AN2890
Rev. 0, 12/2005

FPGA MDR Interface for the MRC6011

A VHDL Reference Design for the ROBIN Motherboard

By Dejan Minic

This application note describes how to implement the MRC6011 MDR antenna bus interface and the supporting ROBIN system design on the Xilinx® field-programmable gate array (FPGA) using VHDL. VHDL is an acronym that stands for *VHSIC hardware description language*. VHSIC is yet another acronym that stands for *very high speed integrated circuits*.

For the project reported in this document, the ROBIN motherboard was used for development and testing, and you should use this board if you have access to it. However, other development boards can be used.

This is one of two application notes that describe an FPGA reference design code for the ROBIN motherboard. The other application note is AN2889, *FPGA System Bus Interface for the MPC8260: A VHDL Reference Design for the ROBIN Motherboard*. This application note and the zip file of code that accompanies it are available at the website listed on the back cover of this document.

CONTENTS

1 Baseband System Example

Figure 1 illustrates the architecture of a generic ROBIN motherboard baseband system FPGA processor. The FPGA connects to external devices via the MPC8260 system bus. The data is exchanged and/or stored via the two internal memory structures. On the system bus, the MPC8260 processor can access either the control registers for FPGA/board configuration or all 128 KB of the FPGA internal storage BlockRAM memory. The control registers control the ROBIN motherboard settings and modes of operation. Additional space is allocated for custom registers integrated into a more complex system design. When the data is loaded into the FPGA, it is streamed via the multiplexed data router (MDR) bus to the MRC6011 processor.

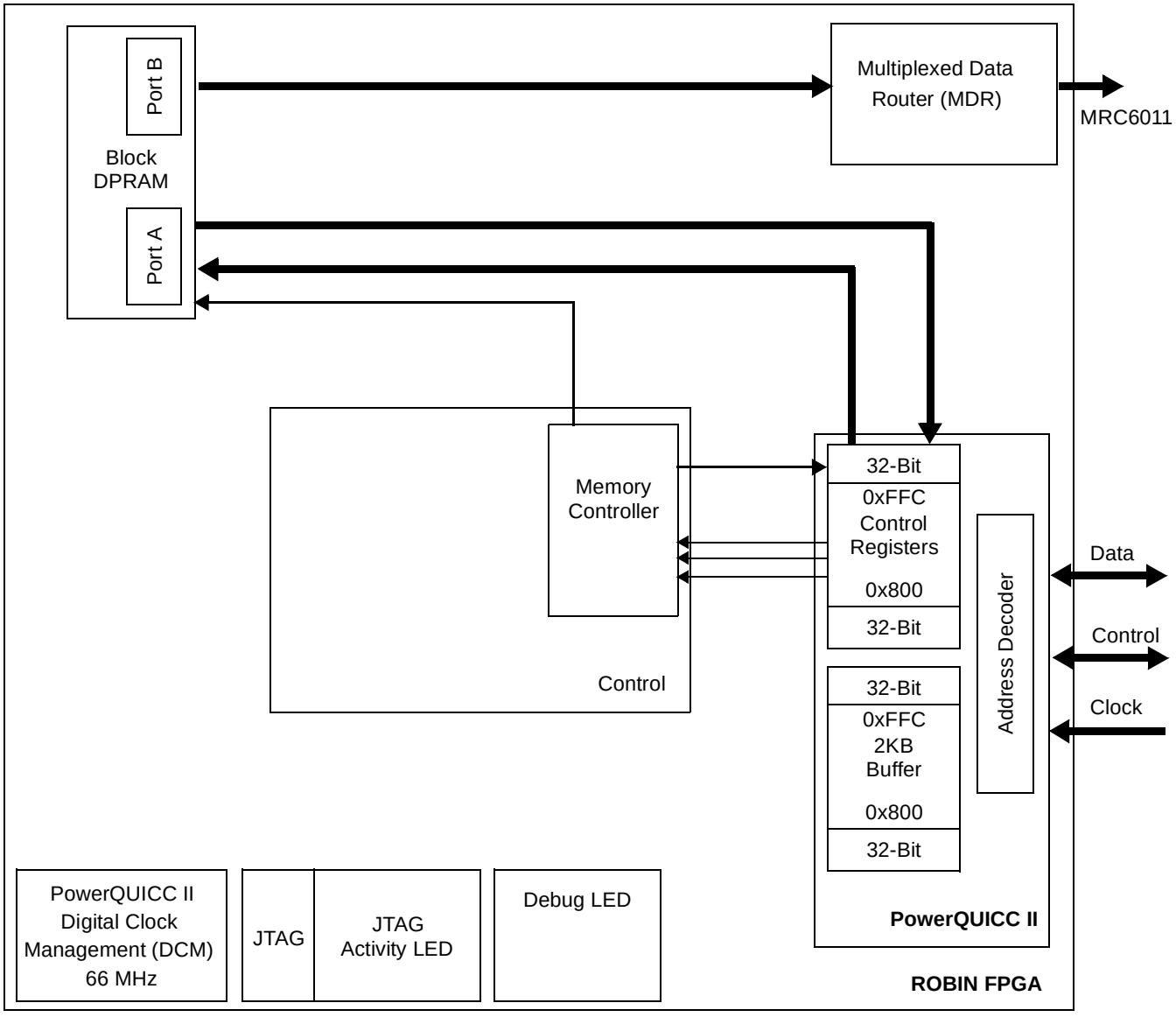

Figure 1. Robin Motherboard Baseband FPGA System

2 MDR Bus

The MRC6011 is the first Freescale processor based on the reconfigurable compute fabric (RCF) technology. The MRC6011 device contains six RCF cores that compose a single homogeneous compute node. The device runs at up to 250 MHz and delivers a performance of 24 Giga complex correlations per second with 8 bits for I and Q. It has two external buses, the system bus and the multiplexed data router (MDR). The MDR is a bus interconnect that directs the input data to the input buffers of targeted RCF cores. The MDR receives data on two multiplexed data input (MDI) ports at a maximum rate of 400 MB per second. The MDI ports are 32-bit data buses with clock, data valid, and sync signals. The sync signal indicates the start of a backplane frame. The data can be captured on either the positive or negative edge of the MDI clock when the MDI data valid signal is asserted. The broadcast router can be configured to distribute data words to one or more RCF cores in a module.

The host can synchronize the multiplexed data to several RCF cores. The host uses the Release Input Buffer Reset (RIBR) signal to deassert the RCF core input buffer reset signal at the start of the next backplane frame in Sync mode.

In some applications, both MDR blocks receive the same data stream through one of the MDI ports. Either multiplexed data capture unit can be configured as the source for both broadcast routers, which distribute the time-slots in the multiplexed data stream according to their own configuration.

2.1 MDR Bus Signals

This reference design uses a total of 71 signals for the MDR bus communication, and each MDR bus has 36 total signals as shown in **Table 1**, and the MDR bus clock is shared between both MDR bus interfaces.

Table 1. MDR Bus Signals in the FPGA Reference Design

Signal Type	Signals
Control signals	fpga_mrc6011_data_valid1c fpga_mrc6011_data_valid2c fpga_mrc6011_clk1c fpga_mrc6011_clk2c fpga_mrc6011_fsync1c fpga_mrc6011_fsync2c fpga_mrc6011c_releaseibrst
Uni-direction data bus signals	fpga_mrc6011_data1c fpga_mrc6011_data2c
MDR data valid signals	fpga_mrc6011_data_valid1c fpga_mrc6011_data_valid2c
Data clock signals	fpga_mrc6011_clk1c fpga_mrc6011_clk2c
Data frame signals to indicate the beginning of the new data frame	fpga_mrc6011_fsync1c fpga_mrc6011_fsync2c
Reset and set-up	fpga_mrc6011_releaseibrst

Figure 2 illustrates the interconnections between the MRC6011 and FPGA devices. The signal connection for the second MDR port is simply doubled, with the exception of the `fpga_mrc6011_releaseibrst` signal, which is shared between both MDR ports. For details on the MRC6011 and the MDR bus, consult the *MRC6011 Reference Manual*.

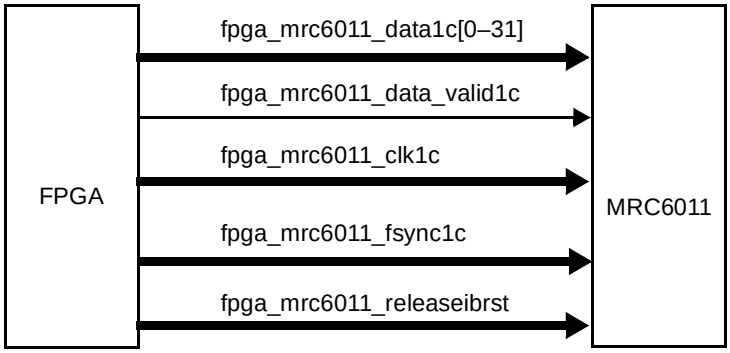

Figure 2. MDR Bus Signal Connection

2.2 MDR Signal Waveform

Figure 3 illustrates the operation of the MDR bus. To initialize the MDR bus, the `fpga_mrc6011_releaseibrst` reset is applied two frames before the data valid is asserted high. When the data valid is asserted, the FPGA can place the data on the data bus. The MRC6011 processor samples the data at either the rising or falling clock edge while the data valid signal is asserted high. When the data is loaded into the MRC6011, the data valid signal is cleared and the MRC6011 stops sampling data. To continue data streaming from the FPGA to the MRC6011, the data valid signal can be asserted again, but it should be aligned with the rising edge of the next frame sycn pulse.

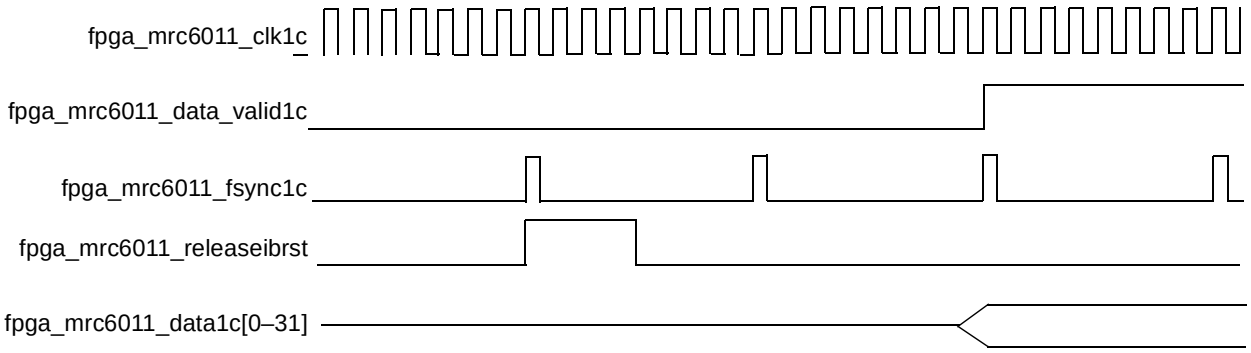

Figure 3. MDR Signal Waveform

3 System Bus

The MPC8260 system bus is a flexible communication medium between the core and internal/external peripheral devices or other bus masters/slaves. The system bus provides 32-bit addressing for a 32-bit or 64-bit wide data base. The burst mode operation can transfer up to 256 bits of data in a four-beat burst. The system bus also supports 8-bit, 16-bit, and 32-bit data ports. Accesses of 1, 2, 3, and 4 bytes can be aligned or unaligned on 4-byte (word) boundaries. The 64-bit, 128-bit, 192-bit, and 256-bit accesses are supported as well. The address and data buses are set up to handle a one-level pipeline, synchronous transaction. The system bus operates in external and internal master modes. For our reference design, the UPM and system bus are configured as follows:

- 32-bit wide port and 32-bit addressing
- Synchronous, single-access transactions
- Single-Bus mode

For details on the system bus, signals used, MPC8260 UPM programming, and FPGA implementation consult AN2889, *FPGA System Bus Interface for the MPC8260: A VHDL Reference Design for the ROBIN Motherboard.*.

4 VHDL Code For the FPGA-MDR Interface

This section covers the top-level architecture of the FPGA-MDR interface, the FPGA control logic, PowerQUICC II logic, memory layout, and other aspects of the interface.

4.1 Top-Level Architecture of the Interface

The top level architecture is presented in **Figure 1**. The main components of the design are the MDR interface, the PowerQUICC II™ interface, JTAG interface, FPGA internal memory register space, FPGA internal memory 4 KB buffer space, and FPGA internal memory BlockRAM data storage space.

