XAPP729 (v1.0.1) March 4, 2007

Interfacing a 64-Bit DDR Memory Bus to a 32-Bit Microprocessor Bus

Author: Marc Defossez

Summary

In today's processor, digital signal processor (DSP), and other applications, memory data widths can be 64 bits and higher. This application note shows how the 32-bit MicroBlaze™ processor can easily access these wide memories. This design is also suitable for use with the IBM PowerPC™ (PPC405) processor because it connects to the On-chip Peripheral Bus (OPB). The reference design provides a modification to an existing Xilinx EDK SDRAM interface, enabling a 32-bit processor to access a 64-bit data bus.

Introduction

When available solutions are too wide or narrow, custom memory width designs are created by connecting standard memory devices together in parallel or serial configurations. This application note describes the modifications to the standard EDK 32-bit wide SDRAM interface, widening it to 64 bits. For the PPC405 or MicroBlaze processor, the connected memory appears to be 32 bits wide, while in reality the memory is 64 bits wide. The reference design described in this application note uses two 32-bit SDRAMs configured in a 64-bit data setup. These memories share all control and address lines, as shown in Figure 1.

In this design, the memory is always accessed in 64-bit mode, using two consecutive memory addresses from the processor. An access must first call the lower addressed 32-bit word followed by the address of the higher 32-bit word.

The least-significant (LS) address bit is used as a multiplexer switch between even and odd addresses to the memory. All lower 32-bit data is stored in one SDRAM bank, and the higher 32-bit data is stored in the second SDRAM block.

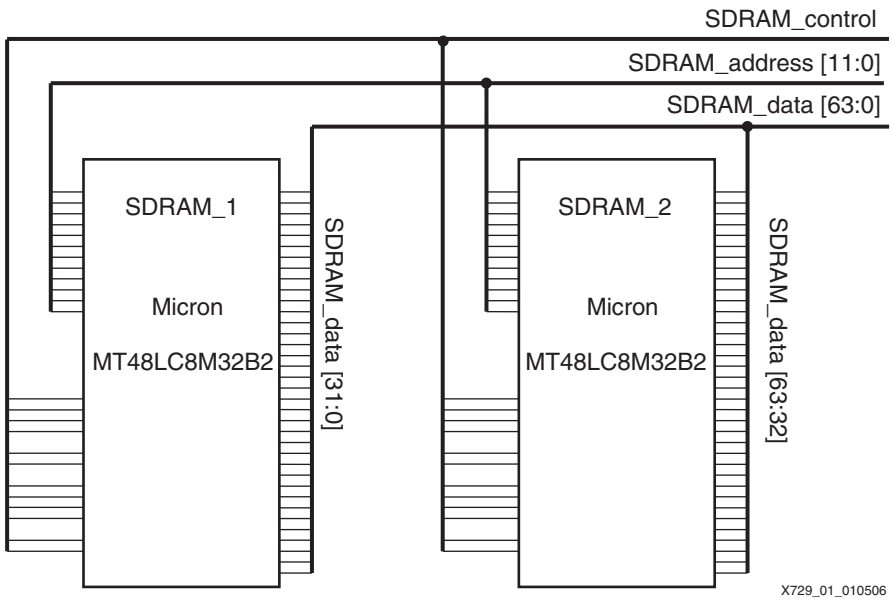


Figure 1: **64-Bit Memory Setup**

Memory Configurations

Custom-width memory systems are built by interconnecting 32-bit memories. An economic and easy way to double the data bus is to share the address bus and all control signals. Using this method, the data bus is multiplied in size with each additional memory block, as shown in Figure 1.

A disadvantage to this method is that the SDRAMs are accessed simultaneously. Data must be assembled and disassembled in the FPGA interface to the SDRAM. An advantage to this method is the easy setup for PCB layout.

This method is tested using a Virtex™-4 XC4VSX35 FPGA connected to a set of Micron SDRAMs (MT48LC8M32B2: 2 Mb x 32 x 4 banks). "Appendix A," page 17 provides schematics of this setup.

FPGA Interface

The easiest way to test a memory on a given design is to use a processor system to write and read the memory and compare the results. A system build with the EDK development system cannot be used as provided because its SDRAM controller has a maximum width of 32 bits.

There are two solutions to this design problem. One solution is to develop a new 64-bit SDRAM controller to connect into the EDK software. The second solution uses existing IP structures wherever possible. This application note uses the second solution. It describes how to modify the existing SDRAM controller to make a customized 64-bit version. The OPB_SDRAM controller is used because it can interconnect with the MicroBlaze and PPC processors.

Solution

For wide memories, the address bus and control signal bus remain unchanged as the data width increases. If needed, this concept can be used to construct wider data memories than the 64-bit interface derived from the existing SDRAM controller.

This solution ensures that the processor in the FPGA can use its native 32-bit read and write operations as if it is accessing 32-bit memories. To access the 64-bit memory, the processor *must* issue a Low address (LS address bit = 0) followed by a consecutive High address (LS address bit = 1) for write and read operations. This is not a limitation for this type of memory application, however, for normal 32-bit memory access it can be a limitation since two memory accesses (two consecutive addresses) are always needed.

Because all control lines are shared between the two memory devices, operations such as byte write and bursting can be performed as in normal 32-bit use. For byte operations, a byte write happens like a byte write in a 32-bit system. The address accessed determines the memory device on which the byte operation is performed. For bursting, the interface automatically switches between the connected memories as long as the bursting operation lasts.

The external 64-bit SDRAM memory is built out of two 32-bit memories with 2 Mb of address space. This needs the construction of a 4-Mb-deep memory interface for the processor. The LS address bit is used as a data multiplexer, where the lower memory data bank is selected when the LS address bit is 0 and the upper data bank is selected when the LS address bit is 1. With this mechanism in place, the memory appears as a 4-Mb-deep by 32-bit-wide memory to the processor, although the external memory is actually 64 bits wide and 2 Mb deep.

The bursting must start at an address with the LSB-bit at zero, as single 64-bit assesses. The interface automatically switches between the memory devices as long as the bursting operation lasts.

Byte access is done at a 32-bit boundary. Byte access to a 64-bit memory address is accomplished in two steps.

1. The processor accesses the lower address with the correct byte mask in place to access only the wanted byte of the lower 32-bit word.
2. The processor accesses the higher address with the byte mask bits set depending the needed byte of the higher 32-bit word.

Assume byte 6 of the 64-bit must be accessed.

- Set the byte mask to omit all bytes in the lower address and access the memory.
- Set the byte mask to omit bytes 1, 3, and 4 and access the higher memory address.

When bursting with byte access, the byte mask will count for all addresses accessed though the bursting operation. The lower and higher addresses representing the 64-bit will be accessed all with the same byte mask.

Figure 2 shows the memory layout, and Figure 3 shows the interface setup.

