guideline.	R/W	0x40
23	Reserved	R/W	0
22:16	NON-BACK-TO-BACK INTER-PACKET GAP PART 2 (IPGR2): This programmable field represents the Non-Back-to-Back Inter-Packet-Gap in bit times. Default is 0x60 (96d), which represents the minimum IPG of 96 bits.	R/W	0x60
15:8	MINIMUM IFG ENFORCEMENT: This programmable field represents the minimum size of IFG to enforce between frames (expressed in bit times). A frame whose IFG is less than that programmed is dropped. The default setting of 0x50 (80d) represents half of the nominal minimum IFG which is 160 bits.	R/W	0x50
7	Reserved	R/W	0
6:0	BACK-TO-BACK INTER-PACKET GAP: This programmable field represents the IPG between Back-to-Back packets (expressed in bit times). This is the IPG parameter used exclusively in Full-Duplex mode when two transmit packets are sent back-to-back. Set this field to the desired number of bits. The default setting of 0x60 (96d) represents the minimum IPG of 96 bits.	R/W	0x60

Table 7-25 Port F GMAC Blocking Register Definitions — Half-Duplex

Bits	Description	Read/Write	Reset Value
31:24	Reserved	R/W	0x00
23:20	ALTERNATE BINARY EXPONENTIAL BACKOFF TRUNCATION: This field is used when ALTERNATE BINARY EXPONENTIAL BACKOFF ENABLE is set. The value programmed is substituted for the Ethernet standard value of ten.	R/W	0xA
19	ALTERNATE BINARY EXPONENTIAL BACKOFF ENABLE: Setting this bit will configure the Tx MAC to use the ALTERNATE BINARY EXPONENTIAL BACKOFF TRUNCATION setting instead of the 802.3 standard tenth collision. The Standard specifies that any collision after the tenth uses $2^{10} - 1$ as the maximum backoff time. Clearing this bit will cause the Tx MAC to follow the standard binary exponential backoff rule.	R/W	0
18	BACKPRESSURE NO BACKOFF: Setting this bit will configure the Tx MAC to immediately re-transmit following a collision during backpressure operation. Clearing this bit will cause the Tx MAC to follow the binary exponential backoff rule.	R/W	0
17	NO BACKOFF: Setting this bit will configure the Tx MAC to immediately re-transmit following a collision. Clearing this bit will cause the Tx MAC to follow the binary exponential backoff rule.	R/W	0

Table 7-25 Port F GMAC Blocking Register Definitions — Half-Duplex (continued)

Bits	Description	Read/Write	Reset Value
16	EXCESSIVE DEFER: Setting this bit will configure the Tx MAC to allow the transmission of a packet that has been excessively deferred. Clearing this bit will cause the Tx MAC to abort the transmission of a packet that has been excessively deferred.	R/W	1
15:12	RETRANSMISSION MAXIMUM: This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The Standard specifies the maximum number of attempts to be 0xF (15d).	R/W	0xF
11:10	Reserved	R/W	0x0
9:0	COLLISION WINDOW: This programmable field represents the slot time or collision window during which collisions might occur in a properly configured network. Since the collision window starts at the beginning of transmission, the preamble and SFD are included. The default of 0x037 (55d) corresponds to the count of frame bytes at the end of the window. If the value is larger than 0x03F, the TPST signal will no longer work correctly.	R/W	0x037

Table 7-26 Port F GMAC Blocking Register Definitions — Maximum Frame Length

Bits	Description	Read/Write	Reset Value
31:16	Reserved	R/W	0x0000
15:0	MAXIMUM FRAME LENGTH: This programmable field sets the maximum frame size in both the transmit and receive directions. It resets to 0x0600 (1536d).	R/W	0x0600

Table 7-27 Port F GMAC Blocking Register Definitions — Test Register

Bits	Description	Read/Write	Reset Value
31:4	Reserved	R/W	0x0000000
3	MAXIMUM BACKOFF: Setting this bit will cause the MAC to backoff for the maximum possible length of time. This test bit is used to predict backoff times in Half-Duplex mode.	R/W	0
2	REGISTERED TRANSMIT FLOW ENABLE: Registered Transmit Half-Duplex Flow Enable.	R/W	0
1	TEST PAUSE: Setting this bit allows the MAC to be paused via the host interface for testing purposes.	R/W	0
0	SHORTCUT SLOT TIME: This bit allows the slot time counter to expire regardless of the current count. This bit is for testing purposes only.	R/W	0

In the Interface Control Register, the range of bits that are active depends on which optional interfaces are connected.

Table 7-28 Port F GMAC Blocking Register Definitions — Interface Control

Bits	Description	Read/Write	Reset Value
31	RESET INTERFACE MODULE: Setting this bit resets the Interface module. Clearing this bit allows for normal operation. This bit can be used in place of bits 23, 15, and 7 when just 1 Interface module is connected.	R/W	0
30:28	Reserved	R/W	0x0
27	TBIMODE: Setting this bit configures the A-RGMII module to expect TBI signals at the GMII interface as described in A-RGMII documentation. This bit should not be asserted unless this mode is being used.	R/W	0
26	GHDMODE: Setting this bit configures the A-RGMII module to expect half-duplex GMII at the GMII interface. It also enables the use of CRS and COL signals as described in A-RGMII documentation. When the M-MCXWOL module is integrated, this bit performs the function of WOLDTCTDCLR. When WOLDTCTDCLR is asserted, WOLDTCTD is held low. When WOLDTCTDCLR is cleared, WOLDTDCTD may become asserted appropriately.	R/W	0
25	LHDMODE: Setting this bit configures the A-RGMII module to expect 10 or 100 half-duplex MII at the GMII interface and will enable the use of CRS and COL signals as described in A-RGMII documentation. This bit should not be asserted unless this mode is being used.	R/W	0
24	PHY MODE: Setting this bit configures the PESMII Serial MII module to be in PHY Mode. Link characteristics are taken directly from the Rx segments supplied by the PHY. Clearing this bit configures the PESMII to be in MAC-to-MAC mode. In this configuration, the Serial MII module reverts to the pre-defined settings of 100 Mbps, Full-Duplex.	R/W	0
23	RESET PERMII: Setting this bit resets the PERMII module. Clearing this bit allows for normal operation	R/W	0
22:17	Reserved	R/W	0x00
16	SPEED: This bit configures the PERMII Reduced MII module with the current operating speed. When set, 100 Mbps mode is selected. When cleared, 10 Mbps mode is selected.	R/W	0
15	RESET PE100X: This bit resets the PE100X module, which contains the 4B / 5B symbol encipher / decipher logic.	R/W	0
14:11	Reserved	R/W	0x0
10	FORCE QUIET: When enabled, transmit data is quieted which allows the contents of the cipher to be output. When cleared, normal operation is enabled. Affects PE100X module only.	R/W	0
9	NO CIPHER: When enabled, the raw transmit 5B symbols are transmitted without ciphering. When disabled, normal ciphering occurs. Affects PE100X module only.	R/W	0
8	DISABLE LINK FAIL: When enabled, the 330 ms Link Fail timer is disabled, allowing shorter simulations. Removes the 330 ms link-up time before reception of streams is allowed. When cleared, normal operation occurs. Affects PE100X module only.	R/W	0
7	RESET GPSI: This bit resets the PE10T module which converts MII nibble streams to the serial bit stream of ENDEC PHYs. Affects PE10T module only.	R/W	0
6:1	Reserved	R/W	0x00
0	ENABLE JABBER PROTECTION: This bit enables the Jabber Protection logic within the PE10T in ENDEC mode. Jabber is the condition where a transmitter is stuck on for longer than 50 ms, preventing other stations from transmitting. Affects PE10T module only.	R/W	0

In the Interface Status Register, the range of bits that are active depends on which optional interfaces are connected.

Table 7-29 Port F GMAC Blocking Register Definitions — Interface Status

Bits	Description	Read/Write	Reset Value
31:11	Reserved	RO	0x000000
10	WOLDTCTD: This bit is only used when the optional M-MCXWOL module is integrated. It is set when the MAC detects a **Magic Packet** and stays high until it is cleared by the assertion of WOLDTCTDCLR. Its reset value is low.	RO/LH	0
9	EXCESS DEFER: This bit sets when the MAC excessively defers a transmission. It clears when read. This bit latches high.	RO/LH	0
8	CLASH: When read as a 1, the Serial MII module is in MAC-to-MAC mode with the partner in 10 Mbps and / or Half-Duplex mode indicative of a configuration error. When read as a 0, the Serial MII module is either in PHY mode or in a properly configured MAC-to-MAC mode.	RO	0
7	JABBER: When read as a 1, the Serial MII PHY has detected a jabber condition on the link. When read as a 0, the Serial MII PHY has not detected a jabber condition.	RO	0
6	LINK OK: When read as a 1, the Serial MII PHY has detected a valid link. When read as a 0, the Serial MII PHY has not detected a valid link.	RO	0
5	FULL DUPLEX: When read as a 1, the Serial MII PHY is operating in Full-Duplex mode. When read as a 0, the Serial MII PHY is operating in Half-Duplex mode.	RO	0
4	SPEED: When read as a 1, the Serial MII PHY is operating at 100 Mbps mode. When read as a 0, the Serial MII PHY is operating at 10 Mbps.	RO	0
3	LINK FAIL: When read as a 1, the MII Management module has read the PHY link fail register to be 1. When read as a 0, the MII Management module has read the PHY link fail register to be 0. Note that for asynchronous host accesses, this bit must be read at least once every scan read cycle of the PHY.	RO	0
2	LOSS OF CARRIER: When read as a 1, the PE10T module has detected a Loss of Carrier. When read as a 0, the PE10T module has not detected a Loss of Carrier. This bit latches high.	RO/LH	0
1	SQE ERROR: When read as a 1, the PE10T module has detected an SQE Error. When read as a 0, the PE10T module has not detected an SQE Error. This bit latches high.	RO/LH	0
0	JABBER: When read as a 1, the PE10T module has detected a Jabber condition. When read as a 0, the PE10T module has not detected a Jabber condition. This bit latches high.	RO/LH	0

Table 7-30 Port F GMAC Blocking Register Definitions — Station Address, Part 1

Bits	Description	Read/Write	Reset Value
31:24	STATION ADDRESS, 1st octet: This field holds the first octet of the station address.	R/W	0x00
24:16	STATION ADDRESS, 2nd octet: This field holds the second octet of the station	R/W	0x00
15:8	STATION ADDRESS, 3rd octet: This field holds the third octet of the station address.	R/W	0x00
7:0	STATION ADDRESS, 4th octet: This field holds the fourth octet of the station address.	R/W	0x00

Table 7-31 Port F GMAC Blocking Register Definitions — Station Address, Part 2

Bits	Description	Read/Write	Reset Value
31:24	STATION ADDRESS, 5th octet: This field holds the fifth octet of the station address.	R/W	0x00
23:16	STATION ADDRESS, 6th octet: This field holds the sixth octet of the station address.	R/W	0x00
15:0	Reserved	R/W	0x00

7.12   Port G Registers

7.12.1   Port G Function 1 (DDR SDRAM)

This section describes the function-specific attributes of the registers used when Port G Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

The DDR SDRAM interface uses both non-blocking registers and blocking registers. The non-blocking registers are listed first.

7.12.1.1   Port G DDR SDRAM Function Register

Bits	Field Name	Description	Read/ Write	Reset Value
31:13	Reserved		RO	0x00000
12:8	BR_TNUM	Blocking region thread number. The number of the thread that is allowed to access the blocking region for this port.	R/W	0x00
7:5	Not applicable to the DDR SDRAM function.		RO	0x0
4	FN_RESET	DDR SDRAM Function Reset	R/W	0x0
3	RX_FIFO_SEL	Not used by DDR SDRAM.	R/W	0x0
2:0	FN_SEL	'3h1 selects DDR SDRAM.	R/W	0xX

7.12.1.2   Port G DDR SDRAM Interrupt Status

Bits	Field Name	Description	Read/ Write	Reset Value
31:1	Reserved		RO	0x00000000
0	Controller Interrupt	DDR Controller level sensitive interrupt.	RO	0

7.12.1.3   Port G DDR SDRAM Interrupt Set

Bits	Field Name	Description	Read/ Write	Reset Value
31:1	Reserved		WO	0x00000000
0	Interrupt Set	Writing a 1 to this bit position sets the corresponding bit in the Interrupt Status register.	WO	0x0

7.12.1.4   Port G DDR SDRAM Transmit FIFO LO

Bits	Field Name	Description	Read/ Write	Reset Value
31:0	Not used by DDR SDRAM.		WO	0x00000000

7.12.1.5   Port G DDR SDRAM Transmit FIFO HI

Bits	Field Name	Description	Read/ Write	Reset Value
31:0	Not used by DDR SDRAM.		WO	0x00000000

7.12.1.6 Port G DDR SDRAM Receive FIFO LO

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used by DDR SDRAM.		RO	0xXXXXXXXX

7.12.1.7 Port G DDR SDRAM Receive FIFO HI

Bits	Field Name	Description	Read/Write	Reset Value
31:0	Not used by DDR SDRAM.		RO	0xXXXXXXXX

7.12.1.8 Port G DDR SDRAM Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31:4	Reserved		R/W	0x0
3:0	RX_FIFO_WATERMARK	Read Data FIFO Watermark.	R/W	0x0

7.12.1.9 Port G DDR SDRAM Blocking Region Registers

Table 7-32 shows the address map for these registers, and lists the parameters contained within each byte of each register. For each parameter, there is a high-level definition, access restrictions, reset value, and the range of possible values. Table 7-33 gives detailed definitions for the parameters listed in Table 7-32.