4.1.1 Top Port Declaration

The `top_vhdl.vhd` file contains definitions of all top-level ports of the system bus module. The top-level ports used in this design are as follows:.

```
fpga_clock    : in std_logic;
fpga_reset    : in std_logic;
```

- `fpga_clock`. An input clock connected to the FPGA primary clock pad. This pad is routed to the clock input of the digital clock manager (DCM).
- `fpga_reset`. Connected to the reset switch on the ROBIN motherboard and resets the DCM module. In conjuction with the internal reset signal, this signal is the main reset signal to the internal FPGA components.

```
fpga_odyc_irq5_out: out std_logic;
fpga_dspc_irq5_out: out std_logic;
fpga_dspc_irq6_out: out std_logic;
fpga_pq2_irq4_out: out std_logic;
fpga_pq2_pd7_out: out std_logic;
```

- These interrupt lines connect to the MRC6011, MSC8102, and MPC8260 and can be used for various communication schemes among FPGA, MRC6011, MSC8102, and MPC8260.

```
fpga_pq2_clock : in std_logic;
fpga_pq2_csb : in std_logic;
fpga_pq2_rwb : in std_logic;
fpga_pq2_addr : in std_logic_vector (0 to 9);
fpga_pq2_data : inout std_logic_vector (0 to 31);
```

- `fpga_pq2_clock`. A system bus clock source to drive the PowerQUICC II interface logic.
- `fpga_pq2_csb`. A MPC8260 UPM chip select (chip enable) active low signal for FPGA internal memory.
- `fpga_pq2_rwb`. A MPC8260 UPM read/write signal. Logical “0” or low determines write and logical 1 or high determines read bus access.
- `fpga_pq2_addr`. Ten address lines used by the MPC8260 UPM for memory addressing.
- `fpga_pq2_data`. A data bus composed of 32 bidirectional data lines for use by the MPC8260 device.

```
fpga_pq2_trstb: in std_logic;
fpga_pq2_tms : in std_logic;
fpga_pq2_tdo : in std_logic;
fpga_pq2_tdi : in std_logic;
fpga_pq2_tck : in std_logic;
fpga_pq2_hreset: in std_logic;
```

- These MPC8260 JTAG signals are used in this reference design only to display the PowerQUICC II JTAG activity on the board LED. These signals can be used to implement any custom JTAG controller or logic needed in a system.

```
fpga_jtag_conndsp_tdi      : in std_logic;
fpga_jtag_conndsp_tdo      : out std_logic;
fpga_jtag_conndsp_commun   : in std_logic_vector (0 to 3);
fpga_jtag_connmrc6011_tdi  : in std_logic;
fpga_jtag_connmrc6011_tdo  : out std_logic;
fpga_jtag_connmrc6011_commun: in std_logic_vector (0 to 3);
fpga_jtag_dsp_a_tdi        : out std_logic;
fpga_jtag_dsp_a_tdo        : in std_logic;
fpga_jtag_dsp_a_commun     : out std_logic_vector (0 to 3);
fpga_jtag_dsp_b_tdi        : out std_logic;
fpga_jtag_dsp_b_tdo        : in std_logic;
fpga_jtag_dsp_b_commun     : out std_logic_vector (0 to 3);
fpga_jtag_dsp_c_tdi        : out std_logic;
fpga_jtag_dsp_c_tdo        : in std_logic;
fpga_jtag_dsp_c_commun     : out std_logic_vector (0 to 3);
fpga_jtag_mrc6011_a_tdi    : out std_logic;
fpga_jtag_mrc6011_a_tdo    : in std_logic;
fpga_jtag_mrc6011_a_commun : out std_logic_vector (0 to 3);
fpga_jtag_mrc6011_b_tdi    : out std_logic;
fpga_jtag_mrc6011_b_tdo    : in std_logic;
fpga_jtag_mrc6011_b_commun : out std_logic_vector (0 to 3);
fpga_jtag_mrc6011_c_tdi    : out std_logic;
fpga_jtag_mrc6011_c_tdo    : in std_logic;
fpga_jtag_mrc6011_c_commun : out std_logic_vector (0 to 3);
fpga_jtag_dipswdsp         : in std_logic_vector (0 to 2);
fpga_jtag_dipswmrc6011     : in std_logic_vector (0 to 2);
```

- The JTAG logic uses these signals to configure the JTAG chain. Six JTAG enabled devices connect to the FPGA, three MRC6011 and three MSC8102 devices. Each device has six signals for routing standard JTAG signals. In addition, there are two JTAG connectors for use by the JTAG master controller.

```
fpga_mrc6011_data1c : out std_logic_vector(31 downto 0);
fpga_mrc6011_data2c : out std_logic_vector(31 downto 0);
fpga_mrc6011_fsync2c : out std_logic;
fpga_mrc6011_fsync1c : out std_logic;
fpga_mrc6011_data_valid2c : out std_logic;
fpga_mrc6011_data_valid1c : out std_logic;
fpga_mrc6011_clk2c  : out std_logic;
fpga_mrc6011_clk1c  : out std_logic;
fpga_mrc6011c_releaseibrst : out std_logic;
```

- `fpga_mrc6011_data1c` and `fpga_mrc6011_data2c`. Two 32-bit uni-directional buses used by the FPGA to stream MDR data to the MRC6011.
- `fpga_mrc6011_fsync2c` and `fpga_mrc6011_fsync1c`. Carry the MDR frame sync signals.

- `fpga_mrc6011_data_valid2c` and `fpga_mrc6011_data_valid1c`. Carry the MDR data valid signals to indicate when the MRC6011 processor should start sampling the MDR data.
- `fpga_mrc6011_clk2c` and `fpga_mrc6011_clk1c`. Provide the data sampling clock signals to the MRC6011 processor.
- `fpga_mrc6011c_releaseibrst`. Provides the reset release signal to the MDR port, and is used during MDR port initialization.

```
debug_mictor : out std_logic_vector(31 downto 0);
debug_led    : out std_logic_vector(3 downto 0)
```

- The debug_mictor port consists of 32 debug lines routed to the P4 Mictor connector on the ROBIN motherboard. All 32 lines can be routed internally in the FPGA to any internal FPGA signal for probing/debugging. The debug_led port consists of four lines that connect to four LEDs on the ROBIN motherboard. The four LEDs are marked as FPGA_LED_0 (D4), FPGA_LED_1 (D5), FPGA_LED_2 (D6), and FPGA_LED_3 (D7).

4.1.2 Internal Signals

All internal signals (wires) have a prefix of `i_` and interconnect the modules and FPGA I/O pins. Other internal signals have a prefix of `r_` to indicate that they are used as registers.

```
signal i_module_reset: std_logic;
signal r_jtag_reg   : std_logic_vector(5 downto 0);
```

- `i_module_reset`. The main reset to all FPGA internal blocks, with the exception of DCM and the MPC8260 component. This line is active high and is driven by an inverted DCM lock signal ANDed with an MPC8260-accessible write reset register. This line is asserted either when the DCM is not locked or when the user manually resets the FPGA by writing to the FPGA configuration reset register via the MCP8260.
- `r_jtag_reg`. An internal bus reserved for future expansion of the reconfigurable JTAG chain.

```
signal i_clk1x_out  : std_logic;
signal i_clkdv_out  : std_logic;
signal i_clk2x_out  : std_logic;
signal i_clk2x180_out: std_logic;
signal i_clkfx_out  : std_logic;
signal i_dcm_lock_out: std_logic;
```

- The DCM uses these signals to provide various clock sources to the FPGA logic. Our reference design uses the `i_clkdv_out` clock source for the clock division capability of the DCM component. Other sources can be used. For example, use `i_clk1x_out` for the clock source when frequency is not altered but the clock is de-skewed. Use `i_clk2x_out` for a de-skewed clock source with twice the input frequency.

```
signal i_sdram_dout : std_logic_vector(31 downto 0);
signal i_sdram_addra: std_logic_vector(14 downto 0);
signal i_sdram_ena  : std_logic;
signal i_sdram_wea  : std_logic;
signal i_sdram_addrb: std_logic_VECTOR(14 downto 0);
signal i_sdram_dinb : std_logic_VECTOR(31 downto 0);
signal i_sdram_doutb: std_logic_VECTOR(31 downto 0);
signal i_sdram_enb  : std_logic;
signal i_sdram_web  : std_logic;
signal i_sdram_a_data: std_logic_vector(31 downto 0);
```

- These signals connect the internal SDRAM (FPGA BlockRAM) component with other FPGA components. They can connect to the external SDRAM chip on the ROBIN motherboard.

```
signal i_pq2_data_out           : std_logic_vector (31 downto 0);
signal i_pq2_data_in            : std_logic_vector (31 downto 0);
signal i_pq2_ctrl_rst           : std_logic;
signal i_pq2_ctrl_sdram_load    : std_logic;
signal i_pq2_ctrl_sdram_unload  : std_logic;
signal i_pq2_ctrl_sdram_load_done: std_logic;
signal i_pq2_ctrl_sdram_load_done_clr : std_logic;
signal i_pq2_ctrl_sdram_unload_begin: std_logic;
signal i_pq2_ctrl_sdram_unload_begin_clr : std_logic;
signal i_pq2_ctrl_simulation    : std_logic;
signal i_pq2_ctrl_simulation_on : std_logic;
signal i_reg03_sdram_words      : std_logic_vector(31 downto 0);
signal i_pq2mem_addr            : std_logic_vector(9 downto 0);
signal i_pq2mem_en              : std_logic;
signal i_pq2mem_we              : std_logic;
signal i_pq2mem_dout            : std_logic_vector(31 downto 0);
```

These signals carry the control logic from the MCP8260 interfaces to various locations in the internal FPGA logic:

- `i_pq2_data_out` and `i_pq2_data_in`. 32-bit output and input buses.
- `i_pq2_ctrl_rst`. Issues a master internal FPGA reset signal.
- `i_pq2_ctrl_sdram_load`. Used by the main control state machine to select the SDRAM loading mode.
- `i_pq2_ctrl_sdram_unload`. Used by the main control state machine to select the SDRAM unloading mode.
- `i_pq2_ctrl_sdram_load_done` and `i_pq2_ctrl_sdram_unload_begin`. Trigger the internal memory controller and its state machine to begin loading/unloading data to/from FPGA memory.
- `i_pq2_ctrl_sdram_load_done_clr` and `i_pq2_ctrl_sdram_unload_begin_clr`. Internally generated FPGA signals to notify the FPGA memory controllers that loading and unloading of data is complete and the memory controllers should be temporarily disabled.
- `i_pq2_ctrl_simulation` and `i_pq2_ctrl_simulation_on`. Reserved for additional unloading of data to the MRC6011 device.
- `i_reg03_sdram_words`. A bus used by the FPGA memory access counter to count the number of MPC8260 memory read/writes. These signals connect to the MPC8260 memory-mapped read registers, providing debugging visibility into how many words are read or written in/out of the FPGA.

4.1.3 Debug Signals

The following signals are strictly for debugging and are routed to the Mictor connector P4 on the ROBIN motherboard. A Mictor-compatible logic analyzer can be used to sample and view these signals.

```
signal debug_addr          : std_logic_vector (9 downto 0);
signal debug_ena           : std_logic;
signal debug_wea           : std_logic;
signal debug_ctrl_fsm      : std_logic_vector(7 downto 0);
```

To simplify the design, the number of functional logic blocks is kept to minimum. When you add logical blocks for more complex designs, keep in mind that the design should be as modular and hierarchical as possible.

4.2 Control Logic

The control logic module is the brains of the FPGA architecture, which controls all FPGA modules. It uses the PowerQUICC II interface logic to receive configuration and operation instructions from the MPC8260 device and to send the FPGA status and debug information back to the MPC8260 (see **Figure 4)**. The heart of the control module is the finite state machine (see **Figure 5)**. Two smaller counter blocks serve as memory controllers for the two dual-ported memory blocks. The input and output port of the control logic is as follows:

```
p_clock                                 : in std_logic;
p_reset                                 : in std_logic;
p_pq2_ctrl_sdram_load_in                : in std_logic;
p_pq2_ctrl_sdram_unload_in              : in std_logic;
p_pq2_ctrl_simulation_in                : in std_logic;
p_pq2_ctrl_sdram_load_done_in           : in std_logic;
p_pq2_ctrl_sdram_load_done_clr_out      : out std_logic;
p_pq2_ctrl_sdram_unload_begin_in        : in std_logic;
p_pq2_ctrl_sdram_unload_begin_clr_out   : out std_logic;
p_pq2_ctrl_simulation_on_out            : out std_logic;

p_pq2mem_addrb_out                      : out std_logic_VECTOR(9 downto 0);
p_pq2mem_enb_out                        : out std_logic;
p_pq2mem_web_out                        : out std_logic;
p_sdram_addrb_out                       : out std_logic_VECTOR(14 downto 0);
p_sdram_enb_out                         : out std_logic;
p_sdram_web_out                         : out std_logic;
p_sdram_addra_out                       : out std_logic_VECTOR(14 downto 0);
p_sdram_wea_out                         : out std_logic;
p_sdram_ena_out                         : out std_logic;
debug_ctrl_fsm                          : out std_logic_vector(7 downto 0)
```

- `p_clock`. The main clock tree fed from the DCM DV output (`i_clkdv`) clock source.
- `p_reset`. The main reset line to the block. It is primarily used to reset the main finite state machine. It is driven from the i_module_reset (global reset) line.
- The remaining signals control the finite state machine (see **Figure 5**).
- `p_pq2mem_addrb_out`, `p_pq2mem_enb_out`, and `p_pq2mem_web_out`. Output ports to control the 4 KB buffer DPRAM in the PowerQUICC II logic block.
- `p_sdram_addrb_out`, `p_sdram_enb_out`, `p_sdram_web_out`, p_sdram_addra_out, `p_sdram_wea_out`, and `p_sdram_ena_out`. Output ports to control the dual-ported BlockRAM, which is 128 KB of FPGA internal storage memory.