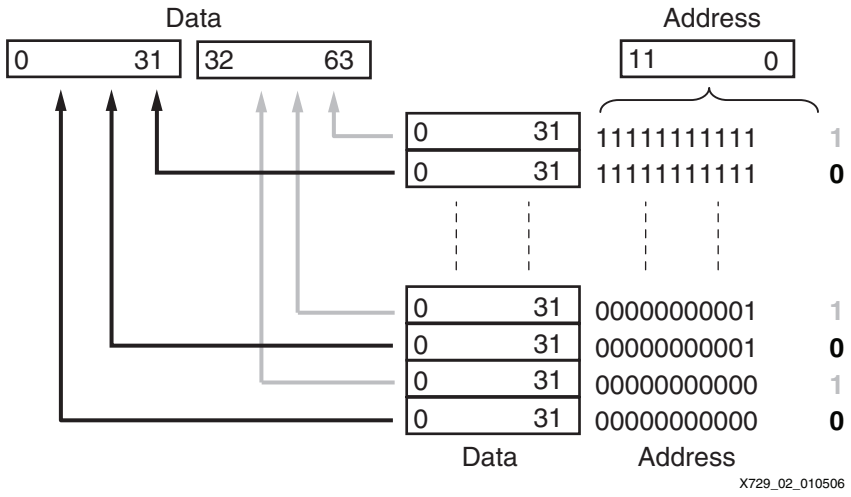


Figure 2: **Reference Design Memory Address Layout**

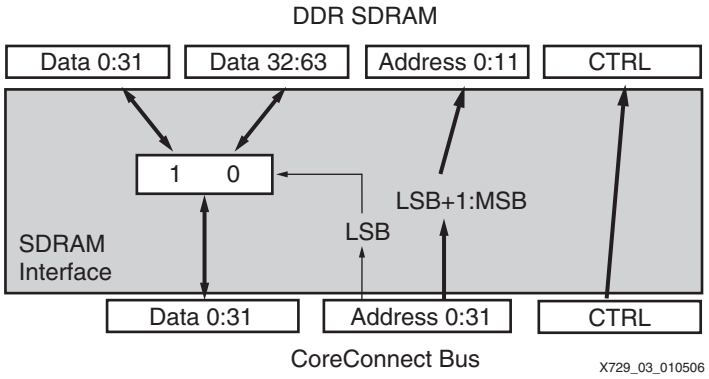


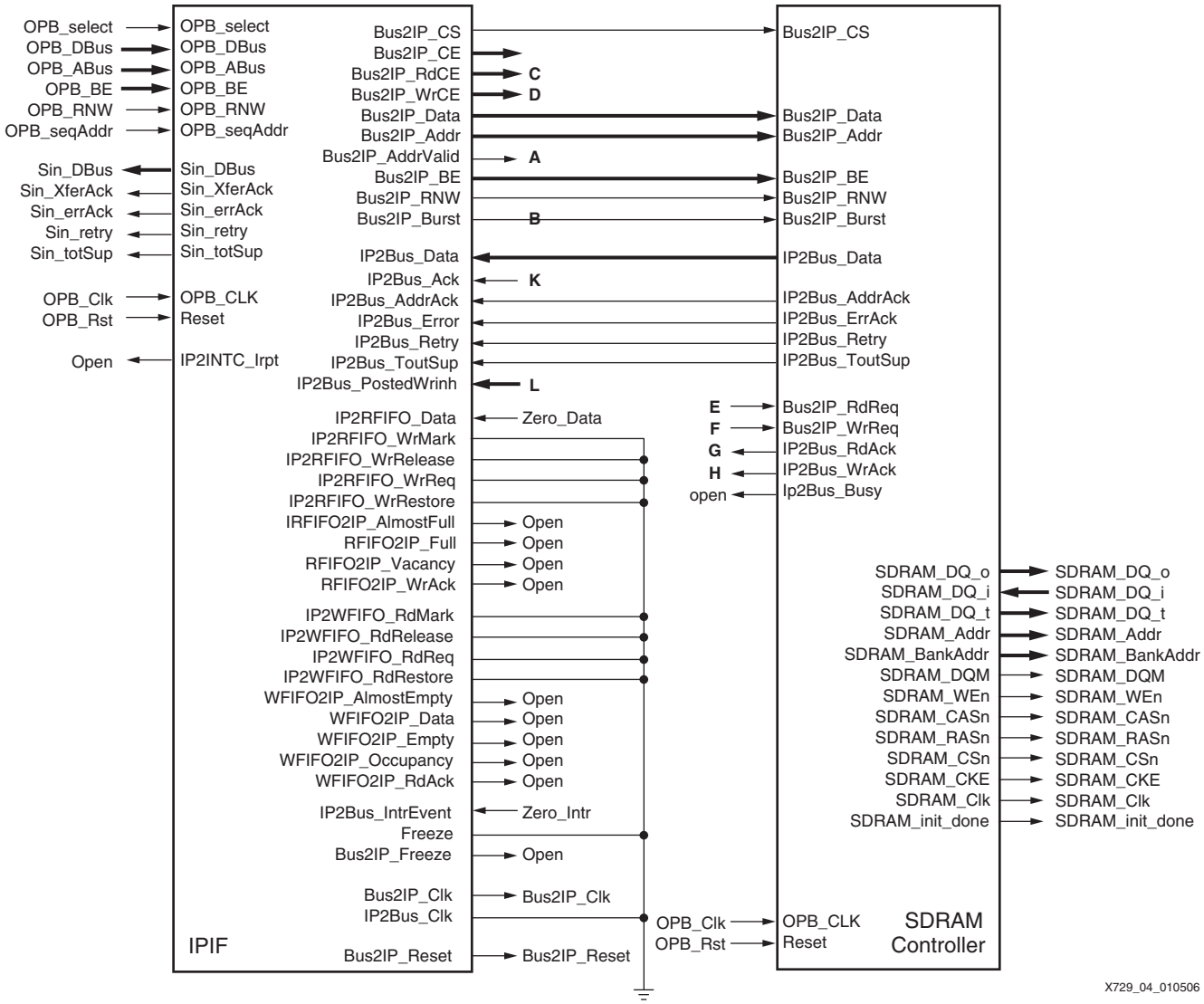
Figure 3: **Reference Design Interface Setup**

The "SDRAM Interface" section explains the SDRAM interface and modifications, and the "Interface Implementation" section describes the implementation and flow.

SDRAM Interface

As with all Xilinx OPB controllers, the SDRAM interface controller is built out of a Xilinx Intellectual Property Interface (IPIF) and the interface controller itself, as shown in Figure 4.

This reference design modifies the SDRAM controller. Modifications cannot alter the IPIF because this is the standard engine to connect different IP to the IBM CoreConnect™ buses. Details on the OPB-IPIF interface are found in the EDK documentation at http://www.xilinx.com/ise/embedded/edk_docs.htm.



Notes:

1. Signals with labels A to H are routed through logic gates.

Figure 4: **OPB-SDRAM Interface**

The SDRAM controller has seven different hierarchical levels and is completely parameterized through the use of VHDL generic statements. The values of these generics are set in the Xilinx Processor System (XPS) development tool and are passed to VHDL through intermediate files.

The seven hierarchical levels of the SDRAM controller are:

1. Ipic_If

   Interface between the processor bus and the peripheral. The address bus is split and rewired in this level.

2. Data_statemachine

   Data handling at read and write operations. In this level, the datapath doubles in width.

3. Command_statemachine

   State machine handling RAS, CAS, and so forth.

4. Init_statemachine

   Initialization of the memory.

5. Io_registers

   Interface to memory/FPGA connections. The data inputs and outputs double in size in this level.

6. Counters

   Counters used in this interface.

7. Clock_gen

   Clock generation logic and feedthrough.

The Ipic-if, data_statemachine, and Io_registers hierarchical levels and their required logic are described in more detail in these subsections:

- "IPIC Interface (ipic-if)"
- "Data State Machine (data_statemachine)"
- "I/O State Machine (Io_registers)"

IPIC Interface (ipic-if)

This hierarchical level provides the interface between the IPIF IP block and the SDRAM controller. It is a bus translation and bus size adaptation layer that generates new signals depending on incoming signal states.