Table 7-32 Port G DDR SDRAM Blocking Register Definitions

Offset	Register Name	Bits	Parameter Name	Description	Read/Write	Reset Value
0x00	DDR_CTL_00	31:27	Reserved		WO	0x00
		26:24	INT_ACK	Clear mask of the INT_STATUS parameter.	WO	0x0
		23:20	Reserved		RO	0x0
		19:16	INT_STATUS	Status of interrupt features in the controller.	RO	0x0
		15:8	DLL_INCREMENT	Number of elements to add to DLL_START_POINT when searching for lock.	R/W	0x00
		7:0	DLL_START_POINT	Initial delay count when searching for lock in master DLL.	R/W	0x00

Table 7-32 Port G DDR SDRAM Blocking Register Definitions (continued)

Offset	Register Name	Bits	Parameter Name	Description	Read/Write	Reset Value
0x01	DDR_CTL_01	31:25	Reserved		R/W	0x00
		24	START	Initiate command processing in the controller.	R/W	0x0
		23:17	Reserved		RO	0x00
		16	OUT_OF_RANGE_TYPE	Type of command that caused an Out-of-Range interrupt.	RO	0x0
		15:13	Reserved		RO	0x0
		12:8	OUT_OF_RANGE_LENGTH	Length of command that caused an Out-of-Range interrupt.	RO	0x00
		7:4	Reserved		R/W	0x0
		3:0	INT_MASK	Mask for controller_int signals from the INT_STATUS parameter.	R/W	0x0
0x02	DDR_CTL_02	31:28	Reserved		R/W	0x0
		27:24	INITAREF	Number of auto-refresh commands to execute during DRAM initialization.	R/W	0x0
		23:17	Reserved		RO	0x
		16	MAX_CS_REG	Maximum number of chip selects available.	RO	0x1
		15:12	Reserved		RO	0x0
		11:8	MAX_COL_REG	Maximum width of column address in DRAMs.	RO	0xC
		7:4	Reserved		RO	0x0
		3:0	MAX_ROW_REG	Maximum width of memory address bus.	RO	0xE
0x03	DDR_CTL_03	31:19	Reserved		R/W	0x0000
		18:16	BSTLEN	Encoded burst length sent to DRAMs during initialization.	R/W	0x0
		15:11	Reserved		R/W	0x00
		10:8	CASLAT	Encoded CAS latency sent to DRAMs during initialization.	R/W	0x0
		7:4	Reserved		R/W	0x00
		3:0	CASLAT_LIN	Sets latency from read command send to data receive from / to controller.	R/W	0x0
0x04	DDR_CTL_04	31:19	Reserved		R/W	0x0000
		18:16	TRRD	DRAM TRRD parameter in cycles.	R/W	0x0
		15:13	Reserved		R/W	0x0
		12:8	TFAW	DRAM TFAW parameter in cycles.	R/W	0x00
		7:3	Reserved		R/W	0x00
		2:0	TRTP	DRAM TRTP parameter in cycles.	R/W	0x0
0x05	DDR_CTL_05	31:27	Reserved		R/W	0x00
		26:24	TEMRS	DRAM TEMRS parameter in cycles.	R/W	0x0
		23:20	Reserved		R/W	0x0
		19:16	TRP	DRAM TRP parameter in cycles.	R/W	0x0
		15:8	TRAS_MIN	DRAM TRAS_MIN parameter in cycles.	R/W	0x00
		7:5	Reserved		R/W	0x0
		4:0	TRC	DRAM TRC parameter in cycles.	R/W	0x00

Table 7-32 Port G DDR SDRAM Blocking Register Definitions (continued)

Offset	Register Name	Bits	Parameter Name	Description	Read/Write	Reset Value
0x06	DDR_CTL_06	31:27	Reserved		R/W	0x00
		26:24	TWTR	DRAM TWTR parameter in cycles.	R/W	0x0
		23:16	TDLL	DRAM TDLL parameter in cycles.	R/W	0x00
		15:14	Reserved		R/W	0x0
		13:8	TRFC	DRAM TRFC parameter in cycles.	R/W	0x00
		7:5	Reserved		R/W	0x0
		4:0	TMRD	DRAM TMRD parameter in cycles.	R/W	0x00
0x07	DDR_CTL_07	31:25	Reserved		R/W	0x00
		24	NO_CMD_INIT	Disable DRAM commands until TDLL has expired during initialization.	R/W	0x0
		23:17	Reserved		R/W	0x00
		16	EIGHT_BANK_MODE	Number of banks on the DRAM(s).	R/W	0x
		15:9	Reserved		WO	0x00
		8	AREFRESH	Initiate auto-refresh when specified by AUTO_REFRESH_MODE.	WO	0x0
		7:1	Reserved		R/W	0x00
		0	WRITEINTERP	Allow controller to interrupt write bursts to the DRAMs with a read command.	R/W	0x0
0x08	DDR_CTL_08	31:27	Reserved		R/W	0x00
		26:24	WRLAT	DRAM WRLAT parameter in cycles.	R/W	0x0
		23:16	TCPD	DRAM TCPD parameter in cycles.	R/W	0x00
		15:1	Reserved		R/W	0x0000
		0	DDRII_SDRAM_MODE	DDRI or DDRII mode.	R/W	0x0
0x09	DDR_CTL_09	31:25	Reserved		R/W	0x00
		24	AP	Enable auto pre-charge mode of controller.	R/W	0x0
		23:2	Reserved		R/W	0x0
		1:0	RTT_0	On-Die termination resistance setting for chip select 0.	R/W	0x0
0x0A	DDR_CTL_10	31:25	Reserved		R/W	0x00
		24	INTRPTAPBURST	Allow the controller to interrupt an auto pre-charge command with another command.	R/W	0x0
		23:17	Reserved		R/W	0x00
		16	INTRPTWRITEA	Allow the controller to interrupt a combined write command with auto pre-charge with another write command.	R/W	0x0
		15:9	Reserved		R/W	0x00
		8	INTRPTREADA	Allow the controller to interrupt a combined read with auto pre-charge command with another read command.	R/W	0x0
		7:1	Reserved		R/W	0x00
		0	CONCURRENTAP	Allow controller to issue commands to other banks while a bank is in auto pre-charge.	R/W	0x0

Table 7-32 Port G DDR SDRAM Blocking Register Definitions (continued)

Offset	Register Name	Bits	Parameter Name	Description	Read/ Write	Reset Value
0x0B	DDR_ CTL_11	31:25	Reserved		R/W	0x00
		24	DLL_BYPASS_ MODE	Enable the DLL bypass feature of the controller.	R/W	0x0
		23:17	Reserved		R/W	0x00
		16	REDUC	Enable the half datapath feature of the controller.	R/W	0x0
		15:8	TRCD_INT	DRAM TRCD parameter in cycles.	R/W	0x00
		7:1	Reserved		R/W	0x00
		0	TRAS_LOCKOUT	Allow the controller to execute auto pre-charge commands before TRAS_MIN expires.	R/W	0x0
0x0C	DDR_ CTL_12	31:24	DLL_LOCK	Number of delay elements in master DLL lock.	RO	0x00
		23:20	Reserved		R/W	0x0
		19:16	APREBIT	Location of the auto pre-charge bit in the DRAM address.	R/W	0x0
		15:11	Reserved		R/W	0x00
		10:8	COLUMN_SIZE	Difference between number of column pins available and number being used.	R/W	0x0
		7:3	Reserved		R/W	0x00
		2:0	ADDR_PINS	Difference between number of addr pins available and number being used.	R/W	0x0
0x0D	DDR_ CTL_13	31:23	Reserved		R/W	0x000
		22:16	DQS_OUT_SHIFT	Fraction of a cycle to delay the write dqs signal to the DRAMs during writes.	R/W	0x00
		15	Reserved		R/W	0x0
		14:8	DLL_DQS_ DELAY_1	Fraction of a cycle to delay the dqs signal from the DRAMs for dll_rd_dqs_slice 1 during reads.	R/W	0x00
		7	Reserved		R/W	0x0
		6:0	DLL_DQS_ DELAY_0	Fraction of a cycle to delay the dqs signal from the DRAMs for dll_rd_dqs_slice 0 during reads.	R/W	0x00
0x0E	DDR_ CTL_14	31:30	Reserved		R/W	0x0
		29:16	TREF	DRAM TREF parameter in cycles.	R/W	0x0000
		15:0	VERSION	Controller version number.	RO	0x2040
0x0F	DDR_ CTL_15	31:16	TINIT	DRAM TINIT parameter in cycles.	R/W	0x0000
		15:0	TRAS_MAX	DRAM TRAS_MAX parameter in cycles.	R/W	0x0000
0x10	DDR_ CTL_16	31:30	Reserved		R/W	0x0
		29:16	EMRS2_DATA	EMRS2 Data written during DDRII initialization.	R/W	0x0000
		15:14	Reserved		R/W	0x0
		13:0	EMRS_DATA	Extended mode register data written during initialization or when WRITE_MODEREG set.	R/W	0x0000
0x11	DDR_ CTL_17	31:14	Reserved		R/W	0x00000
		13:0	EMRS3_DATA	EMRS3 Data written during DDRII initialization.	R/W	0x0000

Table 7-32 Port G DDR SDRAM Blocking Register Definitions (continued)

Offset	Register Name	Bits	Parameter Name	Description	Read/Write	Reset Value
0x12	DDR_CTL_18	31:30	Reserved		RO	0x0
		29:0	OUT_OF_RANGE_ADDR[29:0]	Address of command that caused an Out-of-Range interrupt.	RO	0x00000000
0x13	DDR_CTL_19	31:24	WR_DQS_SHIFT_BYPASS	Fraction of a cycle to delay the clk_wr signal in the controller when DLL is being bypassed.	R/W	0x00
		23	Reserved		R/W	0x0
		22:16	WR_DQS_SHIFT	Fraction of a cycle to delay the clk_wr signal in the controller.	R/W	0x00
		15:9	Reserved		R/W	0x00
		8	AUTO_REFRESH_MODE	Sets whether auto-refresh will be at next burst or the next command boundary.	R/W	0x0
		7:4	Reserved		R/W	0x0
		3:0	CASLAT_LIN_GATE	Adjusts data capture gate open by half cycles.	R/W	0x0
0x14	DDR_CTL_20	31:24	DLL_DQS_DELAY_BYPASS_1	Fraction of a cycle to delay the dqs signal from the DRAMs for dll_rd_dqs_slice 1 during reads when DLL is being bypassed.	R/W	0x00
		23:16	DLL_DQS_DELAY_BYPASS_0	Fraction of a cycle to delay the dqs signal from the DRAMs for dll_rd_dqs_slice 0 during reads when DLL is being bypassed.	R/W	0x00
		15:12	Reserved		R/W	0x0
		11:8	TDAL	DRAM TDAL parameter in cycles.	R/W	0x0
		7:3	Reserved		R/W	0x00
		2:0	TWR_INT	DRAM TWR parameter in cycles.	R/W	0x0
0x15	DDR_CTL_21	31:10	Reserved		R/W	0x000000
		9:8	RTT_PAD_TERMINATION	Set termination resistance in controller pads.	R/W	0x0
		7:0	DQS_OUT_SHIFT_BYPASS	Fraction of a cycle to delay the write dqs signal to the DRAMs during writes when DLL is being bypassed.	R/W	0x00

Table 7-33 Detailed Descriptions of the Parameters in Table 7-32

Parameter	Description
ADDR_PINS [2:0]	Defines the difference between the maximum number of address pins configured (14) and the actual number of pins being used. The user address is automatically shifted so that the user address space is mapped contiguously into the memory map based on the value of this parameter.
AP [0]	Enables auto pre-charge mode for DRAM devices. • 0 = Auto pre-charge mode disabled. Memory banks will stay open until another request requires this bank, the maximum open time (tras_max) has elapsed, or a refresh command closes all the banks. • 1 = Auto pre-charge mode enabled. All read and write transactions must be terminated by an auto pre-charge command. If a transaction consists of multiple read or write bursts, only the last command is issued with an auto pre-charge.
APREBIT [3:0]	Defines the location of the auto pre-charge bit in the DRAM address in decimal encoding.
AREFRESH [0]	Initiates an automatic refresh to the DRAM devices based on the setting of the auto_refresh_mode parameter. If there are any open banks when this parameter is set, the DDR SDRAM controller will automatically close these banks before issuing the auto-refresh command. This parameter will always read back 0. • 0 = No action. • 1 = Issue refresh to the DRAM devices.
AUTO_REFRESH_MODE [0]	Sets the mode for when the automatic refresh will occur. If auto_refresh_mode is set and a refresh is required to memory, the controller will delay this refresh until the end of the current transaction (if the transaction is fully contained inside a single page), or until the current transaction hits the end of the current page. • 0 = Issue refresh on the next DRAM burst boundary, even if the current command is not complete. • 1 = Issue refresh on the next command boundary.
BSTLEN [2:0]	Defines the burst length encoding that will be programmed into the DRAM devices at initialization. • 001 = 2 words. • 010 = 4 words. • 011 = 8 words. • All other settings are reserved.
CASLAT [2:0]	Sets the CAS (Column Address Strobe) latency encoding that the memory uses. The binary value programmed into this parameter depends on the memory device, since the same caslat value may have different meanings to different memories. This will be programmed into the DRAM devices at initialization. The CAS encoding will be specified in the DRAM spec sheet, and should correspond to the caslat_lin parameter. Refer to the regconfig files in the release for actual settings for each particular device.

Table 7-33 Detailed Descriptions of the Parameters in Table 7-32 (continued)

Parameter	Description
CASLAT_LIN [3:0]	Sets the CAS latency linear value in 1/2 cycle increments. This sets an internal adjustment for the delay from when the read command is sent from the controller to when data will be received back. The window of time in which the data is captured is a fixed length. The caslat_lin parameter adjusts the start of this data capture window. Note: Not all linear values will be supported for the memory devices being used. Refer to the specification for the memory devices being used. • 0000 - 0010 = Reserved. • 0011 = 1.5 cycles. • 0100 = 2 cycles. • 0101 = 2.5 cycles. • 0110 = 3 cycles. • 0111 = 3.5 cycles. • 1000 = 4 cycles. • 1001 - 1111 = Reserved.
CASLAT_LIN_GATE [3:0]	Adjusts the data capture gate open time by 1/2 cycle increments. This parameter is programmed differently from caslat_lin when there are fixed offsets in the flight path between the memories and the controller for clock gating. When caslat_lin_gate is a larger value than caslat_lin, the data capture window will become shorter. A caslat_lin_gate value smaller than caslat_lin may have no effect on the data capture window, depending on the fixed offsets in the ASIC and the board. • 0000 - 0010 = Reserved. • 0011 = 1.5 cycles. • 0100 = 2 cycles. • 0101 = 2.5 cycles. • 0110 = 3 cycles. • 0111 = 3.5 cycles. • 1000 = 4 cycles. • 1001 - 1111 = Reserved.
COLUMN_SIZE [2:0]	Shows the difference between the maximum column width available (12) and the actual number of column pins being used. The user address is automatically shifted so that the user address space is mapped contiguously into the memory map based on the value of this parameter.
CONCURRENTAP [0]	Enables concurrent auto pre-charge. Some DRAM devices do not allow one bank to be auto pre-charged while another bank is reading or writing. The JEDEC standard allows concurrent auto pre-charge. Set this parameter for the DRAM device being used. • 0 = Concurrent auto pre-charge disabled. • 1 = Concurrent auto pre-charge enabled.
DDRII_SDRAM_MODE [0]	Enables DDRII mode. • 0 = DDRI mode. • 1 = DDRII mode.
DLL_BYPASS_MODE [0]	Defines the behavior of the DLL bypass logic. When set to 1, the values programmed into the parameters dll_dqs_delay, dqs_out_shift, and wr_dqs_shift become absolute values rather than fractional values of delays in the delay chains. In this mode, the DCC locking mechanism is bypassed. If the total delay time programmed into the delay parameters exceeds the number of delay elements in the delay chain, then the delay will be set to the maximum number of delay elements in the delay chain. • 0 = Normal operational mode. • 1 = Bypass the DLL master delay line.