- `debug_fsm`. For debugging purposes only.

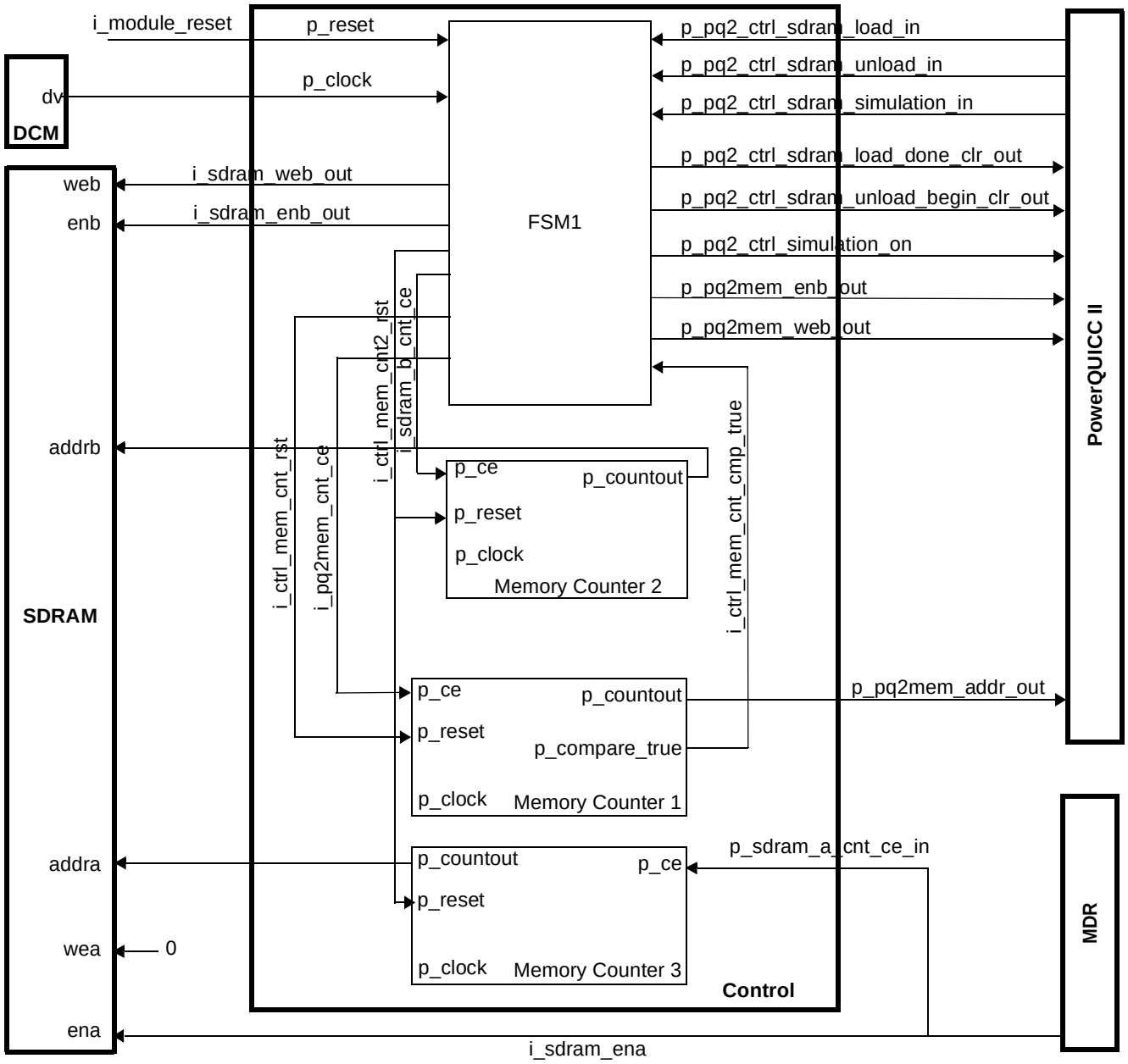

Figure 4. Control Logic Module

4.2.1 Memory Counter 1

Memory counter 1 is controlled by the FSM1 state machine and addresses the FPGA 4 KB buffer DPRAM in the PowerQUICC II logic block. The component has following input and output ports:

```
p_clock             : in std_logic;
p_reset             : in std_logic;
p_ce                : in std_logic;
p_countout          : out std_logic_vector(9 downto 0);
p_compare_true      : out std_logic
```

- `p_clock`. A clock signal supplied from the `i_clkdv` DCM clock source.

- p_reset. Carries a reset signal to reset the addressing counter to 0. This reset signal is i_ctrl_mem_cnt_rst, and it is generated by the FSM1 state machine. The reset is asserted at FPGA initialization, at the beginning of the SDRAM load and unload memory operation sequence.
- p_ce. Enables the address counter. This port connects to the i_pq2mem_cnt_ce signal, which is controlled by the FSM1 state machine. This signal, and therefore the counter, is active during the SDRAM load and unload sequence. This address counter has address compare capability to detect when the address counter reaches the end of the 4 KB DPRAM address space.
- port p_compare_true. When the end of the address space is detected, this signal notifies the FSM1 state machine. When this signal is active, the SDRAM load and unload sequence in the FSM1 state machine completes, and the state machine is sent to the next state. The remaining port p_countout is connected to the p_addrb_in port in the PowerQUICC II logical block via the i_pq2mem_addrb signal. This is a 10-bit bus carrying an address value to address port B of the 4 KB DPRAM. For details on memory control signal generation and the state machine, refer to **Figure 5** and **Table 2**.

4.2.2 Memory Counter 2

Memory counter 2 is controlled by the FSM1 state machine and addresses the FPGA internal BlockRAM storage DPRAM. The component has following input and output ports:

```
p_clock             : in std_logic;
p_reset             : in std_logic;
p_ce                : in std_logic;
p_countout          : out std_logic_vector(14 downto 0)
```

- p_clock. A clock signal supplied from the i_clkdv DCM clock source.
- p_reset. Carries a reset signal to reset the addressing counter to 0. This reset signal is i_ctrl_mem_cnt2_rst, and it is generated by the FSM1 state machine. The reset is asserted at FPGA initialization and the beginning of the SDRAM unload memory operation sequence.
- p_ce. Enables the address counter and connects to the i_sdram_b_cnt_cnt_ce signal, which is controlled by the FSM1 state machine. This signal, and therefore the counter, is active during the SDRAM load and unload sequence.
- p_countout. Connects to address port B in the internal BlockRAM of DPRAM, via the i_sdram_addrb signal. This is a 15-bit bus carrying address values to address port B. For details on memory control signal generation and the state machine, refer to **Figure 5** and **Table 2**.

4.2.3 Memory Counter 3

Memory counter 3 is controlled by the FSM1 state machine and the MDR block. This counter addresses port A of the FPGA internal BlockRAM storage DPRAM. It has following input and output ports:

```
p_clock                 : in std_logic;
p_reset                 : in std_logic;
p_ce                    : in std_logic;
p_countout              : out std_logic_vector(14 down to 0)
```

- p_clock. A clock signal supplied from the i_clkdv DCM clock source.

- p_reset. Carries a reset signal to reset the addressing counter to 0. This reset signal is i_ctrl_mem_cnt2_rst and is generated by the FSM1 state machine. The reset is asserted at FPGA initialization and begins the unload SDRAM memory operation sequence.
- p_ce. Enables the address counter and connects to p_sdram_a_cnt_cnt_ce_in, which is controlled by the MDR logic block. This signal and therefore the counter is active during the simulation sequence.
- p_countout. Connects to the addra port A in the internal BlockRAM of DPRAM via the i_sdram_addra signal. This 15-bit bus carries the address value to address port A. For details on memory control signal generation and the state machine, see the state machine diagram in **Figure 5** and **Table 2**, *FSM1 Control Signal Assignment*, on page 14.

4.2.4 Finite State Machine 1 (FSM1)

The FPGA FSM1 state machine is illustrated in the **Figure 9**. It has 14 states, 25 transitions, 6 inputs, and 16 outputs. The clock source is provided by port p_clock, which is driven by the i_clkdv clock. The rising clock edge is used for clocking. The p_reset port provides the positive edge of the reset signal. The state machine encoding type used is one-hot encoding. The progression of states is as follows:

1. In Finit, both address counters are reset.
2. F0 indicates the beginning of the state machine's repetitive operation. In this state, the ctrl_mem_cnt counter is the only counter that is reset.
3. F1 is a decision or wait state. The machine waits until the p_pq2_ctrl_sdram_load_done_in, p_pq2_ctrl_sdram_unload_in, or p_pq2_ctrl_simulation_in signals are asserted. Then, the next state is either F1A, F4, or F9, respectively.

   In the F1A, F4 and F9 states, the SDRAM load, unload, and simulation operations are performed. When the p_pq2_ctrl_sdram_load_done_in signal is asserted high, the state machine proceeds to the next wait state, F1A. In this wait state, memory counter 1 is enabled, along with the PowerQUICC II 4 KB buffer DPRAM.
4. In F2, memory counter 2 and the FPGA internal storage BlockRAM are enabled. During the F1A and F2 states, the data is unloaded from the 4 KB buffer DPRAM and loaded into the FPGA internal storage BlockRAM. One clock cycle delay is required before data is transmitted from the 4 KB buffer DPRAM.
5. F2 terminates when the address memory counter 1 detects the end of the 4 KB buffer space.
6. In the F2A state, the 4 KB buffer DPRAM and its address counter are disabled, but the FPGA internal storage BlockRAM is enabled for one more clock cycle to allow the last data word to propagate.
7. In the F3 state, the FPGA PowerQUICC II memory-mapped register in the PQ2 memory block notifies the MPC8260 that loading of the 4 KB data block is complete. The MPC8260 can use this register to determine when the next 4 KB of data should be sent.
8. On the next clock cycle, the state machine returns to the F0, the decision wait state. In the F0 state, memory counter 1 is reset to zero.
9. The operation of loading the 4 KB data can repeat.

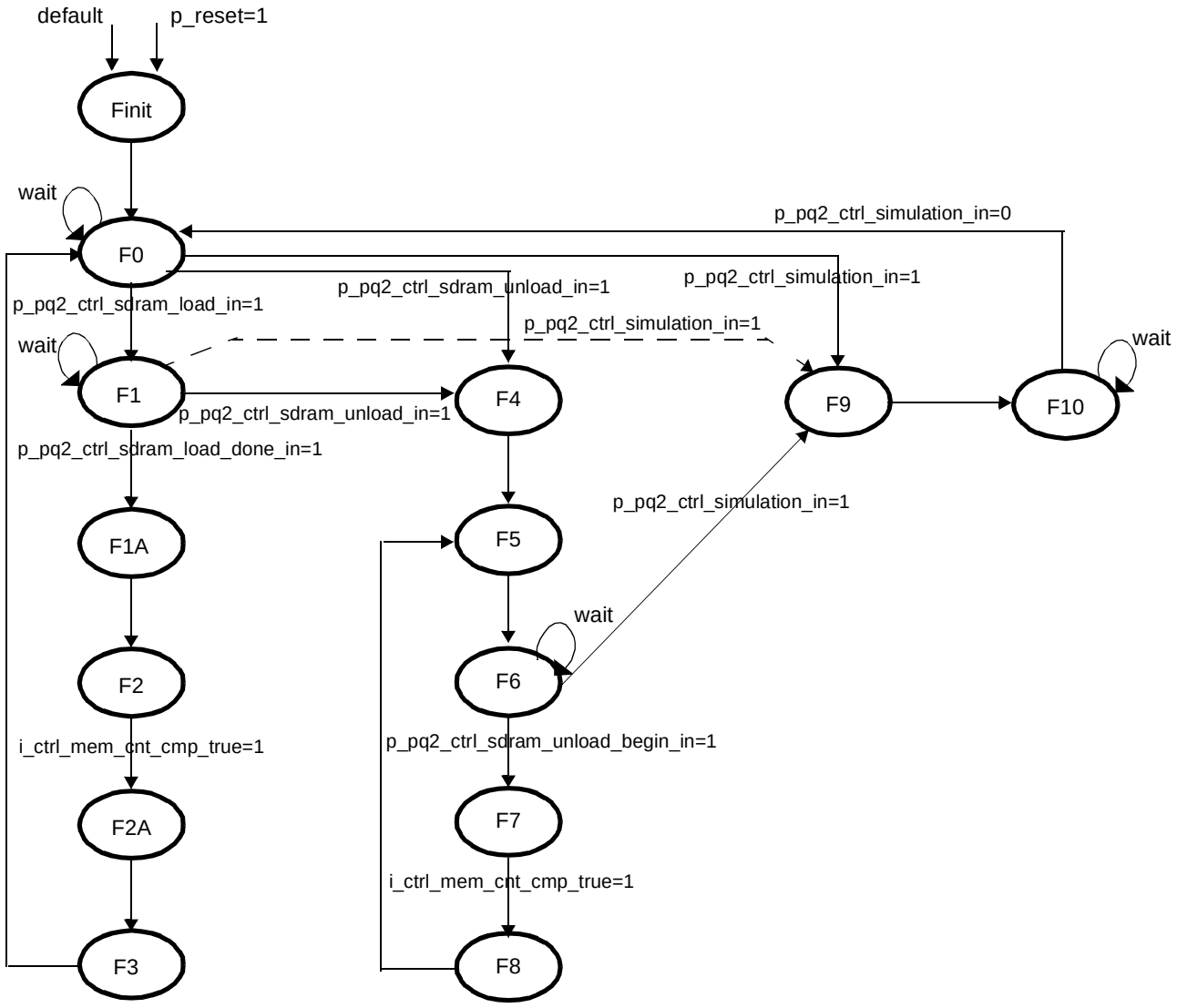

Figure 5. Main Control Finite State Machine (FSM1) Operation

1. When data loading completes, the `p_pq2_ctrl_sdram_unload_in` signal is asserted high and the state machine proceeds to the F4 state, in which memory counter 2 is reset to zero.
2. On the next clock cycle, the state increments to F5, in which memory counter 1 is reset to zero.
3. On the next clock cycle, the state increments to F6, in which the state machine awaits further instructions. Based on the values of `p_pq2_ctrl_sdram_unload_begin_in` and `p_pq2_ctrl_simulation_in`, it proceeds to the SDRAM unload or simulation operations.
4. When `p_pq2_ctrl_sdram_unload_begin_in` is asserted high, the next state is F7, in which both memory controllers and both memories are enabled to perform the SDRAM unload operation.
5. At the end of the 4 KB address space, the state machine increments to the F8 state, in which the memory unloading signal is sent to the MPC8260 via the FPGA PQ2 memory-mapped register in the PowerQUICC II memory block. The MPC8260 can use this register to determine when the next block of 4 KB data should be read from the FPGA.