The incoming address bus is split into row, column, and bank address buses, and the CoreConnect read and write data buses are passed to the correct buses at the SDRAM controller side. This configuration is just a rewiring of the incoming and outgoing buses to the correct sizes.

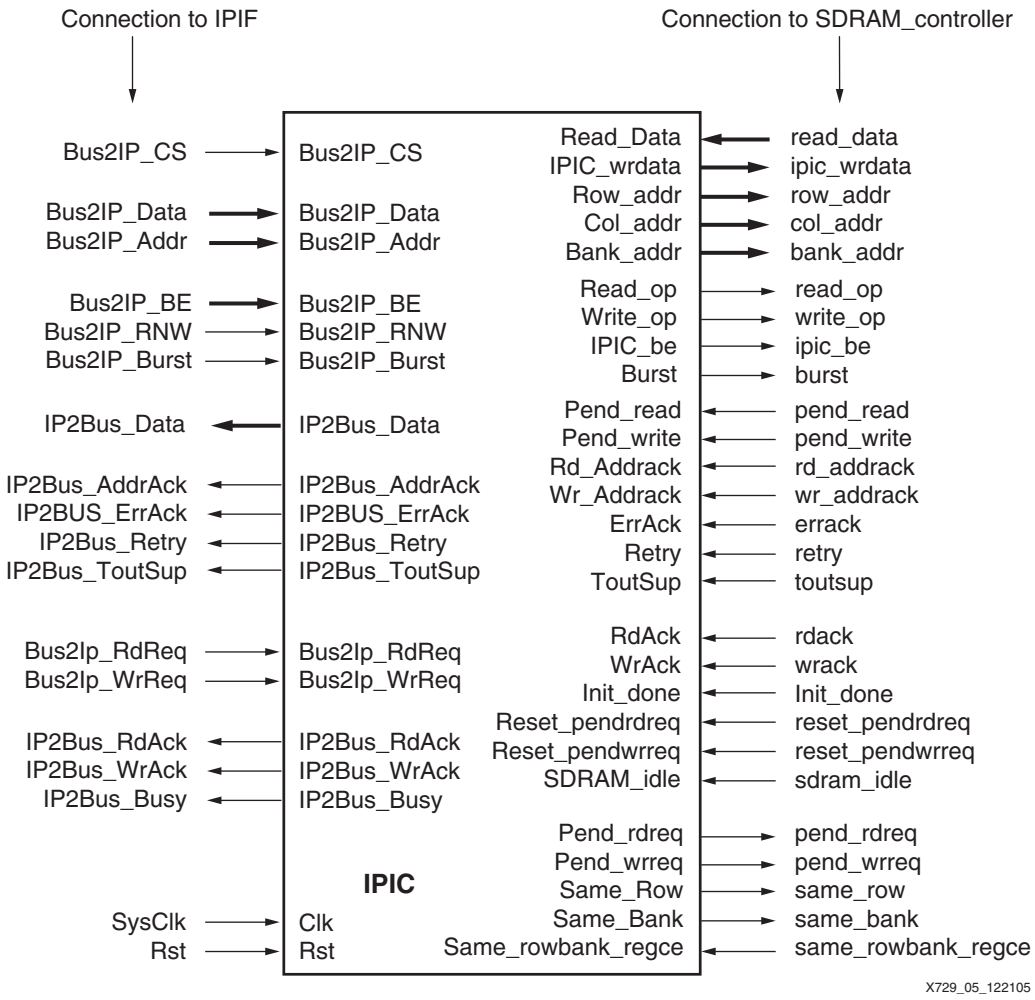


Figure 5: **Top-Level Connections of the IPIC_IF Hierarchical Level**

In this level, the LS address bit is split from the address bus structure, and the rest of the address bus is shifted. The LS address bit adds a new signal to the IPIC component to allow switching between Low and High memory SDRAM data blocks (generation of the multiplexer enable signal).

From the top level “opb_sdram” on, all bus structures are sized through VHDL generic syntax settings. The OPB_SDRAM controller documentation ([Ref 1]) provides a complete description of each generic. Table 1 lists the important generics for this application note. “Appendix B” shows the generic settings for the Micron SDRAM device.

Table 1: **Important VHDL Generics for Buses**

OPB_sdram	Value	Sdram_controller	Ipic-if
C_OPB_DWIDTH	32	C_IPIF_DWIDTH	C_IPIF_DWIDTH
C_OPB_AWIDTH	32	C_IPIF_AWIDTH	C_IPIF_AWIDTH
C_SDRAM_DWIDTH	32	C_SDRAM_DWIDTH	C_SDRAM_DWIDTH
C_SDRAM_AWIDTH	12	C_SDRAM_AWIDTH	C_SDRAM_AWIDTH
C_SDRAM_COL_AWIDTH	9	C_SDRAM_COL_AWIDTH	C_SDRAM_COL_AWIDTH
C_SDRAM_BANK_AWIDTH	2	C_SDRAM_BANK_AWIDTH	C_SDRAM_BANK_AWIDTH

When examining and calculating the values for the generics:

- The data bus at the CoreConnect level is wired to the SDRAM data buses.
- The address bus from the CoreConnect bus system is split into row, column, and bank address buses for the SDRAM controller as shown in Table 2 and Figure 6.

Table 2: **SDRAM Address Calculations**

Generic Variable	Equation
SDRAM_ADDR_OFFSET	Log2(C_SDRAM_DWIDTH/8)
OPB_ADDR_OFFSET	Log2(C_OPB_DWIDTH/8)
COLADDR_STARTBIT	C_OPB_AWIDTH – (C_SDRAM_COL_AWIDTH + SDRAM_ADDR_OFFSET)
COLADDR_ENDBIT	C_OPB_AWIDTH – OPB_ADDR_OFFSET-1
NUM_ZEROADDR_BITS	OPB_ADDR_OFFSET – SDRAM_ADDR_OFFSET
ROWADDR_STARTBIT	COLADDR_STARTBIT – C_SDRAM_AWIDTH
ROWADDR_ENDBIT	ROWADDR_STARTBIT + C_SDRAM_AWIDTH-1
BANKADDR_STARTBIT	ROWADDR_STARTBIT – C_SDRAM_BANK_AWIDTH
BANKADDR_ENDBIT	BANKADDR_STARTBIT + C_SDRAM_BANK_AWIDTH-1
ZERO_COL_PAD	(0 to C_SDRAM_AWIDTH – C_SDRAM_COL_AWIDTH-1)
Column Address	ZERO_COL_PAD & OPB_ABus (COLADDR_STARTBIT to COLADDR_ENDBIT)
Row Address	OPB_ABus (ROWADDR_STARTBIT to ROWADDR_ENDBIT)
Bank Address	OPB_ABus (BANKADDR_STARTBIT to BANKADDR_ENDBIT)

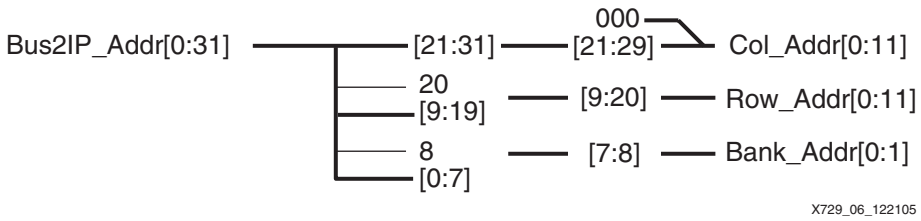


Figure 6: **Address Bus Split in IPIC_IF**

Required Modifications

Because the LS address bit is used to switch the data bus between the different SDRAMs, the address bus must be shifted by one bit. The LS address bit, which acts as the multiplexer select bit for the data bus, forms a new output of the “ipic_if” interface.