Table 7-33 Detailed Descriptions of the Parameters in Table 7-32 (continued)

Parameter	Description
DLL_DQS_DELAY_X [6:0]	Sets the delay for the read_dqs signal from the DDR SDRAM devices for dll_rd_dqs_slice X. This delay is used to center the edges of the read_dqs signal so that the read data will be captured in the middle of the valid window in the I/O logic. Delay is added in increments of 1/128 of the system clock. The same delay will be added to the read_dqs signal for each byte of the read data.
DLL_DQS_DELAY_BYPASS_X [7:0]	Sets the delay for the read_dqs signal from the DDR SDRAM devices for dll_rd_dqs_slice X for reads when the DLL is being bypassed. This delay is used center the edges of the read_dqs signal so that the read data will be captured in the middle of the valid window in the I/O logic. Delay is added in increments of 1/128 of the system clock. The same delay will be added to the read_dqs signal for each byte of the read data.
DLL_INCREMENT [7:0]	Defines the number of delay elements by which to recursively increment the dll_start_point parameter when searching for lock.
DLL_LOCK [7:0]	Defines the actual number of delay elements used to capture one full clock cycle. This parameter is automatically updated every time a refresh operation is performed. This parameter is read-only.
DLL_START_POINT [7:0]	Sets the number of delay elements to place in the master delay line to start searching for lock in master DLL.
DQS_OUT_SHIFT [6:0]	Controls the amount of delay in fractions of a cycle introduced into the clk_dqs_out signal for the dll_wr_dqs_slice to ensure correct data capture in the I/ O logic.
DQS_OUT_SHIFT_BYPASS [7:0]	Controls the amount of delay in fractions of a cycle for the clk_dqs_out signal for the dll_wr_dqs_slice, when the DLL is being bypassed. This delay is introduced to ensure correct data capture in the I/O logic.
EIGHT_BANK_MODE [0]	Indicates whether the memory devices have four banks or eight banks. • 0 = Memory devices have 4 banks. • 1 = Memory devices have 8 banks.
EMRS_DATA [13:0]	Holds the Extended Mode Register data. The contents of this parameter will be programmed into the DRAM at initialization. Consult the DRAM specification for the correct settings for this parameter.
EMRS2_DATA [13:0]	Holds the EMRS2 data written during DDRII initialization.
EMRS3_DATA [13:0]	Holds the EMRS3 data written during DDRII initialization.
INITAREF [3:0]	Defines the number of auto-refresh commands needed by the DRAM devices to satisfy the initialization sequence.
INT_ACK [2:0]	Controls the clearing of the int_status parameter. If any of the int_ack bits are set to a 1, the corresponding bit in the int_status parameter will be set to 0. Any int_ack bits written with a 0 will not alter the corresponding bit in the int_status parameter. This parameter will always read back as 0.
INT_MASK [3:0]	Active-high mask bits that control the value of the controller_int signal on the ASIC interface. This mask is inverted and then logically ANDed with the outputs of the int_status parameter.

Table 7-33 Detailed Descriptions of the Parameters in Table 7-32 (continued)

Parameter	Description
INT_STATUS [3:0]	Shows the status of all possible interrupts generated by the controller. The MSB is the result of a logical OR of all the lower bits. This parameter is read-only. Note: Backwards compatibility is available for register parameters across configurations. However, even with this compatibility, the individual bits, their meaning and the size of the int_status parameter may change. The int_status bits correspond to these interrupts: • 0 = A single access outside the defined PHYSICAL memory space detected. • 1 = Multiple accesses outside the defined PHYSICAL memory space detected. • 2 = DRAM initialization complete. • 3 = Logical OR of all lower bits.
INTRPTAPBURST [0]	Enables interrupting an auto pre-charge command with another command for a different bank. If enabled, the current operation will be interrupted. However, the bank will be pre-charged as if the current operation were allowed to continue. • 0 = Disable interrupting an auto pre-charge operation on a different bank. • 1 = Enable interrupting an auto pre-charge operation on a different bank.
INTRPTREADA [0]	Enables interrupting of a combined read with auto pre-charge command with another read command to the same bank before the first read command is completed. • 0 = Disable interrupting the combined read with auto pre-charge command with another read command to the same bank. • 1 = Enable interrupting the combined read with auto pre-charge command with another read command to the same bank.
INTRPTWRITEA [0]	Enables interrupting of a combined write with auto pre-charge command with another read or write command to the same bank before the first write command is completed. • 0 = Disable interrupting a combined write with auto pre-charge command with another read or write command to the same bank. • 1 = Enable interrupting a combined write with auto pre-charge command with another read or write command to the same bank.
MAX_COL_REG [3:0]	Defines the maximum width of column address in the DRAM devices. This value can be used to set the column_size parameter. This parameter is read-only. column_size = max_col_reg - <number of column bits in memory device>.
MAX_CS_REG [0]	Defines the maximum number of chip selects for the controller.This parameter is read-only.
MAX_ROW_REG [3:0]	Defines the maximum width of the memory address bus (number of row bits) for the controller. This value can be used to set the addr_pins parameter. This parameter is read-only. addr_pins = max_row_reg - <number of row bits in memory device>.
NO_CMD_INIT [0]	Disables DRAM commands until DLL initialization is complete and tdll has expired. • 0 = Issue only REF and PRE commands during DLL initialization of the DRAM devices. • 1 = Do not issue any type of command during DLL initialization of the DRAM devices.
OUT_OF_RANGE_ ADDR [29:0]	Holds the address of the command that caused an out-of- range interrupt request to the memory devices. This parameter is read-only.
OUT_OF_RANGE_ LENGTH [4:0]	Holds the length of the command that caused an out-of-range interrupt request to the memory devices. This parameter is read-only.
OUT_OF_RANGE_TYPE [0]	Holds the type of command that caused an out-of-range interrupt request to the memory devices. This parameter is read-only.

Table 7-33 Detailed Descriptions of the Parameters in Table 7-32 (continued)

Parameter	Description
REDUC [0]	For controllers that can operate with a memory datapath width of half the standard with, this parameter sets the width. The entire user datapath is used regardless of this setting. • 0 = Standard operation using full memory bus. • 1 = Memory datapath width is half of the maximum size.
RTT_X [1:0]	Defines the On-Die termination resistance for a DRAM for chip select X. • 00 = Termination Disabled. • 01 = 75 Ohm. • 10 = 150 Ohm. • 11 = Reserved.
RTT_PAD_ TERMINATION [1:0]	Sets the termination resistance in the controller pads. The controller decodes this information and sets the param_75_ohm_sel output signal accordingly. The param_75_ohm_sel signal will be asserted if this parameter is set to 'b01 and deasserted for all other cases. This parameter also disables the output signal tsel, an active-high, dynamic signal which is used in the pads to enable termination on reads. If this parameter is set to 'b00, the tsel signal will be held low. • 00 = Termination Disabled. • 01 = 75 Ohm. • 10 = 150 Ohm. • 11 = Reserved.
START [0]	With this parameter set to 0, the controller will not issue any commands to the DRAM devices or respond to any signal activity except for reading and writing parameters. Once this parameter is set to 1, the controller will respond to inputs from the ASIC. The user should not set the start bit until the controller is fully initialized as indicated by the controller_int bit of the int_status parameter being set. Once operational, the start bit should not be cleared during operation. • 0 = Controller is not in active mode. • 1 = Initiate active mode for the controller.
TCPD [7:0]	Defines the clock enable to pre-charge delay time for the DRAM devices, in cycles.
TDAL [3:0]	Defines the auto pre-charge write recovery time when auto pre-charge is enabled (ap is set), in cycles. This is defined internally as tRP (pre-charge time) + Auto precharge write recovery time. Note that not all memories use this parameter. If tDAL is defined in the memory specification, then program this parameter to the specified value. If the memory does not specify a tDAL time, then program this parameter to tWR + tRP. DO NOT program this parameter with a value of 0x0, or the controller will not function properly when auto pre-charge is enabled.
TDLL [7:0]	Defines the DRAM DLL lock time, in cycles.
TEMRS [2:0]	Defines the DRAM extended mode parameter set time, in cycles.
TFAW [4:0]	Defines the DRAM tFAW parameter, in cycles.
TINIT [15:0]	Defines the DRAM initialization time, in cycles.
TMRD [4:0]	Defines the DRAM mode register set command time, in cycles.
TRAS_LOCKOUT [0]	Defines the tRAS lockout setting for the DRAM device. tRAS lockout allows the controller to execute auto pre-charge commands before the tras_min parameter has expired. • 0 = Tras lockout not supported by memory device. • 1 = Tras lockout supported by memory device.
TRAS_MAX [15:0]	Defines the DRAM maximum row active time, in cycles.
TRAS_MIN [7:0]	Defines the DRAM minimum row activate time, in cycles.
TRC [4:0]	Defines the DRAM period between active commands for the same bank, in cycles.

Table 7-33 Detailed Descriptions of the Parameters in Table 7-32 (continued)

Parameter	Description
TRCD_INT [7:0]	Defines the DRAM RAS to CAS delay, in cycles
TREF [13:0]	Defines the DRAM cycles between refresh commands.
TRFC [5:0]	Defines the DRAM refresh command time, in cycles.
TRP [3:0]	Defines the DRAM pre-charge command time, in cycles.
TRRD [2:0]	Defines the DRAM activate to activate delay for different banks, in cycles.
TRTP [2:0]	Defines the DRAM tRTP (read to pre-charge time) parameter, in cycles.
TWTR [2:0]	Sets the number of cycles needed to switch from a write to a read operation, as dictated by the DDR SDRAM specification.
VERSION [15:0]	Holds the version number for this controller. This parameter is read-only.
WR_DQS_SHIFT [6:0]	Controls the amount of delay introduced to the write datapath (the clk_wr signal) in fractions of a cycle, to ensure the correct capture of data internally in the I/O logic.
WR_DQS_SHIFT_BYPASS [7:0]	Controls the amount of delay introduced to the write datapath (the clk_wr signal) when the DLL is being bypassed. This delay is introduced in fractions of a cycle to ensure the correct capture of data internally in the I/O logic.
WRITEINTERP [0]	Defines whether the controller can interrupt a write burst with a read command. Some memory devices do not allow this functionality. • 0 = The device does not support read commands interrupting write commands. • 1 = The device does support read commands interrupting write commands.
WRLAT [2:0]	Defines the write latency from when the write command is issued to the time the write data is presented to the DRAM devices, in cycles.

7.13   Port H Registers

7.13.1   Port H Function 1 (DDR SDRAM)

The DDR SDRAM signals on Port H are an extension to those on Port G. These signals are used only when the DDR SDRAM device has a 16-bit wide data bus. In that case, Function Select for Port H must be set to 1 (DDR SDRAM). Control of these signals occurs via the Port G DDR SDRAM registers (see Section 7.12.1).

Port H is not used for DDR SDRAM if the DDR SDRAM has an 8-bit wide data bus.

7.14   Port I Registers

7.14.1   Port I Function 3 (MII)

The MII / RMII signals on Port I are an extension to those on Port E. Control of these signals occurs via the Port E MII / RMII registers (see Section 7.10.2).

7.15   USB Port Registers

7.15.1   USB Port Function 1 (USB)

This section describes the function-specific attributes of the registers used when USB Port Function 1 is selected.

NOTE: When a register is not mentioned, its function is the same as the generic descriptions given in Table 7-18.

The High-Speed USB interface uses both non-blocking registers and blocking registers. The non-blocking registers are listed first.

7.15.1.1   USB Port Interrupt Status

Bits	Field Name	Description	Read/Write	Reset Value
31:12	Refer to Table 7-18.		RO	0x00000
11:4	Reserved		RO	0x00
3	USB_CPU_Interrupt	If enabled in the controller, asserted upon assertion of TX or RX interrupt signal inside the controller.	RO	0x0
2	USB_SOF_PULSE	If enabled in the controller, asserted upon reception of an SOF packet.	RO	0x0
1:0	Reserved		RO	0x0

7.15.1.2   USB Port Interrupt Set

Bits	Field Name	Description	Read/Write	Reset Value
31	TX_FIFO _RESET	TX FIFO Reset. Writing a 1 to this bit resets the Transmit FIFO.	WO	0x0
30	RX_FIFO _RESET	RX FIFO Reset. Writing a 1 to this bit resets the Receive FIFO selected by the RX FIFO Select bit in the Function register.	WO	0x0
29:4	Reserved		WO	0x0000000
3:2	Interrupt Set	Writing a 1 to a given bit position sets the corresponding bit in the Interrupt Status register.	WO	0x0
1:0	Reserved		WO	0x0

7.15.1.3   USB Port Interrupt Clear

Bits	Field Name	Description	Read/Write	Reset Value
31:16	Reserved		WO	0x000
15:0	Interrupt Clear	Writing a 1 to a given bit position clears the corresponding bit in the Interrupt Status register.	WO	0x0000

7.15.1.4 USB Port Function Control 0

Bits	Field Name	Description	Read/Write	Reset Value
31:5	Reserved		R/W	0x0000000
4	USB_PHY_ RESETN	Active-Low logic Reset for the USB PHY. Does not stop any of the USB PHY clock outputs when asserted. Asserted at chip reset by default. Must be deasserted after the USB clock generator is active.	R/W	0x0
3:0	USB_CLK_DIV	Clock divisor value for the USB Core Clock Generator. 0x0: Stops the USB Core Clock at the end of the current USB Core Clock cycle. 0x1- Valid clock divisors. The clock frequency is core clock 0xF: divided by the value of this field + 1. For example, 1 produces core clock / 2, 2 produces core clock / 3, 3 produces core clock / 4, etc.	R/W	0x1

7.15.1.5 USB Port Function Status 0

Bits	Field Name	Description	Read/Write	Reset Value
31:4	Reserved		RO	0x0000000
3	USB_ POWERDWN	USB Core Powerdown Status	RO	0xX
2	USB_NRSTO	USB Core Reset Status	RO	0xX
1	USB cable power	When equal to 1, this bit indicates that 5 V power is present on the VBUS pin of the USB connector.	RO	0xX
0	USB clock valid	When equal to 1, this bit indicates that the 60 MHz clock output of the USB PHY is present and is accurate. When equal to 0, it indicates that the clock output is stopped and held in the low position.	RO	0xX

7.15.1.6   USB Port Blocking Region Registers

The USB Port blocking region registers reside at offsets 0x00 - 0xFF in the USB Port blocking region (see Table 7-16).

Table 7-34 shows the overall organization of the USB Port blocking region. The tables beginning with Table 7-35 describe the individual registers.

Table 7-34 Organization of the USB Port Blocking Region

Offset	Register Name	Description
Common USB Registers (0x00 – 0x0F)		
0x00	FAddr	Function address register.
0x01	Power	Power management register.
0x02-0x03	IntrTx	Interrupt register for Endpoint 0 plus Tx Endpoints 1 to 5.
0x04-0x05	IntrRx	Interrupt register for Rx Endpoints 1 to 5.
0x06-0x07	IntrTxE	Interrupt enable register for IntrTx.
0x08-0x09	IntrRxE	Interrupt enable register for IntrRx.
0x0A	IntrUSB	Interrupt register for common USB interrupts.
0x0B	IntrUSBE	Interrupt enable register for IntrUSB.
0x0C-0c0D	Frame	Frame number.
0x0E	Index	Index register for selecting the endpoint status and control registers.
0x0F	Testmode	Enables the USB test modes.
Indexed registers – Peripheral mode (0x10 – 0x1F) Control Status registers for endpoint selected by the Index register when the Host Mode bit (DevCtl[2]) = 0		
0x10-0x11	TxMaxP	Maximum packet size for peripheral Tx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x12-0x13	CSR0	Control Status register for Endpoint 0. (Index register set to select Endpoint 0).
	TxCSR	Control Status register for peripheral Tx endpoint. (Index register set to select Endpoints 1 – 5).
0x14-0x15	RxMaxP	Maximum packet size for peripheral Rx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x16-0x17	RxCSR	Control Status register for peripheral Rx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x18-0x19	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0).
	RxCount	Number of bytes in peripheral Rx endpoint FIFO. (Index register set to select Endpoints 1 – 5).
0x1A-0x1B	Reserved. Value returned affected by use in Host mode (see the following register group).	
Ox1C-0x1E	Unused, always returns 0.	
0x1F	ConfigData	Returns details of core configuration. (Index register set to select Endpoint 0.)
	FIFOSize	Returns the configured size of the selected Rx FIFO and Tx FIFOs (Endpoints 1 – 5 only).
Indexed registers – Host mode (0x10 – 0x1F) Control Status registers for endpoint selected by the Index register when the Host Mode bit (DevCtl[2]) = 1		
0x10-0x11	TxMaxP	Maximum packet size for host Tx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x12-0x13	CSR0	Control Status register for Endpoint 0. (Index register set to select Endpoint 0).
	TxCSR	Control Status register for host Tx endpoint. (Index register set to select Endpoints 1 – 5).

