XAPP496 (v1.0) June 3, 2010

# Creating Wider Memory Interfaces Using Multiple Spartan-6 FPGA Memory Controller Blocks

Author: Derek Curd


## Summary

The Memory Controller Block (MCB) is a dedicated embedded multi-port memory controller that greatly simplifies the task of interfacing Spartan-6 devices to DDR3, DDR2, DDR, and LPDDR memories. Spartan®-6 devices contain two to four MCBs, each of which can implement a single component interface to a 4-bit, 8-bit, or 16-bit memory. Some applications with higher memory bandwidth or density requirements benefit from using memory interfaces wider than the 16-bits offered by a single MCB. This application note describes how to merge the operation of two or more MCBs to implement effective 32-bit or wider memory interfaces. Both MCBs must be in a single-port configuration mode. This application note and reference design does not support merging MCBs configured in the multi-port configuration mode. The associated reference design has been verified in hardware, and analyzed for both performance and device utilization.


## Introduction

The MCB addresses the memory interface needs for the majority of Spartan-6 FPGA applications by providing an interface to the most common, lowest-cost, and lowest-power SDRAM memory standards. The MCB eliminates the complexities of communicating with these memory devices, and presents a single-data rata (SDR) user interface to the rest of the FPGA user logic. As shown in Figure 1, the MCB is a multi-port memory controller with up to six available ports. Each port consists of a command interface and a read and/or write data interface.

