

Lab 4: Master IP Design



National Chiao Tung University
Chun-Jen Tsai
5/8/2018

Lab Description

- ❑ In this lab, you will learn how to design a master IP to read and write the DRAM contents in burst mode
 - A sample IP will be given to you that demonstrates how to copy one 16-word burst of memory data from a source pointer to a destination pointer in the DDR memory
 - You must design a general-purpose data-copy DMA IP

- ❑ Demo to your TA on 5/23

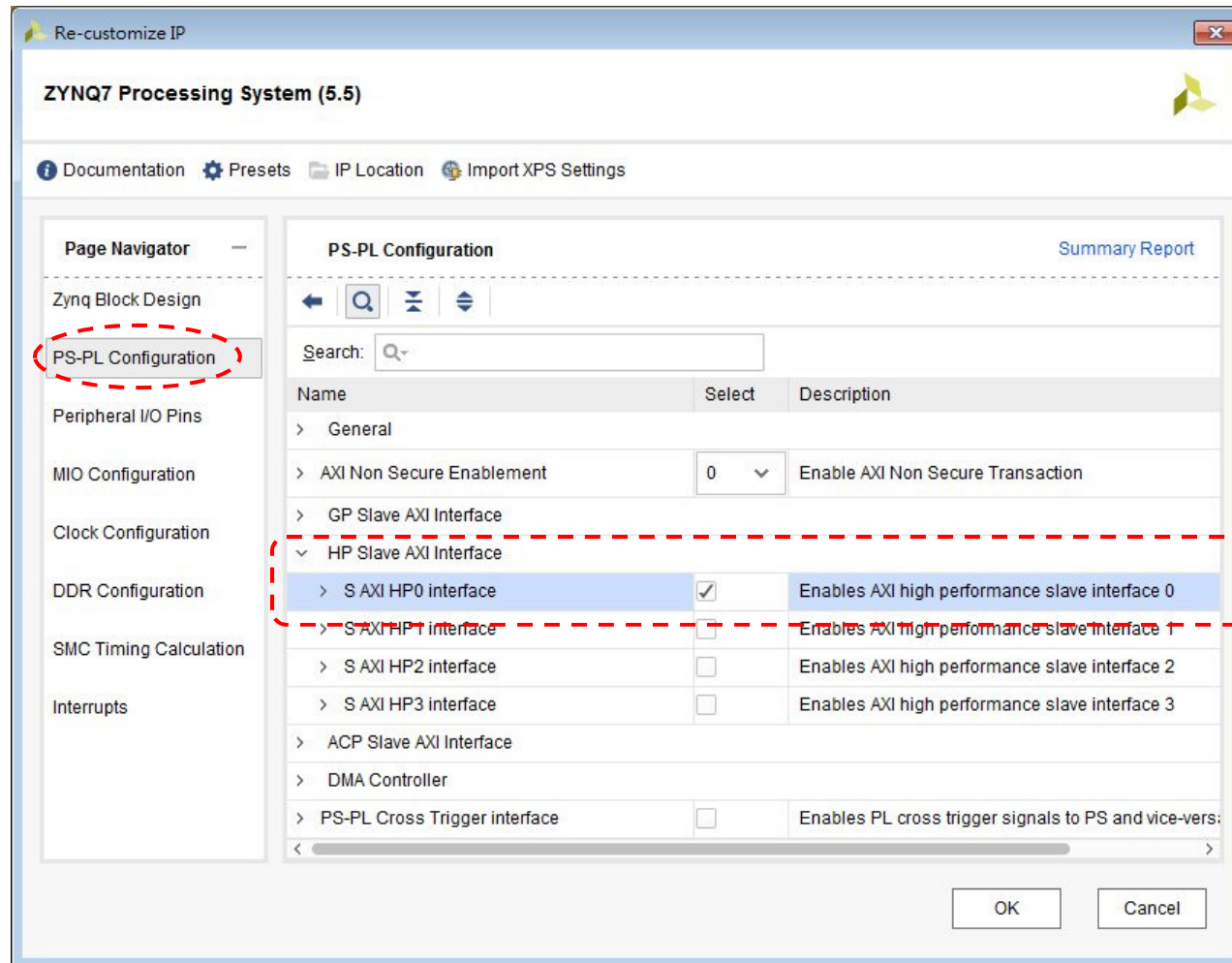
Enabling Zynq HP Data Port (1/2)

- ❑ First, create a basic Zynq block diagram

The screenshot displays the Vivado 2017.4.1 interface. The main window shows the 'BLOCK DESIGN - lab4 *' tab with a 'Diagram' view. A 'Re-customize IP' dialog box is open, showing the 'ZYNQ7 Processing System (5.5)' configuration. The 'Page Navigator' on the left lists various configuration options, including 'Zynq Block Design', 'PS-PL Configuration', 'Peripheral I/O Pins', 'MIO Configuration', 'Clock Configuration', 'DDR Configuration', 'SMC Timing Calculation', and 'Interrupts'. The 'Zynq Block Design' section is currently selected, showing a detailed block diagram of the Zynq7 Processing System. The diagram includes components like the Application Processor Unit (APU), System Level Control Rags, GIC, DAP, DEVIC, DMA Channels, Config AE/SHA, IRQ, High Performance AXI 32bit/64bit Slave Ports, XADC, Processing System (PS), Programmable Logic (PL), and various interfaces like I/O MUX (MIO), Bank0 MIO (150), Bank1 MIO (53-16), I/O Peripherals, General Settings, SWGT, TTC, SMC, and SMC Timing Calculation. A red dashed circle highlights the 'ZYNQ7 Processing System' block in the diagram, with a red arrow pointing to it and the text 'Double-click' in red. The 'ZYNQ7 Processing System' block is shown with its ports: DDR, FIXED_IO, USBIND_0, M_AXI_GP0, TTC0_WAVE0_OUT, TTC0_WAVE1_OUT, TTC0_WAVE2_OUT, FCLK_CLK0, and FCLK_RESET0_N. The 'ZYNQ7 Processing System' block is also shown in the 'Re-customize IP' dialog box, with a red dashed circle highlighting it and a red arrow pointing to it and the text 'Double-click' in red.