Table 7-34 Organization of the USB Port Blocking Region (continued)

Offset	Register Name	Description
0x14-0x15	RxMaxP	Maximum packet size for host Rx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x16-0x17	RxCSR	Control Status register for host Rx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x18-0x19	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0).
	RxCount	Number of bytes in host Rx endpoint FIFO. (Index register set to select Endpoints 1 – 5).
0x1A	Type0	Defines the speed of Endpoint 0. (Index register set to select Endpoint 0).
	TxType	Sets the transaction protocol, speed, and peripheral endpoint number for the host Tx endpoint. (Index register set to select Endpoints 1 – 5).
0x1B	NAKLimit0	Sets the NAK response timeout on Endpoint 0. (Index register set to select Endpoint 0).
	TxInterval	Sets the polling interval for Interrupt / ISOC transactions or the NAK response timeout on Bulk transactions for host Tx endpoint. (Index register set to select Endpoints 1 – 5 only).
Ox1C	RxType	Sets the transaction protocol, speed, and peripheral endpoint number for the host Rx endpoint. (Index register set to select Endpoints 1 – 5 only).
Ox1D	RxInterval	Sets the polling interval for Interrupt / ISOC transactions or the NAK response timeout on Bulk transactions for host Rx endpoint. (Index register set to select Endpoints 1 – 5 only).
0x1E	Unused, always returns 0.	
0x1F	ConfigData	Returns details of core configuration. (Index register set to select Endpoint 0.)
	FIFOSize	Returns the configured size of the selected Rx FIFO and Tx FIFOs (Endpoints 1 – 5 only).
FIFOs(0x20 – 0x5F)		
0x20-0x37	FIFOx	FIFOs for Endpoints 0 – 5.
0x38-0x5F	Reserved	
Device Control, Dynamic FIFO, Version & Vendor Registers (0x60 – 0x6F)		
0x60	DevCtl	Device Control register.
0x61	Unused	
0x62	TxFIFOsz	Tx Endpoint FIFO size. Used only when Dynamic FIFO sizing option is selected. Otherwise returns 0.
0x63	RxFIFOsz	Rx Endpoint FIFO size. Used only when Dynamic FIFO sizing option is selected. Otherwise returns 0.
0x64-0x65	TxFIFOadd	Tx Endpoint FIFO address. Used only when Dynamic FIFO sizing option is selected. Otherwise returns 0.
0x66-0x67	RxFIFOadd	Rx Endpoint FIFO address. Used only when Dynamic FIFO sizing option is selected. Otherwise returns 0.
0x68-0x6B	VControl/ VStatus	UTMI+ PHY Vendor registers
0x6C-0x6D	HWVers	Hardware Version Number Register
0x6E-0x6F	Unused	
Not Used (0x70 – 0x7F)		

Table 7-34 Organization of the USB Port Blocking Region (continued)

Offset	Register Name	Description
Target Address Registers (0x80 – 0xFF)		
0x80+8*n	TxFuncAddr	Transmit Endpoint n Function Address (Host Mode only)
0x81+8*n	Unused, always returns 0.	
0x82+8*n	TxHubAddr	Transmit Endpoint n Hub Address (Host Mode only)
0x83+8*n	TxHubPort	Transmit Endpoint n Hub Port (Host Mode only)
0x84+8*n	RxFuncAddr	Receive Endpoint n Function Address (Host Mode only)
0x85+8*n	Unused, always returns 0.	
0x86+8*n	RxHubAddr	Receive Endpoint n Hub Address (Host Mode only)
0x87+8*n	RxHubPort	Receive Endpoint n Hub Port (Host Mode only)

The tables beginning with Table 7-35 provide detailed information about the registers listed in Table 7-34.
The abbreviations in the Access columns have the following meanings:

- R/W means that the bit can be read or written.
- RO means that the bit is read only.
- Set means that the bit can be written only to set it.
- Clr means that the bit can be written only to clear it.
- RSet means that the bit can be read or set, but it cannot be cleared.

- RClr means that the bit can be read or cleared, but it cannot be set.
- SelfClr means that the bit will be cleared automatically when the associated action has been executed.
- p: means Peripheral Mode
- h: means Host Mode

Table 7-35 USB Port Blocking Region: Common USB Registers

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x00	Faddr			Function address register (peripheral mode only). This register should be written with the 7-bit address of the peripheral part of the transaction. When the USB controller is being used in Peripheral mode (DevCtl[2]=0), this register should be written with the address received through a SET_ADDRESS command, which will then be used for decoding the function address in subsequent token packets. This register applies only to operations carried out when the USB controller is in Peripheral mode. In Host mode, this register is ignored.			0x00
		7		Unused, always returns 0.	RO	-	
		6:0	Func Addr	The function address.	R/W	RO	

Table 7-35 USB Port Blocking Region: Common USB Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x01	Power			Power management register. This is an 8-bit register that is used for controlling Suspend and Resume signaling, and some basic operational aspects of the USB controller.			0x20
		7	ISO Update	When set by the CPU, the USB controller will wait for an SOF token from the time TxPktRdy is set before sending the packet. If an IN token is received before an SOF token, then a zero length data packet will be sent. Note: Valid in Peripheral Mode only. Also, this bit only affects only endpoints performing Isochronous transfers.	p:R/W	p:RO	
		6	Soft Conn	If Soft Connect/Disconnect feature is enabled, then the USB D+/D- lines are enabled when this bit is set by the CPU and tri-stated when this bit is cleared by the CPU. Valid only in Peripheral Mode.	p:R/W	p:RO	
		5	HS Enab	When set by the CPU, the USB controller will negotiate for High-speed mode when the device is reset by the hub. If not set, the device will operate only in Full-speed mode.	R/W	RO	
		4	HS Mode	When set, this read-only bit indicates High-speed mode successfully negotiated during USB reset. In Peripheral Mode, it becomes valid when USB reset completes (as indicated by USB reset interrupt). In Host Mode, it becomes valid when the Reset bit is cleared. It remains valid for the duration of the session. Note: Allowance is made for Tiny-J signaling in determining the transfer speed to select.	RO	R/W	
		3	Reset	This bit is set when Reset signaling is present on the bus. Note: This bit is Read/Write from the CPU in Host Mode but Read-Only in Peripheral Mode.	p:RO h:R/W	R/W	
		2	Resume	Set by the CPU to generate Resume signaling when the function is in Suspend mode. The CPU should clear this bit after 10 ms (a maximum of 15 ms) to end Resume signaling. In Host mode, this bit is also automatically set when Resume signaling from the target is detected while the USB controller is suspended.	R/W	p:RO h:RSet	
		1	Suspend Mode	In Host mode, this bit is set by the CPU to enter Suspend mode. In Peripheral mode, this bit is set on entry into Suspend mode. It is cleared when the CPU reads the interrupt register, or sets the Resume bit above.	p:RO h:Set	p:R/W h:Clr	
		0	Enable SuspendM	Set by the CPU to enable the SUSPENDM signal.	R/W	RO	

Table 7-35 USB Port Blocking Region: Common USB Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x02-0x03	IntrTx			Interrupt register for Endpoint 0 plus Tx Endpoints 1 to 5. This is a 16-bit read-only register that indicates which interrupts are currently active for Endpoint 0 and the Tx Endpoints 1 to 5. Bits relating to endpoints that have not been configured will always return 0. Note also that all active interrupts are cleared when this register is read.	RO	Set	0x0000
		15:14		Unused, always returns 0.			
		13	EP5 Tx	Tx Endpoint 5 interrupt.			
		12	EP4 Tx	Tx Endpoint 4 interrupt.			
		11	EP3 Tx	Tx Endpoint 3 interrupt.			
		10	EP2 Tx	Tx Endpoint 2 interrupt.			
		9	EP1 Tx	Tx Endpoint 1 interrupt.			
		8	EP0	Endpoint 0 interrupt.			
		7:0		Unused, always returns 0.			
0x04-0x05	IntrRx			Interrupt register for Rx Endpoints 1 to 5. This is an 16-bit read-only register that indicates which of the interrupts for Rx Endpoints 1 to 5 are currently active. Bits relating to endpoints that have not been configured will always return 0. Note also that all active interrupts are cleared when this register is read.	RO	Set	0x0000
		15:14		Unused, always returns 0.			
		13	EP5 Rx	Rx Endpoint 5 interrupt.			
		12	EP4 Rx	Rx Endpoint 4 interrupt.			
		11	EP3 Rx	Rx Endpoint 3 interrupt.			
		10	EP2 Rx	Rx Endpoint 2 interrupt.			
		9	EP1 Rx	Rx Endpoint 1 interrupt.			
		8:0		Unused, always returns 0.	RO	RO	
0x06-0x07	IntrTxE			Interrupt enable register for IntrTx. This is a 16-bit register that provides interrupt enable bits for the interrupts in IntrTx. On reset, the bits corresponding to Endpoint 0 and the Tx endpoints included in the design are set to 1, while the remaining bits are set to 0. Bits relating to endpoints that have not been configured will always return 0.	R/W	RO	0x0F00
		15:14		Unused, always returns 0.			
		13	EP5 Tx	Tx Endpoint 5 interrupt enable.			
		12	EP4 Tx	Tx Endpoint 4 interrupt enable.			
		11	EP3 Tx	Tx Endpoint 3 interrupt enable.			
		10	EP2 Tx	Tx Endpoint 2 interrupt enable.			
		9	EP1 Tx	Tx Endpoint 1 interrupt enable.			
		8	EP0 Tx	Endpoint 0 interrupt enable.			
		7:0		Unused, always returns 0.			

Table 7-35 USB Port Blocking Region: Common USB Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x08-0x09	IntrRxE			Interrupt enable register for IntrRx. This is a 16-bit register that provides interrupt enable bits for the interrupts in IntrRx. On reset, the bits corresponding to the Rx endpoints included in the design are set to 1, while the remaining bits are set to 0. Bits relating to endpoints that have not been configured will always return 0.	R/W	RO	0xFE00
		15:14	Unused, always returns 0.				
		13	EP5 Rx	Rx Endpoint 5 interrupt enable.			
		12	EP4 Rx	Rx Endpoint 4 interrupt enable.			
		11	EP3 Rx	Rx Endpoint 3 interrupt enable.			
		10	EP2 Rx	Rx Endpoint 2 interrupt enable.			
		9	EP1 Rx	Rx Endpoint 1 interrupt enable.			
		8:0	Unused, always returns 0.		RO	RO	
0x0A	IntrUSB			Interrupt register for common USB interrupts. This is an 8-bit read-only register that indicates which USB interrupts are currently active. All active interrupts will be cleared when this register is read.	RO	Set	0x00
		7	VBus Error	Set when VBus drops below the VBus Valid threshold during a session. Valid only when the USB controller is a Host device.			
		6	Sess Req	Set when Session Request signaling has been detected. Valid only when the USB controller is a Host device.			
		5	Discon	Set in Host mode when a device disconnect is detected. Set in Peripheral mode when a session ends. Valid at all transaction speeds.			
		4	Conn	Set when a device connection is detected. Valid only in Host mode. Valid at all transaction speeds.			
		3	SOF	Start of Frame. Set when a new frame starts.			
		2	Reset	Set in Peripheral mode when Reset signaling is detected on the bus.			
			Babble	Set in Host mode when babble is detected.			
		1	Resume	Set when Resume signaling is detected on the bus while the USB controller is in Suspend mode.			
		0	Suspend	Set when Suspend signaling is detected on the bus. Valid only in Peripheral mode.			
0x0B	IntrUSBE			Interrupt enable register for IntrUSB. This is an 8-bit register that provides interrupt enable bits for each of the interrupts in IntrUSB.	R/W	RO	0x06
		7	VBus Error	VBus Error interrupt enable.			
		6	Sess Req	Sess Req interrupt enable.			
		5	Discon	Discon interrupt enable.			
		4	Conn	Conn interrupt enable.			
		3	SOF	SOF interrupt enable.			
		2	Reset / Babble	Reset / Babble interrupt enable.			
		1	Resume	Resume interrupt enable.			
		0	Suspend	Suspend interrupt enable.			

Table 7-35 USB Port Blocking Region: Common USB Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x0C-0x0D	Frame			Frame number. Frame is a 16-bit read-only register that holds the last received frame number.	RO	WO	0x0000
		7:3	00000	Always zero.			
		2:0, 15:8	Frame Number	The last received frame number.			
0x0E	Index			Index register for selecting the endpoint status and control registers. Each Tx endpoint and each Rx endpoint has its own set of control / status registers located between 0x100 and 0x1FF. In addition, one set of Tx control / status and one set of Rx control / status registers appear at 0x10 – 0x19. Index is a 4-bit register that determines which endpoint control / status registers are accessed. Before accessing an endpoint's control / status registers at 0x10 – 0x19, the endpoint number should be written to the Index register to ensure that the correct control / status registers appear in the memory map.			0x00
		7:4	Unused, always returns 0.		RO	RO	
		3:0	Selected Endpoint	The 4-bit selected endpoint value. Bit 3 is the MSB and Bit 0 is the LSB.	R/W	RO	

Table 7-35 USB Port Blocking Region: Common USB Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x0F	Test-mode			Enables the USB test modes. This is an 8-bit register that is primarily used to put the USB controller into one of the four test modes for High-speed operation described in the USB specification – in response to a SET FEATURE: TESTMODE command. It is not used in normal operation.			0x00
		7	Force_Host	The CPU sets this bit to instruct the core to enter Host mode when the Session bit is set, regardless of whether it is connected to any peripheral. The state of the CID input, HostDisconnect and LineState signals are ignored. The core will then remain in Host mode until the Session bit is cleared, even if a device is disconnected, and if the Force_Host bit remains set, will re-enter Host mode the next time the Session bit is set. While in this mode, the status of the HOSTDISCON signal from the PHY may be read from bit 7 of the DevCtl register. The operating speed is determined from the Force_HS and Force_FS bits as follows: 	Force_HS	Force_FS	Operating Speed
---	---	---
0	0	Undefined
0	1	Full Speed
1	0	High Speed
1	1	Undefined		R/W	RO	
		6	FIFO_Access (self-clearing)	The CPU sets this bit to transfer the packet in the Endpoint 0 Tx FIFO to the Endpoint 0 Rx FIFO. It is cleared automatically when the packet has been transferred.	Set	RO	
		5	Force_FS	The CPU sets this bit either in conjunction with bit 7 above or to force the USB controller into Full-Speed mode when it receives a USB reset.	R/W	RO	
		4	Force_HS	The CPU sets this bit either in conjunction with bit 7 above or to force the USB Controller into High-Speed mode when it receives a USB reset.	R/W	RO	
		3	Test_Packet	(High-speed mode) The CPU sets this bit to enter the Test_Packet test mode. In this mode, the USB controller repetitively transmits on the bus a 53-byte test packet, defined in the Universal Serial Bus Specification Revision 2.0, Section 7.1.20. The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered.	R/W	RO	
		2	Test_K	(High-speed mode) The CPU sets this bit to enter the Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.	R/W	RO	
		1	Test_J	(High-speed mode) The CPU sets this bit to enter the Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.	R/W	RO	
		0	Test_SE0_NAK	(High-speed mode) The CPU sets this bit to enter the Test_SE0_NAK test mode. In this mode, the USB controller remains in High-speed mode but responds to any valid IN token with a NAK.	R/W	RO	