The CoreConnect bus is coded using the big-endian format shown in Figure 7.

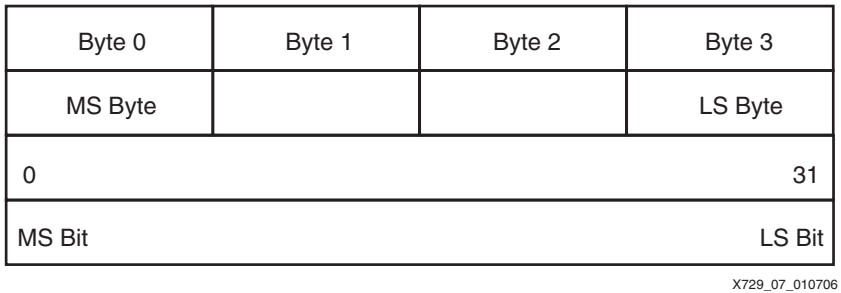


Figure 7: **Big-Endian Data Organization**

To obtain the shifting effect, a new generic variable called C_SDRAM_WIDE_DBUS is created in the top level of the IP. Legal values for this variable are: 0 and 1.

When the variable is:

- 0: A normal SDRAM controller is generated with a maximum data width of 32 bits.
- 1: A dual-width SDRAM controller is generated following the principle explained in this document.

An extra output pin for the "ipic_if" called Sel_Wide_DBus, reflecting the LS address bit status, is created.

The generic variable is subtracted from the value calculated for:

- SELADDR_MUXBIT = (C_IPIF_AWIDTH – IPIF_ADDR_OFFSET – 1)
- COLADDR_STARTBIT = (C_IPIF_AWIDTH – (C_SDRAM_COL_AWIDTH + SDRAM_ADDR_OFFSET)) – C_SDRAM_WIDE_DBUS
- COLADDR_ENDBIT = (C_IPIF_AWIDTH – IPIF_ADDR_OFFSET – 1) – C_SDRAM_WIDE_DBUS

All other calculated values use the modified address range values, resulting in an address setup as shown in Figure 8.

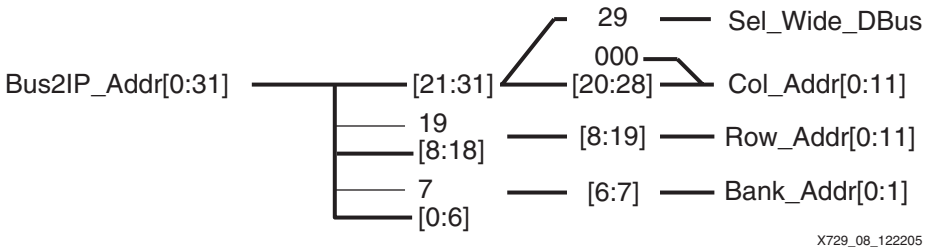


Figure 8: **Wide Data Bus Address Setup**

Data State Machine (data_statemachine)

This hierarchical level routes the data from the processor bus to the I/O section and vice-versa. It ensures that writing and reading happen at the correct moment and data bus actions are signaled to other parts of the SDRAM controller interface. Figure 9 shows the external connections of this hierarchical level before any modification. The signals of interest are schematically shown in Figure 10.

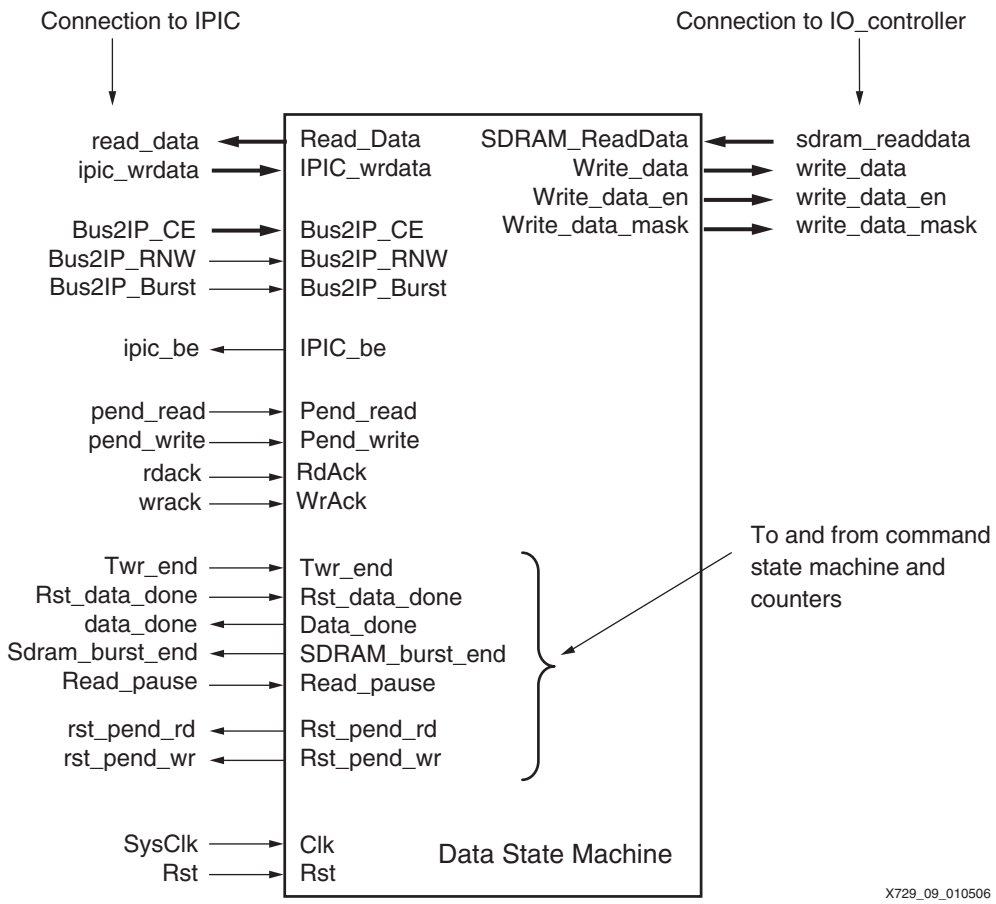


Figure 9: **Top-Level Connections of a 32-Bit Data State Machine**

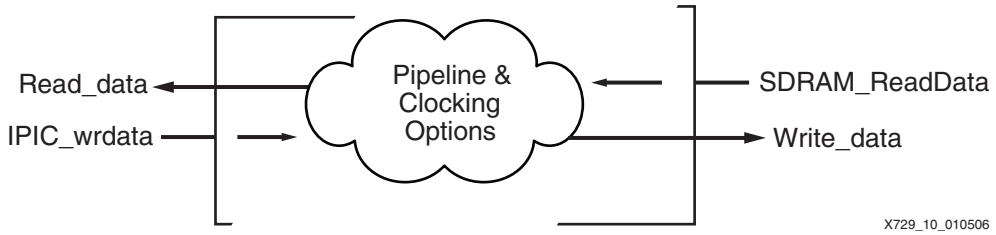


Figure 10: **Data Flow Through the Data State Machine Hierarchical Level**

A set of generic variables controls the data flow between the processor bus and the I/O registers (pads). The values of these generics are determined and/or changed when the interface is added to a design in the EDK software. All variables determine pipelining (high-speed designs), clocking possibilities, and bursting (see Table 3).