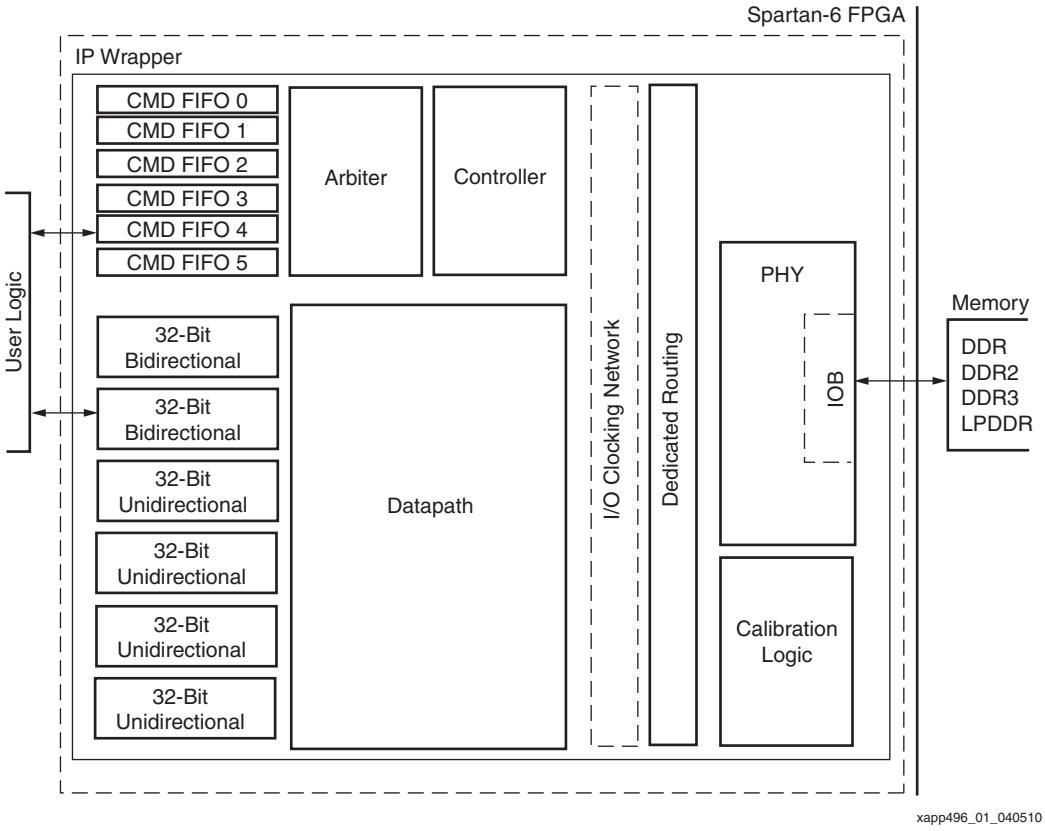


*Figure 1:* **MCB Block Diagram**

The two 32-bit bidirectional and four 32-bit unidirectional native ports of the MCB can be combined in five possible port configurations to create user interfaces of different widths. For example, the native ports can be combined to create four 32-bit bidirectional ports, two 64-bit bidirectional ports, or a single 128-bit bidirectional port. When used in a multi-port configuration, the arbiter inside the MCB determines which port currently has access to the external memory device.

The Memory Interface Generator (MIG) tool is used to create an external memory interface based on the MCB. The MIG tool is a GUI wizard launched from within the CORE Generator™ application and guides the user through a series of steps to configure the MCB. The MIG tool automatically generates the necessary RTL code, user constraints files (UCF), and script files for simulation and implementation of an MCB memory interface.

The MCB supports memory densities up to 4 Gb and data rates up to 800 Mb/s (DDR2, DDR3 SDRAM), providing up to 12.8 Gb/s of peak bandwidth when interfacing to the widest supported 16-bit single-component memory device. This provides more than enough memory density and bandwidth for the majority of Spartan-6 FPGA applications.

Some systems require memory interface configurations that cannot be supported by a single MCB. This application note explains how to take the RTL code produced by the MIG tool for two (or more) separate MCB single-component interfaces, and merge them together to create an effective wider interface for greater memory density and bandwidth.

## Design Overview

The Reference Design discussed in this application note only supports merging the operation of MCBs that are each configured in a single-port mode, meaning that no arbitration is occurring within each of the MCBs. Merging MCBs that are configured in one of the multi-port modes is not recommended because of the complexities of having separate arbiters in each MCB. If a multi-port interface is required, it is recommended that the procedures in this document be performed to create a wider effective single-port interface, followed by a "soft" multi-port interface with an arbiter built into the FPGA logic that can then be attached to the wider single-port interface.

Figure 2 illustrates the concept of merging two MCBs to create a wider effective interface. Each MCB still implements a single-component interface with completely separate address, command, and data pins connected to the individual external memory devices. However, inside the FPGA, some additional logic is used to combine the user interfaces of the MCBs to implement a single, wider, user interface. The full read and write datapaths from each MCB are merged to create twice the data bus width, and the address and command signals are distributed to both MCBs to synchronize their operation.

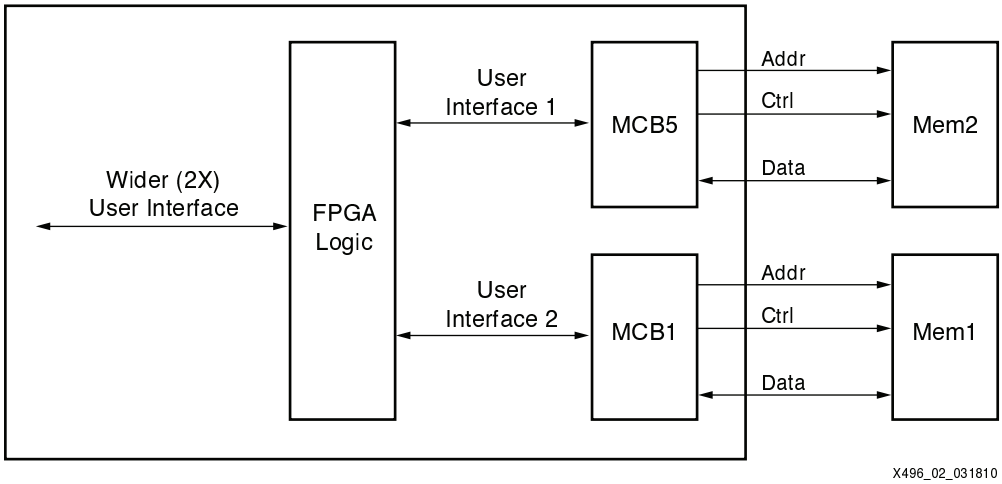


*Figure 2:* **Merging Two MCBs to Create a Wider Effective Interface**

Both MCBs must be configured in the exact same manner, and the external memory devices must also be identical. This allows the two interfaces to operate in unison as much as possible. However, it is possible, even likely, that the two external interfaces do not operate in complete synchronization because of slight timing variations between the interfaces. For example, it is likely that the two memory devices enter refresh at slightly different times, causing a potential interruption of the data flow for one MCB with respect to the other.