Table 7-36 USB Port Blocking Region: Indexed registers – Peripheral mode

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x10-0x11	TxMaxP			Maximum packet size for peripheral Tx endpoint. (Index register set to select Endpoints 1 – 5 only). The maximum amount of data that can be transferred through the selected Tx endpoint in a single operation. There is a TxMaxP register for each Tx endpoint (except Endpoint 0).	R/W	RO	0x0000
		7:3	m-1	Where the option of High-bandwidth Isochronous / Interrupt endpoints or of packet splitting on Bulk endpoints has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.			
		2:0 15:8	Max Payload (2:0 are MSB, 15:8 are LSB)	Maximum payload per transaction. These 11 bits define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB 2.0 Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations. The total amount of data represented by the value written to this register (specified payload × m) must not exceed the FIFO size for the Tx endpoint, and should not exceed half the FIFO size if double-buffering is required.			
0x12-0x13	CSR0			Control Status register for Endpoint 0. (Index register set to select Endpoint 0)			0x0000
		15	ServicedSetupEnd	The CPU writes a 1 to this bit to clear the SetupEnd bit. It is cleared automatically.	Set	RO	
		14	ServicedRxPktRdy	The CPU writes a 1 to this bit to clear the RxPktRdy bit. It is cleared automatically.	Set	RO	
		13	SendStall	The CPU writes a 1 to this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically.	Set	RO	
		12	SetupEnd	This bit will be set when a control transaction ends before the DataEnd bit has been set. An interrupt will be generated and the FIFO flushed at this time. The bit is cleared by the CPU writing a 1 to the ServicedSetupEnd bit.	RO	Set	
		11	DataEnd	The CPU sets this bit: 1. When setting TxPktRdy for the last data packet. 2. When clearing RxPktRdy after unloading the last data packet. 3. When setting TxPktRdy for a zero length data packet. It is cleared automatically.	Set	RO	
		10	SentStall	This bit is set when a STALL handshake is transmitted. The CPU should clear this bit.	RClr	Set	
		9	TxPktRdy	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.	RSet	RO	

Table 7-36 USB Port Blocking Region: Indexed registers – Peripheral mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		8	RxPktRdy	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. The CPU clears this bit by setting the ServicedRxPktRdy bit.	RO	Set	
		7:1	Unused. Returns 0 when read.		RO	RO	
		0	FlushFIFO	The CPU writes a 1 to this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the TxPktRdy / RxPktRdy bit (below) is cleared. FlushFIFO should be used only when TxPktRdy/RxPktRdy is set. At other times, it may cause data to be corrupted.	Set	RO	
0x12-0x13	TxCSR		Control Status register for peripheral Tx endpoint. (Index register set to select Endpoints 1 – 5). There is a TxCSR register for each configured Tx endpoint (not including Endpoint 0).			0x0000
		15	IncompTx	When the endpoint is being used for high-bandwidth Isochronous / Interrupt transfers, this bit is set to indicate where a large packet has been split into 2 or 3 packets for transmission but insufficient IN tokens have been received to send all the parts. Note: In anything other than a high bandwidth transfer, this bit will always return 0.	RClr	Set	
		14	ClrDataTog	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.	Set	RClr	
		13	SentStall	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the TxPktRdy bit is cleared (see below). The CPU should clear this bit.	RClr	Set	
		12	SendStall	The CPU writes a 1 to this bit to issue a STALL handshake to an IN token. The CPU clears this bit to terminate the stall condition. Note: This bit has no effect where the endpoint is being used for Isochronous transfers.	R/W	RO	
		11	FlushFIFO	The CPU writes a 1 to this bit to flush the latest packet from the endpoint Tx FIFO. The FIFO pointer is reset, the TxPktRdy bit (below) is cleared and an interrupt is generated. May be set simultaneously with TxPktRdy to abort the packet that is currently being loaded into the FIFO. FlushFIFO should only be used when TxPktRdy is set. At other times, it may cause data to be corrupted. If the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.	Set	RO	
		10	UnderRun	The USB sets this bit if an IN token is received when the TxPktRdy bit not set. The CPU should clear this bit.	RClr	Set	
		9	FIFONotEmpty	The USB sets this bit when there is at least one packet in the Tx FIFO.	RClr	Set	

Table 7-36 USB Port Blocking Region: Indexed registers – Peripheral mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		8	TxPktRdy	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). TxPktRdy is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	RSet	Clr	
		7	AutoSet	If the CPU sets this bit, TxPktRdy will be automatically set when data of the maximum packet size (value in TxMaxP) is loaded into the Tx FIFO. If a packet of less than the maximum packet size is loaded, then TxPktRdy will have to be set manually. This bit should not be set for high-bandwidth Isochronous endpoints.	R/W	RO	
		6	ISO	The CPU sets this bit to enable the Tx endpoint for Isochronous transfers, and clears it to enable the Tx endpoint for Bulk or Interrupt transfers. Note: This bit has any effect only in Peripheral mode. In Host mode, it always returns zero.	R/W	RO	
		5	Mode	The CPU sets this bit to enable the endpoint direction as Tx, and clears it to enable the endpoint direction as Rx. Note: This bit has any effect only where the same endpoint FIFO is used for both Tx and Rx transactions.	R/W	RO	
		4	Unused. Returns 0 when read.		RO	RO	
		3	FrcDataTog	The CPU sets this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints.	R/W	RO	
		2:0	Unused, always returns 0.		RO	RO	

Table 7-36 USB Port Blocking Region: Indexed registers – Peripheral mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x14-0x15	RxMaxP			Maximum packet size for peripheral Rx endpoint. (Index register set to select Endpoints 1 – 5 only). The maximum amount of data that can be transferred through the selected Rx endpoint in a single operation. There is an RxMaxP register for each Rx endpoint (except Endpoint 0).			
		7:3	m-1	Where the option of High-bandwidth Isochronous / Interrupt endpoints or of combining Bulk packets has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.			
		2:0 15:8	Max Payload (2:0 are MSB, 15:8 are LSB)	Maximum payload per transaction. These 11 bits define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the USB 2.0 Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations. The total amount of data represented by the value written to this register (specified payload × m) must not exceed the FIFO size for the OUT endpoint, and should not exceed half the FIFO size if double-buffering is required.			
0x16-0x17	RxCSR			Control Status register for peripheral Rx endpoint. (Index register set to select Endpoints 1 – 5 only). There is an RxCSR register for each configured Rx endpoint (not including Endpoint 0).			0x0000
		15	ClrDataTog	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.	Set	RClr	
		14	SentStall	This bit is set when a STALL handshake is transmitted. The CPU should clear this bit.	RClr	Set	
		13	SendStall	The CPU writes a 1 to this bit to issue a STALL handshake. The CPU clears this bit to terminate the stall condition. This bit has no effect where the endpoint is being used for Isochronous transfers.	R/W	RO	
		12	FlushFIFO	The CPU writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO. The FIFO pointer is reset and the RxPktRdy bit (below) is cleared. FlushFIFO should be used only when RxPktRdy is set. At other times, it may cause data to be corrupted. If the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.	Set	RO	
		11	DataError	This bit is set when RxPktRdy is set if the data packet has a CRC or bit-stuff error. It is cleared when RxPktRdy is cleared. This bit is valid only when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.	RO	Set	
		10	OverRun	This bit is set if an OUT packet cannot be loaded into the Rx FIFO. The CPU should clear this bit. This bit is valid only when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.	RClr	Set	

Table 7-36 USB Port Blocking Region: Indexed registers – Peripheral mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		9	FIFOFull	This bit is set when no more packets can be loaded into the Rx FIFO.	RO	Set	
		8	RxPktRdy	This bit is set when a data packet has been received. The CPU should clear this bit when the packet has been unloaded from the Rx FIFO. An interrupt is generated when this bit is set.	RCIR	Set	
		7	AutoClear	If the CPU sets this bit, then the RxPktRdy bit will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, RxPktRdy will have to be cleared manually. Note: Should not be set for high-bandwidth Isochronous endpoints.	R/W	RO	
		6	ISO	The CPU sets this bit to enable the Rx endpoint for Isochronous transfers, and clears it to enable the Rx endpoint for Bulk / Interrupt transfers.	R/W	RO	
		5	Unused. Returns 0 when read.		RO	RO	
		4	DisNyet	Bulk/Interrupt Transactions: The CPU sets this bit to disable the sending of NYET handshakes. When set, all successfully received Rx packets are ACKed, including at the point at which the FIFO becomes full. Note: This bit only has any effect only in High-speed mode, in which mode it should be set for all Interrupt endpoints.	R/W / RO	RO / R/W	
			PID Error	ISO Transactions: The core sets this bit to indicate a PID error in the received packet.			
		3:1	Unused, always returns 0.		RO	RO	
		0	IncompRx	This bit is set in a high-bandwidth Isochronous transfer if the packet in the Rx FIFO is incomplete because parts of the data were not received. It is cleared when RxPktRdy is cleared. In anything other than a high-bandwidth Isochronous transfer, this bit will always return 0.	RO	Set	
0x18-0x19	Count0	Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0). The value returned changes as the contents of the FIFO change and is valid only while RxPktRdy (CSR0[0]) is set.					0x00
		14:8	Count0	Endpoint 0 Rx Count.	RO	WO	
		15 7:0	Unused, always returns 0.		RO	RO	
0x18-0x19	RxCount	Number of bytes in peripheral Rx endpoint FIFO. (Index register set to select Endpoints 1 – 5). The value returned changes as the FIFO is unloaded and is valid only while RxPktRdy (RxCSR[0]) is set.					0x0000
		4:0 15:8	RxCount (4:0 are MSB, 15:8 are LSB)	Endpoint Rx Count.	RO	WO	
		7:3	Unused, always returns 0.		RO	RO	
0x1A-0x1B	Reserved. Value returned affected by use in Host mode (see Table 7-37).						

Table 7-36 USB Port Blocking Region: Indexed registers – Peripheral mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x1C-0x1E	Unused, always returns 0.						
0x1F	Config Data		Returns details of core configuration. (Index register set to select Endpoint 0.)		RO	RO	Configuration Dependent
		7	MPRxE	Set to 1 when automatic amalgamation of bulk packets is selected.			
		6	MPTxE	Set to 1 when automatic splitting of bulk packets is selected.			
		5	BigEndian	Set to 1 when Big Endian ordering is selected.			
		4	HBRxE	Set to 1 when High-bandwidth Rx ISO Endpoint Support is selected.			
		3	HBTxE	Set to 1 when High-bandwidth Tx ISO Endpoint Support is selected.			
		2	DynFIFO Sizing	Set to 1 when Dynamic FIFO Sizing option is selected.			
		1	SoftConE	Set to 1 when Soft Connect/Disconnect option is selected.			
		0	UTMI DataWidth	Indicates selected UTMI+ data width: 1: 16 bits 0: 8 bits			
0x1F	FIFO Size			Returns the sizes of the FIFOs associated with the selected additional Tx and Rx endpoints. Values 3 – 13 correspond to a FIFO size of 2^n bytes (8 – 8192 bytes). If an endpoint has not been configured, a value of 0 will be displayed. Where the Tx and Rx endpoints share the same FIFO, the Rx FIFO size will be encoded as 0xF. Note: This register has this interpretation only when the Index register is set to select one of Endpoints 1 – 5 and Dynamic FIFO Sizing is not selected. It has a special interpretation when the Index register is set to select Endpoint 0 (see ConfigData above). The result returned is not valid where Dynamic FIFO sizing is used.	RO	RO	Configuration Dependent
		7:4	Rx FIFO Size	Size of the Rx FIFO for the selected Rx endpoint.			
		3:0	Tx FIFO Size	Size of the Tx FIFO for the selected Tx endpoint.			

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x10-0x11	TxMaxP			Maximum packet size for host Tx endpoint. (Index register set to select Endpoints 1 – 5 only). The maximum amount of data that can be transferred through the selected Tx endpoint in a single operation. There is a TxMaxP register for each Tx endpoint (except Endpoint 0).	R/W	RO	0x0000
		7:3	m-1	Where the option of High-bandwidth Isochronous / Interrupt endpoints or of packet splitting on Bulk endpoints has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.			
		2:0 15:8	Max Payload (2:0 are MSB, 15:8 are LSB)	Maximum payload per transaction. These 11 bits define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the USB 2.0 Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations. The total amount of data represented by the value written to this register (specified payload × m) must not exceed the FIFO size for the Tx endpoint, and should not exceed half the FIFO size if double-buffering is required.			
0x12-0x13	CSR0			Control Status register for Endpoint 0. (Index register set to select Endpoint 0)			0x0000
		15	NAK Timeout	This bit will be set when Endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the NAKLimit0 register. The CPU should clear this bit to allow the endpoint to continue.	RClr	Set	
		14	StatusPkt	The CPU sets this bit at the same time as the TxPktRdy or ReqPkt bit is set, to perform a status stage transaction. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the Status Stage transaction.	R/W	RO	
		13	ReqPkt	The CPU sets this bit to request an IN transaction. It is cleared when RxPktRdy is set.	R/W	R/W	
		12	Error	This bit will be set when three attempts have been made to perform a transaction with no response from the peripheral. The CPU should clear this bit. An interrupt is generated when this bit is set.	RClr	Set	
		11	SetupPkt	The CPU sets this bit, at the same time as the TxPktRdy bit is set, to send a SETUP token instead of an OUT token for the transaction. Note: Setting this bit also clears the Data Toggle.	RClr	RO	
		10	RxStall	This bit is set when a STALL handshake is received. The CPU should clear this bit.	RClr	Set	
		9	TxPktRdy	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.	RSet	Clr	

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		8	RxPktRdy	This bit is set when a data packet has been received. An interrupt is generated (if enabled) when this bit is set. The CPU should clear this bit when the packet has been read from the FIFO.	RClr	R/W	
		7:4	Unused. Returns 0 when read.		RO	RO	
		3	Dis Ping	The CPU writes a 1 to this bit to instruct the core not to issue PING tokens in data and status phases of a high-speed Control transfer (for use with devices that do not respond to PING).	R/W	RO	
		2	Data Toggle Write Enable	The CPU writes a 1 to this bit to enable the current state of the Endpoint 0 data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.	Set	RO	
		1	Data Toggle	When read, this bit indicates the current state of the Endpoint 0 data toggle. If bit 10 is high, this bit may be written with the required setting of the data toggle. If bit 10 is low, any value written to this bit is ignored.	R/W	R/W	
		0	FlushFIFO	The CPU writes a 1 to this bit to flush the next packet to be transmitted / read from the Endpoint 0 FIFO. The FIFO pointer is reset and the TxPktRdy / RxPktRdy bit (below) is cleared. FlushFIFO should be used only when TxPktRdy / RxPktRdy is set. At other times, it may cause data to be corrupted.	Set	RO	
0x12-0x13	TxCSR	Control Status register for host Tx endpoint. (Index register set to select Endpoints 1 – 5). There is a TxCSR register for each configured Tx endpoint (not including Endpoint 0).					0x0000
		15	NAK Timeout	This bit will be set when the Tx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the TxInterval register. The CPU should clear this bit to allow the endpoint to continue. Valid only for Bulk endpoints.	RClr	Set	
		14	ClrDataTog	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.	Set	RClr	
		13	RxStall	This bit is set when a STALL handshake is received. When this bit is set, the FIFO is completely flushed and the TxPktRdy bit is cleared (see below). The CPU should clear this bit.	RClr	Set	
		12	SetupPkt	The CPU sets this bit, at the same time as the TxPktRdy bit is set, to send a SETUP token instead of an OUT token for the transaction. Setting this bit also clears the Data Toggle.	R/W	RO	