6. On the next clock cycle, the state machine returns to the F5 state, and the operation of unloading the 4 KB of data can repeat.
7. When the `p_pq2_ctrl_simulation_in` signal is asserted in the F6 state, the state machine increments to the F9 state, a simulation state in which data streams from the FPGA into the MRC6011 via the MDR port. This operation is beyond the scope of this application note. See AN2890, *FPGA MDR Antenna Interface for MRC6011: A VHDL Reference Design for the ROBIN Motherboard.*

Table 2. FSM1 Control Signal Assignment

Signal Name	Finit	F0	F1	F1A	F2	F2A	F3	F4	F5	F6	F7	F8	F9	F10
i_ctrl_mem_cnt_rst	1	1							1					
i_ctrl_mem_cnt2_rst	1							1						
i_pq2mem_cnt_ce				1	1						1			
p_pq2mem_enb_out				1	1						1			
p_pq2mem_web_out											1			
i_sdram_a_cnt_ce					1	1					1			
p_sdram_enb_out					1	1					1			
p_sdram_web_out					1	1								
p_pq2_ctrl_sdram _load_done_clr_out							1							
p_pq2_ctrl_sdram_unload_ begin_clr_out												1		
p_pq2_ctrl_simulation_on_out														1

4.3 PowerQUICC II Logic Module

The PowerQUICC II logic module is the second largest module in the FPGA. It provides a system bus interface between the FPGA and MPC8260. Its main components are the 4 KB buffer DPRAM, sixteen 32-bit registers, memory and register detectors, output enable data multiplexes, and miscellaneous glue logic (see **Figure 6**). The PowerQUICC II logic module is strongly connected with the main control module as it receives and sends the PowerQUICC II control signals to the main FPGA state machine.

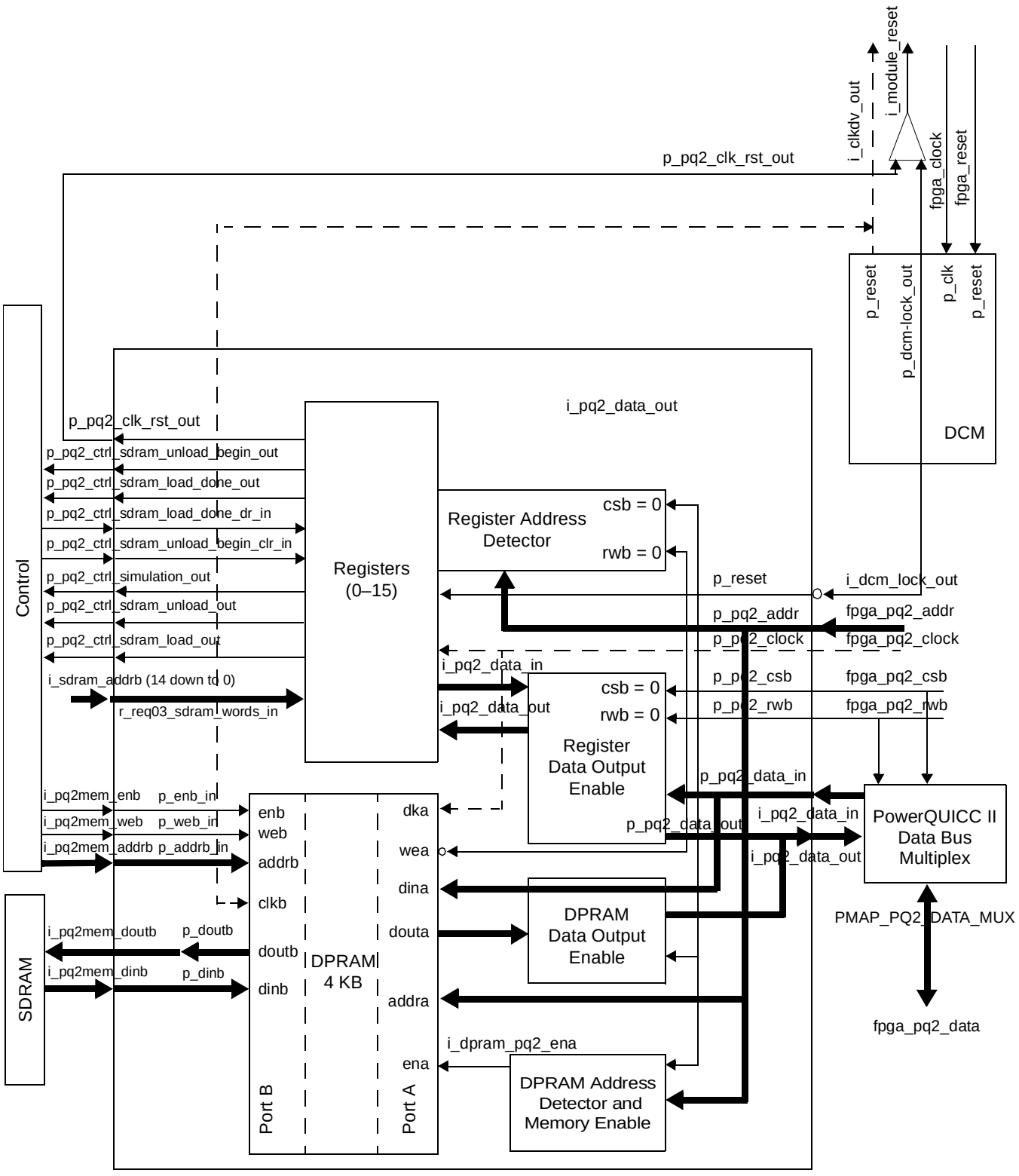

Figure 6. Architecture of the PowerQUICC II Logic Module

```
p_pq2_clock                 : in std_logic;
p_reset                     : in std_logic;
```

- p_pq2_clock. The clock source for the PowerQUICC II logic module.
- p_reset. Connected to fpga_reset, which in turn connects to the FPGA reset button on the ROBIN motherboard.

```
p_pq2_csb                   : in std_logic;
p_pq2_rwb                   : in std_logic;
p_pq2_addr                  : in std_logic_vector (9 downto 0);
p_pq2_data_in               : in std_logic_vector (31 downto 0);
p_pq2_data_out              : out std_logic_vector (31 downto 0);
```

- p_pq2_csb, p_pq2_rwb, and p_pq2_addr. PowerQUICC II UPM system bus control and address bus signals that connect directly to the MPC8260 processor.
- p_pq2_data_in and p_pq2_data_out. 32-bit buses that connect to the PowerQUICC II multiplexed data bus module. This multiplex is controlled by the fpga_pq2_csb and fpga_pq2_rwb signals to determine the value of the p_pq2_data_in, p_pq2_data_out, and fpga_pq2_data buses.

```
p_pq2_ctrl_rst_out              : out std_logic;
p_pq2_ctrl_sdram_load_out       : out std_logic;
p_pq2_ctrl_sdram_unload_out:    : out std_logic;
p_pq2_ctrl_simulation_out       : out std_logic;
p_pq2_ctrl_sdram_load_done_out  : out std_logic;
p_pq2_ctrl_sdram_load_done_clr_in: in std_logic;
p_pq2_ctrl_sdram_unload_begin_out : out std_logic;
p_pq2_ctrl_sdram_unload_begin_clr_in: in std_logic;
p_reg03_sdram_words_in          : in std_logic_vector(31 downto 0);
```

- These ports are routed directly to the ports of the control logic module. The functionality of these ports is the same as described for the port of the control logic module. See **Section 4.2**, *Control Logic*, on page 9.

```
p_addrb_in                  : IN std_logic_VECTOR(9 downto 0);
p_clkb_in                   : IN std_logic;
p_enb_in                    : IN std_logic;
p_web_in                    : IN std_logic;
```

- p_addrb_in. A 10-bit address bus connected to the 4 KB DPRAM and driven by the control logic module.
- p_clkb_in. A clock source for port B of the 4 KB DPRAM that is run from the i_clkdv clock tree.
- p_enb_in and p_web_in. Driven by the control logic module and connect to the memory enable and read/write enable ports of the port B DPRAM.

```
p_dinb_in                   : IN std_logic_VECTOR(31 downto 0);
p_doutb_out                 : OUT std_logic_VECTOR(31 downto 0);
```

- p_dinb_in and p_doutb_out. 32-bit input and output buses to connect the 4 KB buffer DPRAM and FPGA internal storage BlockRAM.

```
debug_addr                : out std_logic_vector(9 downto 0);
debug_ena                 : out std_logic;
debug_wea                 : out std_logic
```

- Used exclusively for debug purposes.

4.3.1 4 KB Buffer DPRAM

The Xilinx CORE Generator™ was used to create the 4 KB buffer dual-ported random access memory (DPRAM) module. Each memory port has an address bus, a data input bus, a data output bus, an individual clock signal, an individual memory enable signal, and an individual write enable signal. Both port A and B are 32 bits wide and 1024 words deep. To address the 1024 words, the 10 address lines are used for both port A and B.

```
Check:
Port A and B-> (2^10) address lines * 32 bits port size = 32,768 bits = 4,096 bytes
```

```
component dpram_pq2
port (
addra  : IN std_logic_VECTOR(9 downto 0);
addrb  : IN std_logic_VECTOR(9 downto 0);
clka   : IN std_logic;
clkb   : IN std_logic;
dina   : IN std_logic_VECTOR(31 downto 0);
dinb   : IN std_logic_VECTOR(31 downto 0);
douta  : OUT std_logic_VECTOR(31 downto 0);
doutb  : OUT std_logic_VECTOR(31 downto 0);
ena    : IN std_logic;
enb    : IN std_logic;
wea    : IN std_logic;
web    : IN std_logic);
end component;
```

- `addra`, `clka`, `dina`, `douta`, `ena`, and `wea`. Port A signals to connect to the MPC8260 system bus. The `dina` and `douta` 32-bit buses are routed to the DPRAM data output enable logic block. The `ena` port is driven by `i_dpram_pq2_ena`, which is generated by the DPRAM address detector and memory enable logic block. The `clka` port is driven by the `p_pq2_clock` tree.
- `addrb`, `clkb`, `dinb`, `doutb`, `enb` and `web`. Port B signals to connect to FPGA internal storage BlockRAM. The `dinb` and `doutb` 32-bit ports directly connect to the FPGA internal storage BlockRAM data input and output buses. The `enb`, `web`, and `addrb` ports connect to the control logic module that drives these signals. The `clkb` port is driven by the `i_clkdv` clock tree.

4.3.2 Xilinx CORE Generator

To create or modify the DPRAM core using the Xilinx CORE Generator, select the core in the source window within the Xilinx Project Navigator. Next, in the Process window run **MANAGE CORES** within the Coregen process. When the Xilinx CORE Generator application launches, you can chose either to edit or create a new core. The design core window for the dual-port block memory is straightforward. On the first page, set the width of port A to 32 bits and the depth to 1024. Set the width of port B to 32 bits and the depth to 1024, the same as for port A. Both port A and port B options for configuration should be set to **READ AND WRITE** and for write mode should be set to **READ AND WRITE**.

On the second core design page for port A, select **ENABLE PIN**. The **HANDSHAKING PIN** and **REGISTER INPUT** features are not used. In the output register options, set the **ADDITIONAL OUTPUT PIPE STAGES** to 0. The **SINIT PIN** functionality is not used, so the initialization value (hexadecimal) can be left blank. The pin polarity options are **RISING EDGE TRIGGERED** for an active clock edge, **ACTIVE HIGH** for the enable pin, and **ACTIVE HIGH** for the write enable pin.

