## MicroBlaze MCS Register Descriptions

**Table 13: MicroBlaze MCS Address Map**

| Address (hex) | Name | Access Type | Description |
|---|---|---|---|
| 0x0-<br>C_MEMSIZE-1 | Local Memory | RW | Local Memory for MicroBlaze software |
| C_MEMSIZE-<br>0x7FFFFFFF | Reserved | | |
| 0x80000000 | UART_RX | R | UART Receive Data Register |
| 0x80000004 | UART_TX | W | UART Transmit Data Register |
| 0x80000008 | UART_STATUS | R | UART Status Register |
| 0x8000000C | Reserved | | |
| 0x80000010 | GPO1 | W | General Purpose Output 1 Register |
| 0x80000014 | GPO2 | W | General Purpose Output 2 Register |
| 0x80000018 | GPO3 | W | General Purpose Output 3 Register |
| 0x8000001C | GPO4 | W | General Purpose Output 4 Register |
| 0x80000020 | GPI1 | R | General Purpose Input 1 Register |
| 0x80000024 | GPI2 | R | General Purpose Input 2 Register |
| 0x80000028 | GPI3 | R | General Purpose Input 3 Register |
| 0x8000002C | GPI4 | R | General Purpose Input 4 Register |
| 0x80000030 | IRQ_STATUS | R | Interrupt Status Register |
| 0x80000034 | IRQ_PENDING | R | Pending Interrupt Register |
| 0x80000038 | IRQ_ENABLE | W | Interrupt Enable Register |
| 0x8000003C | IRQ_ACK | W | Interrupt Acknowledge Register |
| 0x80000040 | PIT1_PRELOAD | W | PIT1 Preload Register |
| 0x80000044 | PIT1_COUNTER | R | PIT1 Counter Register |
| 0x80000048 | PIT1_CONTROL | W | PIT1 Control Register |
| 0x8000004C | Reserved | | |
| 0x80000050 | PIT2_PRELOAD | W | PIT2 Preload Register |
| 0x80000054 | PIT2_COUNTER | R | PIT2 Counter Register |
| 0x80000058 | PIT2_CONTROL | W | PIT2 Control Register |
| 0x8000005C | Reserved | | |
| 0x80000060 | PIT3_PRELOAD | W | PIT3 Preload Register |
| 0x80000064 | PIT3_COUNTER | R | PIT3 Counter Register |
| 0x80000068 | PIT3_CONTROL | W | PIT3 Control Register |
| 0x8000006C | Reserved | | |
| 0x80000070 | PIT4_PRELOAD | W | PIT4 Preload Register |
| 0x80000074 | PIT4_COUNTER | R | PIT4 Counter Register |
| 0x80000078 | PIT4_CONTROL | W | PIT4 Control Register |

**Table 13: MicroBlaze MCS Address Map**

| Address (hex) | Name | Access Type | Description |
|---|---|---|---|
| 0x8000007C | Reserved | | |
| 0x80000080-<br>0xBFFFFFFF | Reserved | | |
| 0xC0000000-<br>0xFFFFFFFF | IO Bus | RW | Mapped to IO Bus address output IO_Address |

## UART Receive Data Register (UART_RX)

A register contains data received by the UART. Reading of this location will result in reading the current word from the register. When a read request is issued without having received a new character, the previously read data will be read again. This register is a read-only register. Issuing a write request to the register will do nothing but generate the write acknowledgement.

The register is implemented if C_USE_UART_RX is set to 1.

*Table 14:* **UART Receive Data Register (UART_RX) (C_DATA_BITS=8)**

| Reserved | | UART_RX | |
|---|---|---|---|
| 31 | 8 | 7 | 0 |

*Table 15:* **UART Receive Data Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:C_UART_DATA_BITS | - | R | 0 | Reserved |
| [C_UART_DATA_BITS-1]:0 | UART_RX | R | 0 | UART Receive Data |

## UART Transmit Data Register (UART_TX)

A register contains data to be output by the UART. Data to be transmitted is written into this register. This is write only location. Issuing a read request to this register generates the read acknowledgement with zero data. Writing this register when the character has not been transmitted will overwrite previously written data, resulting in loss of data.

The register is implemented if C_USE_UART_TX is set to 1.

*Table 16:* **UART Transmit Data Register (UART_TX) (C_DATA_BITS=8)**

| Reserved | | UART_TX | |
|---|---|---|---|
| 31 | 8 | 7 | 0 |

*Table 17:* **UART Transmit Data Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:C_UART_DATA_BITS | - | R | 0 | Reserved |
| [C_UART_DATA_BITS-1]:0 | UART_TX | R | 0 | UART Transmit Data |

## UART Status Register (UART_Status)

The UART Status Register contains the status of the receive and transmit registers, and if there are any errors. This is read only register. If a write request is issued to status register it will do nothing but generate write acknowledgement.