This semi-synchronous behavior of the two MCBs does not present any issues with respect to creating the wider internal user interface as long as care is taken to keep the MCBs synchronized from the perspective of their individual user-interface ports. The MCB was specifically designed with a user interface that allows asynchronous operation, meaning that the user interface clock can have a completely undefined relationship with respect to the MCB system clocks that are responsible for synchronizing the physical interface (PHY) of the MCB with the external memory device. The FIFOs in the user interface manage the necessary clock domain crossing. Refer to the clocking section of the *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for more details.

The MCB is an in-order controller, both MCBs always execute read and write commands sent by the common user interface, in the same sequence. As long as the FIFO status flags from both MCBs are monitored and used to properly regulate the requests from the common user interface (e.g., prevent any overflow or under-run conditions on the FIFOs), the two MCBs operate in perfect unison from the perspective of the user application. The following section explains the necessary logic additions to the common user interface to ensure that the two MCBs stay in unison.

## Implementation Details

The user interface for a single MCB is designed to provide a means of communicating with an external memory device by abstracting away the complex timing relationships and protocols required to communicate with a DDR SDRAM memory device. The user interface uses a collection of command and data FIFOs to allow the user application to issue memory transaction requests using a recommended protocol.

The controller sub-block within the MCB then automatically translates these requests into the necessary sequence of DDR device commands sent to the memory device, via the PHY of the MCB, to execute the requested action. The MCB also automatically handles periodic refresh of the memory device. From the perspective of the user application, the external DDR memory resembles a byte-addressable SDR memory space. For more details on the user interface and protocol, refer to the *Spartan-6 FPGA Memory Controller User Guide* [Ref 1].

Figure 3 shows the primary signals involved when combining the user interfaces of two MCBs. The command port-related signals (cmd_byte_addr, cmd_bl, and cmd_instr) are connected in parallel to both user interfaces, ensuring that both MCBs are simultaneously fed with the same instruction, address, and data burst-length information. The read and write datapaths from the individual MCB user interfaces are concatenated to create single data buses with twice the width.