On the third core design page for port B, select the optional **ENABLE PIN**. The **HANDSHAKING PIN** and **REGISTER INPUT** features are not used. In the output register options, set the **ADDITIONAL OUTPUT PIPE STAGES** to 0. The **SINIT PIN** functionality is not used, so the initialization value (hexadecimal) can be left blank. The pin polarity options are **RISING EDGE TRIGGERED** for active clock edge, **ACTIVE HIGH** for the enable pin, and **ACTIVE HIGH** for the write enable pin.

On the final, fourth core design page within the Initial Contents, the global initialization value is set to 0, but it can be set to any value desired. You can preload the memory with the initialization hexadecimal file (`.coe`), which can be useful in debugging. If desired, check the **LOAD INIT FILE** check box and load the initialization hexadecimal file. The bottom of this page presents the summary of the designed DPRAM core. For our example, the information panel should display as follows:

```
Address Width A           10
Address Width B           10
Blocks Used               2
Port A Read Pipeline Latency 1
Port B Read Pipeline Latency 1
```

If these values are correct, click **GENERATE**, and the new core is generated.

4.3.3 PowerQUICC II FPGA Control Register Space

In addition to the 4 KB memory mapped buffer space, the MPC8260 can access the sixteen 32-bit register space. For the purpose of this generic reference design, only first four registers are used. The remaining twelve registers are available for future use.

FPGA_CTRL FPGA Control Register BASE+0xFFC

	Bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	R	—														
TYPE	R/W															
RESET	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
	—													SIM	UNL	LD
TYPE	R/W.															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FPGA_CTRL is accessed from the MPC8260.

Table 3. FPGA_CTRL Bit Descriptions

Name	Reset	Description	Settings
R 0	1	**Soft Reset** When this bit is set, the FPGA can be reset or held in the reset state from the MPC8260. The MPC8260 reference code uses this bit to reset the FPGA during FPGA initialization. This bit is set by default, so the FPGA is kept in reset by default until instructed otherwise by the MCP8260. The p_pq2_ctrl_rst_out signal carries the value of the reset bit to the reset of the FPGA logic.	0 Deasserts the reset state of the internal FPGA modules. 1 Holds the internal FPGA modules in the reset state.
— 1–28	0	Reserved. Cleared to zero for future compatibility.	
SIM 29	0	**Simulation** Unloads the data from the SDRAM into the MDR port of the MRC6011 processor. This operation is covered in **Section 6**, *FPGA Operating Sequence,* on page 31. The p_pq2_ctrl_simulation_out signal carries the value of this bit to the master control state machine (FSM1).	0 OFF. 1 ON.
UNL 30	0	**Unload SDRAM** Initiates the FPGA unload SDRAM sequence within the FSM1 in the main control block. When this bit is set, the state machine is instructed to unload the SDRAM state. The p_pq2_ctrl_sdram_unload_out signal carries the value of this bit to the master control state machine (FSM1).	0 OFF. 1 ON.
LD 31	0	**Load SDRAM** Initiates the FPGA load SDRAM sequence within FSM1 in the main control block. When this bit is set, the state machine is instructed to load the SDRAM state. The p_pq2_ctrl_sdram_load_out signals carries the value of this bit to the master control state machine (FSM1).	0 OFF. 1 ON.

FPGA_VER FPGA Image Version Register BASE+0xFF8

Bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
YEAR								MONTH							
TYPE								R							

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DAY								VER							
TYPE								R							

FPGA_VER determines the FPGA image version.

Table 4. FPGA_VER Bit Descriptions

Name	Description	Settings
YEAR 0–7	**Year** Indicates the last two digits of the year in which the FPGA image was created.	2004 would appear as 0x04.

Table 4. FPGA_VER Bit Descriptions (Continued)

Name	Description	Settings
MONTH 8–15	**Month** Indicates the month in which the FPGA image was created.	0x01 Corresponds to January. 0x0C Corresponds to December.
DAY 16–23	**Day** Indicates the day in which the FPGA image was created.	0x01 Corresponds to day 01 of the given month. 0x07 Corresponds to day 31 of the given month.
VER 24–31	**Version** Indicates the version number of the FPGA image.	

SDRAM_CTRL SDRAM Control Register BASE+0xFF4

	Bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
	—														ULB	LD
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SDRAM_CTRL can be accessed from the MPC8260.

The FPGA `p_pq2_ctrl_sdram_load_done_clr_in` and `p_pq2_ctrl_sdram_unload_begin_clr_in` internal signals are used to clear the SDRAM_CTRL register bit flags 30 and 31. If these signals are high, the SDRAM_CTRL register is cleared. The two signals are controlled by the master control state machine (FSM1).

Table 5. SDRAM_CTRL Bit Descriptions

Name	Reset	Description
— 0–29	0	Reserved. Cleared to zero for future compatibility.
ULB 30	0	**SDRAM Block of 512 32-Bit Words Begin Unload** Sends a request to the FPGA FSM1 to begin the transfer of 512 32-bit words from the storage BlockRAM to the 4 KB buffer space.
LD 31	0	**SDRAM Block of 512 32-Bit Words Begin Load Done** Sends a request to the FPGA FSM1 to begin the transfer of 512 32-bit words from the 4 KB buffer space into the storage BlockRAM**.**

SDRAM_NO SDRAM Number of 32-Bit Words Register BASE+0xFF0

	Bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SDRAM_NO is used by MPC8260 to keep track of how many 32-bit words the FPGA reads or writes into the storage BlockRAM. The FPGA address counter logic holds the count.

Table 6. SDRAM_NO Bit Descriptions

Name	Reset	Description
C0–C31 0–31	0	**SDRAM Word Count**

4.3.4 PowerQUICC II Memory/Register Detector

The memory/register detector determines whether the MPC8260 is requesting the 4 KB buffer space or one of the sixteen 32–bit registers. The internal 32-bit data bus, `p_pq2_dat_out` signal, connects to the 4 KB buffer data out bus only when `p_pq2_csb` and `i_dpram_pq2_addra(9)` are equal to 0. Similarly, the 4 KB buffer DPRAM is enabled only when both of `p_pq2_csb` and `i_dpram_pq2_addra`(9) are equal to 0. If the `i_dpram_pq2_addra`(9) signal is equal to 1, the access is interpreted as a register read or write, and the data is read and written from one of the sixteen 32-bit registers.

4.4 PowerQUICC II Data Multiplex

The data multiplex component is used as a smart driver for the `fpga_pq2_data` 32-bit bidirectional data bus. The ports of the component are as follows:

```
p_pq2_oeb     : IN std_logic;
p_pq2_csb     : IN std_logic;
p_pq2_data_out: IN std_logic_vector(31 downto 0);
p_pq2_data_inout: INOUT std_logic_vector(31 downto 0);
p_pq2_data_in: OUT std_logic_vector(31 downto 0)
```

- `p_pq2_oeb`. Connects to the `fpga_pq2_rwb` signal to determine whether the system bus transaction is a read or write.
- `p_pq2_csb. Connects` to the `fpga_pq2_csb` signal, which is a system bus signal used by the UPM as a chip select.
- `p_pq2_data_out`. A 32-bit input bus connected to the `i_pq2_data_out` that provides connection to the register or 4 KB buffer memory-mapped space.

- p_pq2_data_inout. A bidirectional bus directly connected directly to the MPC8260 via the fpga_pq2_data port. p_pq2_data_inout is equal to p_pq2_data_out when p_pq2_oeb = 1 and p_pq2_csb = 0. Otherwise, p_pq2_data_inout is held in *high z* state, allowing the MCP8260 UPM controller to drive the bus.
- p_pq2_data_in. A 32-bit output bus connected to the i_pq2_data_in to provide data to the register or 4 KB buffer memory-mapped space.

4.5 BlockRAM Storage Memory

The Xilinx CORE Generator™ was used to create the dual-ported random access memory (DPRAM) module used as 128 KB storage BlockRAM. Each memory port has an address bus, a data input bus, a data output bus, an individual clock signal, an individual memory enable signal, and an individual write enable signal. Both port A and B are 32 bits wide and 32768 words deep. To address the 32768 words, the 15 address lines were used for both port A and B.

```
Check:
Port A and B-> (2^15) address lines * 32 bits port size = 1,048,576 bits = 131,072
bytes = 128 Kilo Bytes

component mdr_test_dpram
port (
addra  : IN std_logic_VECTOR(14 downto 0);
addrb  : IN std_logic_VECTOR(14 downto 0);
clka   : IN std_logic;
clkb   : IN std_logic;
dina   : IN std_logic_VECTOR(31 downto 0);
dinb   : IN std_logic_VECTOR(31 downto 0);
douta  : OUT std_logic_VECTOR(31 downto 0);
doutb  : OUT std_logic_VECTOR(31 downto 0);
ena    : IN std_logic;
enb    : IN std_logic;
wea    : IN std_logic;
web    : IN std_logic);
end component;
```

- addrb, clkb, dinb, doutb, enb, and web. Port B signals that connect to the FPGA 4 KB buffer DPRAM and control logic module. The 32-bit dinb and doutb ports directly connect to the FPGA 4 KB buffer DPRAM data input and output buses. The enb, web, and addrb ports connect to the control logic module, which drives these signals. clkb is driven by the i_clkdv clock tree.
- addra, clka, dina, douta, ena, and wea. Port A signals not used in this reference design. These signals are reserved for connection to the MDR interface, which is covered in second part of this reference design.

4.5.1 Xilinx CORE Generator

To create or modify the DPRAM core using the Xilinx CORE Generator, select the core in the source window within the Xilinx Project Navigator. Next, in the Process window run **MANAGE CORES** within the Coregen process. When the Xilinx CORE Generator application launches, you can chose either to edit or create a new core. The design core window for the dual-port block memory is straightforward. On the first page, set the width of port A to

32 bits and the depth to 32768. Set the width of port B to 32 bits and the depth to 32768, the same as for Port A. Both Port A and port B options for configuration should be set to **READ AND WRITE** and for write mode should be set to **READ AND WRITE**.

On the second core design page for port A, select **ENABLE PIN**. The **HANDSHAKING PIN** and **REGISTER INPUT** features are not used. In the output register options, set the **ADDITIONAL OUTPUT PIPE STAGES** to 0. The **SINIT PIN** functionality is not used, so the initialization value (hexadecimal) can be left blank. The pin polarity options are **RISING EDGE TRIGGERED** for an active clock edge, **ACTIVE HIGH** for the enable pin, and **ACTIVE HIGH** for the write enable pin.

On the third core design page for port B, select the optional **ENABLE PIN**. The **HANDSHAKING PIN** and **REGISTER INPUT** features are not used. In the output register options, set the **ADDITIONAL OUTPUT PIPE STAGES** to 0. The **SINIT PIN** functionality is not used, so the initialization value (hexadecimal) can be left blank. The pin polarity options are **RISING EDGE TRIGGERED** for active clock edge, **ACTIVE HIGH** for the enable pin, and **ACTIVE HIGH** for the write enable pin.

On the final, fourth core design page within the Initial Contents, the global initialization value is set to 0, but it can be set to any value desired. You can preload the memory with the initialization hexadecimal file (.coe), which can be useful in debugging. If desired, check the **LOAD INIT FILE** check box and load the initialization hexadecimal file. The bottom of this page presents the summary of the designed DPRAM core. For our example, the information panel should display as follows:

```
Address Width A15
Address Width B           15
Blocks Used               58
Port A Read Pipeline Latency 1
Port B Read Pipeline Latency 1
```

If these values are correct, click **GENERATE**, and the new core is generated.