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		11	FlushFIFO	The CPU writes a 1 to this bit to flush the latest packet from the endpoint Tx FIFO. The FIFO pointer is reset, the TxPktRdy bit (below) is cleared and an interrupt is generated. May be set simultaneously with TxPktRdy to abort the packet that is currently being loaded into the FIFO. FlushFIFO should be used only when TxPktRdy is set. At other times, it may cause data to be corrupted. If the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.	Set	RO	
		10	Error	The USB sets this bit when three attempts have been made to send a packet and no handshake packet has been received. When the bit is set, an interrupt is generated, TxPktRdy is cleared and the FIFO is completely flushed. The CPU should clear this bit. Valid only when the endpoint is operating in Bulk or Interrupt mode.	RClr	R/W	
		9	FIFONotEmpty	The USB sets this bit when there is at least one packet in the Tx FIFO.	RClr	Set	
		8	TxPktRdy	The CPU sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). TxPktRdy is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	RSet	Clr	
		7	AutoSet	If the CPU sets this bit, TxPktRdy will be automatically set when data of the maximum packet size (value in TxMaxP) is loaded into the Tx FIFO. If a packet of less than the maximum packet size is loaded, then TxPktRdy will have to be set manually. This bit should not be set for high-bandwidth Isochronous endpoints.	R/W	RO	
		6	Unused, always returns zero.		RO	RO	
		5	Mode	The CPU sets this bit to enable the endpoint direction as Tx, and clears it to enable the endpoint direction as Rx. This bit has any effect onlywhere the same endpoint FIFO is used for both Tx and Rx transactions.	R/W	RO	
		4	Unused. Returns 0 when read.		RO	RO	
		3	FrcDataTog	The CPU sets this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints.	R/W	RO	
		2	Unused. Returns 0 when read.		RO	RO	
		1	Data Toggle Write Enable	The CPU writes a 1 to this bit to enable the current state of the Tx Endpoint data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.	Set	RO	

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		0	Data Toggle	When read, this bit indicates the current state of the Tx Endpoint data toggle. If bit 1 is high, this bit may be written with the required setting of the data toggle. If bit 1 is low, any value written to this bit is ignored.	R/W	R/W	
0x14-0x15	RxMaxP			Maximum packet size for host Rx endpoint. (Index register set to select Endpoints 1 – 5 only). The maximum amount of data that can be transferred through the selected Rx endpoint in a single operation. There is an RxMaxP register for each Rx endpoint (except Endpoint 0).			
		7:3	m-1	Where the option of High-bandwidth Isochronous / Interrupt endpoints or of combining Bulk packets has been taken when the core is configured, the register includes either 2 or 5 further bits that define a multiplier m which is equal to one more than the value recorded.			
		2:0 15:8	Max Payload (2:0 are MSB, 15:8 are LSB)	Maximum payload per transaction. These 11 bits define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the USB 2.0 Specification on packet sizes for Bulk, Interrupt and Isochronous transfers in Full-speed and High-speed operations. The total amount of data represented by the value written to this register (specified payload × m) must not exceed the FIFO size for the OUT endpoint, and should not exceed half the FIFO size if double-buffering is required.			
0x16-0x17	RxCSR			Control Status register for host Rx endpoint. (Index register set to select Endpoints 1 – 5 only). There is an RxCSR register for each configured Rx endpoint (not including Endpoint 0).			0x0000
		15	ClrDataTog	The CPU writes a 1 to this bit to reset the endpoint data toggle to 0.	Set	RClr	
		14	RxStall	When a STALL handshake is received, this bit is set and an interrupt is generated. The CPU should clear this bit.	RClr	Set	
		13	ReqPkt	The CPU writes a 1 to this bit to request an IN transaction. It is cleared when RxPktRdy is set.	R/W	R/W	
		12	FlushFIFO	The CPU writes a 1 to this bit to flush the next packet to be read from the endpoint Rx FIFO. The FIFO pointer is reset and the RxPktRdy bit (below) is cleared. FlushFIFO should be used only when RxPktRdy is set. At other times, it may cause data to be corrupted. If the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.	Set	RO	

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		11	DataError	ISO mode: this bit is set when RxPktRdy is set if the data packet has a CRC error or bit-stuff error and cleared when RxPktRdy is cleared.	R(/Clr)	Set	
			NAK Timeout	Bulk mode: This bit will be set when the Rx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the RxInterval register. The CPU should clear this bit to allow the endpoint to continue.			
		10	Error	The USB sets this bit when three attempts have been made to receive a packet and no data packet has been received. The CPU should clear this bit. An interrupt is generated when the bit is set. This bit is valid only when the Tx endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero.	RClr	Set	
		9	FIFOFull	This bit is set when no more packets can be loaded into the Rx FIFO.	RO	Set	
		8	RxPktRdy	This bit is set when a data packet has been received. The CPU should clear this bit when the packet has been unloaded from the Rx FIFO. An interrupt is generated when this bit is set.	RClr	Set	
		7	AutoClear	If the CPU sets this bit, then the RxPktRdy bit will be automatically cleared when a packet of RxMaxP bytes has been unloaded from the Rx FIFO. When packets of less than the maximum packet size are unloaded, RxPktRdy will have to be cleared manually. This bit should not be set for high-bandwidth Isochronous endpoints.	R/W	RO	
		6	AutoReq	If the CPU sets this bit, the ReqPkt bit will be automatically set when the RxPktRdy bit is cleared.	R/W	RO	
		5	Unused. Returns 0 when read.		RO	RO	
		4	PID Error	ISO Transactions Only: The core sets this bit to indicate a PID error in the received packet. Bulk / Interrupt Transactions: The setting of this bit is ignored.	RO	R/W	
		3	Unused. Returns 0 when read.		RO	RO	
		2	Data Toggle Write Enable	The CPU writes a 1 to this bit to enable the current state of the Rx Endpoint data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.	RO	RO	
		1	Data Toggle	When read, this bit indicates the current state of the Rx Endpoint data toggle. If bit 2 is high, this bit may be written with the required setting of the data toggle. If bit 2 is low, any value written to this bit is ignored.	RO	RO	

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
		0	IncompRx	This bit will be set in a high-bandwidth Isochronous transfer if the packet received is incomplete. It will be cleared when RxPktRdy is cleared. If USB protocols are followed correctly, this bit should never be set. This bit becoming set indicates a failure of the associated Peripheral device to behave correctly. (In anything other than a high-bandwidth Isochronous transfer, this bit will always return 0.)	RO	Set	
0x18-0x19	Count0			Number of received bytes in Endpoint 0 FIFO. (Index register set to select Endpoint 0). The value returned changes as the contents of the FIFO change and is valid only while RxPktRdy (CSR0[0]) is set.			0x00
		14:8	Count0	Endpoint 0 Rx Count.	RO	WO	
		15 7:0	Unused, always returns 0.		RO	RO	
0x18-0x19	RxCount			Number of bytes in host Rx endpoint FIFO. (Index register set to select Endpoints 1 – 5). The value returned changes as the FIFO is unloaded and is valid only while RxPktRdy (RxCSR[0]) is set.			0x0000
		4:0 15:8	RxCount (4:0 are MSB, 15:8 are LSB)	Endpoint Rx Count.	RO	WO	
		7:3	Unused, always returns 0.		RO	RO	
0x1A	Type0			Defines the speed of Endpoint 0. (Index register set to select Endpoint 0).			
		7:6	Speed	Operating speed of the target device: 00: Unused (If selected, the target will be assumed to be using the same connection speed as the core.) 01: High speed 10: Full speed 11: Reserved	R/W	RO	0x00
		5:0	Not implemented.			

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x1A	TxType			Sets the transaction protocol, speed, and peripheral endpoint number for the host Tx endpoint. (Index register set to select Endpoints 1 – 5). There is a TxType register for each configured Tx endpoint (except Endpoint 0, which has its own Type0 register at 0x1A).	R/W	RO	0x00
		7:6	Speed	Operating speed of the target device: 00: Unused (If selected, the target will be assumed to be using the same connection speed as the core.) 01: High speed 10: Full speed 11: Reserved			
		5:4	Protocol	:The CPU should set this to select the required protocol for the Tx endpoint: 00: Control 01: Isochronous 10: Bulk 11: Interrupt			
		3:0	Target Endpoint Number	The CPU should set this value to the endpoint number contained in the Tx endpoint descriptor returned to the USB controller during device enumeration.			
0x1B	NAK Limit0			Sets the NAK response timeout on Endpoint 0. (Index register set to select Endpoint 0). NAKLimit0 sets the number of frames / microframes (High-speed transfers) after which Endpoint 0 should timeout on receiving a stream of NAK responses. (Equivalent settings for other endpoints can be made through their TxInterval and RxInterval registers). The number of frames / microframes selected is $2^{(m-1)}$ (where m is the value set in the register, valid values 2 – 16). If the host receives NAK responses from the target for more frames than the number represented by the Limit set in this register, the endpoint will be halted. A value of 0 or 1 disables the NAK timeout function.	R/W	RO	0x00
		4:0	NAK Limit	Endpoint 0 NAK Limit (m).			
0x1B	Tx Interval			For Interrupt and Isochronous transfers, TxInterval defines the polling interval for the currently-selected Tx endpoint. For Bulk endpoints, this register sets the number of frames / microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is a TxInterval register for each configured Tx endpoint (except Endpoint 0).	R/W	RO	0x00
		7:0		The value that is set defines a number of frames / microframes (High-speed transfers), as follows:			

Transfer Type	Speed	Valid Values (m)	Interpretation
Interrupt	Full	1 - 255	Polling interval is m frames.
	High	1 - 16	Polling interval is $2^{(m-1)}$ microframes.
Isochronous	Full or High	1 - 16	Polling interval is $2^{(m-1)}$ frames/microframes.
Bulk	Full or High	2 - 16	NAK Limit is $2^{(m-1)}$ frames / microframes. A value of 0 or 1 disables the NAK timeout function.

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x1C	RxType			Sets the transaction protocol, speed, and peripheral endpoint number for the host Rx endpoint. (Index register set to select Endpoints 1 – 5). There is an RxType register for each configured Rx endpoint (except Endpoint 0, which has its own Type0 register at 0x1A).	R/W	RO	0x00
		7:6	Speed	Operating speed of the target device: 00: Unused (If selected, the target will be assumed to be using the same connection speed as the core.) 01: High speed 10: Full speed 11: Reserved			
		5:4	Protocol	The CPU should set this to select the required protocol for the Tx endpoint: 00: Control 01: Isochronous 10: Bulk 11: Interrupt			
		3:0	Target Endpoint Number	The CPU should set this value to the endpoint number contained in the Rx endpoint descriptor returned to the USB controller during device enumeration.			
0x1D	Rx Interval			For Interrupt and Isochronous transfers, RxInterval defines the polling interval for the currently-selected Rx endpoint. For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is a RxInterval register for each configured Rx endpoint (except Endpoint 0).	R/W	RO	0x00
		7:0		The value that is set defines a number of frames / microframes (High-speed transfers), as follows:			
0x1E	Unused, always returns 0.						

The value that is set defines a number of frames / microframes (High-speed transfers), as follows:

Transfer Type	Speed	Valid Values (m)	Interpretation
Interrupt	Full	1 - 255	Polling interval is m frames.
	High	1 - 16	Polling interval is $2^{(m-1)}$ microframes.
Isochronous	Full or High	1 - 16	Polling interval is $2^{(m-1)}$ frames/microframes.
Bulk	Full or High	2 - 16	NAK Limit is $2^{(m-1)}$ frames / microframes. A value of 0 or 1 disables the NAK timeout function.

Table 7-37 USB Port Blocking Region: Indexed registers – Host mode (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x1F	Config Data			Returns details of core configuration. (Index register set to select Endpoint 0.)	RO	RO	Configuration Dependent
		7	MPRxE	Set to 1 when automatic amalgamation of bulk packets is selected.			
		6	MPTxE	Set to 1 when automatic splitting of bulk packets is selected.			
		5	BigEndian	Set to 1 when Big Endian ordering is selected.			
		4	HBRxE	Set to 1 when High-bandwidth Rx ISO Endpoint Support is selected.			
		3	HBTxE	Set to 1 when High-bandwidth Tx ISO Endpoint Support is selected.			
		2	DynFIFO Sizing	Set to 1 when Dynamic FIFO Sizing option is selected.			
		1	SoftConE	Set to 1 when Soft Connect/Disconnect option is selected.			
		0	UTMI DataWidth	Indicates selected UTMI+ data width: 1: 16 bits 0: 8 bits			
0x1F	FIFO Size			Returns the sizes of the FIFOs associated with the selected additional Tx and Rx endpoints. Values 3 – 13 correspond to a FIFO size of 2^n bytes (8 – 8192 bytes). If an endpoint has not been configured, a value of 0 will be displayed. Where the Tx and Rx endpoints share the same FIFO, the Rx FIFO size will be encoded as 0xF. Note: This register has this interpretation only when the Index register is set to select one of Endpoints 1 – 5 and Dynamic FIFO Sizing is not selected. It has a special interpretation when the Index register is set to select Endpoint 0 (see ConffigData above). The result returned is not valid where Dynamic FIFO Sizing is used.	RO	RO	Configuration Dependent
		7:4	Rx FIFO Size	Size of the Rx FIFO for the selected Rx endpoint.			
		3:0	Tx FIFO Size	Size of the Tx FIFO for the selected Tx endpoint.			

Table 7-38 USB Port Blocking Region Definitions: FIFOs

Offset	Register Name	Description
0x20-0x37	FIFOs	This address range provides 16 addresses for CPU access to the FIFOs for each endpoint. Writing to these addresses loads data into the TxFIFO for the corresponding endpoint. Reading from these addresses unloads data from the RxFIFO for the corresponding endpoint. The FIFOs are located on 32-bit double-word boundaries (Endpoint 0 at 0x20, Endpoint 1 at 0x24 ... Endpoint 5 at 0x34). **Notes:** 1. Transfers to and from FIFOs may be 8-bit, 16-bit, or 32-bit as required, and any combination of access is allowed, provided that the data accessed is contiguous. However, all the transfers associated with one packet must be of the same width so that the data is consistently byte-, word-, or double-word-aligned. The last transfer may, however, contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer. 2. Depending on the size of the FIFO and the expected maximum packet size, the FIFOs support either single-packet or double-packet buffering. However, burst writing of multiple packets is not supported, since flags need to be set after each packet is written. 3. Following a STALL response or a Tx Strike Out error on Endpoint 1 – 5, the associated FIFO is completely flushed.