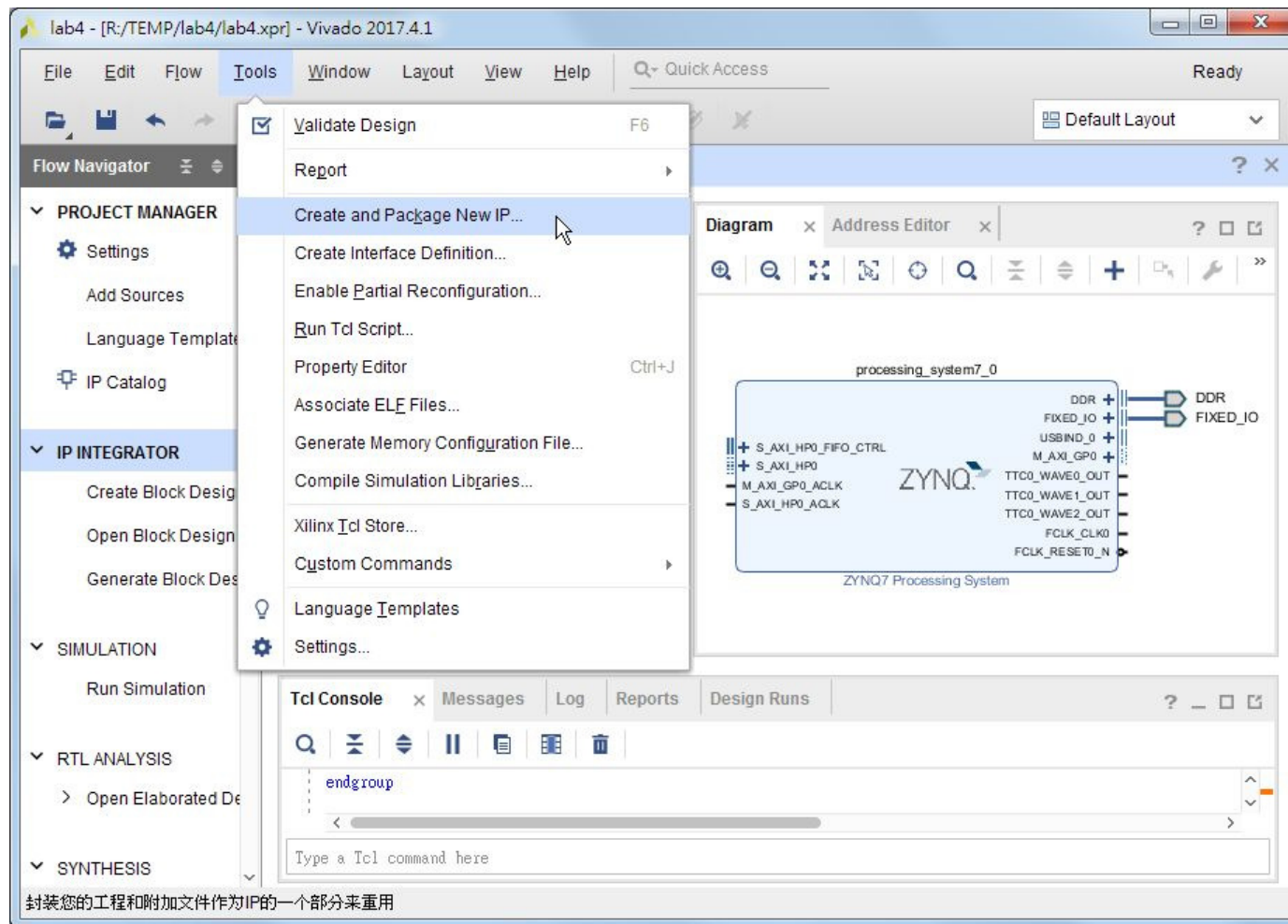
Enabling Zynq HP Data Port (2/2)

- ❑ Enable the Zynq High Performance (HP) Data Port:



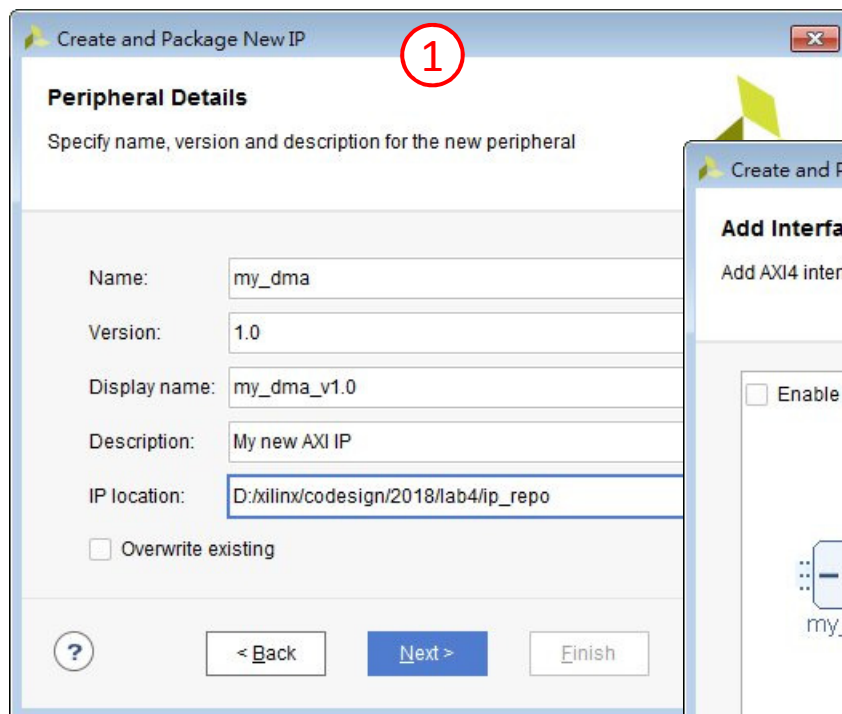
Master IP Creation (1/3)

- Now, create a master IP template:



Master IP Creation (2/3)

- ❑ To create a master IP, in addition to the default slave interface, you must add a master interface:

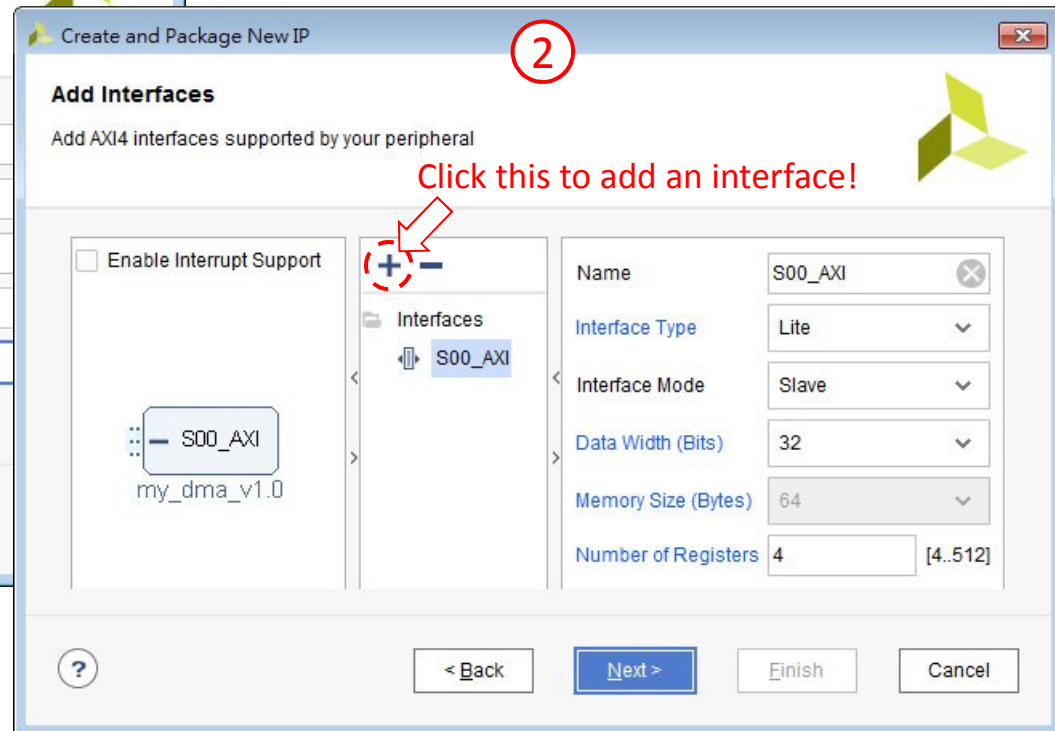


Create and Package New IP (1)

Peripheral Details
Specify name, version and description for the new peripheral

Name: my_dma
Version: 1.0
Display name: my_dma_v1.0
Description: My new AXI IP
IP location: D:\xilinx\codesign\2018\lab4\ip_repo
☐ Overwrite existing

< Back Next > Finish



Create and Package New IP (2)

Add Interfaces
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces
+ - S00_AXI

