Version for EDK 10.1.03 as of June 14, 2010

Goals

- To familiarize with overall architecture of multiprocessors
- To understand how to synchronize data among processors using interprocessor peripheral
- To understand how to access external memory with several processors running simultaneously
- To learn how to configure debug module UART as other processor standard input output interface other than RS232 UART peripheral

Requirements

- If you are using Virtex-II Pro XUP Evaluation Platform before, you will need to utilize a more advanced version of library. Please download the EDK 10.1 Board Definition File for XUPV2 Pro Development System from Xilinx website.
- Please check the memory size of your DDR SDRAM on your board. If it is **KVR266X64C25/256** memory module, you can configure the program as default; however, if the module is **KVR266X64C25/512**, please follow the instruction on 512MB DDR SDRAM Fix after you start BSB.

Preparation

- Please read the document Designing Multiprocessor Systems in Platform Studio to understand the architecture of multiprocessor and interprocessor communication peripherals.
- Please read the document Dual Processor Reference Design Suite to better understands how to create multiprocessor structure using EDK. (Note: the reference note is not based on XUPV2P, so the sample design suite will not work on this board.)
- Please download the ml505_dual_mb design suite and read the sample codes in **sw** folder. (Note: You will need to sign up an account to download the design suite. You may ignore part of the codes that are written for PowerPC processor.)

Configure Hardware

1. Using the Base System Builder Wizard (BSB), create a new design and set the project peripheral repositories to **virtex2pro_board_lib/lib** file after you extract the **EDK 10.1 Board Definition File**. In the board selection part, choose **XUP Virtex-II Pro Development System** as board name.
2. Select microblaze with frequency *100.00Hz* as DDR SDRAM will need higher system frequency to be functional.
3. Select *On-chip H/W* debug module, local memory *8KB* and enable cache setup.
4. For this tutorial, you only need **RS232 Uart** and **DDR_SDRAM** with MPMC peripheral. Uncheck the other IO devices in the interface.
5. Check the **ICache** and **DCache** to allow cache memory configuration in the external memory. Leave the cache memory size as default.

6. Uncheck the **memory test** and **peripheral selftest** to shorten the compilation time. Leave the standard IO and boot memory as default. (Note: Changing the boot memory to external memory will cause fatal error in Microblaze, so leave it as local memory configuration.)

7. Now, create a similar Microblaze processor by adding peripherals and connect them manually as instructed in step 8. (Note: remember to check your DDR SDRAM as stated in Requirement section above before you proceed.)

8. Include the following peripherals (in **IP Catalog**) and rename them as ≺peripheral≻_≺no.≻ so that you can easily recognize which processor is referred to:-

   Bus and Bridge

   - 1 PLBV46 to PLBV46 Bridge
   - 1 Processor Local Bus (PLB) 4.6
   - 2 Local Memory Bus (LMB) 1.0

   Clock, Reset and Interrupt

   - 2 XPS Interrupt Controller

   DMA and Timer

   - 2 XPS Timer/Counter

   Interprocessor Communication

   - 1 XPS Mutex
   - 1 XPS Mailbox

   Memory and Memory Controller

   - 2 XPS BRAM controller
   - 2 LMB BRAM Controller
   - 2 Block RAM (BRAM) Block

9. Instead of making 3 connections (1 PLB and 2 XCL) to the MPMC for each processor, we will use only 2 XCL connections. This method causes a processor to make uncached memory accesses over XCL instead of PLB to reduce system size.

10. Double click on the **microblaze_0** in **System Assembly View**, scroll to **Cache**, enable both caches if you haven't done so, and enable *Use Cache Links for all I-Cache Memory Accesses* and *Use Cache Links for all D-Cache Memory Accesses*. (Note: You may change the Cache Line Length if you prefer to speed up the system. If you do so, make sure you change all of them in the processor setting and DDR SDRAM setting in **Port Configuration**.)

11. Change the base address and high address of **I-cache** and **D-cache** to your DDR SDRAM base address and high address. By default, 256 MB memory module address ranges from *0x90000000 to 0x9FFFFFFF* while 512 MB memory module address ranges from *0xA0000000 to 0xBFFFFFFF*.

12. Double click on the **microblaze_0**, scroll to **Debug** and enable *Microblaze debug module interface*. Repeat the same procedure in steps 10 to 12 for **microblaze_1**.

13. Double click on the **DDR SDRAM**, create four XCL connections in **Base configuration**. Scroll to **Advanced** ⟶ **EEC/PM/PHY** and uncheck *Enable Static PHY*.

14. Double click on the **debug module** to increase the *number of MicroBlaze debug ports* to the number of processors you have.

15. Now, switch to **System Assembly View**. Click on the + sign on the left of **Bus Interfaces** to view all the bus connections. Connect the bus for **microblaze_1** in the same manner as **microblaze_0** corresponding to their number. For example, the **IPLB** under **microblaze_0** is connected to **mb_plb_0**, so you must connect **IPLB** under **microblaze_1** to **mb_plb_1**.