Table 3: **Generic Variables Involved in Determining the Data State Machine Setup**

Generic Variable	Default Value	Equation or Explanation
C_SDRAM_BRST_LEN	–	IPIF_DWIDTH / SDRAM_DWIDTH
C_SDRAM_CAS_LAT	2	CAS latency determined by memory devices.
C_USE_POSEDGE_OUTREGS	0	When set to 1, uses positive-edge registers. When cleared to 0, uses negative-edge registers.
C_INCLUDE_HIGHSPEED_PIPE	1	Adds a pipeline stage for high frequency operation. When C_USE_POSEDGE_OUTREGS is set to 1, this variable should be cleared to 0.

Required Modifications

The size of the incoming and outgoing datapaths must change so that data written from the IPIF side is widened to 64 bits and data from the SDRAM side is reduced to 32 bits for the processor. The control of the datapath sizes is managed by a new signal and a new generic variable created in the IPIC interface logic and by the existing generic variables.

An input signal “Sel_Wide_DBus” must be created for use as a multiplexer control signal or a logic enable signal. For write operations, this signal is used to swap the data from the processor’s 32-bit data bus onto the Low and High 32-bit SDRAM buses. For read operations, this signal is used to transfer data from the Low and High SDRAM data buses onto the 32-bit processor bus.

The modifications to the different data buses comprise all existing generic variables so that the pipeline registers and positive-edge or negative-edge registers are automatically generated for these new extensions. To carry out these modifications in the existing VHDL, a substantial amount of code must be added (mostly cut-and-paste operations).

Read Data Flow from SDRAM to Processor

The standard read data flow can be customized through a set of generic VHDL values to contain pipeline, burst, and positive- or negative-clocked registers. When the newly introduced generic value C_SDRAM_WIDEBUS is set to "1", this logic is duplicated and the output is passed through a multiplexer controlled by the Sel_Wide_DBus signal. Figure 11 shows how the 64-bit extension is generated.

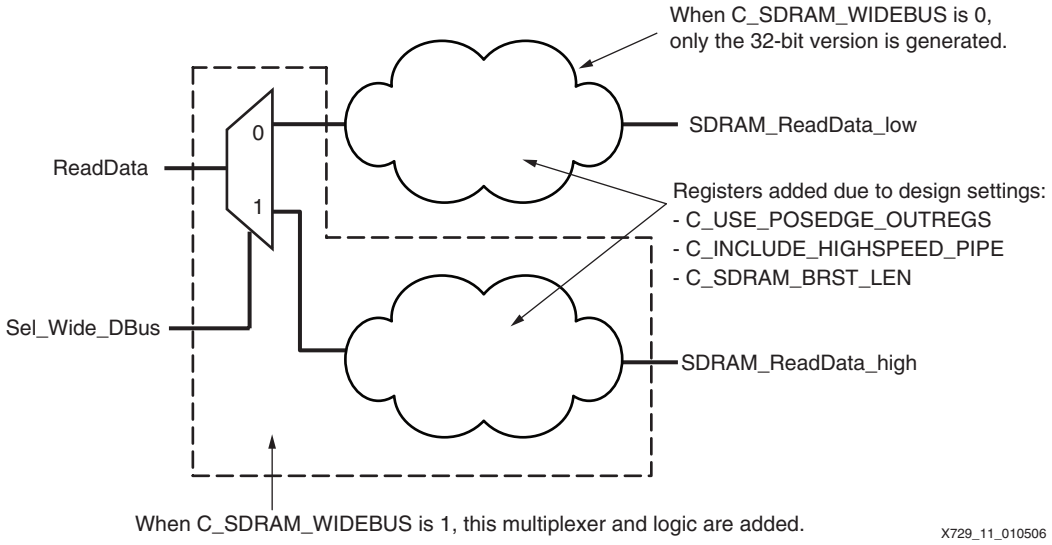


Figure 11: **Read Datapath and Additions**

Write Data Flow from Processor to SDRAM

Figure 12 shows the write data flow and its modifications.

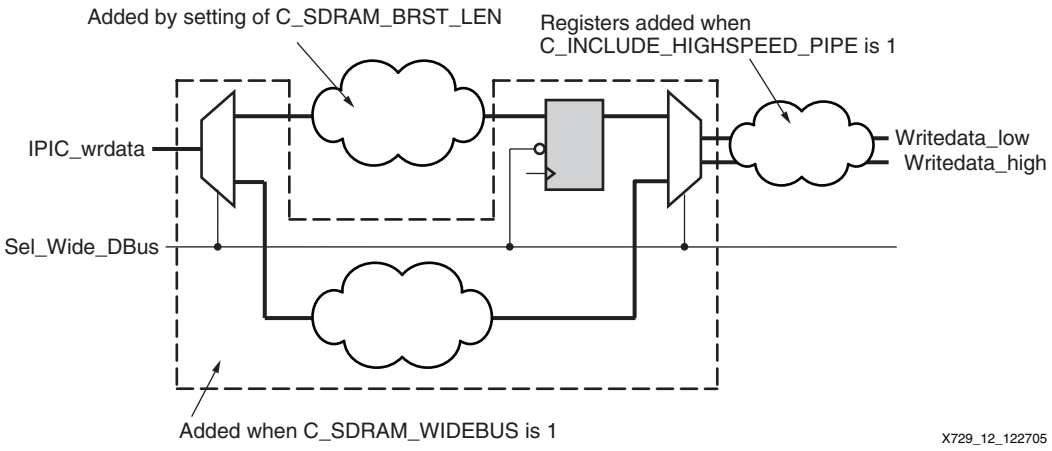


Figure 12: **Write Datapath**

The 32-bit write datapath is customized through the same set of generic VHDL values as used for the read datapath. Output registers, clocking edge polarity, pipeline, and burst registers can be added through these generics. For the 64-bit version, C_SDRAM_WIDEBUS must be set to "1", which duplicates the 32-bit path and adds a demultiplexer and register set.

Because all control lines and address lines are shared between the connected SDRAMs, a protection must be built into the interface to prevent the same data to be written into both connected SDRAMs. A switch at the front of the block directs incoming data to the Low or High data bus. Data is then passed through possible register sets added through the use of normal interface generics. A register and data-combining multiplexer are required to combine the data into a 64-bit format.

A set of pipeline registers can be added in the path after any extra burst registers. In this case, an additional register and data-combining multiplexer are required in front of the set of pipeline registers to handle the tying together of all control lines.

Here is a description of the data-combining register and multiplexer:

- When writing the first 32 bits of data to the lower address, data is written into this added register. The SDRAM controller acts as if the data is written into the external memory.
- When the second 32 bits of data are written to the high address, this data is combined with the lower-address data, and the full 64 bits are passed to the external SDRAM.

I/O State Machine (Io_registers)

This hierarchical level provides the interface between the memory and I/O, as shown in Figure 13. To modify this level to run in 64-bit mode, a set of output and input registers must be added.