my_dma_v1.0

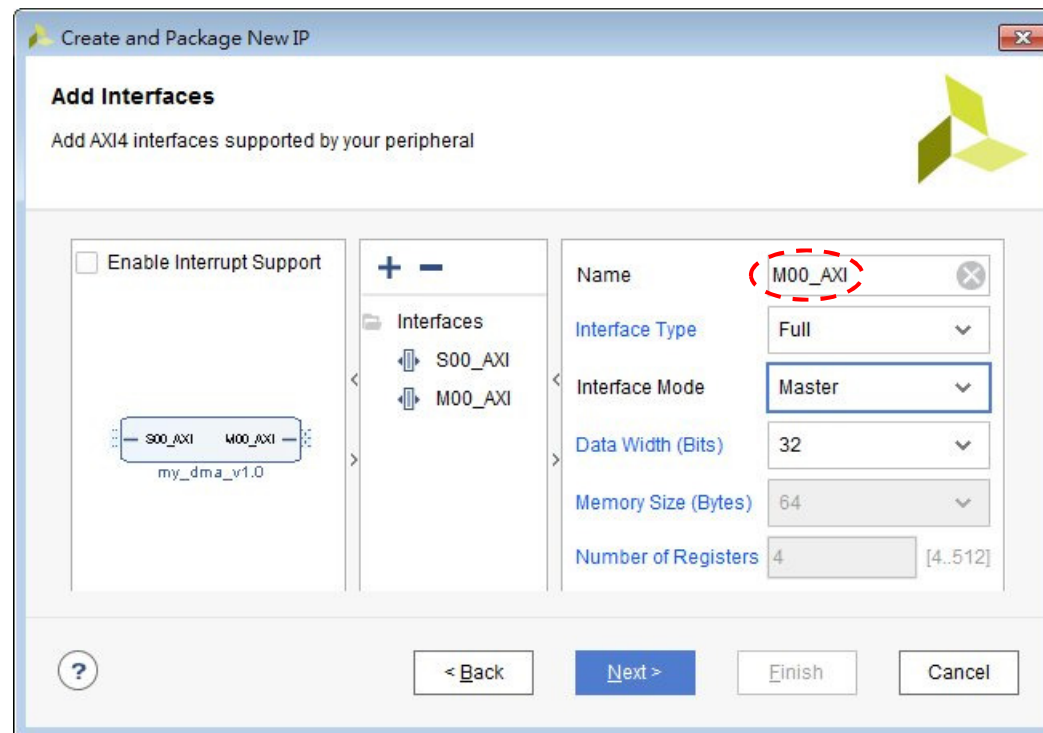
Name: S00_AXI
Interface Type: Lite
Interface Mode: Slave
Data Width (Bits): 32
Memory Size (Bytes): 64
Number of Registers: 4 [4..512]

< Back Next > Finish Cancel

Click this to add an interface!

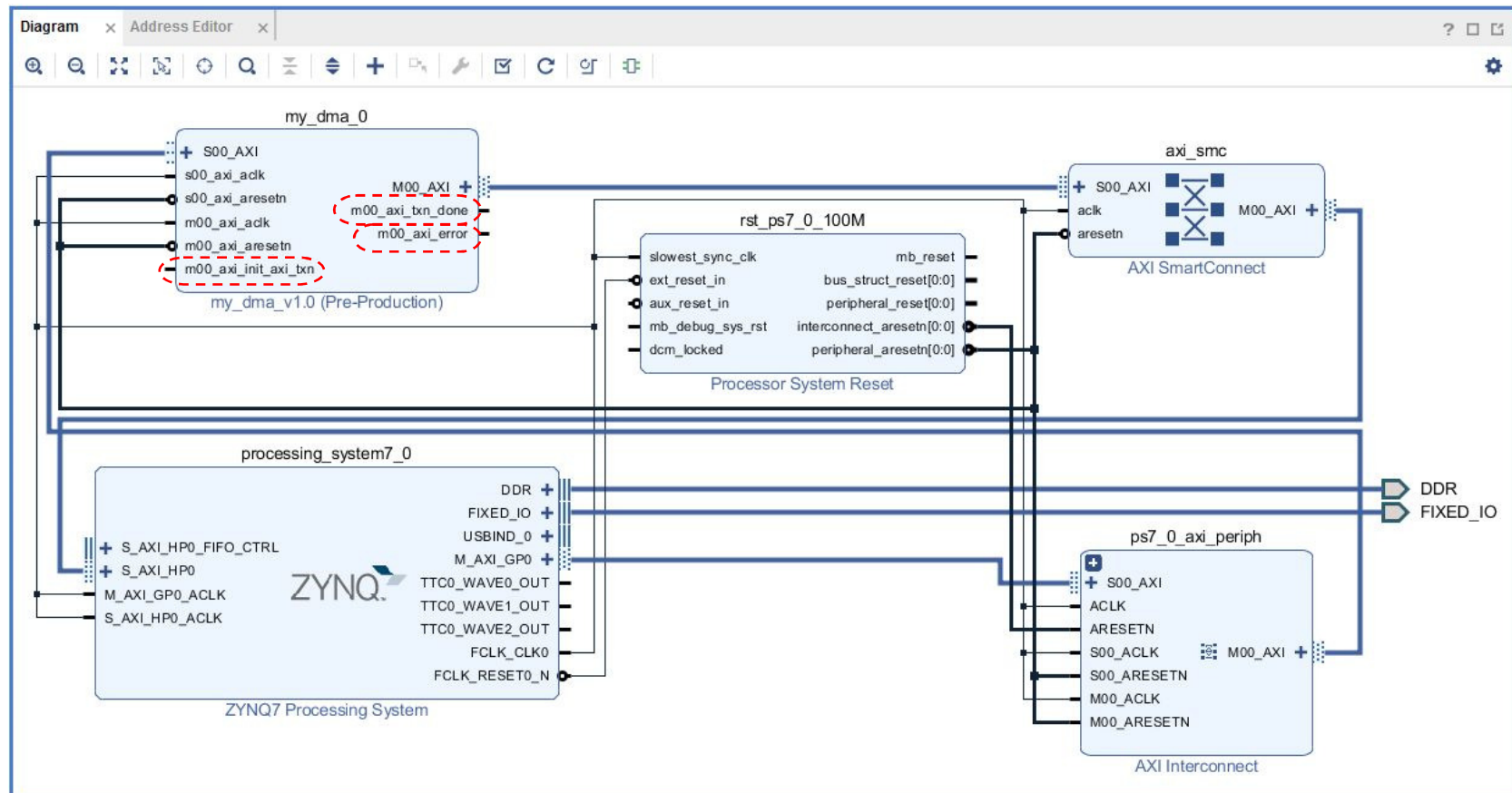
Master IP Creation (3/3)

- ❑ For lab4, we should create a master IP, my_dma.