The register is implemented if C_USE_UART_RX or C_USE_UART_TX is set to 1.

*Table 18:* **UART Status Register (UART_Status)**

| Reserved | | UART_Status | |
|---|---|---|---|
| 31 | 8 | 7 | 0 |

*Table 19:* **UART Status Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 7 | Parity Error | R | ’0’ | Indicates that a parity error has occurred after the last time the status register was read. If the UART is configured without any parity handling, this bit is always ‘0’. The received character is written into the receive register. This bit is cleared when the status register is read.<br>’0’ = No parity error has occurred<br>’1’ = A parity error has occurred |
| 6 | Frame Error | R | ’0’ | Indicates that a frame error has occurred after the last time the status register was read. Frame Error is defined as detection of a stop bit with the value ’0’. The receive character is ignored and not written to the receive register. This bit is cleared when the status register is read.<br>’0’ = No Frame error has occurred<br>’1’ = A frame error has occurred |
| 5 | Overrun Error | R | ’0’ | Indicates that a overrun error has occurred since the last time the status register was read. Overrun occurs when a new character has been received but the receive register has not been read. The received character is ignored and not written into the receive register. This bit is cleared when the status register is read.<br>’0’ = No interrupt has occurred<br>’1’ = Interrupt has occurred |
| 4 | - | R | ’0’ | Reserved |

*Table 19:* **UART Status Register Bit Definitions *(Cont'd)***

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 3 | Tx Used | R | '0' | Indicates if the transmit register is in use<br>'0' = Transmit register is not in use<br>'1' = Transmit register is in use |
| 2 | - | R | '0' | Reserved |
| 1 | - | R | '0' | Reserved |
| 0 | Rx Valid Data | R | '0' | Indicates if the receive register has valid data<br>'0' = Receive register is empty<br>'1' = Receive register has valid data |

## General Purpose Output x Register (GPOx) (x = 1, 2, 3 or 4)

This register holds the value that will be driven to the corresponding bits in the IO Module GPOx port output signals. All bits in the register are updated when the register is written.

This register is not implemented if the value of C_USE_GPOx is 0.

*Table 20:* **General Purpose Output x Register (GPOx)**

| Reserved | | GPOx | |
|---|---|---|---|
| 31 | C_GPOx_SIZE | C_GPOx_SIZE-1 | 0 |

*Table 21:* **General Purpose Output x Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:C_GPOx_SIZE | - | - | - | Reserved |
| [C_GPOx_SIZE-1]:0 | GPOx | W | 0 | Register holds data driven to corresponding bits in the GPO port |

## General Purpose Input x Register (GPIx) (x=1, 2, 3 or 4)

This register reads the value that is input on the corresponding IO Module GPIx port input signal bits.

This register is not implemented if the value of C_USE_GPIx is 0.

*Table 22:* **General Purpose Input x Register (GPIx)**

| Reserved | | GPIx | |
|---|---|---|---|
| 31 | C_GPIx_SIZE | C_GPIx_SIZE-1 | 0 |

*Table 23:* **General Purpose Input x Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:C_GPIx_SIZE | - | R | 0 | Reserved |
| [C_GPIx_SIZE-1]:0 | GPIx | R | 0 | Register reads value input on the IO Module GPIx port input signals |

## Interrupt Status Register (IRQ_STATUS)

The Interrupt Status Register holds information on interrupt events that have occurred. The register is read-only and the IRQ_ACK register should be used to clear individual interrupts.

*Table 24:* **Interrupt Status Register (IRQ_STATUS)**

| Reserved | | INTC_Interrupt | | Reserved | | Internal Interrupts | |
|---|---|---|---|---|---|---|---|
| 31 | C_INTC_EXT_INTR+16 | C_INTC_EXT_INTR+15 | 16 | 15 | 11 | 10 | 0 |

*Table 25:* **Interrupt Status Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:[C_INTC_EXT_INTR + 16] | - | R | 0 | Reserved |
| [C_INTC_EXT_INTR+15]:16 | INTC_Interrupt | R | 0 | IO Module external interrupt input signal INTC_Interrupt [C_INTC_EXT_INTR-1:0] mapped to corresponding bit positions in IRQ_STATUS |
| 15:11 | - | R | 0 | Reserved |
| 10 | FIT4 | R | 0 | FIT4 strobe |
| 9 | FIT3 | R | 0 | FIT3 strobe |
| 8 | FIT2 | R | 0 | FIT2 strobe |
| 7 | FIT1 | R | 0 | FIT1 strobe |
| 6 | PIT4 | R | 0 | PIT4 lapsed |
| 5 | PIT3 | R | 0 | PIT3 lapsed |
| 4 | PIT2 | R | 0 | PIT2 lapsed |
| 3 | PIT1 | R | 0 | PIT1 lapsed |
| 2 | UART_RX | R | 0 | UART Received Data |
| 1 | UART_TX | R | 0 | UART Transmitted Data |
| 0 | UART_ERR | R | 0 | UART Error |