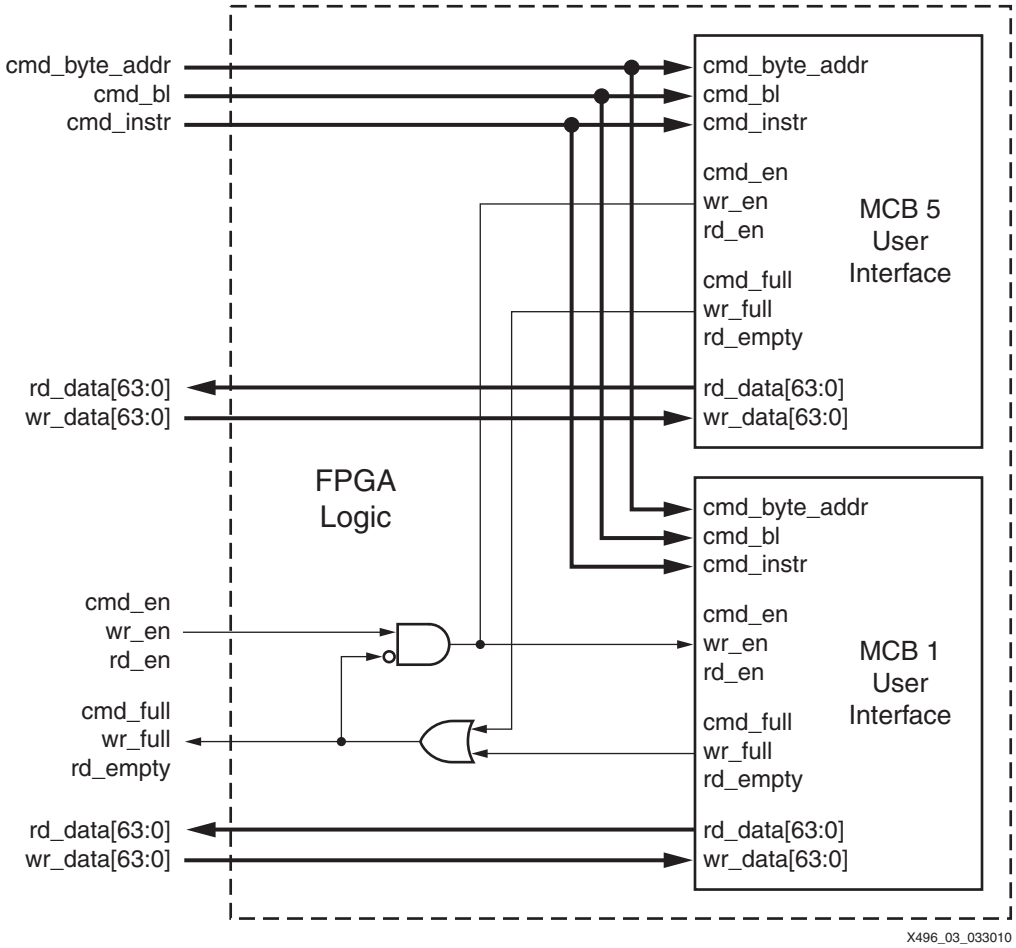


*Figure 3:* **User Interface with Two MCBs**

The next step is to synchronize the operation of the two user interfaces using logic to merge the FIFO status and control signals. The FIFO status flags (cmd_full, wr_full, and rd_empty) from the two MCBs are logically OR'ed together to create a single status flag indicating whether the FIFOs in either MCB are in a full or empty condition. These combined status flags are then used as acknowledge signals to gate the respective FIFO enable inputs (cmd_en, wr_en, and rd_en) to ensure that the two user interfaces stay synchronized. The resulting local versions of the enable signals are fed to both MCB user interfaces.

For example, if one of the two command path FIFOs becomes full, the acknowledge logic prevents a new command from being entered into either MCB, keeping the user interfaces synchronized. Similarly, if one of the read data FIFOs runs empty before the other, possibly because its data flow is interrupted by a DRAM refresh, the acknowledge logic prevents clocking-out any additional data from either MCB, maintaining synchronization.

The acknowledge logic ensures that the two user interfaces stay synchronized by regulating the FIFO access directly, but the user application must to monitor the combined FIFO status flags to ensure that memory transaction requests are paused when either of the MCBs indicates a full or empty condition. Continuing to issue requests when the local MCB enables are deactivated by the acknowledge logic results in lost transactions and invalid assumptions about the associated data.

The clocking section of the *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] describes the recommended clocking architecture for the common case of a single MCB interfacing to a single memory component. The clock infrastructure automatically generated by the MIG tool is

based on the recommended architecture. However, this scheme requires some modification when using two or more MCBs in parallel.

Figure 4 illustrates two possible clocking architectures for combining MCBs. In Figure 4 part (a), two MCBs are shown on the same side of the device. This situation only occurs for devices with four MCBs, two on each side. In this situation, the MCB system clocks driven by BUFPLL_MCB, and the user interface clocks driven by a single BUFG, can be routed to both the upper and lower MCBs. This is a requirement when using two MCBs on the same side of the device, regardless of whether they are being merged into a common user interface. Since there is only one BUFPLL_MCB signal and two available I/O clock routes per-side of the device, there is no means to provide separate system clocks to MCBs on the same side.