Table 7-39 USB Port Blocking Region: Device Control, Dynamic FIFO, Version & Vendor Registers

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x60	DevCtl			Device Control register.			0x80

		7	Peripheral Device	This read-only bit indicates whether the USB controller is operating as the Host device or the Peripheral device. Valid only while a session is in progress.
				Note: If the core is in Force_Host mode (i.e. a session has been started with Testmode[7] = 1), this bit will indicate the state of the HOSTDISCON input signal from the PHY.
				    1: Peripheral device
				    0: Host device
					RO	R/W	

		6	FSDev	This read-only bit is set when a Full-speed or High-speed device has been detected being connected to the port. (High-speed devices are distinguished from full-speed by checking for high-speed chirps when the device is reset.) Valid only in Host mode.	RO	R/W	
		5		Unused, always returns 0.	RO	RO	

		4:3	VBus[1:0]	These read-only bits encode the current VBus level as follows:
				    00: Below SessionEnd
				    01: Above SessionEnd, below AValid
				    10: Above AValid, below VBusValid
				    11: Above VBusValid
					RO	R/W	

		2	Host Mode	This read-only bit is set when the USB controller is acting as a Host.	RO	R/W	
		1		Unused, always returns 0.	RO	RO	
		0	Session	When operating as a Host device, this bit is set or cleared by the CPU to start or end a session. When operating as a Peripheral device, this bit is set / cleared by the USB controller when a session starts / ends.	R/W	R/W	
0x61	Unused						
0x62	TxFIFOsz				R/W	RO	0x00
0x63	RxFIFOsz				R/W	RO	0x00
0x64-0x65	TxFIFO add			Used only if Dynamic FIFO sizing option is selected. Otherwise returns 0.	R/W	RO	0x0000
0x66-0x67	RxFIFO add				R/W	RO	0x0000

Table 7-39 USB Port Blocking Region: Device Control, Dynamic FIFO, Version & Vendor Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x68-0x6B	VControl			VControl is a UTMI+ PHY Vendor register that may optionally be included in the core when the core is configured. Its size is also configurable and may be up to 32 bits. The structure of the register is up to the system designer, although users should note that the UTMI+ specification defines a 4-bit VControl register. **Notes:** 1. If a register of more than 8 bits is specified, any write must be made to the entire register, using either a 16-bit or a 32-bit write as appropriate. Writes to individual bytes are not supported. 2. The latency for the write (as measured between the positive edge of CLK at the end of the AHB write cycle and the positive edge of XCLK when the UTMI+ PHY VControl register is loaded) will be between Hc + 3Xc and Hc + 4Xc, where Hc is a cycle of CLK and Xc is a cycle of XCLK. The minimum period between successive writes to the core's VControl, register must therefore be Hc + 4Xc to ensure that the value is not corrupted while it is being synchronized to the XCLK domain.	WO	WO	Configuration Dependent
0x68-0x6B	VStatus			VStatus is a UTMI+ PHY Vendor register that may optionally be included in the core when the core is configured. Its size is also configurable and may be up to 32 bits. The structure of the register is up to the system designer, although users should note that the UTMI+ specification defines an 8-bit VStatus register. **Notes:** 1. The VSTATUS input bus is sampled once every 6 XCLK cycles. 2. The latency between the VSTATUS input bus from the PHY changing and the new value being read from the VStatus register (measured to the positive edge of CLK at the end of the AHB read cycle) will be between 2Hc + Xc and 3Hc + 6Xc, where Hc is a cycle of CLK and Xc is a cycle of XCLK.	RO	RO	Configuration Dependent
0x6C-0x6D	HWVers			Hardware Version Number Register	RO	RO	Version Dependent
		7	RC	Set to 1 if RTL is used from a Release Candidate rather than from a full release of the core.			
		6:2	xx	Major Version Number. Range 0 – 31 (0x00 – 0x1F).			
		1:0 15:8	yyy 1:0 are MSB, 15:8 are LSB)	Minor Version Number. Range 0 – 999 (0x000 – 0x3E7).			
0x6E-0x6F	Unused						

Table 7-40 USB Port Blocking Region: Target Address Registers

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x80+ 8*n	TxFunc Addr			TxFuncAddr and RxFuncAddr record the address of the target function that is to be accessed through the associated endpoint (EPn). TxFuncAddr must be defined for each Tx endpoint that is used; RxFuncAddr must be defined for each Rx endpoint that is used. Both TxFuncAddr and RxFuncAddr must be defined for Endpoint 0. These registers (together with the Tx/RxHubAddr and Tx/RxHubPort registers) allow the allocation of multiple devices to the USB controller's endpoints.			0x00
		6:0	TxFuncAddr	Address of Target Function			
0x81+ 8*n	Unused, always returns 0.						
0x82+ 8*n	TxHub Addr			TxHubAddr and RxHubAddr need to be written only when a Full- or Low-speed device is connected to Tx/Rx Endpoint EPn via a high-speed USB hub which carries out the necessary transaction translation to convert between High-speed transmission and Full- / Low-speed transmission. If Endpoint 0 is connected to a hub, then both TxHubAddr and RxHubAddr must be defined for Endpoint 0.			0x00
		7	Multiple Translators	This bit should record whether the hub has multiple transaction translators: 1: Multiple transaction translators 0: Single transaction translator			
		6:0	TxHub Addr	The address of this USB hub.			
0x83+ 8*n	TxHub Port			TxHubPort and RxHubPort need to be written where a Full- or Low-speed device is connected to Tx Endpoint EPn via a high-speed USB hub which carries out the necessary transaction translation. If Endpoint 0 is connected to a hub, then both TxHubPort and RxHubPort need to be defined for Endpoint 0.			0x00
		6:0	TxHub Port	The port of the USB hub through which the target associated with this endpoint is accessed.			
0x84+ 8*n	RxFunc Addr			TxFuncAddr and RxFuncAddr records the address of the target function that is to be accessed through the associated endpoint (EPn). TxFuncAddr must be defined for each Tx endpoint that is used; RxFuncAddr must be defined for each Rx endpoint that is used. Both TxFuncAddr and RxFuncAddr must be defined for Endpoint 0. These registers (together with the Tx/RxHubAddr and Tx/RxHubPort registers) allow the allocation of multiple devices to the USB controller's endpoints.			0x00
		6:0	RxFunc Addr	Address of Target Function			
0x85+ 8*n	Unused, always returns 0.						

Table 7-40 USB Port Blocking Region: Target Address Registers (continued)

Offset	Register Name	Bits	Field Name	Description	CPU Access	USB Access	Reset Value
0x86+ 8*n	RxHub Addr			TxHubAddr and RxHubAddr need to be written only when a Full- or Low-speed device is connected to Rx Endpoint EPn via a high-speed USB hub which carries out the necessary transaction translation to convert between High-speed transmission and Full- / Low-speed transmission. If Endpoint 0 is connected to a hub, then both TxHubAddr and RxHubAddr must be defined for Endpoint 0.			0x00
		7	Multiple Translators	This bit should record whether the hub has multiple transaction translators: 1: Multiple transaction translators 0: Single transaction translator			
		6:0	TxHub Addr	The address of this USB hub.			
0x87+ 8*n	RxHub Port			TxHubPort and RxHubPort need to be written where a Full- or Low-speed device is connected to Tx/Rx Endpoint EPn via a high-speed USB hub which carries out the necessary transaction translation. If Endpoint 0 is connected to a hub, then both TxHubPort and RxHubPort need to be defined for Endpoint 0.			0x00
		6:0	RxHub Port	The port of the USB hub through which the target associated with this endpoint is accessed.			

8.0   Electrical Specifications

8.1   Absolute Maximum Ratings

Absolute Maximum Ratings, beyond which permanent damage may occur. Correct operation not guaranteed outside of DC Specifications listed below.

Parameter	Min	Typ	Max	Units	Conditions
Ambient temperature under bias	-40		125	ºC	
Storage temperature	-65		150	ºC	
Voltage on VDD_D, VDD_PLLDDR, VDD_PLLCG, and VDDU_P with respect to VSS	-0.3		1.5	V	
Voltage on VDD_IO, VDDG, VDDH, VDDG_RC, VDDH_RC, VDDF_PD, VDDU, and VDDF with respect to VSS	-0.3		4.0	V	
Input voltage on Port A-E, I pins, RSTN, TEST0, TEST1, TEST2, TSSN, TSCK, and TSI pins	-0.3		5.7	V	
Input voltage on USB2_N, USB2_P, and USB2_VBUS pins	-0.3		5.25	V	
Input voltage on Port F	-0.3		VDDF + 0.3	V	
Input voltage on Ports G and H, and on DDR_CAL, DDR_CLK,DDR_CLKN, DDR_CLKFB, DDR_CLKFBN, and DDR_ODT pins	-0.3		VDDG/H + 0.3	V	
Input voltage on DDRH_VREF, DDRG_VREF	-0.3		VDDG/H	V	These pins should normally be at half the VDDG/H voltage.
Input voltage on PF_VREF	-0.3		VDDF	V	This pin should normally be at half the VDDF voltage.
Output voltage on Ports A-E, I pins, TSO pin.	-0.3		4.5	V	
Output voltage on all other output pins	-0.3		Supply for that pin + 0.3	V	

8.2 DC Specifications

Operating Temperature as shown in Section 10.0, except where otherwise noted.

Symbol	Parameter	Min	Typ	Max	Units	Conditions
VDD_D = VDD_PLLDDR = VDD_PLLCG = VDDU_P	Supply Voltages (Digital Core, PLLs, USB PHY PLL) See Note 2.	1.16		1.28	V	IP5160U: 0 to 70ºC, 270 MHz max. IP5160U-I: -40 to 85ºC with a heat sink, 270 MHz max.
		1.30		1.40		IP5170U: 0 to 70ºC, 350 MHz max with a heat sink.
VDDG (DDR2 mode) = VDDH (DDR2 mode)	Supply Voltage for DDR (ports G and H), when in DDR2 mode	1.7	1.8	1.9	V	
VDDF (RGMII mode) = VDDG (DDR1 mode) = VDDG_RC = VDDH (DDR1 mode) = VDDH_RC	Supply Voltages for RGMII, DDR1, DDR receivers	2.3	2.5	2.7	V	
VDD_IO = VDDF (GPIO, RMII, or MII mode) = VDDF_PD = VDDU	Supply Voltages for GPIOs and for USB port, and for VDDF supply when using F port in GPIO, RMII, or MII mode.	3.0	3.3	3.6	V	Supplies should be powered up from highest voltage to lowest voltage, and powered down from lowest to highest (because there is an internal diode from VDD_D to VDD_IO).
Idd12	Supply Current, Active: All 1.2V supplies		750	1000	mA	Vdd = 1.23V for Typ, 1.28V for Max, Core = 270 MHz, Typical Router Gateway
Idd25	Supply Current, Active: All 2.5V supplies		246	304	mA	Vdd = 2.5V for Typ, 2.7V for Max Typical Router Gateway
Idd33	Supply Current, Active: All 3.3V supplies		57	124	mA	Vdd = 3.3V for Typ, 3.6V for Max Typical Router Gateway
DDRG_VREF	Reference Input voltage on DDRG_VREF	0.49x VDDG	0.50x VDDG	0.51x VDDG	V	DDR1 and DDR2. Peak to peak AC noise on DDRx_VREF may not exceed ±2% of DDRx_VREF (DC).
DDRH_VREF	Reference Input voltage on DDRH_VREF	0.49x VDDH	0.50x VDDH	0.51x VDDH	V	
PF_VREF	Reference Input Voltage on PF_VREF	0.49x VDDF	0.50x VDDF	0.51x VDDF	V	Peak to peak AC noise on PF_VREF may not exceed ±2% of PF_VREF (DC).

Symbol	Parameter	Min	Typ	Max	Units	Conditions
Vih	Input high voltage: Port A-E, I pins, RSTN, TSSN, TSCK, and TSI pins	2.0		5.5	V	VDD_IO = 3.0 - 3.6V
	DC Input high voltage: Port F in GPIO mode	2.0		VDDF +0.3	V	VDDF = 3.0 - 3.6V
	AC Input high voltage: Port F in RGMII mode	PF_ VREF + 0.2			V	VDDF = 2.3 - 2.7V
	DC Input high voltage: Port F in RGMII mode	1.7		VDDF +0.3	V	VDDF = 2.3 - 2.7V
	DC Input high voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR1 mode	DDRG /H_ VREF +0.15		VDDG /H + 0.3	V	VDDG = VDDH = 2.3 - 2.7V
	AC Input high voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR1 mode	DDRG /H_ VREF + 0.31		-	V	VDDG = VDDH = 2.3 - 2.7V
	DC Input high voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR2 mode	DDRG /H_ VREF + 0.125		VDDG /H + 0.3	V	VDDG = VDDH = 1.7 - 1.9V
	AC Input high voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR2 mode	DDRG /H_ VREF + 0.25			V	VDDG = VDDH = 1.7 - 1.9V
	Input high voltage: USB2_VBUS pin	4.4		5.25	V	This pin must be above the minimum Vih for normal operation (host mode).

Symbol	Parameter	Min	Typ	Max	Units	Conditions
Vil	Input low voltage: Port A-E, I pins, RSTN, TSSN, TSCK, TSI, and TEST0, TEST1, TEST2 pins, and Port F in GPIO with VDDF = 3.0-3.6V	-0.3		0.8	V	
	DC Input low voltage: Port F, in RGMII mode	-0.3		0.7	V	VDDF = 2.3 - 2.7V
	AC Input low voltage: Port F in RGMII mode	-		PF_VREF - 0.2	V	VDDF = 2.3 - 2.7V
	DC Input low voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR1 mode	-0.3		VDDG/H_VREF - 0.15	V	VDDG = VDDH = 2.3-2.7V
	AC Input low voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR1 mode	-		VDDG/H_VREF - 0.31	V	VDDG = VDDH = 2.3-2.7V
	DC Input low voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR2 mode	-0.3		VDDG/H_VREF - 0.125	V	VDDG = VDDH = 1.7-1.9V
	AC Input low voltage: Ports G and H, and DDR_CLKFB, DDR_CLKFBN in DDR2 mode	-		VDDG/H_VREF - 0.25	V	VDDG = VDDH = 1.7-1.9V
	Input low voltage: USB2_ID pin			0.4	V	This pin must be below the maximum Vil for normal operation (host mode).
Ioh	Output high current: Ports A, B, C, D, E, I, except PB18. Also includes TSO pin.	8	22		mA	Voh = 2.4V, VDD_IO = VDDF = 3.0 - 3.6V.
	Output high current: PB18 pin	12	29		mA	
	Output high current: Port F in GPIO mode	8			mA	
	Output high current: Port F, RGMII mode	8			mA	Voh = 1.8V, VDDF = 2.3 - 2.7V
	Output high current: Ports G, H, and DDR_CLK, DDR_CLKN pins (DDR1)	8			mA	Voh = 1.8V, VDDG/VDDH = 2.3 - 2.7V
	Output high current: Ports G, H, and DDR_CLK, DDR_CLKN pins (DDR2)	13			mA	Voh = 1.0V, VDDG/VDDH = 1.7 - 1.9V

Symbol	Parameter	Min	Typ	Max	Units	Conditions
Iol	Output low current: Ports A, B, C, D, E, I, except PB18. Also includes TSO pin.	8	15		mA	Vol = 0.4V, VDD_IO = VDDF = 3.0-3.6V
	Output low current: PB18 pin	12	23		mA	
	Output low current: Port F in GPIO mode	8			mA	
	Output low current: Port F, RGMII mode	8			mA	Vol = 0.4V, VDDF = 2.3 - 2.7V
	Output low current: Ports G, H, and DDR_CLK, DDR_CLKN pins (DDR1)	8			mA	Vol = 0.35V, VDDG/VDDH = 2.3-2.7V
	Output low current: Ports G, H, and DDR_CLK, DDR_CLKN pins (DDR2)	13			mA	Vol = 0.62V, VDDG/VDDH = 1.7-1.9V
Ileak	Input leakage current: Port A-E,I pins in input mode, and tri-stated TSO pin	-10		10	µA	Pins held at 0V or 5.5V. TSO measured while TSSN= high
	Port F, PF_VREF pins	-10		10	µA	Pins held at 0V or VDDF
	Ports G, H, DDR_CLK, CLKN, CLKFB, CLKFBN, ODT, DDRG_VREF, DDRH_VREF, pins	-10		10	µA	Pins held at 0V or VDDG/H
	Input leakage current: RSTN, TSSN, TSI pins	-110		-25	µA	Inputs held at 0V. See Note 3.
		-10		10	µA	Inputs held at 5.5V. See Note 3.
	Input leakage current: TSCK, TEST0, TEST1, TEST2 pins	-10		10	µA	Inputs held at 0V. See Note 4.
		10		140	µA	Inputs held at 5.5V. See Note 4.
	USB2_N, USB2_P pins	-10		500	µA	Pin held at 0V or VDDU.
	USB2_VBUS pin	-10		500	µA	Pin held at 0V or 3.6V.
	USB2_ID pin	-10		10	µA	Pin held at 0V.
	DDR_CAL pin	-10		10	µA	Input held at 0V or VDDG / VDDH

Note 1: Data in the Typical column is at 1.23/1.8/2.5/3.3V, 25ºC, unless otherwise stated.