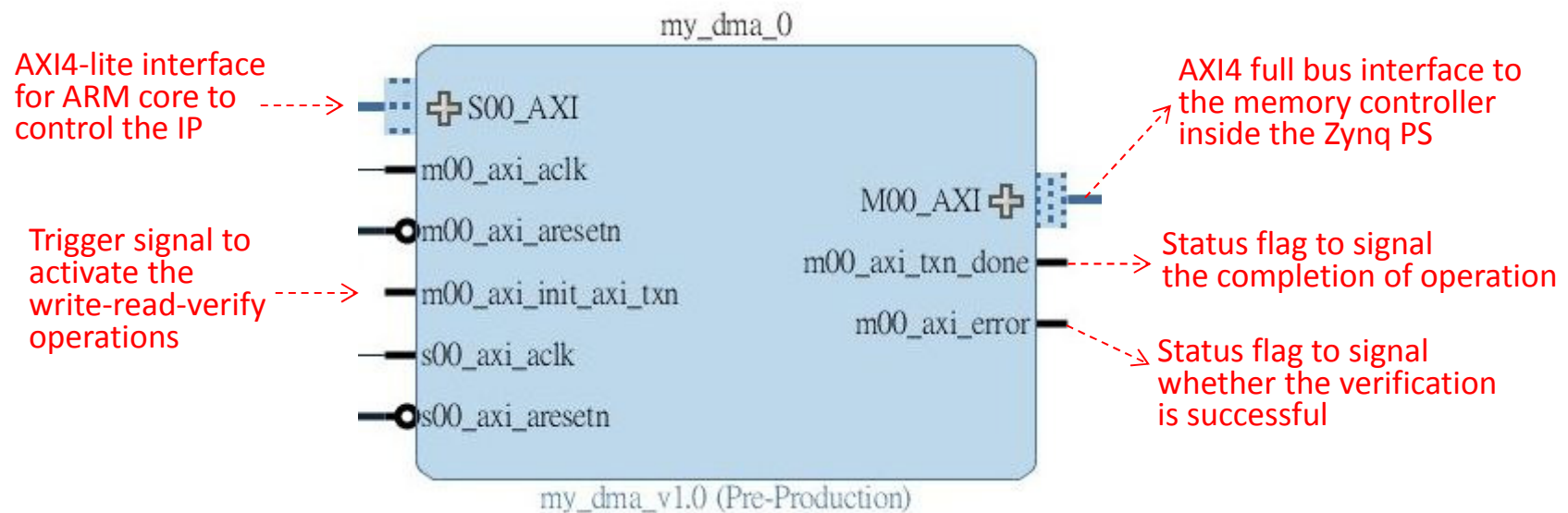
The Full Block Diagram

- ❑ Now, you can “Run Connection Automation.”
 - Note that three ports: `init_axi_txn`, `txn_done`, and `error`, are not connected