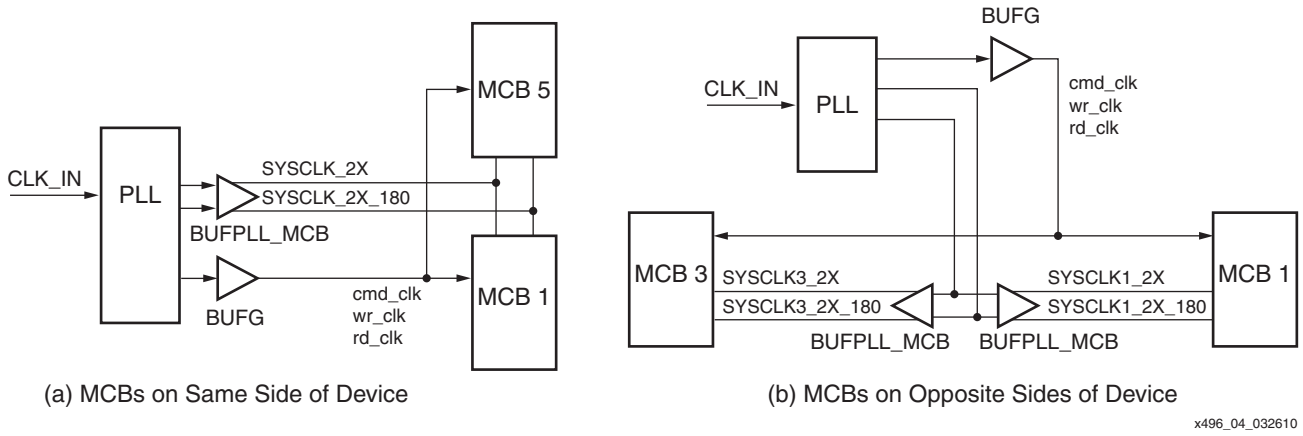


*Figure 4:* **Two Possible Clocking Architectures for Combining MCBs**

In Figure 4 part (b), the two MCBs are on opposite sides of the device. This situation occurs for devices with only two MCBs, or for devices with four MCBs when combining MCBs from opposite sides of the device. In these scenarios, an additional BUFPLL_MCB signal is required to drive the MCB(s) on the other side. There is one BUFPLL_MCB and two available I/O clock routes that it can drive, per side.

## Reference Design Guidelines

The directory structure and files found in the Reference Design are nearly identical to those produced automatically by the MIG tool for a single MCB design, with a few exceptions. The basic directory structure produced by the MIG tool and reproduced for this reference design is shown in Figure 5. For details on the directory structure and files produced by the MIG tool, refer to the *Spartan-6 FPGA Memory Interface Solutions User Guide* [Ref 2].

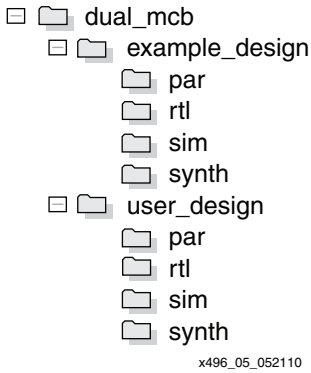


*Figure 5:* **Basic Directory Structure Produced by the MIG Tool**

The MIG tool produces two kinds of MCB memory interface designs; the example_design, and the user_design. The example_design contains all of the necessary RTL files, user constraint files, and scripts, to simulate and implement a completely self-contained MCB-based memory interface. The example_design includes a synthesizable hardware testbench, referred to as the Traffic Generator, that can be used for demonstration or board bring-up purposes. The Traffic Generator can create a variety of read/write stimulus patterns to exercise the MCB user interface.

The user_design does not contain a hardware testbench, but is otherwise essentially the same as the example_design. This version of the MIG design is ready for direct instantiation into an overall system design. Refer to the first chapter of the *Spartan-6 FPGA Memory Interface Solutions User Guide* [Ref 2] for more information on the Traffic Generator, example_design, and user_design.

The Reference Design in this application note combines two MCBs on opposite sides of the device, with each MCB configured to implement a 16-bit interface to a single 1 Gb DDR3 device. The user interface of each MCB is configured as a single 64-bit port. By combining the two MCBs, the reference design effectively implements a 128-bit user interface to a 32-bit wide DDR3 memory device. Any supported single-port configuration of the MCB can be combined in a similar manner.