5 FPGA Memory Space

The FPGA 4 KB buffer DPRAM is memory-mapped via the MPC8260 UPM system bus interface, along with sixteen 32-bit registers. The MPC8260 can access the buffer space from the following address range:

```
START  BASE_ADDRESS + 0x0000
END    BASE_ADDRESS + 0x1000
```

In this reference design, the base address of 0x0300 0000 transfers into the 0x0300 0000–0x0300 1000 memory range reserved for accessing the FPGA 4 KB buffer space.The FPGA control registers are mapped in the following order:

```
BASE_ADDRESS + 0x0FFC     =     r_reg00_FPGA_CTRL
BASE_ADDRESS + 0x0FF8     =     r_reg01_FPGA_VER
BASE_ADDRESS + 0x0FF4     =     r_reg02_SDRAM_CTRL
BASE_ADDRESS + 0x0FF0     =     r_reg03_SDRAM_WORDS
BASE_ADDRESS + 0x0FEC     =     r_test_5
BASE_ADDRESS + 0x0FE8     =     r_test_6
BASE_ADDRESS + 0x0FE4     =     r_test_7
BASE_ADDRESS + 0x0FE0     =     r_test_8
BASE_ADDRESS + 0x0FDC     =     r_test_9
BASE_ADDRESS + 0x0FD8     =     r_test_10
BASE_ADDRESS + 0x0FD4     =     r_test_11
BASE_ADDRESS + 0x0FD0     =     r_test_12
BASE_ADDRESS + 0x0FCC     =     r_test_13
BASE_ADDRESS + 0x0FC8     =     r_test_14
BASE_ADDRESS + 0x0FC4     =     r_test_15
```

5.1 Digital Clock Management (DCM) Logic

DCM is a standard feature in the Xilinx Virtex-II™, Virtex-II Pro™, and Spartan III® devices. Different models of these devices have different numbers of clock managers. The main features are clock de-skewing, frequency synthesis, phase shifting, and duty cycle correction. Our design uses only clock de-skewing and duty cycle correction. To keep the reference design simple, only clock input, clock output, clock reset, and DCM lock signals are used. Other signals, such as status and additional clock outputs, can be used in a larger design. The lock signal and its associated logic should keep all FPGA logic in the reset state until the lock signal goes high to indicate that the DCM DLL is locked and the clock output signal is stable. Until the lock signal activates, the DCM output clocks are not valid and can exhibit glitches, spikes, or other unwanted spurious signals.

```
DLL_FREQUENCY_MODE : string := "LOW";
DUTY_CYCLE_CORRECTION : boolean := TRUE;

CLKDV_DIVIDE : string := "6.0";
STARTUP_WAIT : boolean := TRUE

attribute DLL_FREQUENCY_MODE : string;
attribute DUTY_CYCLE_CORRECTION : string;

attribute CLKDV_DIVIDE : string;
attribute STARTUP_WAIT : string;
attribute DLL_FREQUENCY_MODE of U_DCM: label is "LOW";
attribute DUTY_CYCLE_CORRECTION of U_DCM: label is "TRUE";

attribute CLKDV_DIVIDE of U_DCM2: label is "6.0";
attribute STARTUP_WAIT of U_DCM: label is "TRUE";
```

These parameters set up the DCM for low-frequency operation with the duty cycle correction enabled (50/50) and the start-up sequence wait enabled. If the `clkdv` output is used, it is divided by factor of 6.

```
component dcm_main is
Port (
      p_clock             : in std_logic;
      p_reset             : in std_logic;
      p_clk1x_out         : out std_logic;
      p_clkdv_out         : out std_logic;
      p_clk2x_out         : out std_logic;
      p_clk2x180_out      : out std_logic;
      p_clkfx_out         : out std_logic;
      p_dcm_lock_out      : out std_logic
);
end component;
```

- `p_clock` is a non-de-skewed input clock.
- `p_reset`. This signal is inverted because DCM is active high and the master FPGA switch on the board in this reference design is active low.
- `p_clk1x_out`, `p_clkdv_out`, `p_clk2x_out`, `p_clk2x180_out`, `p_clkfx_out`, and `p_dcm_lock_out`. These signals are routed to the DCM output ports for use in the top-level design.

```
-- DCM Instantiation
U_DCM2: DCM
port map
(
       CLKIN         =>     p_clock,
       CLKFB         =>     i_clk0,
       DSSEN         =>     gnd,
       PSINCDEC      =>     gnd,
       PSEN          =>     gnd,
       PSCLK         =>     gnd,
       RST           =>     p_reset,
       CLK0          =>     i_clk0_tmp,
       CLKDV         =>     i_clkdv_tmp,
       CLKFX         =>     i_clkfx_tmp,

       CLK2X         =>     i_clk2x_tmp,
       CLK2X180      =>     i_clk2x180_tmp,
       LOCKED        =>     i_lock
);
```

Notice that the `i_clk0` signal is routed both to `CLKFB` and to the rest of the FPGA system bus logic. This signal is an output of the global clock buffer, `BUFG`, and must be routed to the `CLKFB` port on the DCM component for proper operation of the feedback circuit. In the following BUFG instantiation, `i_clk0_tmp` is a de-skewed output DCM signal serving as an input to the `BUFG`. The output of the `BUFG` is the clock signal to drive the rest of the FPGA logic. The DCM logic uses five `BUFG` global clock buffers. The `BUFG` instantiation for the `i_clk0` DCM clock output is presented as follows. The BUFG instantiation for `i_clk0`, `i_clk2x`, `i_clk2x180`, and `i_clkfx` are the same as for `i_clk0`.

```
-- BUFG Instantiation
U22_BUFG: BUFG
port map
(
       I      => i_clkdv_tmp,
       O      => i_clkdv
);
```

In this example, only clock output that is de-skewed and divided by 6 is used (`i_clkdv`). The following de-skewed clock output options can be used as needed:

- `i_clk1x` = A frequency equal to one of the input clocks.
- `i_clk2x` = A frequency multiplied by two of the input clocks.
- `i_clkfx` = A frequency multiplied by factor of fx of the input clock.
- `i_clk2x180` = A frequency multiplied by two of the input clocks with a phase shift of 180 degrees.

5.2 JTAG Logic

The JTAG logic provides flexible control of the JTAG chain, which is composed of six JTAG-enabled devices: MRC6011A, MRC6011B, MRC6011C, DSPA, DSPB, and DSPC. You can create a custom JTAG chain of any or all six devices. The signals in the JTAG logic are as follows:

```
p_jtag_conndsp_tdi         : in std_logic;
p_jtag_conndsp_tdo         : out std_logic;
p_jtag_conndsp_commun      : in std_logic_vector (0 to 3);
p_jtag_connmrc6011_tdi     : in std_logic;
p_jtag_connmrc6011_tdo     : out std_logic;
p_jtag_connmrc6011_commun  : in std_logic_vector (0 to 3);
```

- `p_jtag_conndsp_tdi` and `p_jtag_conndsp_tdo`. Common JTAG TDI and TDO signals connected to the DSP JTAG connector.
- `p_jtag_conndsp_commun`. A four-signal bus connected to the DSP JTAG connector, providing common JTAG signals.
- `p_jtag_connmrc6011_tdi` and `p_jtag_connmrc6011_td0`. Common JTAG TDI and TDO signals connected to the MRC6011 JTAG connector.
- `p_jtag_connmrc6011_commun`. A four-signal bus connected to the MRC6011 JTAG connector, providing common JTAG signals.

```
p_jtag_dsp_a_tdi           : out std_logic;
p_jtag_dsp_a_tdo           : in std_logic;
p_jtag_dsp_a_commun        : out std_logic_vector (0 to 3);
p_jtag_dsp_b_tdi           : out std_logic;
p_jtag_dsp_b_tdo           : in std_logic;
p_jtag_dsp_b_commun        : out std_logic_vector (0 to 3);
p_jtag_dsp_c_tdi           : out std_logic;
p_jtag_dsp_c_tdo           : in std_logic;
p_jtag_dsp_c_commun        : out std_logic_vector (0 to 3);
```

- `p_jtag_dsp_a_tdi` and `p_jtag_dsp_a_tdo`. Common JTAG TDI and TDO signals directly connected to DSP A.
- `p_jtag_dsp_a_commun`. A four-bit bus directly connected to DSP A, providing common JTAG signals.
- `p_jtag_dsp_b_tdi` and `p_jtag_dsp_b_tdo`. Common JTAG TDI and TDO signals directly connected to DSP B.
- `p_jtag_dsp_b_commun`. A four-bit bus directly connected to DSP B, providing common JTAG signals.
- `p_jtag_dsp_c_tdi` and `p_jtag_dsp_c_tdo`. Common JTAG TDI and TDO signals directly connected to DSP C.
- `p_jtag_dsp_c_commun`. A four-bit bus connected directly to DSP C, providing common JTAG signals.

```
p_jtag_mrc6011_a_tdi       : out std_logic;
p_jtag_mrc6011_a_tdo       : in std_logic;
p_jtag_mrc6011_a_commun    : out std_logic_vector (0 to 3);
p_jtag_mrc6011_b_tdi       : out std_logic;
p_jtag_mrc6011_b_tdo       : in std_logic;
```

```
p_jtag_mrc6011_b_commun    : out std_logic_vector (0 to 3);
p_jtag_mrc6011_c_tdi       : out std_logic;
p_jtag_mrc6011_c_tdo       : in std_logic;
p_jtag_mrc6011_c_commun    : out std_logic_vector (0 to 3);
```

- p_jtag_mrc6011_a_tdi and p_jtag_mrc6011_a_tdo. Common JTAG TDI and TDO signals directly connected to MRC6011 A.
- p_jtag_mrc6011_a_commun. A 4-bit bus directly connected to MRC6011 A, providing common JTAG signals.
- p_jtag_mrc6011_b_tdi and p_jtag_mrc6011_b_tdo. Common JTAG TDI and TDO signals directly connected to MRC6011 B.
- p_jtag_mrc6011_b_commun. A 4-bit bus connected directly to MRC6011 B, providing common JTAG signals.
- p_jtag_mrc6011_c_tdi and p_jtag_mrc6011_c_tdo. Common JTAG TDI and TDO signals directly connected to MRC6011 C.
- p_jtag_mrc6011_c_commun. A 4-bit bus directly connected to MRC6011 C, providing common JTAG signals.

```
p_jtag_dipswdsp            : in std_logic_vector (0 to 2);
p_jtag_dipswmrc6011        : in std_logic_vector (0 to 2);
p_jtag_reg                 : in std_logic_vector (0 to 5)
```

- p_jtag_dipswdsp. A 3-bit bus connected to the DSP JTAG configuration switch (SW6).
- p_jtag_dipswmrc6011. A 3-bit bus connected to the MRC6011 JTAG configuration switch (SW6).
- p_jtag_reg. A 6-bit port connected to the 6-bit register to store the JTAG configuration settings.

The first three switches configure the DSP JTAG chain. Each switch corresponds to a DSP JTAG-enabled device. If a switch is ON, the corresponding DSP is part of the DSP JTAG chain. Otherwise, it is excluded from the DSP JTAG chain (signals are tri-stated). If these three switches are OFF, the FPGA takes the DSP JTAG chain configuration from its dedicated register.[1] The ROBIN DSP daughter card JTAG connector is P6. By default, SW6[1–3] are all OFF so that the DSP JTAG chain configuration is the one present in the FPGA. The DSP JTAG register after reset does not include any of the DSPs as part of the JTAG chain.

SW6[4–6] configure the MRC6011 JTAG chain. Each switch corresponds to an MRC6011 device. If a switch is ON, the corresponding MRC6011 is part of the MRC6011 JTAG chain. Otherwise, it is excluded from the MRC6011 JTAG chain (signals are tri-stated). If these three switches are OFF, the FPGA takes the MRC6011 JTAG chain configuration from its dedicated register (read/write accessible by the MPC8260 via its system bus). The MRC6011 JTAG connector is P7. By default, SW6[4–6] are all OFF so that the DSP JTAG chain configuration is the one present in the FPGA The MRC6011 JTAG register after reset does not include any MRC6011 device as part of the JTAG chain.

1. For simplicity, this option is not implemented but is available as a further expansion.

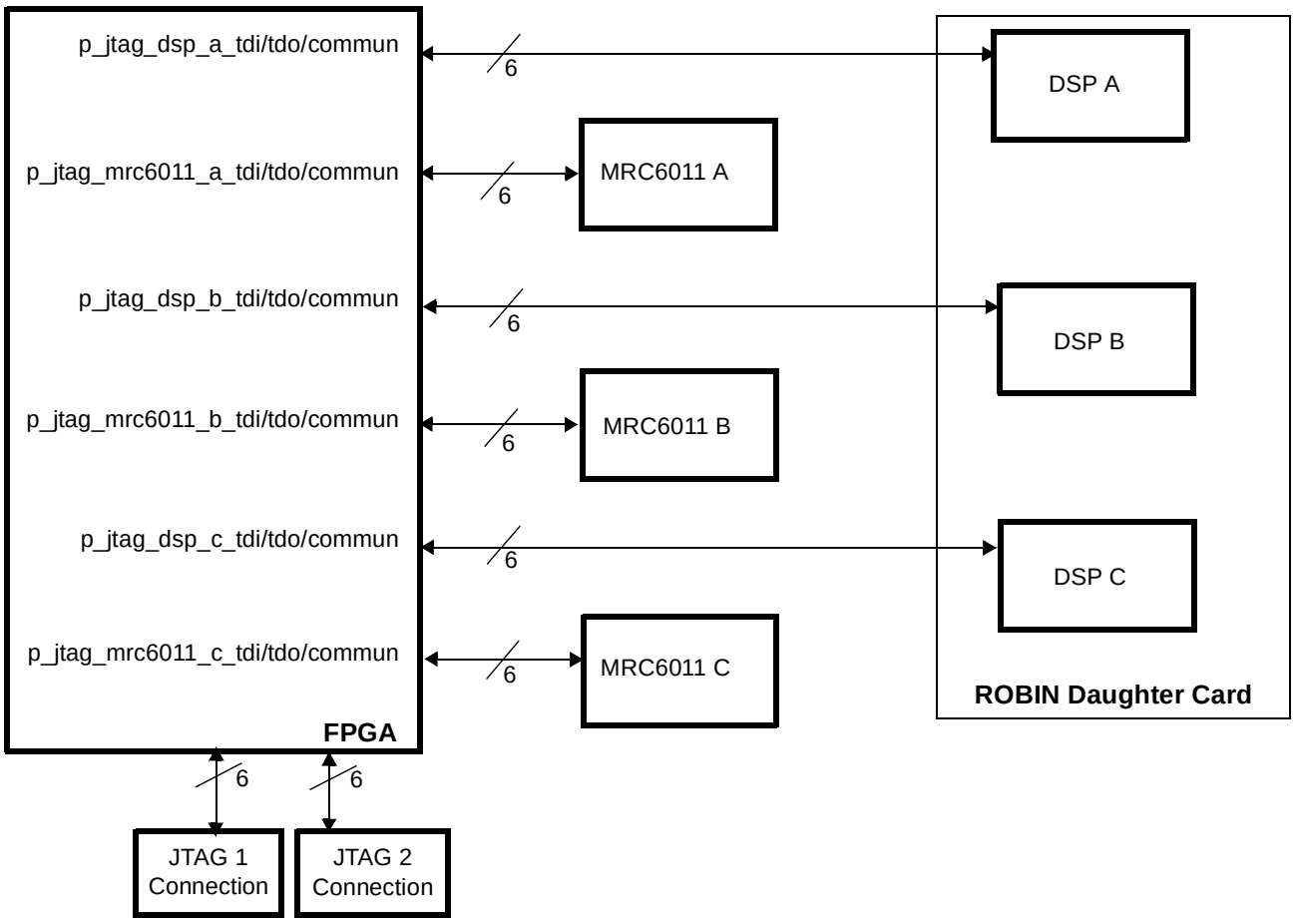

Figure 7. JTAG Reconfigurable Chain

5.3 MDR Logic

The MDR logic provides an antenna data interface to the FPGA for a direct connection to the MRC6011 MDR bus. From the FPGA perspective, the MDR bus is an output-only bus. The ports of the MDR logic are as follows:

```
p_clock                    : in std_logic;
p_reset                    : in std_logic;
p_MRC6011_data_valid1c     : out std_logic;
p_MRC6011_data_valid2c     : out std_logic;
p_MRC6011_fsync1c          : out std_logic;
p_MRC6011_fsync2c          : out std_logic;
p_MRC6011c_releaseibrst    : out std_logic;
p_pq2_ctrl_simulation_on_in : in std_logic;
p_sdram_en_out             : out std_logic
```