## Interrupt Pending Register (IRQ_PENDING)

The Interrupt Pending Register holds information on enabled interrupt events that have occurred. IRQ_PENDING is the contents of IRQ_STATUS bit-wised masked with the IRQ_ENABLE register. The register is read-only and the IRQ_ACK register should be used to clear individual interrupts.

*Table 26:* **Interrupt Pending Register (IRQ_PENDING)**

| Reserved | | INTC_Interrupt | | Reserved | | Internal Interrupts | |
|---|---|---|---|---|---|---|---|
| 31 | C_INTC_EXT_INTR+16 | C_INTC_EXT_INTR+15 | 16 | 15 | 11 | 10 | 0 |

*Table 27:* **Interrupt Pending Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:[C_INTC_EXT_INTR+16] | - | R | 0 | Reserved |
| [C_INTC_EXT_INTR+15]:16 | INTC_Interrupt | R | 0 | IO Module external interrupt input signal INTC_Interrupt [C_INTC_EXT_INTR-1:0] mapped to corresponding bit positions in IRQ_STATUS |
| 15:11 | - | R | 0 | Reserved |
| 10 | FIT4 | R | 0 | FIT4 strobe |
| 9 | FIT3 | R | 0 | FIT3 strobe |
| 8 | FIT2 | R | 0 | FIT2 strobe |
| 7 | FIT1 | R | 0 | FIT1 strobe |
| 6 | PIT4 | R | 0 | PIT4 lapsed |
| 5 | PIT3 | R | 0 | PIT3 lapsed |
| 4 | PIT2 | R | 0 | PIT2 lapsed |
| 3 | PIT1 | R | 0 | PIT1 lapsed |
| 2 | UART_RX | R | 0 | UART Received Data |
| 1 | UART_TX | R | 0 | UART Transmitted Data |
| 0 | UART_ERR | R | 0 | UART Error |

## Interrupt Enable Register (IRQ_ENABLE)

The Interrupt Enable Register enables assertion of the IO Module interrupt output signal INTC_IRQ by individual interrupt sources. The contents of this register is also used to mask the value of the IRQ_STATUS register when registering enabled interrupts in the IRQ_PENDING register.

*Table 28:* **Interrupt Enable Register (IRQ_ENABLE)**

| Reserved | | INTC_Interrupt | | Reserved | | Internal Interrupts | |
|---|---|---|---|---|---|---|---|
| 31 | C_INTC_EXT_INTR+16 | C_INTC_EXT_INTR+15 | 16 | 15 | 11 | 10 | 0 |

*Table 29:* **Interrupt Enable Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:[C_INTC_EXT_INTR+16] | - | - | 0 | Reserved |
| [C_INTC_EXT_INTR+15]:16 | INTC_Interrupt | W | 0 | Enable IO Module external interrupt input signal INTC_Interrupt(16-C_INTC_EXT_INTR) |
| 15 - 11 | - | - | 0 | Reserved |
| 10 | FIT4 | W | 0 | FIT4 interrupt enabled |
| 9 | FIT3 | W | 0 | FIT3 interrupt enabled |
| 8 | FIT2 | W | 0 | FIT2 interrupt enabled |
| 7 | FIT1 | W | 0 | FIT1 interrupt enabled |
| 6 | PIT4 | W | 0 | PIT4 interrupt enabled |
| 5 | PIT3 | W | 0 | PIT3 interrupt enabled |
| 4 | PIT2 | W | 0 | PIT2 interrupt enabled |
| 3 | PIT1 | W | 0 | PIT1 interrupt enabled |
| 2 | UART_RX | W | 0 | UART Received Data interrupt enabled |
| 1 | UART_TX | W | 0 | UART Transmitted Data interrupt enabled |
| 0 | UART_ERR | W | 0 | UART Error interrupt enabled |

## Interrupt Acknowledge Register (IRQ_ACK)

This register is used as a command register for clearing individual interrupts in IRQ_STATUS and IRQ_PENDING registers. All bits written ‘1’ will clear the corresponding bits in the IRQ_STATUS and IRQ_PENDING registers. The register is write-only.

*Table 30:* **Interrupt Acknowledge Register (IRQ_ACK)**

| IRQ_ACK | |
|---|---|
| 31 | 0 |

*Table 31:* **Correctable Error First Failing Address Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:0 | IRQ_ACK | W | 0 | All bit position written with '1' will clear corresponding bits in both the IRQ_STATUS and the IRQ_PENDING registers |

## PITx Preload Register (PITx_PRELOAD) (x = 1, 2, 3 or 4)

The value written to this register determines the period between two consecutive PITx_Interrupt events. The period will be the value written to the register + 2 count events.

The register is implemented if C_USE_PITx is 1.

*Table 32:* **PITx Preload Register (PITx_PRELOAD)**

| Reserved | | PITx_PRELOAD | |
|---|---|---|---|
| 31 | C_PITx_SIZE | C_PITx_SIZE-1 | 0 |

*Table 33:* **PITx Preload Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:C_PITx_SIZE | - | - | - | Reserved |
| [C_PITx_SIZE-1]:0 | PITx_ PRELOAD | W | 0 | Register holds the timer period |

## PITx Counter Register (PITx_COUNTER) (x = 1, 2, 3 or 4)

When reading this register the obtained data will be a sample of the current counter value.

The register is implemented if C_USE_PITx is 1 and C_PITx_READABLE is 1.

*Table 34:* **PITx Counter Register (PITx_COUNTER)**

| Reserved | | PITx_PRELOAD | |
|---|---|---|---|
| 31 | C_PITx_SIZE | C_PITx_SIZE-1 | 31 |

*Table 35:* **PITx Counter Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:C_PITx_SIZE | - | - | - | Reserved |
| [C_PITx_SIZE-1]:0 | PITx_ COUNTER | R | 0 | PITx counter value at time of read |

## PITx Control Register (PITx_CONTROL) (x=1, 2, 3 or 4)

The EN bit in this register enables/disables counting. The PRELOAD bit determines if the counting is continuous with automatic reload of the PITx_PRELOAD value when lapsing (PITx_COUNTER = 0) or if the counting is stopped after counting the number of cycles defined in PITx_PRELOAD.

The register is implemented if C_USE_PITx is 1.

*Table 36:* **PITx Control Register (PITx_CONTROL)**

| Reserved | | RELOAD | EN |
|---|---|---|---|
| 31 | 2 | 1 | 0 |

*Table 37:* **PITx Control Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| 31:2 | - | - | 0 | Reserved |
| 1 | PRELOAD | W | 0 | 0 = Counter counts PITx_PRELOAD value cycles and the stops<br>1 = Counter value is automatically reloaded with the PITx_PRELOAD value when counter lapses |
| 0 | EN | W | 0 | 0 = Counting Disabled<br>1 = Counter Enabled |

# Design Implementation

## Design Tools

See the Tool Flow chapter.

## Target Technology

The target technology is an FPGA listed in the Supported Device Family field of the LogiCORE Facts table.

## Device Utilization and Performance Benchmarks

Because the MicroBlaze MCS is a module that is used together with other parts of the design in the FPGA, the utilization and timing numbers reported in this section are just estimates, and the actual utilization of FPGA resources and timing of the MicroBlaze MCS design will vary from the results reported here. All parameters not given in the table below have their default values.

*Table 38:* **Performance and Resource Utilization Benchmarks on Virtex-6 (xc6vlx240t-1-ff1156)**

| Parameter Values (other parameters at default value) | | | | | | | | | | | | | | Device Resources | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C_USE_UART_RX | C_USE_UART_TX | C_INTC_USE_EXT_INTR | C_INTC_INTR_SIZE | C_USE_FIT1 | C_FIT1_No_CLOCKS | C_USE_PIT1 | C_PIT1_SIZE | C_USE_GPI1 | C_GPI1_SIZE | C_USE_GPO1 | C_GPO1_SIZE | C_USE_IO_BUS | C_DEBUG_ENABLE | LUTs | Flip-Flops |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 716 | 299 |
| 1 | 1 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 733 | 330 |
| 1 | 1 | 1 | 5 | 1 | 65000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 740 | 342 |
| 1 | 1 | 1 | 5 | 1 | 65000 | 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 783 | 434 |
| 1 | 1 | 1 | 5 | 1 | 65000 | 1 | 32 | 1 | 32 | 0 | 0 | 0 | 0 | 804 | 466 |
| 1 | 1 | 1 | 5 | 1 | 65000 | 1 | 32 | 1 | 32 | 1 | 32 | 0 | 0 | 805 | 498 |
| 1 | 1 | 1 | 5 | 1 | 65000 | 1 | 32 | 1 | 32 | 1 | 32 | 1 | 0 | 820 | 602 |
| 1 | 1 | 1 | 5 | 1 | 65000 | 1 | 32 | 1 | 32 | 1 | 32 | 1 | 1 | 1022 | 959 |

# Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY.*