The most direct way to build the Reference Design is to create two MCB interfaces in a single-pass using the MIG tool. Refer to the *Spartan-6 FPGA Memory Interface Solutions User Guide* [Ref 2] for a complete step-by-step description of how to use the MIG tool to create an MCB interface.

On the Memory Selection page of the MIG tool, select the same memory interface standard (e.g., DDR3 SDRAM) for two of the MCBs. The same configuration choices (e.g., memory device selection and speed, port configuration, termination scheme, etc.,) should be made for both MCB interfaces throughout all steps of the MIG tool GUI flow.

When all of the necessary base files are generated in a single run using the MIG GUI wizard, the modifications detailed in the following sections can be implemented to merge the two separate MCB interfaces into a single design with twice the user interface and memory interface width. These steps are documented from the perspective of the example_design, but an analogous process can be used for the user_design.

## New Wrapper Creation (`memc13_wrapper.v`)

The MIG tool produces two separate MCB wrapper files (`memc1_wrapper.v` and `memc3_wrapper.v`). A new wrapper that merges the two MCB user interfaces into a single interface is required to implement the reference design files. In this step, a new wrapper file (`memc13_wrapper.v`) is created between these two base wrapper files and the top-level `example_top.v` design file. Figure 5 illustrates the insertion of this new wrapper layer instantiated in the `example_top.v` file, rather than the two individual base wrappers.

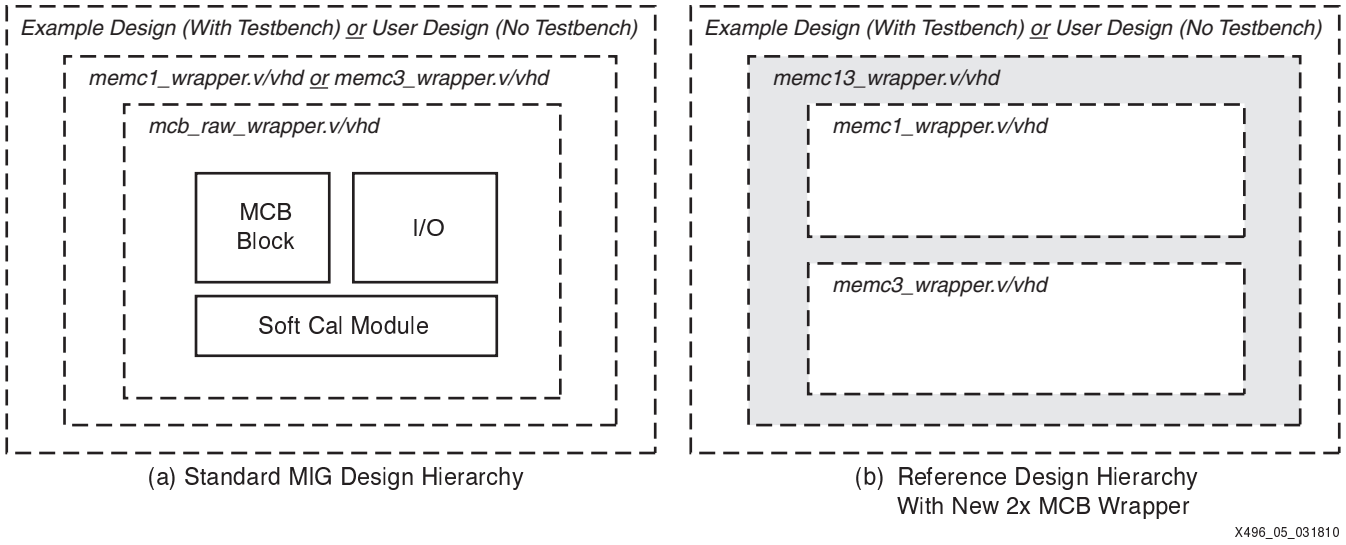


*Figure 6:* **Insertion of New Wrapper Layer**

The new `memc13_wrapper.v` file addresses the following changes, most of which are notated directly in the comments found in the reference design files:

- The new wrapper implements the connectivity and logic from Figure 3 to combine the command and datapaths of the two separate user interface ports. Command signals are sent to both MCBs, the combined datapath is a concatenation of the separate datapaths from both MCBs. The wrapper also implements the OR of the FIFO status signals, and the acknowledge logic to gate the FIFO enables.
- The wrapper passes common parameters to both base wrapper instantiations.
- External memory interface buses are kept separate because they remain as two distinct component interfaces.
- The MCB system clock (sysclk1_2x and sysclk3_2x) and clock enables (pll_ce1_0 and pll_ce3_0) are kept separate because they are driven by different BUFPLL_MCB blocks when MCBs are on opposite sides of the device. This is not necessary when combining MCBs on the same side of the device.

## New Clock Infrastructure Block (`memc13_infrastructure.v`)

The MIG tool produces two clock infrastructure blocks in the base files (`memc1_infrastructure.v` and `memc3_infrastructure.v`), although only one is needed. One of these should be removed, and the other renamed to `memc13_infrastructure.v`, and modified as follows:

- Add a second BUFPLL_MCB block driven by the same outputs of the PLL as the existing BUFPLL_MCB block. This is not necessary when the MCBs are on the same side of the device.
- Create a second set of system clocks (sysclk3_2x) and clock enables (pll_ce3_0) from the outputs of the new BUFPLL_MCB block, and bring these out of the newly named `memc13_infrastructure.v` block.

## New Testbench Block (memc13_tb_top.v)

The MIG tool produces two testbench blocks in the base files (`memc1_tp_top.v` and `memc3_top_tb.v`) when only one is needed. One of these should be removed, and the other renamed to `memc13_tb_top.v`, and modified as follows:

- Double the data port size (C_P0_DATA_PORT_SIZE) and mask size (C_P0_MASK_SIZE) global parameters to accommodate the new user interface width.
- Increase the width of the error_status output by the new combined user interface width (add 128 bits to the error_status bus if the combined user interface is 128-bits wide).
- Set the value of the local DWIDTH parameter (p0_DWIDTH) to the combined user interface width (128 bits).

## Changes to Top-Level Example Design Block (`example_top.v`)

In the base files produced by the MIG tool, the top-level design block (`example_top.v`) contains two instantiations of the base MCB wrappers, clock infrastructure blocks, and test bench blocks. The following changes implement a single, combined MCB interface using the blocks modified in the previous steps:

- Keep input and output signals to the two external memory interfaces separate.
- Create a single set of all global and local parameters to be common for both MCB interfaces (e.g., C13_MEMCLK_PERIOD instead of separate C1_MEMCLK_PERIOD and C3_MEMCLK_PERIOD parameters). This set of common parameters should be passed to the various sub-blocks.
- Double the data port size (C13_P0_DATA_PORT_SIZE) and mask size (C13_P0_MASK_SIZE) global parameters to accommodate the new user interface width.
- Create a single set of input clocks and resets (e.g., c13_sys_clk_p and c13_sys_clk_n) to replace the separate clocks for each interface.
- Modify the testbench address parameters as follows:
  - Double the BEGIN_ADDRESS and END_ADDRESS parameters. For details,see the code comments in the `example_top.v` file.
- Create common wires for internal signals (c13_calib_done) except for the system clocks (c1_sysclk_2x), and clock enables (c1_pll_ce_0).
- The error and calib_done output signals should now be assigned directly to the c13_error and c13_calib_done signals.
- Instantiate the new `memc13_wrapper.v` wrapper, new clock infrastructure block, and new testbench block created in the previous steps (there should be only one copy of each), and connect to the required signals.

## Other Changes for Simulation and Implementation

Additional changes are required to make the reference design files suitable for simulation and design implementation:

- In the `sim/functional` sub-directory, the `sim_tb_top.v` file must be modified to accommodate the signal changes to the `example_top.v` file, including the generation of a single set of clock and reset signals.
- In the `par` sub-directory, the `example_top.ucf` file must be modified to accommodate the signal changes to the `example_top.v` file, including specifying a single set of clock and reset signals.