16. In the same manner as previous step, configure the bus interfaces, ports and assign appropriate addresses to all the peripherals and processors. You should refer to the **System Assembly View** on ml505_dual_mb design suite as a guide to connect and rename all of them. (Note: Except the DDR SDRAM, the rest of the configurations are almost the same.)

17. You should have a similar block diagram as shown below:-

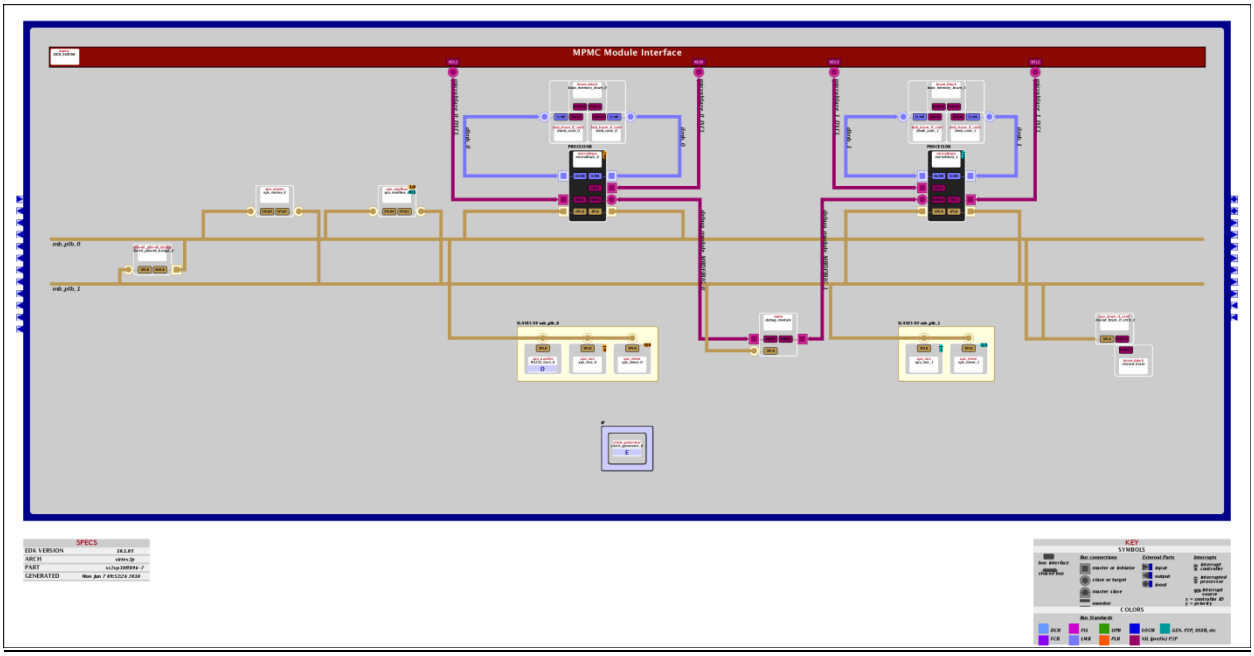

Diagram 1: Dual Processor

Configure Software

1. Scroll to **Software** ⟶ **Software Platform Setting** ⟶ **OS and Libraries**, make sure the stdout and stdin for **microblaze_0** is *RS232_Uart* and for **microblaze_1** is *debug_module*.

2. Scroll to **Debug** ⟶ **Debug Configuration** ⟶ **JTAG UART**, enable *JTAG UART* for **microblaze_1**.

3. From the ml505_dual_mb design suite, copy all the C files in **sw** folder to your project directory. As usual, use **Add Software Application Project** to create 8 different required project files and include their respective source files accordingly. (Note: Refer to ml505_dual_mb design suite **Applications** tab on creating all 8 project files.)

4. Double click on **Project: shm0**, enable **Use Default Linker Script** with:-

Memory Module	256MB	512MB
Program Start Address	0x90000000	0xA0000000
Stack Size	0x3000	0x3000
Heap Size	0x3000	0x3000

5. Double click on **Project: shm1**, enable **Use Default Linker Script** with:-

Memory Module	256MB	512MB
Program Start Address	0x91000000	0xA1000000
Stack Size	0x3000	0x3000
Heap Size	0x3000	0x3000

6. By default, the linker script is using local memory to store all **.elf** data which may not be enough for multiprocessor. The offset address assigned is to ensure that the processors are using data from DDR SDRAM. Two processors should not be allocated the same starting address to avoid data overwriting.

7. Repeat the same procedure for all project files in step 4 and 5.

Running and Debugging Design

1. Generate bitstream for hardware configuration and build all user applications for software application. (Note: You may face several errors in this step, make sure that you follow all the steps above. Refer to the last page on **Common Encountered Bugs** to fix some common problems.)

2. After successful compilation, download bitstream to XUPV2P board and open up a XMD shell. Choose **microblaze_0** when a window pop up, leave the connection type as **hardware** and click **OK**.

3. Open up **hyperterminal** (windows) or **minicom** (linux) to connect to your communication port.

4. After the XMD auto-connect to **microblaze_0**, input the following instructions:-

   `connect mb mdm -debugdevice deviceNr <jtag chain no.> cpunr <processor no.>`
   - This will connect the XMD to other processor. Normally, the jtag chain no. is **3** and processor no. is **2** in this case.

   `connect mdm -uart`
   `terminal -jtag_uart_server 4321`
   - This will connect your processor to microblaze debug module (mdm) uart device. The latter instruction will open a TCP terminal tunnel in port 4321.

   `targets`
   - This command is to check which devices the XMD is connected to. Make sure you have both microblaze processors and mdm uart device connected in this stage. The XMD specifies a unique number for each device.

5. Open up another **hyperterminal** in windows. Select connection *TCP/IP (Winsock)* and enter *localhost* as **host address** and *4321* as **port number**. You should be able to see a notice in XMD saying hyperterminal connection detected. (Note: in Linux OS, open up a **terminal**, enter *telnet* followed by *open localhost 4321* to establish connection.)

6. Enter the following commands to run the program:-

   `targets 0`
   - switch your targeted device to device 0 which is MicroBlaze≺1≻> or microblaze_0 in this case. An asterisk * indicates which device you are currently accessing.

```
dow shm0/executable.elf
run

targets 1
dow shm1/executable.elf
run
```

7. If everything is configured correctly, you should be able to see the same output on Dual Processor Reference Design Suite document. Try to download other software projects file to verify your design is correct.

Extend the Design

1. Now, you can use the similar method to create 4 microblaze processors connected to MPMC.
2. Modify your codes on sharedconsole.c to allow four microblazes running simultaneously.
3. Instead of using shared BRAM memory, you will need to use DDR SDRAM as shared memory for four processors. (Note: Microblaze processor does not provide cache coherency support, thus you must either enforce your coherency in software or disable cache.)
4. You should have a similar block diagram as shown below:-

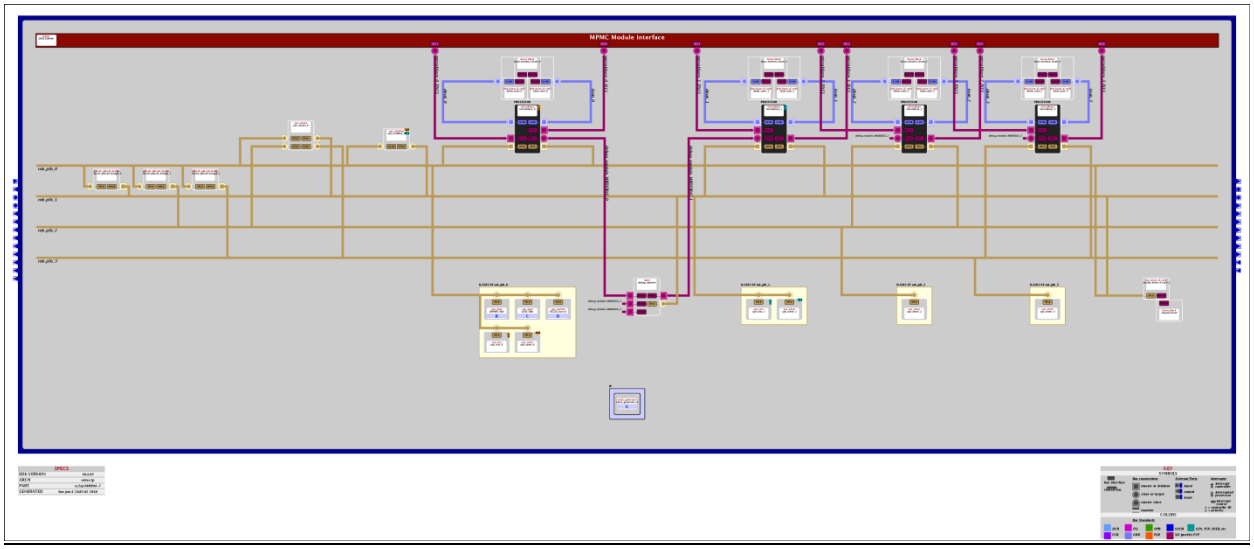

Diagram 2: Quad Processor

Common Encountered Bugs

- If you see "Cable is locked" problem, open your XMD shell and enter **xclean_cablelock** to release all the locks.
- If you encounter "too many RAMB are declared", reduce the size of all your BRAM. The maximum allowable memory size is 32KB.

- When using plbv46_plbv46 bridge, its address range must cover the address of the desired custom peripheral.

Prepared By,
Chia Chen Tan