Note 2: The designer must ensure that the supply voltage on the balls of the device is always above the minimum operating voltage. The typical voltage regulator output voltage should be above the midpoint between the above min and max voltages, to allow for voltage drop on the board's traces between the regulator output and the balls of the device, even when the device is running at full speed.

Note 3: These pins have an internal active pullup to a threshold drop below VDD_IO. The pullup = 39K ohms min, 55K ohms typ, 85K ohms max.

Note 4: These pins have an internal active pulldown to a threshold drop above VSS. The pulldown = 45K ohms min, 93K ohms typ, 198K ohms max.

8.3   AC Specifications

Operating Temperature as shown in Section 10.0, except where otherwise noted.

Note: In this table, # indicates an active low signal.

Function	Symbol	Parameter	Min	Typ	Max	Units	Conditions
System	fCORE	Core Frequency (execution of instructions from on-chip memory)			270	MHz	IP5160
					350	MHz	IP5170
	fPLL	Frequency into Core PLL	2		12	MHz	
	fXTAL	Crystal frequency on OSC_IN / OSC_OUT	12	12	12	MHz	Crystal with ±100ppm rating
	tVdd	VDD_D rise time to ensure Power-On reset			10	ms	Must be > 1V within 10ms of power on when relying on Power-On reset.
		PLL (core and I/O) stabilization time			1000	cycles	Cycles into PLL (after reference clock divider)
	PLL specifications: See Figure 3-4.						
	tRST	External Reset Pulse	1			us	On RSTN pin
	tRHO	Reset hold off time	50		150	ms	Time after deassertion of reset until internal reset released. Allows for full crystal startup.
		Crystal startup time			50	ms	
	fDDR	DDR1 and DDR2 frequency	120		200	MHz	
MII on E or I port	fMII	MII frequency	2.5		25	MHz	
	tMRXS	RXD, RX_DV, RX_ER setup to RX_CLK rising	10			ns	
	tMRXH	RXD, RX_DV, RX_ER hold from RX_CLK rising	10			ns	
	tMTXD	TX_ER, TX_EN, TXD from TX_CLK rising	0		25	ns	
MII on F port	fMII50	MII frequency for F port	2.5		50	MHz	
	tMRXS50	RXD, RX_DV, RX_ER setup to RX_CLK rising	3			ns	
	tMRXH50	RXD, RX_DV, RX_ER hold from RX_CLK rising	2			ns	
	tMTXD50	TX_ER, TX_EN, TXD from TX_CLK rising	0		9	ns	
RMII, Measure all ACs at 50% levels.	tRMRXS	RXD, CRS_DV, RX_ER setup to REF_CLK rising	4			ns	
	tRMRXH	RXD, CRS_DV, RX_ER hold from REF_CLK rising	2			ns	
	tRMTXD	TX_EN, TXD from REF_CLK rising	0		12	ns	

| Function	Symbol	Parameter	Min	Typ	Max	Units	Conditions
RGMII	tRGM-RXS	RXD, RX_CTL setup to RX_CLK rising and falling	1			ns	
	tRGM-RXH	RXD, RX_CTL hold from RX_CLK rising and falling	1			ns	
	tRGM-TXD	TXD, TX_CTL to TX_CLK_O output skew	-0.5		0.5	ns	
Serdes See Note A1.	fIO	I/O PLL output clock frequency (into Serdes divider)			250	MHz	
	fSDCK	Serdes internal (after Serdes divider) clock rate			100	MHz	Core clock must be faster.
	fSDPCK	Serdes pin clock rate	0		25	MHz	This must be at least 4 times slower than tSDCK.
	tEDS	Input data setup to active edge of clock	8			ns	
	tEDH	Input data hold from active edge of clock	8			ns	
	tEDD	Output data from active edge of clock			20	ns	
PCI	tPCIS	AD, DEVSEL#, PERR#, STOP#, SERR#, TRDY#, FRAME#, IRDY#, PAR#, CBE setup to CLK rising	7			ns	
		REQ0 and REQ1 setup to CLK rising	12			ns	
	tPCIH	AD, DEVSEL#, PERR#, STOP#, SERR#, TRDY#, FRAME#, IRDY#, PAR#, CBE, REQ hold from CLK rising	0			ns	
	tPCID	AD, PERR#, STOP#, SERR#, TRDY#, FRAME#, IRDY#, PAR#, CBE signals valid from CLK rising	2		11	ns	
		GNT0 and GNT1 from CLK rising	2		12	ns	

Note A1: The relevant signals and active clock edge for the Serdes depend on the current configuration. These timings are relevant for modes that use an externally supplied clock.

9.0 Package Dimensions

IP51xxU - 256 balls, 17x17x1.3 mm package, 1mm ball pitch.
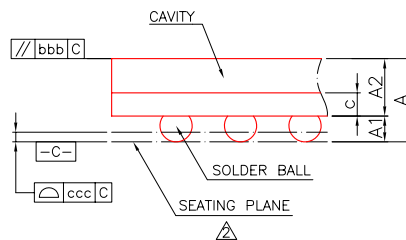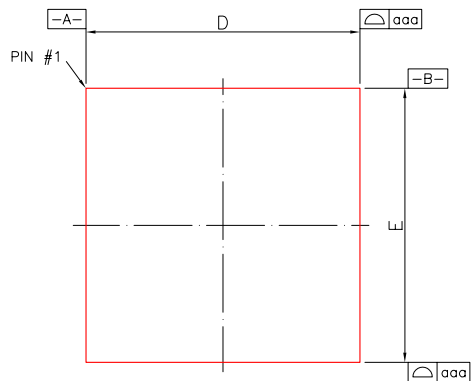
DETAIL : "A"
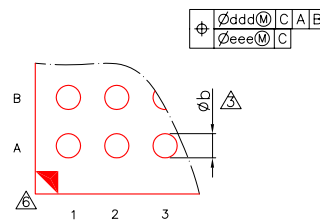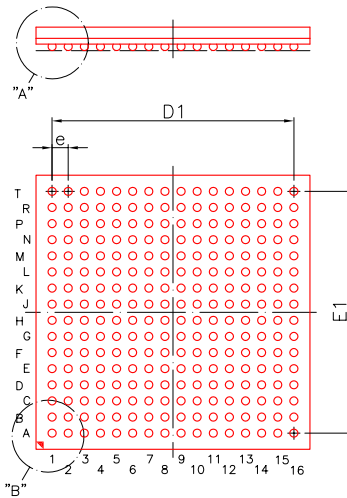
DETAIL : "B"

Symbol	Dimension in mm			Dimension in inch		
	MIN	NOM	MAX	MIN	NOM	MAX
A	---	1.29	---	---	0.051	---
A1	0.35	0.40	0.45	0.014	0.016	0.018
A2	0.84	0.89	0.94	0.033	0.035	0.037
c	0.32	0.36	0.40	0.012	0.014	0.016
D	16.90	17.00	17.10	0.665	0.669	0.673
E	16.90	17.00	17.10	0.665	0.669	0.673
D1	---	15.00	---	---	0.591	---
E1	---	15.00	---	---	0.591	---
e	---	1.00	---	---	0.039	---
b	0.45	0.50	0.55	0.018	0.020	0.022
aaa	---	0.10	---	---	0.004	---
bbb	---	0.20	---	---	0.008	---
ccc	---	0.12	---	---	0.005	---
ddd	---	0.15	---	---	0.006	---
eee	---	0.08	---	---	0.003	---
MD/ME	16/16			16/16		

NOTE :
1. CONTROLLING DIMENSION : MILLIMETER.
2. PRIMARY DATUM C AND SEATING PLANE ARE DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BALLS.
3. DIMENSION b IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER, PARALLEL TO PRIMARY DATUM C.
4. THERE SHALL BE A MINIMUM CLEARANCE OF 0.25mm BETWEEN THE EDGE OF THE SOLDER BALL AND THE BODY EDGE.
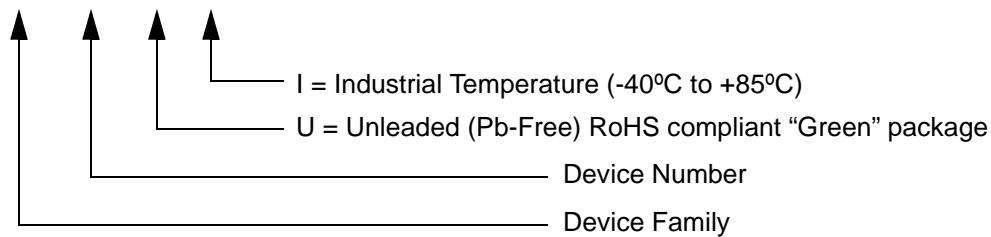5. REFERENCE DOCUMENT : JEDEC MO-216
6. THE PATTERN OF PIN 1 FIDUCIAL IS FOR REFERENCE ONLY.
7. SPECIAL CHARACTERISTICS C CLASS: bbb, ccc

10.0 Part Numbering

Part Number	Pins	I/O	Package	On-Chip Program / Data RAM	Max Core Frequency	Temperature	Heat Sink Required
IP5160U	256	150	BGA, Pb-Free, RoHS compliant	192 KB	270 MHz	0ºC to +70ºC	No
IP5160U-I	256	150	BGA, Pb-Free, RoHS compliant	192 KB	270 MHz	-40ºC to +85ºC	Yes
IP5170U	256	150	BGA, Pb-Free, RoHS compliant	192 KB	350 MHz	0ºC to +70ºC	Yes

IP 5160 U - I

I = Industrial Temperature (-40ºC to +85ºC)

U = Unleaded (Pb-Free) RoHS compliant "Green" package

Device Number

Device Family

11.0 Revision History of This Document

Date	Location	Description of Change
3/28/07	Table 2-3	Added minor clarifications to terminology. Changed description for USB2_VBUS.
	Table 2-4, 6-1, 7-19	Updated port width for Port I and USB Port. Removed FIFO sizes for Port I.
	Table 2-11	Changed RAS to RAS_N.
	Section 3.0	Updated Figure 3-4 and various clock frequency references throughout Section 3.0.
	Table 3-3	Updated addressing for on-chip peripherals and on-chip SRAM.
	Section 4.2	Changed first paragraph characterization of DSP instructions.
	Section 4.2.1	Changed first sentence.
	Section 4.5	Changed lsr instruction description.
	Section 5.0	Added Programmer's Reference (includes some material moved from Section 6.0).
	Section 6.6 7.10.2.7	Removed references to Reverse MII.
	Section 6.9.1	Removed references to USB FIFO sharing.
	Table 7-9	Changed Security Control bits [2:1] definition.
	Section 7.10.2	Updated MII / RMII register definitions.
	Section 7.15.1.2	Changed bit definitions for USB Port Interrupt Set register.
	Section 7.7.1.1	Changed bit definitions for Port A Flash Interrupt Status register.
	Section 7.8.1.2	Changed TX_FIFO_WM definition.
	Section 7.11.1.9, 7.11.1.11	Changed register field names.
	Table 7-21	Removed references to MII Management (also removed all related tables).
	Table 7-23	Changed definition for bits [9:8].
	Section 7.15.1	Removed USB FIFO-related registers.
	Sections 8.1, 8.2, and 8.3.	Updated electrical specifications.
11/15/06	Section 2.2	Added note to Table 2-3 recommending a 200 ohm resistor on the USB2_VBUS pin.
	Section 2.2	Updated Table 2-3 values.
	Section 3.10	Added Figure 3-4.
	Sections 6.4, 6.5.1, 7.10.2.7	Changed text.
	Sections 7.15.1.1, 7.15.1.2	Updated bit definitions.
	Section 8.2, 8.3	Updated DC Specifications and AC Specifications.

Date	Location	Description of Change
10/01/06	Throughout	Changed IP5160 to IP51xx when referring to both IP5160 and IP5170.
	Section 1.0, 10.0	Added part numbers for new parts IP5160U-I and IP5170.
	Section 3.10, 3.9	Updated parameter values. Added paragraph above Figure 3-3.
	Section 4.3, 4.5	Expanded instruction descriptions.
	Section 6.2.3	Added paragraph at end of section.
	Section 8.2, 8.3	Updated DC Specifications and AC Specifications.
7/31/06	Throughout	Changed IP 5160 CPU frequency to 270 MIPS.
	Section 8.2, 8.3	Updated DC Specifications and AC Characteristics.
6/27/06	Section 4.0	Updated entire Instruction Set section.
	Section 3.9	Updated Crystal Oscillator resistor and capacitor values.
5/31/06	Section 7.5.5	Updated Security Module register definitions.
	Section 6.10	Updated Debug Interface Host Command definitions.
5/25/06	Section 6.0	Refined interface descriptions — removed implementation details.
	Section 7.0	Refined register maps — removed unimplemented registers and fields.
	Section 10.0	Removed non-green part numbers.
	Section 7.15	Updated USB register definitions. Changed blocking registers to big-endian.
	Last page	Updated Ubicom address and phone numbers.
4/12/06	Throughout	Corrected errors and misconceptions.
	Section 6.10	Added Debug Port discussion.
3/29/06	Section 2.1	Changed pin assignments for A14 and A15.
3/20/06	Initial Release	

Sales and Technical Support Contact Information

For the latest contact and support information on IP devices, please visit the Ubicom Web site at www.ubicom.com. The site contains technical literature, local sales contacts, tech support, and many other features.

The Products are not authorized for use in life support systems or under conditions where failure of the Product would endanger the life or safety of the user, except when prior written approval is obtained from Ubicom, Inc. Ask your sales representative for details.

510 N. Pastoria Avenue
Sunnyvale, CA 94085

Tel: 408.789.2200
Fax:408.739.2427
Email:sales@ubicom.com
Web:www.ubicom.com

Ubicom, Inc. develops and markets wireless network processor and software platforms that enable all electronic devices to be connected to each other – securely, cost-effectively and transparently. With headquarters in Sunnyvale, California, Ubicom also has offices in Southern California, as well as Netherlands, Taiwan and Hong Kong. For more information, visit www.ubicom.com.

IP5K-DDS-51xx-10