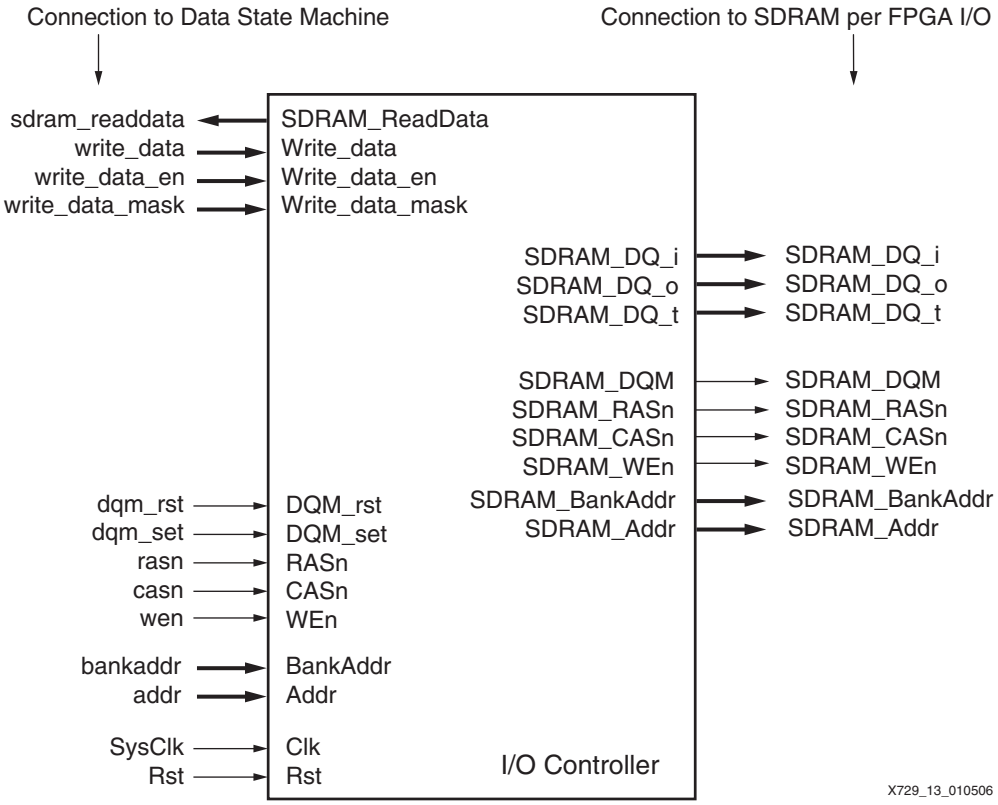


Figure 13: **I/O State Machine**

The write-enable output *must* be combined with the Sel_Wide_Bus signal. Data can be written into the external SDRAM only when Sel_Wide_Bus is High.

When C_SDRAM_WIDEBUS is High, registers must be added for the negative- and positive-edge clocked flip-flops and for the DQ_o and DQ_t outputs. Also input registers must be added when the generic variable is set, resulting in an extra set of inputs and outputs from and to the SDRAM.

Interface Implementation

To complete the whole process and to ease tuning of other cores, this section describes a project where the customized SDRAM controller is used. For this project, the "opb_sdram_v1_00_e" version delivered with EDK 7.1.2i of the OPB_SDRAM controller is used. This version and the "sdram_v1_00_e" version of the SDRAM controller are copied to the local project directory to create a proprietary version of the controller.

Project Setup

The project is carried as a combined ISE/EDK project. The processor and its peripherals are used as a hierarchical level of a top-level design containing all clocking and I/O connections. Figure 14 shows the created directory setup.

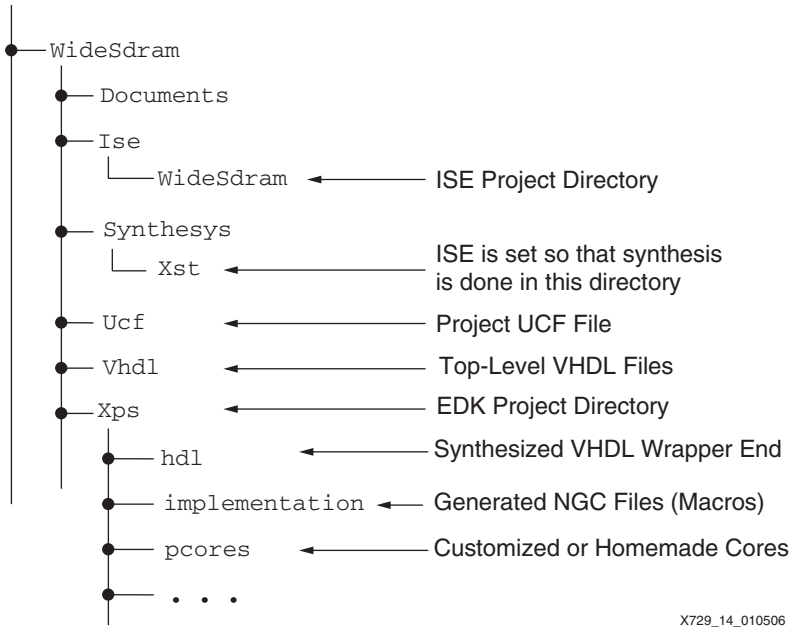


Figure 14: **Project Directory Setup**

Processor Project

These steps describe how to complete the processor project:

1. An XPS base project is created with the Base System Wizard in the `Xps` directory (see Figure 14).
2. The standard OPB_SDRAM is defined as one of the peripherals.
3. The processor and all peripherals are given proper names and the automatically included DCM is removed from the Base System Project. This step can be done from within the XPS tool, or a text editor can be used to modify the MHS and MSS files (see "Appendix C").
4. The project options are set so that the XPS design is generated as a hierarchical level of an already created top-level design.
5. The processor design is passed through the XPS tools, and the VHDL_wrapper and NGC files of this application are built. The processor design's top-level wrapper VHDL file is located in the `hdl` directory of the XPS project.

64-Bit SDRAM Controller

These steps indicate the process to complete the SDRAM controller project:

1. The latest versions of the OPB_SDRAM and SDRAM IP are copied from the EDK installation directory into the `pcores` directory of the project. The entire directory for each IP core is copied.

   The IP cores are located in:

   - `<XILINX_EDK>\hw\XilinxProcessorIPLib\pcores\opb_sdram_v1_00_e`
   - `<XILINX_EDK>\hw\XilinxProcessorIPLib\pcores\sdram_v1_00_e`
2. The directories and all top-level files of the OPB_SDRAM and SDRAM controller are renamed. The modifications described in the "SDRAM Interface" section of this application note are performed.
3. The files in the `\data` directory of each of the cores are also modified. These files are necessary to correctly handle the customer-specific functional blocks. Figure 15 shows the `pcores` directory setup, and an example in the reference design's included files can be used.

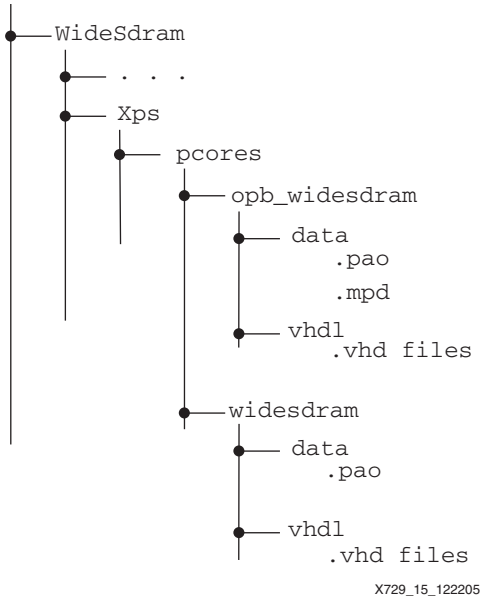


Figure 15: **XPS Project and pcores Subdirectory Setup**

4. The created base system XPS project is opened, and the original SDRAM controller is removed. When everything is correctly set up, the customized SDRAM controller is shown as a component that can be added to the processor design.
5. The customized SDRAM controller is added to the design, and the XPS tools are run.