- The following manual modification to the Traffic Generator is required:

  Open the `mcb_traffic_gen.v` module provided in the `example_design/rtl/traffic_gen` directory. Locate the following code (starting on line 317):

```
MIG 3.4 Code:

     reg mcb_rd_empty;
     always @ (mcb_rd_empty_i, mcb_rd_empty_r)
     if ( FAMILY == "SPARTAN6")
        mcb_rd_empty = mcb_rd_empty_r;
    else
       mcb_rd_empty = mcb_rd_empty_i;

   reg mcb_wr_full;
   always @ (mcb_wr_full_i, mcb_wr_full_r1)
   if ( FAMILY == "SPARTAN6")
       mcb_wr_full = mcb_wr_full_r1;
   else
       mcb_wr_full = mcb_wr_full_i;
```

  Replace with:

```
Workaround Code:
    wire mcb_rd_empty;
    assign     mcb_rd_empty = mcb_rd_empty_i;

    wire mcb_wr_full;
    assign     mcb_wr_full = mcb_wr_full_i;
```

## Design Utilization and Performance

Table 1 shows the design utilization and performance metrics for the associated reference design.

*Table 1:* **Design Utilization and Performance Metrics**

| Parameters | Speed Grade | Specifications and Details |
|---|---|---|
| Maximum data rate: external interface (by speed grade) | -2 | Refer to the Performance Interface section of DS162, *Spartan-6 Data Sheet*. [Ref 3] |
| | -3 | |
| Typical achievable data rate: user interface (by speed grade) | -2 | 87 MHz(1) |
| | -3 | 100 MHz(1) |
| Target Spartan-6 FPGA | | XC6SLX16 |
| Device utilization without testbench | | < 10 slices |
| Effective external interface bus width | | 32 bits |
| Effective user interface bus width | | 128 bits |
| Target memory device for verification | Simulation | MT41J64M16LA_187E (Micron DDR3) |
| | Hardware | MT41J64M16LA_187E (Micron DDR3)<br>EDE1116ACBG_8E_E (Elpida DDR2) |

**Notes:**

1. The maximum data rate for the user interface depends on the specific device and PAR results for a given design. The numbers shown here are representative of the performance that can be achieved with the example design as documented in this application note.

# Reference Design

The reference design for this application note can be found at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=147378

## Reference Design Checklist

The checklist in Table 2 indicates the tool flow and verification procedures used for the reference design.

*Table 2:* **Reference Design Checklist**

| Parameter | Description |
|---|---|
| **General** | |
| Developer | Xilinx |
| Target devices | Spartan-6 LX and LXT FPGAs |
| Source code provided | Yes |
| Source code format | Verilog |
| Other IP incorporated in reference design | MIG generated IP: MCB interface wrappers |
| **Simulation** | |
| Functional simulation performed | Yes |
| Timing simulation performed | No |
| Testbench format | Verilog |
| Simulator | Modelsim 6.5c |
| **Implementation** | |
| Synthesis | XST (ISE® design tools v12.1) |
| Implementation | ISE design tools v12.1 |
| Static timing analysis performed | Yes |
| **Hardware Verfication** | |
| Hardware verified | Yes |
| Hardware platform used for verification | SP601 board |

# Conclusion

The MCB addresses the memory interface needs for the majority of Spartan-6 FPGA applications by providing a simple means of interfacing to the most common, lowest cost, and lowest power SDRAM memory standards. However, some applications with higher memory bandwidth or density requirements can benefit from using memory interfaces wider than the 16-bits offered by a single MCB. This application note describes how to merge the operation of two or more MCBs to implement effective 32-bit or wider memory interfaces. Each MCB still operates at full performance (up to 800 Mb/s), allowing the user application to realize the full benefit of using these dedicated embedded memory controllers for wider interfaces.

# References


1. UG388, *Spartan-6 FPGA Memory Controller User Guide*
2. UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*
3. DS162, *Spartan-6 Data Sheet*

## Revision History

The following table shows the revision history for this document.

| Date | Version | Description of Revisions |
|---|---|---|
| 06/03/10 | 1.0 | Initial Xilinx release. |

## Notice of Disclaimer