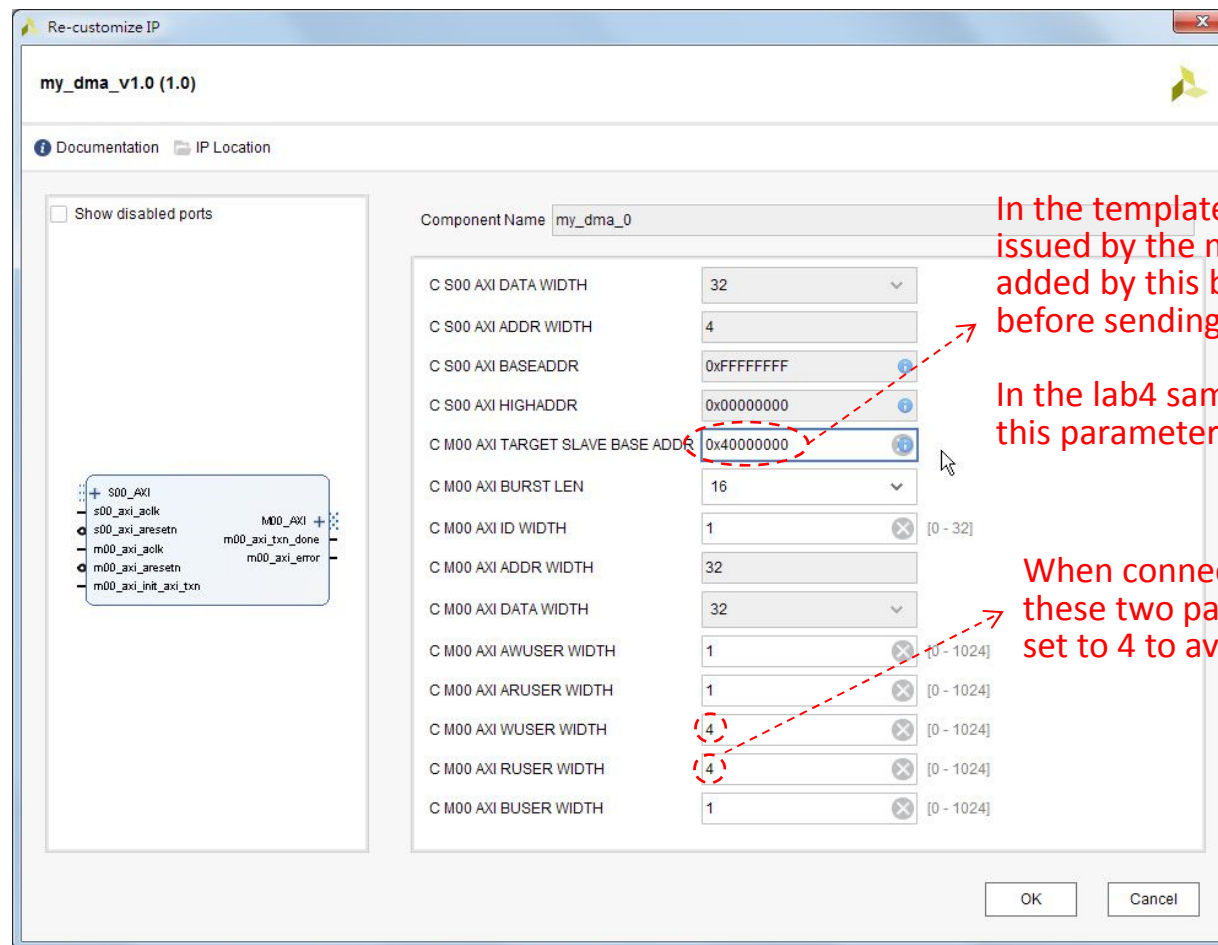
Default RTL Template of Master IP

- ❑ The default RTL template of the master IP writes 4KB of data block to the external memory, then read back the data block from the external memory for verification
 - All write/read operations use AXI4 burst transactions



Parameters in the Template IP

- ❑ We may change a parameter of the master interface:
 - Double-click the `my_dma_0` IP block to get to the dialog box



In the template IP, all addresses issued by the master IP will be added by this base address before sending the bus request.

In the lab4 sample IP, we ignore this parameter.

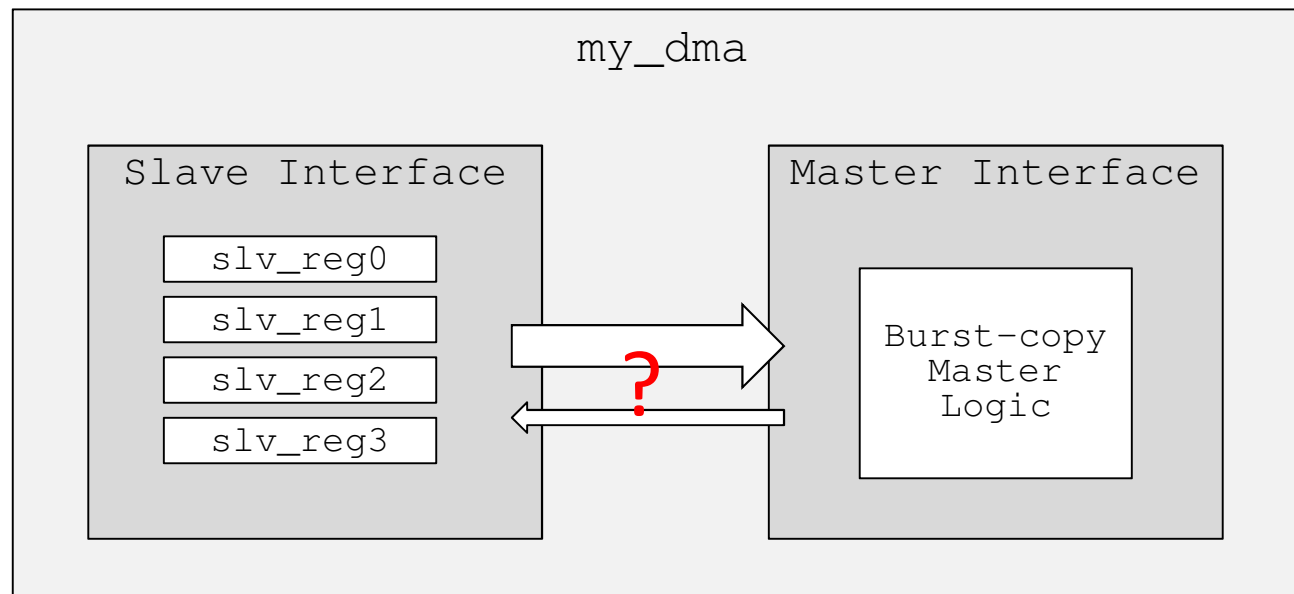
When connected to an AXI_SMC, these two parameters should be set to 4 to avoid critical warnings.

Modifying the Template IP

- ❑ We must modify several places of the template IP to create the DMA IP of Lab4:
 - For `my_dma`, we do not need the three control/status ports: `init_axi_txn`, `txn_done`, and `error`
 - The template IP has four sequential states: `IDLE`→`INIT_WRITE`→`INIT_READ`→`INIT_COMPARE`, but our logic must perform “read” before “write,” and we do not need the verification step
 - The template IP always reads/writes 4KB of data in integer number of bursts, we must support variable-size data
 - The template IP has no internal buffer
 - The master interface has no access to the slave interface registers

Slave/Master Register Sharing

- ❑ The logic `my_dma` has both the slave and the master interface, but they are in two different modules
 - The Zynq PS writes to the slave registers to control the IP, but the master module cannot see these slave registers without explicit connections