Assembling the Project

This project is a normal ISE project. The steps to assemble it are:

1. The highest level VHDL file is created, the needed Digital Clock Manager (DCM) resources and all needed input and output buffers are instantiated.
2. The top-level processor design is copied as a component into the top-level HDL file of the design. All necessary connections are made.
3. A UCF file is created that matches the design and the hardware it needs to run on.
4. This design uses two DCM components: one for the system and one for the SDRAM. The DCM for the SDRAM receives its feedback path from the SDRAM as shown in Figure 16. The proper syntax is put into the UCF file to support this form of feedback.

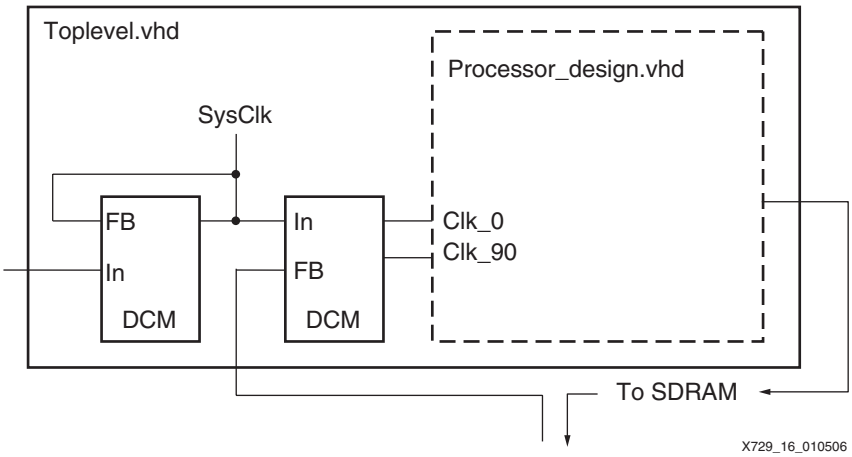


Figure 16: **Top-Level Design Setup**

5. The ISE tools are run.

A 32-bit MicroBlaze processor is connected via the OPB CoreConnect bus to an external, 64-bit SDRAM. The MicroBlaze processor can access the memory in two consecutive reads or writes.

Reference Design

The reference design files are located in a ZIP file, which contains a full project, using the design discussed in this application note and the modified SDRAM controller. The ZIP file is available at: http://www.xilinx.com/bvdocs/appnotes/xapp729.zip.

All EDK IP is written in VHDL as is this project and its modified SDRAM controller. Verilog is not supported.

This project contains a top-level design comprising a MicroBlaze design as a hierarchical level. The flow uses all Xilinx software tools (ISE and EDK/XPS).

This reference design is implemented onto a ML470 (XITI) Xilinx board having a XC4VSX35-FF668 Virtex-4 FPGA connected to Micron MT48LC8M32B2 SDRAMs. As such, the UCF file of the reference design will reflect the pinout of the FPGA on the ML470 demo board.

Resources

When the modified SDRAM controller is synthesized, without the IPIF interface and without the backend processor design, it uses the following resources:

- sdram_controller
  - Registers 512
  - LUTs 390
  - I/O pads 323
- Command_statemachine
  - Registers 60
  - LUTs 162
- Init_statemachine
  - Registers 12
  - LUTs 5
- Data_statemachine
  - Registers 186
  - LUTs 150
- Io_registers
  - Registers 213 — This number also contains registers normally buried in the FPGA because they are connections to the IPIF interface.
  - LUTs 1
- Ipic_interface
  - Registers 8
  - LUTs 13

References

The following documents provide supplemental material useful to this application note:

1. DS426, *OPB Synchronous DRAM (SDRAM) Controller Data Sheet*
2. Micron, MT48LC8M32B2 *256Mb x 32 Synchronous DRAM Data Sheet*

Conclusion

This project shows that existing IP delivered with the EDK tools can be modified and tuned fairly easily. Xilinx provides many powerful IP blocks for free that, due to the open VHDL format, can readily be adapted for a range of different needs.

Appendix A

This appendix provides two example schematics for connecting two different memory devices to a Virtex-4 SX35-FF668 FPGA.