- `p_clock`. The main clock source for the MDR logic driven by the `i_clkdv_out` clock tree.
- `p_reset`. The main reset for the MDR logic, driven by the `i_module_reset` signal.
- `p_MRC6011_data_valid1c` and `p_MRC6011_data_valid2c`. The output ports connected to the `i_MRC6011_data_valid1c` and `i_MRC6011_data_valid2c` signals, respectively. These signals carry the data valid signals to the MRC6011 MDR interface.

- p_MRC6011_fsync1c and p_MRC6011_fsync2c. Output ports connected to the i_MRC6011_fsync1c and p_MRC6011_fsync2c signals, respectively. These signals carry the frame sync signals to the MRC6011 MDR interface.
- p_MRC6011c_releaseibrst. This output port connects to the i_MRC6011c_releaseibrst signal.
- p_pq2_ctrl_simulation_on_in. This input port connects to the i_pq2_ctrl_simulation_on signal, which is driven by the main control block. This signal notifies the MDR control state machine to begin MDR data streaming. When the simulation procedure is initialized, the p_sdram_en_out output port enables port A of the 128 KB storage BlockRAM memory.

```
signal i_MRC6011_mdr_data_valid        : std_logic;
signal i_MRC6011_mdr_fsync             : std_logic;
signal i_MRC6011_mdr_releaseibrst      : std_logic;
signal i_sdram_en_out                  : std_logic;
```

- The MDR state machine uses these signals to drive the various MDR bus control signals, such as the data valid strobe, frame sync, ib reset release, and SDRAM memory enable.

```
type mdr_state_type is (Sinit, S0, S1, S2a, S2b, S2c, S2d, S2e, S2f, S2g, S2h, S3a,
S3b, S3c, S3d, S3e, S3f, S3g, S3h, S4aa, S4a, S4b, S4c, S4d, S4e, S4f, S4g, S4h, S5);
signal mdr_ctrl           : mdr_state_type;
```

- The MDR state machine states. States can be added or subtracted according to the needs of the application. Although the MDR state machine has many states, its basic operation is simple:

1. The MDR state machine begins in the Sinit state, in which it waits for the p_pq2_ctrl_simulation_on_in signal to transition to the high state. This transition triggers the beginning of the MDR bus initialization and later data transfer.
2. In the S0 state, the FSM transitions to the next two states (S1 and S2a) in which the i_MRC6011_mdr_releaseibrst signal is asserted high for the MDR bus initialization. In the S2a state, the i_MRC6011_mdr_fsync signal is asserted high for only one clock cycle to create a one-cycle frame sync strobe necessary for proper MDR operation.
3. Each of the next seven states (S2b, S2c, S2d, S2e, S2f, S2g, and S2h) have a duration of only one clock cycle and are used to count eight clock cycles between frame sycn pulses, which is necessary for proper set-up of the MDR interface in the MRC6011 device.
4. The S3a state increments on the next clock cycle, and the next frame sync pulse is asserted (i_MRC6011_mdr_fsync).
5. The next seven states are incremented (S3b, S3c, S3d, S3e, S3f, S3g and S3h).
6. In the S4aa state, the i_sdram_en_out signal is asserted to enable port A of the 128 KB storage BlockRAM memory. This signal must be enabled one clock cycle before the actual clock cycle desired for the data to appear on the MDR bus due to the one clock cycle read delay out of the FPGA Block-RAM DPRAM.
7. In the 4a state, i_MRC6011_mdr_fsync is asserted, along with the i_MRC6011_mdr_data_valid and i_sdram_en_out signals.
8. The next seven states are incremented (S4b, S4c, S4d, S4e, S4f, S4g, and S4h), and the i_MRC6011_mdr_data_valid and i_sdram_en_out signals are asserted. During this period, the BlockRAM DPRAM and its memory controller are instructed to read the data out of the memory and send it to the MDR bus.

9. After the S4h state, the state machine has two options, depending the status of the `p_pq2_ctrl_simulation_on_in` signal. If this signal is held high (asserted), the state machine is forwarded to the S4a state. Since the S4a state is the first state of the MDR data streaming sequence, the whole process repeats for next eight 32-bit words until the S4h decision state is reached again. If the `p_pq2_ctrl_simulation_on_in` signal is low, indicating that the simulation is complete, the state increments to S5 at the next clock cycle to the Sinit state. At this point, the MDR interface is in the initial state, awaiting the simulation on signal.

Note that following signals are driven during specific states of the MDR control state machine.

- `i_mrc6011_mdr_releaseibrst` is asserted during the S1 and S2a states.
- `i_mrc6011_mdr_fsync` is asserted during the S2a, S3a and S4a states.
- `i_mrc6011_mdr_data_valid` is asserted during the S4a, S4b, S4c, S4d, S4e, S4f, S4g, and S4h states.
- `i_sdram_en_out` is asserted during the S4aa, S4a, S4b, S4c, S4d, S4e, S4f, S4g and S4h states.
- `p_sdram_en_out` is driven by the `i_sdram_en_out` signal.
- `p_mrc6011_data_valid2c` is driven by the `i_mrc6011_mdr_data_valid` signal.
- `p_mrc6011_data_valid1c` is driven by the `i_mrc6011_mdr_data_valid` signal.
- `p_mrc6011_fsync2c` is driven by the `i_mrc6011_mdr_fsync` signal.
- `p_mrc6011_fsync1c` is driven by the `i_mrc6011_mdr_fsync` signal.
- `p_mrc6011c_releaseibrst` is driven by the `i_mrc6011_mdr_releaseibrst` signal.

5.4 UCF file

The `top_vhdl.ucf` file contains all FPGA pin assignments and timing constraints. Timing constraints are located at the end of the UCF file and should be added as needed, keeping the number of constraints to a minimum. For this reference design, the following timing directives were in effect:

```
NET "fpga_clock" TNM_NET = "fpga_clock";
TIMESPEC "TS_fpga_clock" = PERIOD "fpga_clock" 15151 ps HIGH 50 %;
```

For a larger design or a design that operates at higher frequencies, additional timing constraints might be necessary. For the ROBIN motherboard design, it was beneficial to pull up following JTAG signals:

```
NET "fpga_jtag_conndsp_commun<3>" PULLUP;
NET "fpga_jtag_connod1_commun<3>" PULLUP;
```

If the MRC6011 device is not present in the design, these lines can be omitted from the `.ucf` file.

5.5 Registering External Signals at the FPGA IOBs

Registering the external signals at the input/output blocks (IOBs) is optional, but it can be helpful in certain high-speed designs in which the FPGA has difficulty meeting the timing constraints. The disadvantage of this approach is that the bus or interface efficiency decreases. In designs that have problems with control and address signal timing, the following process registers the signals at the FPGA IOBs. This process introduces an additional pipeline stage but solves most timing problems. Use it as necessary, depending on the bus speed, FPGA type, and board design.

```
REG_AT_IOBS_PROC : process (i_clkdv)
begin
       if i_clkdv='1' and i_clkdv'event then
              i_pq2_rwb           <= fpga_pq2_rwb;
              i_pq2_csb           <= fpga_pq2_csb;
              i_pq2_addr          <= fpga_pq2_addr;
       end if;
end process;

REG_AT_IOBS_PROC2 : process (i_clkdv)
begin
       if i_clkdv='1' and i_clkdv'event then
              fpga_mrc6011c_releaseibrst<= i_mrc6011c_releaseibrst;
              fpga_mrc6011_fsync1c      <= i_mrc6011_fsync1c;
              fpga_mrc6011_fsync2c      <= i_mrc6011_fsync2c;
              fpga_mrc6011_data_valid1c <= i_mrc6011_data_valid1c;
              fpga_mrc6011_data_valid2c <= i_mrc6011_data_valid2c;

       end if;
end process;
```

If the additional pipeline stage lowers the bus efficiency below the required level and the design meets timing constraints with this process but fails without it, try changing the synthesis options. Changing the synthesis optimization goal from **AREA** to **SPEED** and/or changing the synthesis optimization effort level from **NORMAL** to **HIGH** can help meet the timing requirements. In addition, disabling the **KEEP HIERARCHY** option (flattening the design) can improve overall timing results. The synthesis options **REGISTER DUPLICATION**, **REGISTER BALANCING**, and **MAX FANOUT** can also improve the timing in a design, but they should be tried one at a time because each has the potential to yield even worse timing results.

If the timing improves but still does not meet requirements, try changing the map properties. Change the **OPTIMIZATION STRATEGY** from **AREA** to **SPEED**. The **ALLOW LOGIC OPTIMIZATION ACROSS HIERARCHY** option also can be beneficial. Changing the place and route **EFFORT LEVEL** from **STANDARD** to **MEDIUM** or **HIGH** along with increasing the **EXTRA EFFORT** level usually produces better timing results but increases the place and route build time. At the end of the process, changing the **STARTING PLACER COST TABLE** and experimenting with **MULTI PASS PLACE AND ROUTE PROPERTIES**, specifically: **PLACE * ROUTE EFFORT LEVEL (OVERALL)**, **EXTRA EFFORT**, **NUMBER OF PAR ITERATIONS** and **NUMBER OF RESULTS TO SAVE** significantly increases the compile time but yields better design timing. Depending on the size of your FPGA design and the number of constraints, some synthesis, map, and place and route options consume too much time to be practical, but others do not. Experimenting with a combination of parameters may be necessary to reach the optimal timing for an FPGA design. In some marginal cases, increasing the I/O pin driving strength and/or I/O pin skew rate can be effective.

If the code still does not achieve the timing goal, the problem may lie in the FPGA external pin placement on the board. Incorrect FPGA I/O pin placement can cause long FPGA internal signal delays because of the great distance these signals must travel within the FPGA.

6 FPGA Operating Sequence

This section discusses the operating sequence and interactions between the FPGA and MPC8260. Because of the default setting of the FPGA_CTRL register reset bit, the MPC8260 is required to take the FPGA out of the reset state by clearing the reset bit. This can be changed by entering the new default value for the FPGA_CTRL register within the PowerQUICC II control logic module.

The first operation for this reference design is loading (programming) the MPC8260 and FPGA. We use used the CodeWarrior® PowerQUICC II IDE to program and develop the MPC8260. For the FPGA, the developer has two options for downloading the FPGA bit image.

- Download the bit image to the FPGA via the MPC8260 using the FPGAPROG() function in the `FPGA_PROG.c` file. First un-comment the FPGAPROG() function in the `main.c` file. This function requires the FPGA image to be in hexadecimal format. For converting the regular FPGA `*.bit` file into the `*.hex` file format, use the utilities in the following project directory:
  `PQ2_debug_SDRAM_LOAD_UNLOAD\PQ2_debug_SDRAM_LOAD_UNLOAD\Src\FPGABoot\Conversion`
  This directory contains the source code of the `stream4c.exe` utility, along with the utility and batch file. Use the `conver_fpga_bit_file.bat` file to run this utility. You must edit the file so that `stream4c.exe name_of_your_FPGA_bit_file.bit fpga_image.hex` `name_of_your_FPGA_bit_file.bit` is replaced with the name of the file you wish to convert to `*.hex` format. Keep the `fpga_image.hex` the same.
- Use the iMPACT FPGA programming tool to download the FPGA image via JTAG. The Parallel-IV JTAG cable the TCK, TDI, TDO, TMS, V_{CC}, and V_{DD} flying leads must be connected to the P5 ROBIN motherboard FPGA JTAG connector. For the P5 pin assignments, refer to the chapter on FPGA JTAG in the *ROBIN Motherboard Reference Manual*. Remember to change the Generate Programming File options in the processes window within the Xilinx ISE Project Navigator. The FPGA Start-Up Clock property should be changed to JTAG clock.

For development or testing, the direct FPGA download via iMPACT and JTAG is the faster method, but it requires FPGA JTAG cable and the iMPACT programming tool. Regardless of the option you select, after the FPGA is programmed, it remains in the reset state until the MPC8260 deasserts the FPGA reset. When the FPGA reset is de-asserted, the MPC8260 can write to or read from the FPGA.

6.1 MPC8260 Programming Sequence

The `FpgaMain.c` file contains the FPGA handling code. In the example discussed here, the FPGAtestPQ2() function is used to interact with the FPGA. The first goal is to load the data into the FPGA from the MPC8260.

1. Initialize the variables to be used later on in the code, and clear all MPC8260 variables, memory, and FPGA-mapped memory space with the default value of 0xF1F2F3F4.

```
for(uliC=0; uliC<64; uliC++)
{
        r[uliC]       = 0xF1F2F3F4;
}

puliFPGAPtr = (UWord32 *) (FPGA_BASE);
usiCountWrites = 0;
while(usiCountWrites != 0x200)
{
        usiCountWrites++;
        *puliFPGAPtr++ = (unsigned long int)0xF1F2F3F4;
}

puliStorePtr = &gauliTestDataRead[0][0];
for(uliJ=0; uliJ<0x404; uliJ++)
{
        *puliStorePtr++ = (unsigned long int)0xF1F2F3F4;
}
```

2. Take the FPGA of reset by clearing the reset bit, FPGA_CTRL[R] (see page 18).

```
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x80000000;
*puliFPGARegPtr = 0x00000000;
```

3. Load the value of the FPGA_VER register into the r[0] register to keep track of which FPGA image version is currently loaded (see page 19).

```
puliFPGARegPtr = (UWord32 *) (0x03000ff8);
r[0] = *puliFPGARegPtr;
```

4. Begin the FPGA SDRAM loading sequence by writing the 0x1 value into the FPGA_CTRL register. The FPGA is ready and awaiting for the data from the MPC8260.

```
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x00000001;
```

5. Initialize the for loop to run NO_OF_4KB times. The while loop performs the 512 FPGA writes (or 0x200 in hexadecimal).

```
uliCount = 0;
usiTotalCountWrites = 0;
for(uliI = 1; uliI<( NO_OF_4KB+1 ); uliI++)
{z

      uliCount++;
      usiCountWrites = 0;
      puliFPGAPtr = (UWord32 *) (FPGA_BASE);
      puliPatternPtr = &gauliTestData[0];
      while(usiCountWrites != 0x200)
      {
            usiCountWrites++;
            *puliFPGAPtr++ = (unsigned long int)usiTotalCountWrites;
            usiTotalCountWrites++;
      }
```

6. After the loop completes, the MPC8260 sets the *SDRAM Block of 512 32 bit words load done* bit in the SDRAM_CTRL register. This operation triggers the FPGA main controller block to transfer data from the 4 KB buffer DPRAM into the storage BlockRAM.

```
puliFPGARegPtr = (UWord32 *) (0x03000ff4);
*puliFPGARegPtr = 0x00000001;
```

7. The following code is for debug purposes only. The last two lines load the next value in the *r* array with the number of successful 32-bit word transactions between the 4 KB buffer DPRAM and storage BlockRAM.

```
uliTemp = 100000;
while(uliTemp)
{
      uliTemp--;
}
puliFPGARegPtr  = (UWord32 *) (0x03000ff0);
r[uliI] = *puliFPGARegPtr ;
```

8. If the code has processed the specified number of 4 KB data blocks, the for loop ends. The MPC8260 next clears the load SDRAM bit (FPGA_CTRL[LD]), thus informing the FPGA main control module that the data loading sequence is complete.

```
}
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x00000000;
```

9. Set the unload SDRAM bit (FPGA_CTRL[UNL]), causing the FPGA main control state machine to enter the SDRAM unloading sequence.

```
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x00000002;
```

10. Initialize the for loop to run NO_OF_4KB times. Two lines of code read the number of words transferred between 4 KB buffer DPRAM and 128 KB FPGA internal storage BlockRAM memory. These two lines are for debug purposes only.

```
uliCount = 0;
puliStorePtr = &gauliTestDataRead[0][0];
for(uliI = 1; uliI<( NO_OF_4KB+1 ); uliI++)
{
      puliFPGARegPtr  = (UWord32 *) (0x03000ff0);
      r[uliI] = *puliFPGARegPtr;
```

11. Set the SDRAM_CTRL[ULB] bit, triggering the FPGA internal memory controller to begin transferring a 4 KB block of data from the 128 KB storage BlockRAM into the 4 KB buffer DPRAM.

```
puliFPGARegPtr = (UWord32 *) (0x03000ff4);
*puliFPGARegPtr = 0x00000002;
```

12. The following code is for debug purposes only. The last two lines load the next value into the *r* array with the number of successful 32-bit word transactions between the 4 KB buffer DPRAM and storage BlockRAM.

```
uliTemp = 100000;
while(uliTemp)
{
      uliTemp--;
}
puliFPGARegPtr  = (UWord32 *) (0x03000ff0);
r[uliI] = *puliFPGARegPtr;
```

13. Read the complete block of 512 words from the FPGA 4 KB memory-mapped buffer, and store the read data into the MPC8260 internal memory (*puliStorePtr).

```
uliCount++;
puliFPGAPtr = (UWord32 *) (FPGA_BASE);
for(uliJ=0; uliJ<0x200; uliJ++)
{
      *puliStorePtr++ = (unsigned long int)*puliFPGAPtr++;
}
```

14. After the main loop completes, clear the unload SDRAM bit, FPGA_CTRL[UNL], to inform the FPGA control state machine that the SDRAM unload sequence is complete.

```
}
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x00000000;
```

15. Run a function check to verify that the data written into the FPGA is the same data as read from the FPGA.

```
Check(&gauliTestDataRead[0][0]);
```

16. Writes a value of 0x4 to the FPGA_CTRL register to notify the FPGA control state machine that Simulation mode can begin. The FPGA control state machine prepares the FPGA internal memory controllers to transfer data out of the 28 KB storage BlockRAM and into the MDR interface of MRC6011 device. The data streams until the Simulation mode is active.

```
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x00000004;
```

17. Clear the FPGA_CTRL register to notify the FPGA control state machine that Simulation mode is over. This operation disables the MDR interface and halts the data transfer into the MRC6011 MDR port. At this point, the FPGA is ready to repeat the whole process.

```
puliFPGARegPtr = (UWord32 *) (0x03000ffc);
*puliFPGARegPtr = 0x00000000;
```

7 ROBIN Motherboard Configuration

Table 7 shows the configuration of all ROBIN motherboard jumpers and switches necessary for the reference design discussed in this application note to work properly. For details on the functionality and location of the jumpers and switches, refer to the *ROBIN Motherboard Reference Manual*.

Table 7. Motherboard Configuration for the FPGA System Bus Interface

Switch/Jumper	Value
SW5	All ON
SW7	All OFF, 6 ON
SW9, SW14	All ON
SW10	1 and 3 OFF, 2 and 4 ON
SW8, SW13	9 and 10 OFF, 1–8 ON
SW11, SW16, SW21	1 ON, 2 OFF, 3 ON, 4 OFF, 5 and 6 ON, 7 OFF, 8–10 ON
SW10, SW15, SW18, SW20	All OFF
SW12, SW17, SW22	1 OFF, 2 and 3 ON, 4 and 5 OFF, 6–8 ON
JP8	1 and 2 connected
JP2, JP9, JP10	Not connected
JP3, JP4, JP5, JP7, JP11, JP12, JP13, JP14, JP15, JP16, JP17, JP18	Closed

8 Integrated Software Environment (ISE) Tool Reports

This section briefly discusses the tool reports for physical synthesis, mapping the logical design to the FPGA, and assessing the performance of various components to prevent/correct timing problems.

8.1 Synthesis Report

The beginning of the synthesis report displays the options summary so that you can review the options for synthesis. The HDL compilation, HDL analysis, and HDL synthesis sections contain the warnings and error messages pertaining to the design. The next section reports on macro usage, which, in our case, should display: 82 total registers (sixty-four 1-bit registers, three 10-bit registers, and fifteen 32-bit registers), seven multiplexers (six 2-to-1 multiplexers and one 32-bit 16-to-1 multiplexed), 21 tri-states (fourteen 1-bit tri-state buffers, one 32-bit tri-state buffer, and six 4-bit tri-state buffers) and three 10-bit adders.

The device use summary for the 2v3000b957-4 FPGA model used in this design should read as follows:

```
Number of Slices:                823    out of  14336       5%
Number of Slice Flip Flops:      645    out of  28672       2%
Number of 4 input LUTs:          1136   out of  28672       3%
Number of bonded IOBs:           218    out of    684       31%
Number of TBUFs:                 6      out of   7168       0%
Number of BRAMs:                 60     out of     96       62%
Number of GCLKs:                 6      out of     16       37%
Number of DCMs:                  1      out of     12       8%
```

Next, the timing summary is displayed. This is only a synthesis estimate, but it can prove very useful in predicting the performance of the map and place and route, without the need to run them.

8.2 Map Report

The map process maps the logical design to a Xilinx FPGA. The input to this process is an NGD file, which contains a logical description of the design in terms of its logical components and lower-level Xilinx primitives. The map process begins with a logical design rule check (DRC) on the design in the NGD file. If the design does not contain any rule violations, the design logic is mapped to the FPGA components of the target FPGA. The output map process is the NCD file, which is used for the place and route process.

The first two sections of the map report are reserved for errors and warnings. The rest of the report is not visible if there are any design rule check errors. Section 3 and 4 of the map report provide information on any removed (optimized) logic. In some cases, the map tool can eliminate the unused logic or FPGA resource, which can produce unwanted effects. Section 6 displays the IOB properties of external FPGA pin declarations. Notice that the `fpga_pq2_data[0-63]` pins are declared as bidirectional.

8.3 Place and Route (PAR) Report

If there are timing problems, the PAR report is the most frequently visited report. As long as the design is successfully mapped to the FPGA and the NCD file is successfully generated, the place and route process is performed. The beginning of the report yields a very helpful device usage summary report. Our design uses a total of 148 out of 684 external IOBs. After the UCF file is created, the number of LOCed external IOBs should read as 148 out of 148. Our design uses 60 out of 96 RAMB16s and 620 out of 14336 slices. It uses 3 out of 16 BUFGMUXs and 1 DCM. The next section of the PAR report indicates the phases and how many iterations the PAR performed, which depends on the PAR parameter settings.

The last section of the PAR report is the generating clock report, which contains a very useful clock summary. The fanout for each clock tree used in the design and the associated net skew and maximum delay are some of the most important parameters. This section summarizes any timing violations and timing constraints. If your design does not meet your timing requirements, this section displays the tree in which the violations occurred and provides measurements of the violations.

8.4 Bitgen Report

The bitgen report is generated after the binary FPGA programming file is generated. If there are problems with the pin constraints or incorrect FPGA resource mapping, the binary FPGA programming file and the report are not generated. The bitgen report starts with a summary of the Bitgen options. Some of the important options for this reference design are: `StartupClk` is set to `Cclk`. The following pins are pulled up: `CclkPin`, `DonePin`, `HswapenPin`, `M0Pin`, `M1Pin`, `M2Pin`, `PowerdownPin`, `ProgPin`, `TckPin`, `TdiPin`, `TdoPin`, and `TmsPin`. The next section in the report is generated after the design rule check is performed. Any errors or warning are presented in this section.

9 VHDL Code Listing

All the FPGA VHDL code is located in the zip file that accompanies this application note, `AN2889SW.zip`. Inside this zip file are the following VHDL files:

- `top_vhdl`.The top-level architecture of the complete FPGA design.
- `top_vhdl.ucf`. All FPGA constraints, including FPGA pin assignments and timing constraints.
- `ctrl.vhdl`, `ctrl_mem_cnt.vhd`, and `ctrl_mem_cnt2.vhd`. The control block logic code.
- `dcm_main.vhdl`. The DCM code to generate all FPGA clock tree sources.
- `pq2.vhd` and `pq2_data_mux.vhdl`. All the PowerQUICC II logic block logic and the register and memory-mapped space.
- `sdram.vhdl`. The 128 KB FPGA internal storage BlockRAM code.
- The companion zip file contains the complete Xilinx ISE 6.1.3 project with different versions of the Xilinx ISE tools to rebuild the project and import source files.
- `jtag.vhdl`. The JTAG module logic.
- `mdr.vhdl`. The MRC6011 MDR interface logic.
- The companion zip file contains the complete Xilinx ISE 6.1.3 project. With the different version of the Xilinx ISE tools the project might need to be rebuilt and source files imported.

10 MPC8260 Code Listing

All MPC8260 source code for this project is located in the SRC directory, in the following subdirectories:

- Board
- DSIboot
- FPGAboot
- FPGAcomm, includes and Main.
- The revalent directories to the FPGA reference design are FPGAboot, FPGAcomm, and Main.

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2890
Rev. 0
12/2005