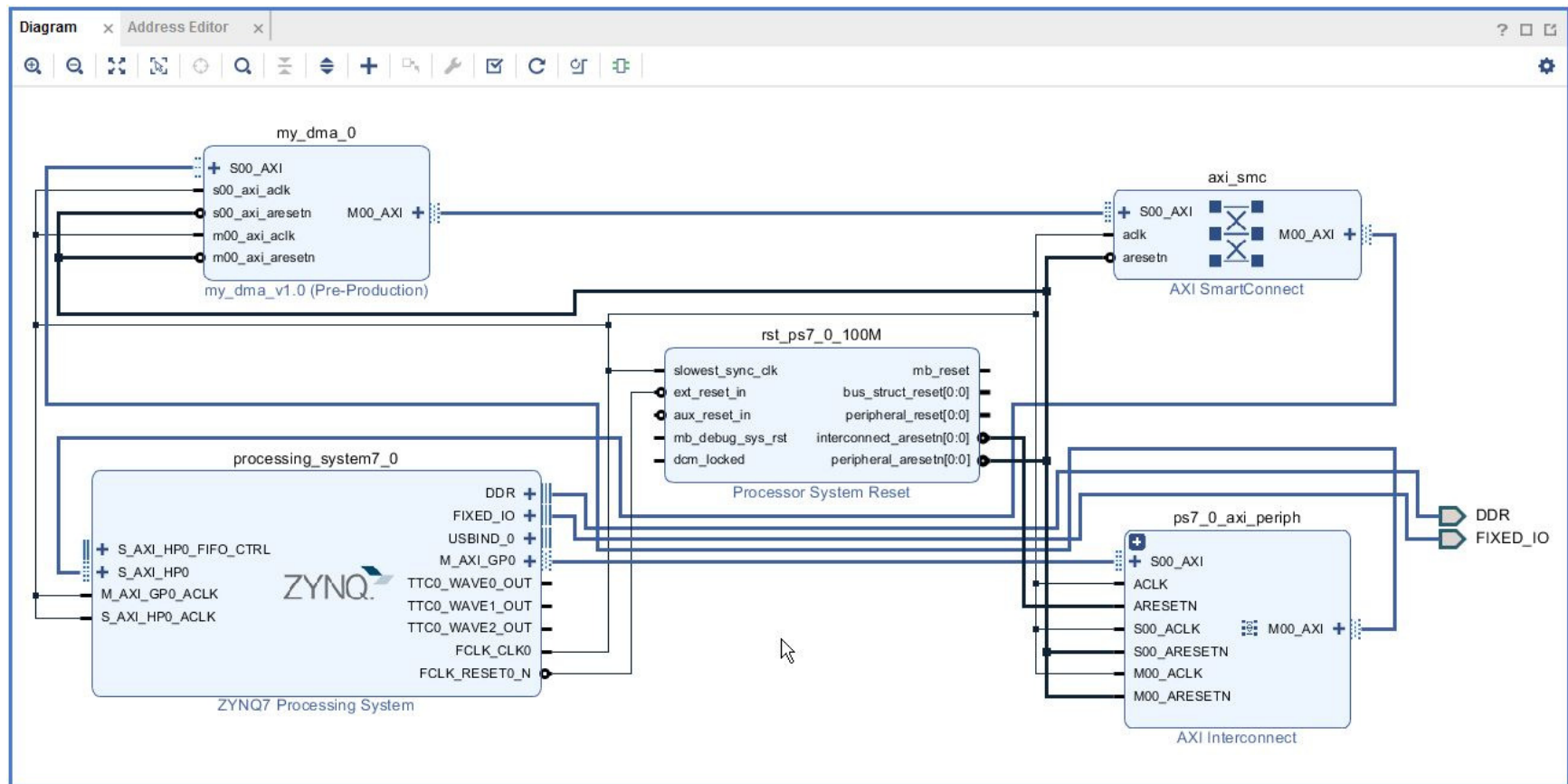


The Sample IP Package

- ❑ A sample IP is provided on the E3 website, in which:
 - The three unnecessary ports `init_axi_txn`, `txn_done`, and `error` are removed
 - The ports to pass the slave registers to the master interface module are implemented
 - The master interface always copies one burst of 16-words (64 bytes) of data from a source pointer to a destination pointer
- ❑ You can replace the template IP repository by the sample IP repository, and then upgrade the DMA IP in your block diagram

The Updated Block Diagram

- ❑ The updated block diagram with the new `my_dma` IP:



Your Goal in Lab 4

- ❑ In Lab4, you must modify the sample IP such that:
 - Its behavior is similar to the `memcpy()` function
 - To simplify the design, you can assume that the source and the destination addresses are 32-bit word-aligned, **the length is a multiple of 4 (bytes)!**
- ❑ Definitions of the four slave interface registers
 - `hw_active` – used to trigger HW and signal the completion
 - `dst_addr` – the destination address pointer
 - `src_addr` – the source address pointer
 - `len_copy` – the length of the transfer in bytes (a multiple of 4)

Cache Issues

- ❑ The memory accessed directly by the hardware cannot be cached by the CPU, otherwise, data inconsistency will happen
- ❑ There are some functions in Xilinx BSP (defined in `xil_cache.h`) that allow you to control the cache behavior of the CPU, such as:
 - `void Xil_DCacheFlushRange(unsigned addr, unsigned size);`
 - `void Xil_DCacheDisable(void);`
- ❑ In Lab6, we will show you an efficient way to maintain data coherence between HW and SW using the ACP of AXI4 bus protocol