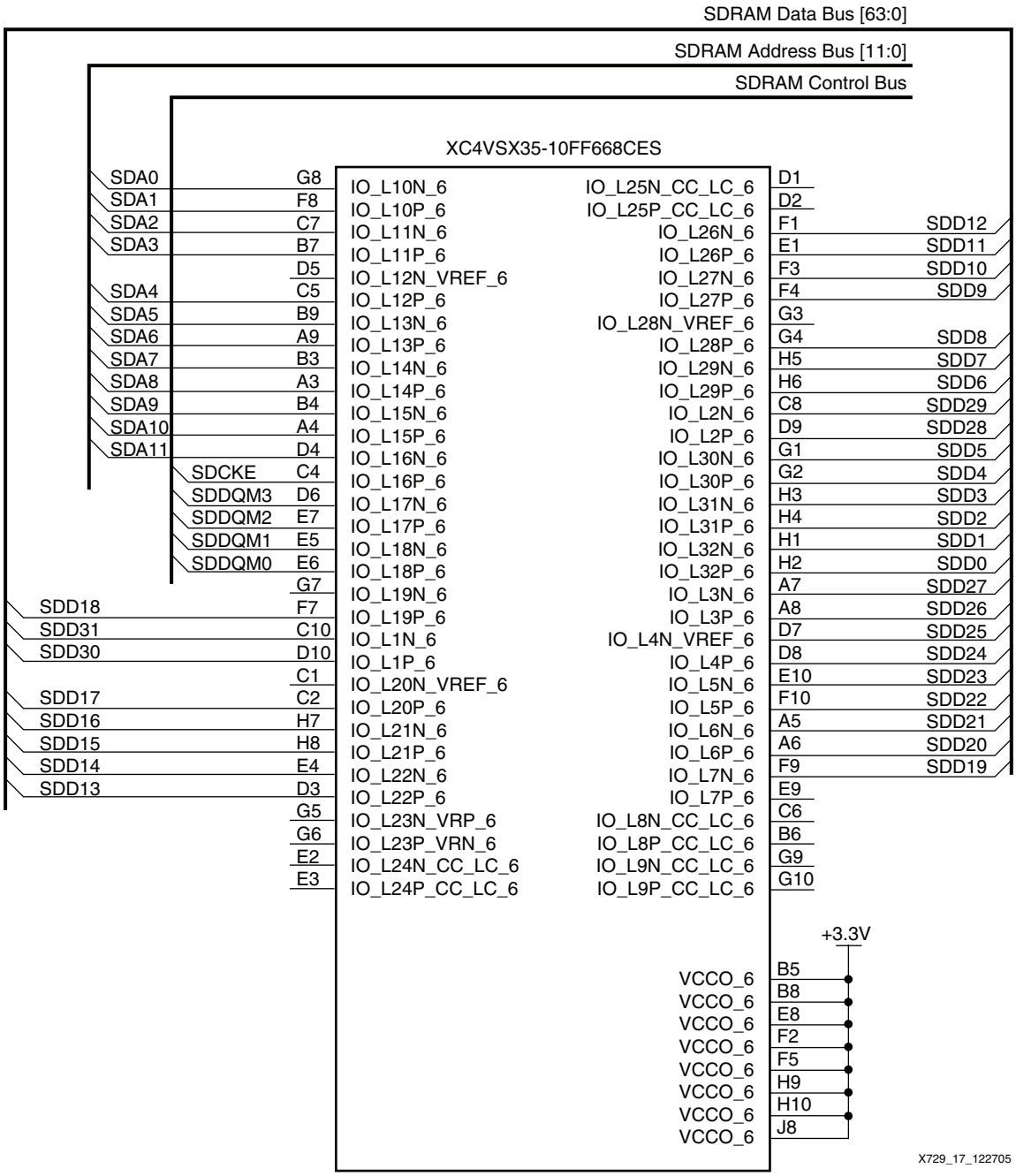


Figure 17: **Example Schematic (Bank 6)**

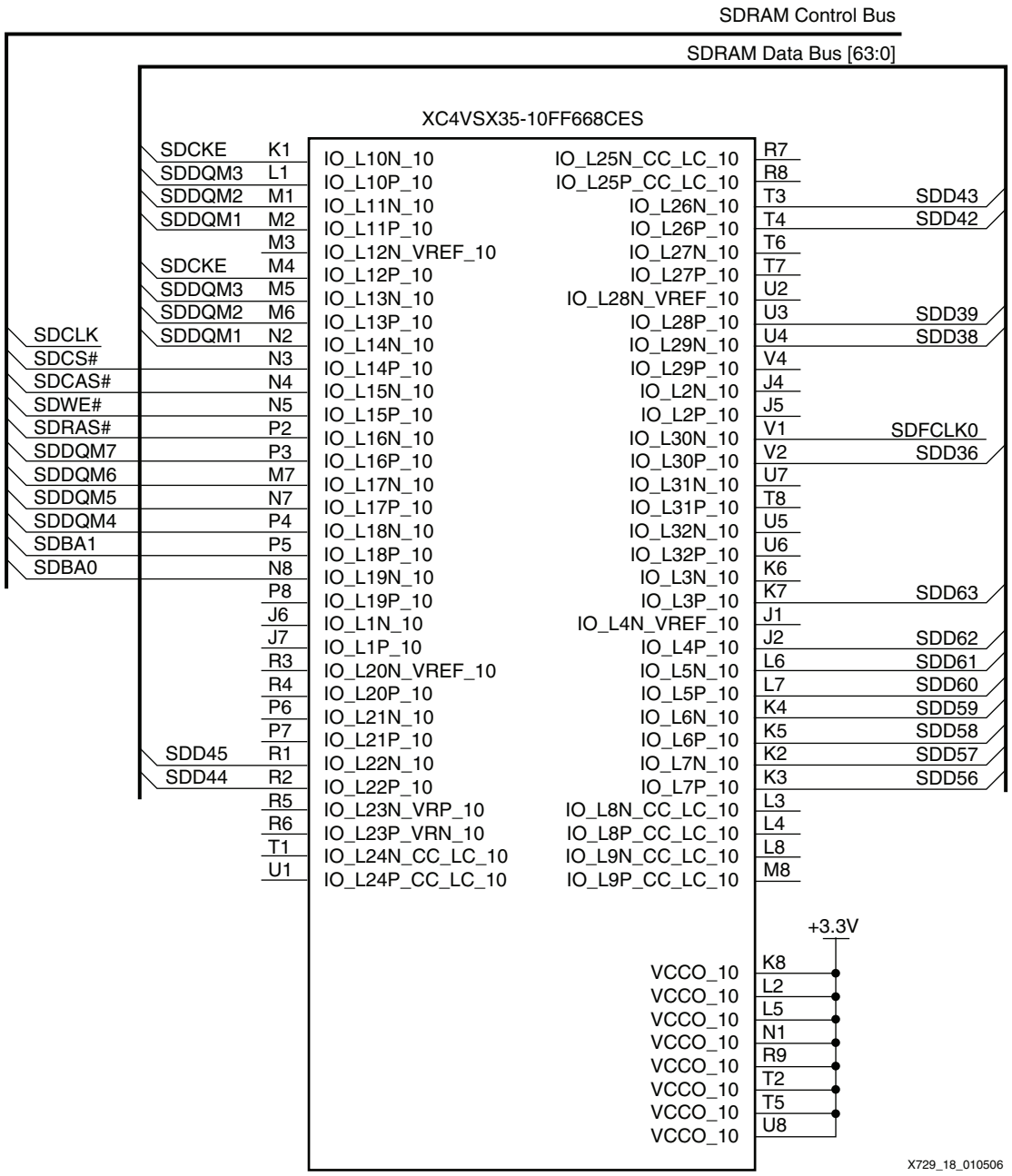


Figure 18: **Example Schematic (Bank 10)**

Appendix B

This appendix provides parameter settings for the Micron MT48LC8M32B2 256 Mb x 32 SDRAM. The data sheet for this SDRAM component is included in the reference design's ZIP file.

The parameters for the CoreConnect OPB bus are:

```
C_FAMILY            virtex4
C_BASEADDR          0x00000000
C_HIGHADDR          0x007FFFFF
C_OPB_DWIDTH        32
C_OPB_AWIDTH        32
```

The parameters for the interface are:

```
C_INCLUDE_BURST_SUPPORT  1
C_INCLUDE_HIGHSPEED_PIPE 1
C_USE_POSEDGE_OUTREGS    0
```

The parameters for the SDRAM hooked to the interface are:

```
C_SDRAM_DWIDTH      32
C_SDRAM_AWIDTH      12
C_SDRAM_COL_AWIDTH  9
C_SDRAM_BANK_AWIDTH 2
C_SDRAM_WIDEBUS     1
```

The Micron SDRAM parameters from the data sheet are:

```
C_SDRAM_TRAS             42000   min. value for -6 and -7
C_SDRAM_TMRD             2
C_SDRAM_TWR              15000   min. 14 ns in -7 SDRAM
C_SDRAM_TCCD             1
C_SDRAM_TRC              75000   min. 70 ns in a -7 SDRAM
C_SDRAM_TRFC             75000   min. 70 ns in a -7 SDRAM
C_SDRAM_TRCD             20000   min. 20 ns in a -7 SDRAM
C_SDRAM_TRRD             15000   min. 14 ns in -7 SDRAM
C_SDRAM_TRP              20000   min. 20 ns in a -7 SDRAM
C_SDRAM_TREF             64
C_SDRAM_CAS_LAT          3       other values are: 1 and 2
C_SDRAM_REFRESH_NUMROWS  4096
```

Appendix C

Examples of both MHS and MSS files are located in the `pcores` directory of the XPS project.

MHS File

The MHS file contains the following major sections:

- A section where all I/O ports are declared

  The port name should be renamed with a meaningful name. It is best is to give the port the same name as the connected net.

  For example: PORT my_name = my_name
- The same number of sections as there are components in the processor design. Each section starts with `BEGIN <component name>` and ends in `END`.
  - The INSTANCE name in each section is renamed to a meaningful name for the design.
  - If the component has access to the external world, the port name is changed to the same name as given in the port section of the file.

MSS File

The file is split into three major sections:

- An OS section

  The “PROC_INSTANCE” parameter is changed to the chosen name of the processor.
- A processor section

  The name of the “HW_INSTANCE” is changed to the name of the processor.
- Driver sections for all peripherals

  The “HW_INSTANCE” name is changed to the chosen peripheral name for all sections.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/05/06	1.0	Initial Xilinx release.
04/04/07	1.0.1	• Modified last paragraph of the “Reference Design” section. • New design files posted with this revision.