The Software Interface

- ❑ The `hw_memcpy()` function that invokes the HW IP to perform data transfer should be as follows:

```
#include <xparameters.h>
#include "xil_cache.h"

volatile int *hw_active = (int *) (XPAR_MY_DMA_0_S00_AXI_BASEADDR + 0);
volatile int *dst_addr = (int *) (XPAR_MY_DMA_0_S00_AXI_BASEADDR + 4);
volatile int *src_addr = (int *) (XPAR_MY_DMA_0_S00_AXI_BASEADDR + 8);
volatile int *len_copy = (int *) (XPAR_MY_DMA_0_S00_AXI_BASEADDR + 12);

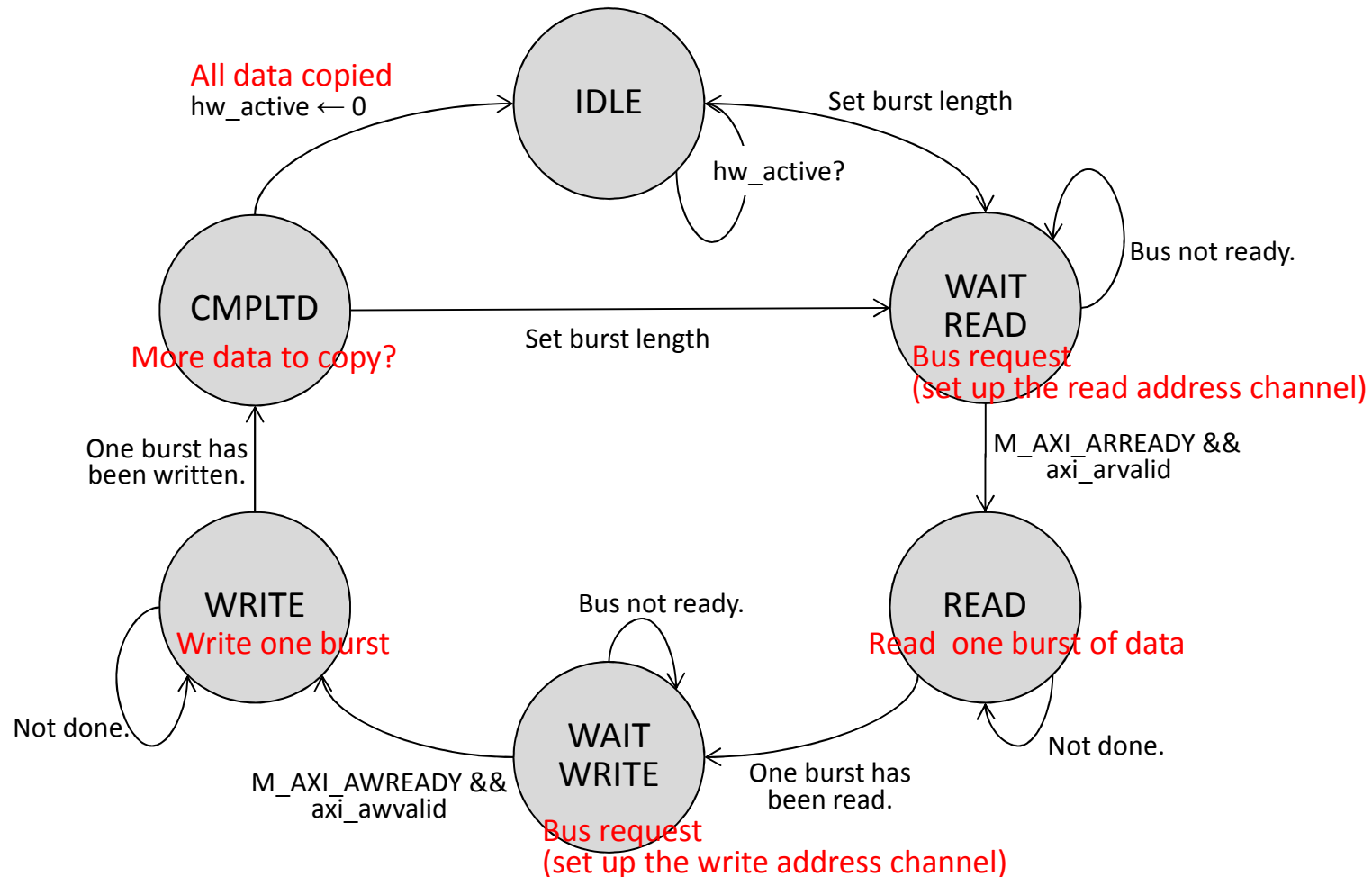
void hw_memcpy(void *dst, void *src, int len)
{
    *dst_addr = (int) dst; // destination word address
    *src_addr = (int) src; // source word address
    *len_copy = len;       // transfer size in bytes
    *hw_active = 1;        // trigger the HW IP
    while (*hw_active);    // wait for the transfer to finish
}
```

AXI4 Burst Transfer

- ❑ An AXI4 burst transfer let the master IP to transfer data to/from consecutive addresses of external memory, one word per each clock cycle
- ❑ To setup a burst transfer, you must set the burst length signal `M_AXI_ARLEN` (or `M_AXI_AWLEN`) of the read (or write) address channel
 - The burst length can be from 1 ~ 256
 - When the burst is long, some wait cycles may be inserted by the target device
 - It is recommended that each burst is ≤ 64 words

Suggested Controller of Lab4

- ❑ The FSM of your DMA IP may need six states:



Expected Performance

❑ DDR-to-DDR memory copy performance on Zynq:

- HW copy, single-word copy : 6.03 MBPS
- HW copy, 8-word burst copy : 39.18 MBPS
- HW copy, 16-word burst copy : 47.68 MBPS
- HW copy, 32-word burst copy : 62.32 MBPS
- HW copy, 64-word burst copy : 125.67 MBPS
- C code memcpy(), cache disable : 31.20 MBPS

❑ As a reference, memory copy within the cache is about ~ 730 MBPS

Reference

- ❑ Xilinx, Vivado Design Suite: AXI Reference Guide, UG1037, June 24, 2015

https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf