

Lab 4: Master IP Design



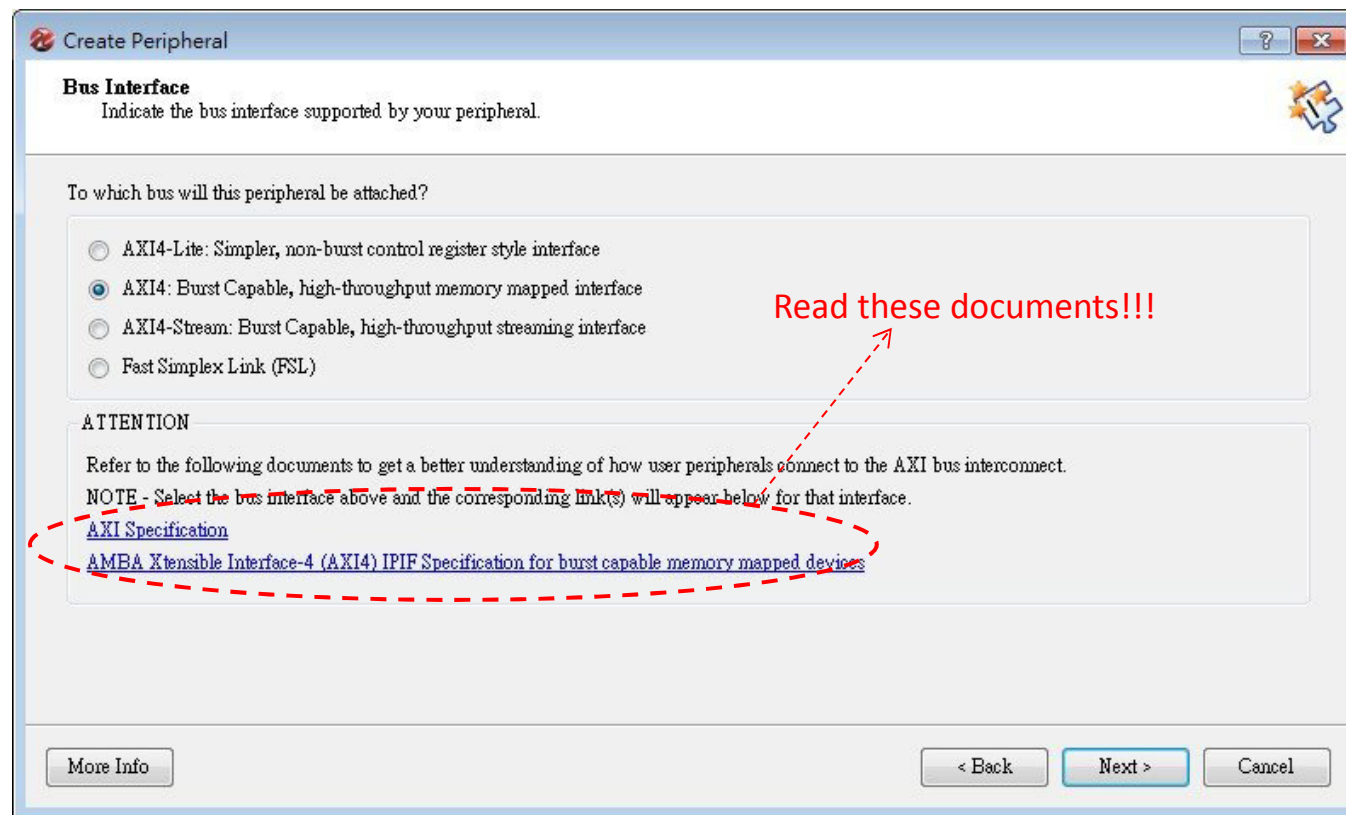
National Chiao Tung University
Chun-Jen Tsai
4/27/2016

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 logic will be given to you that demonstrates how to design a “memcpy” logic that copy a sequence of 32-bit word per bus transaction
 - You must design a logic that copy an 8×8 matrix of n -byte data from the DDR memory to an on-chip BRAM, and vice-versa
- ❑ Make an demo to your TA on 5/10

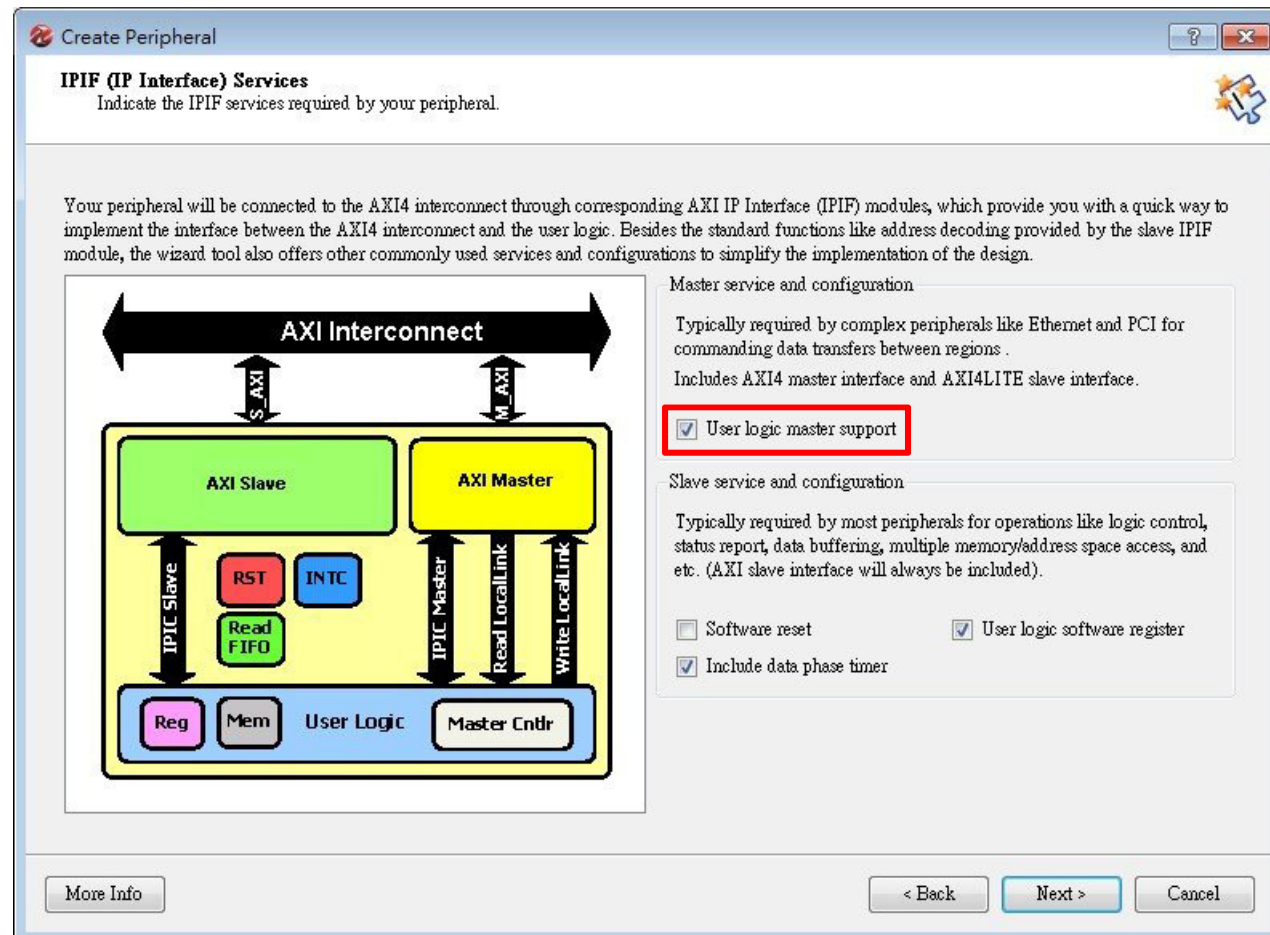
Master IP Creation

- ❑ An IP that can perform burst memory copy must be a master IP, you should select the AXI4 bus:



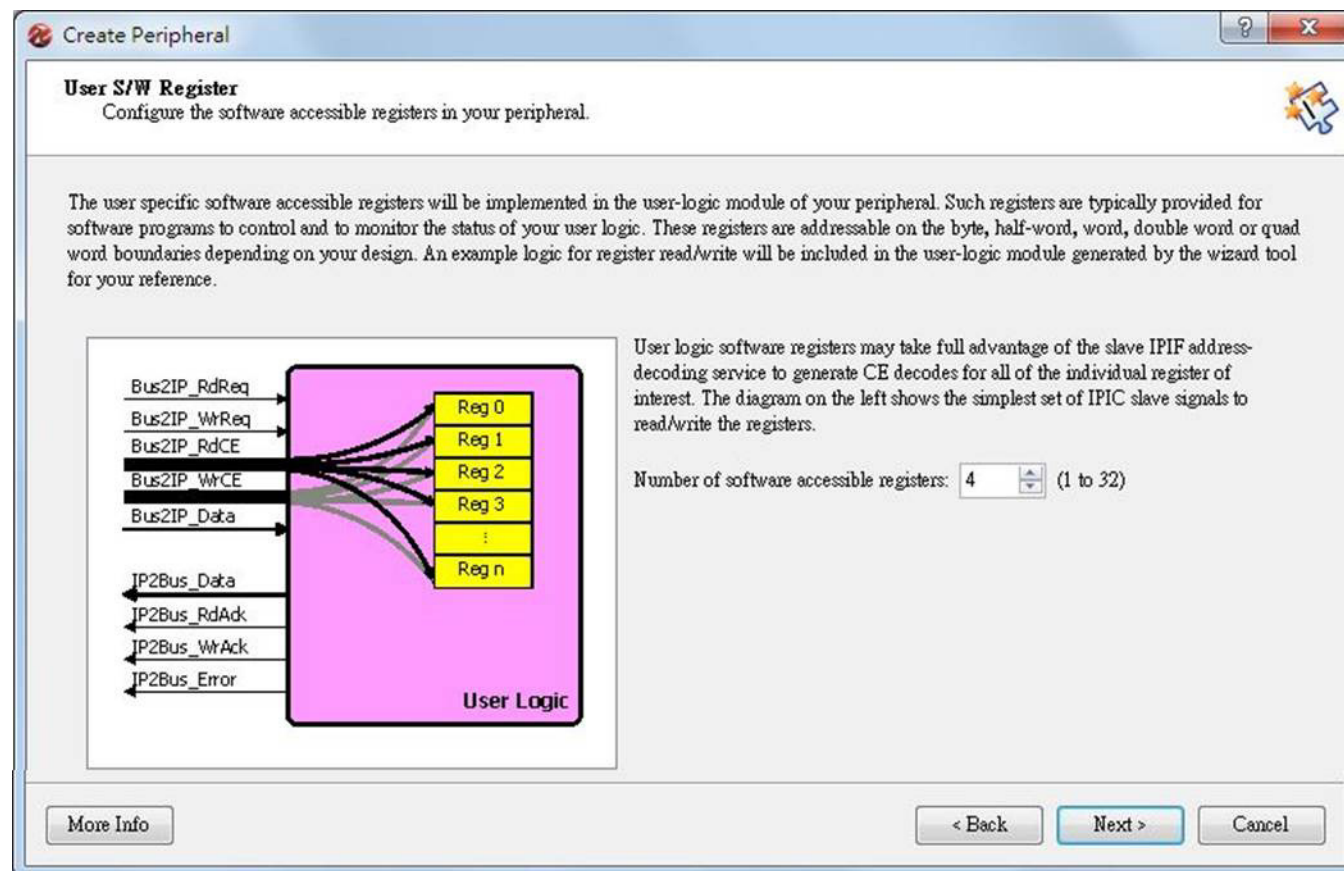
Enable Master IP Signals

- ❑ Master IP requires extra signals to acquire bus



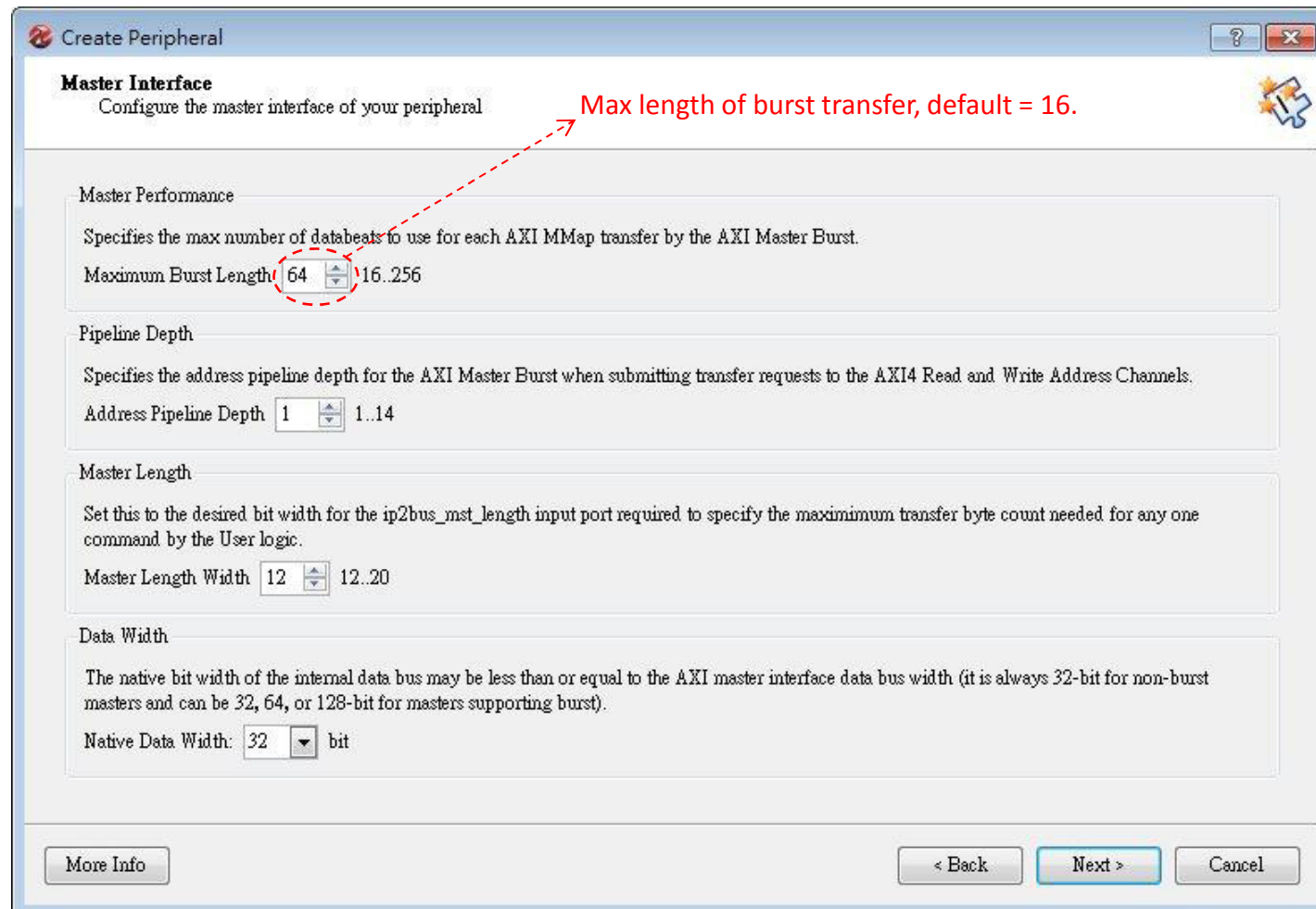
Select Register Interface

- ❑ 8×8 block-copy logic requires four parameters:
 - hw_active, block address, image stride, and bytes per pixel



Configure Master Interface

- ❑ Use the default values here:



Create Peripheral

Master Interface
Configure the master interface of your peripheral

Master Performance
Specifies the max number of databeats to use for each AXI MMap transfer by the AXI Master Burst.

Maximum Burst Length: 64 (dropdown) 16..256

Pipeline Depth
Specifies the address pipeline depth for the AXI Master Burst when submitting transfer requests to the AXI4 Read and Write Address Channels.

Address Pipeline Depth: 1 (dropdown) 1..14

Master Length
Set this to the desired bit width for the ip2bus_mst_length input port required to specify the maximum transfer byte count needed for any one command by the User logic.

Master Length Width: 12 (dropdown) 12..20

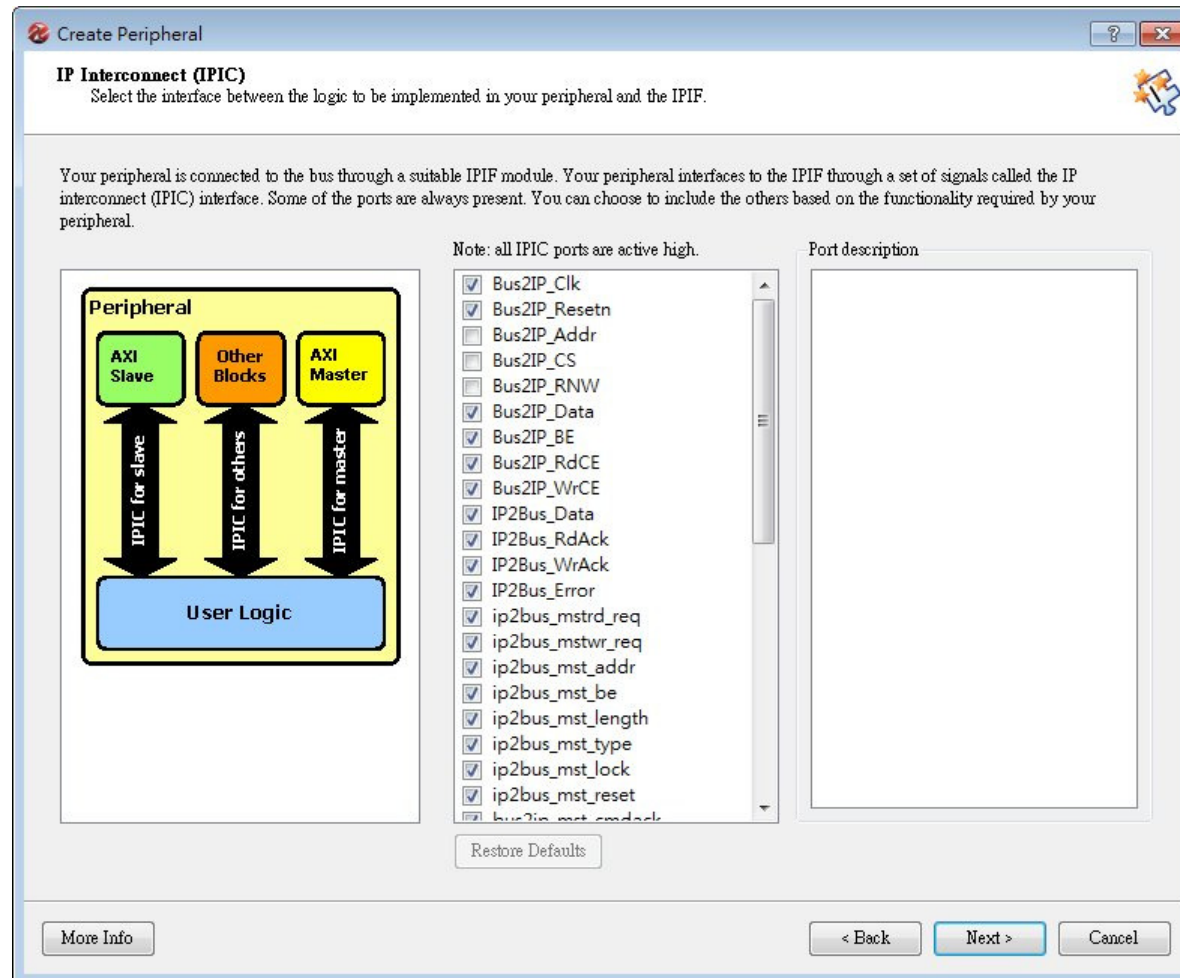
Data Width
The native bit width of the internal data bus may be less than or equal to the AXI master interface data bus width (it is always 32-bit for non-burst masters and can be 32, 64, or 128-bit for masters supporting burst).

Native Data Width: 32 (dropdown) bit

More Info < Back Next > Cancel

Pick the Master Signals

- ❑ Use the default signals here:



Dual-Bus System

- Note that we have two buses in our system now:

Xilinx Platform Studio (EDK_P.20131013) - D:\xps\zynq_memcpy\system.xmp - [System Assembly View]

File Edit View Project Hardware Device Configuration Debug Simulation Window Help

Navigator

Design Flow

Run DRCs

Implement Flow

Generate Netlist

Generate BitStream

Export Design

Simulation Flow

Generate HDL Files

IP Catalog

Description

- EDK Install
- Analog
- Arithmetic
- Bus and Bridge
- Clock, Reset and Interrupt
- Communication High-Speed
- Communication Low-Speed
- DMA and Timer
- Debug
- FPGA Reconfiguration
- General Purpose IO
- Interprocessor Communication

Search IP Catalog: Clear

Project IP Catalog

Legend

- Master Slave Master/Slave Target Initiator Connected Unconnected Monitor
- Production License (paid) License (eval) Local Pre Production Beta Devel
- Superseded Discontinued

Design Summary Graphical Design View System Assembly View

Name	Bus Name
axi4lite_0	
axi_interconnect_1	
processing_system7_0	
M_AXI_GP0	axi4lite_0
S_AXI_HP0	axi_interconnect_1:memcpy_0.M_AXI
BTN_5Bits	
LEDs_8Bits	
SWs_8Bits	
memcpy_0	
M_AXI	axi_interconnect_1
S_AXI	axi4lite_0

M_AXI - Master AXI interface
S_AXI - Slave AXI interface

Console

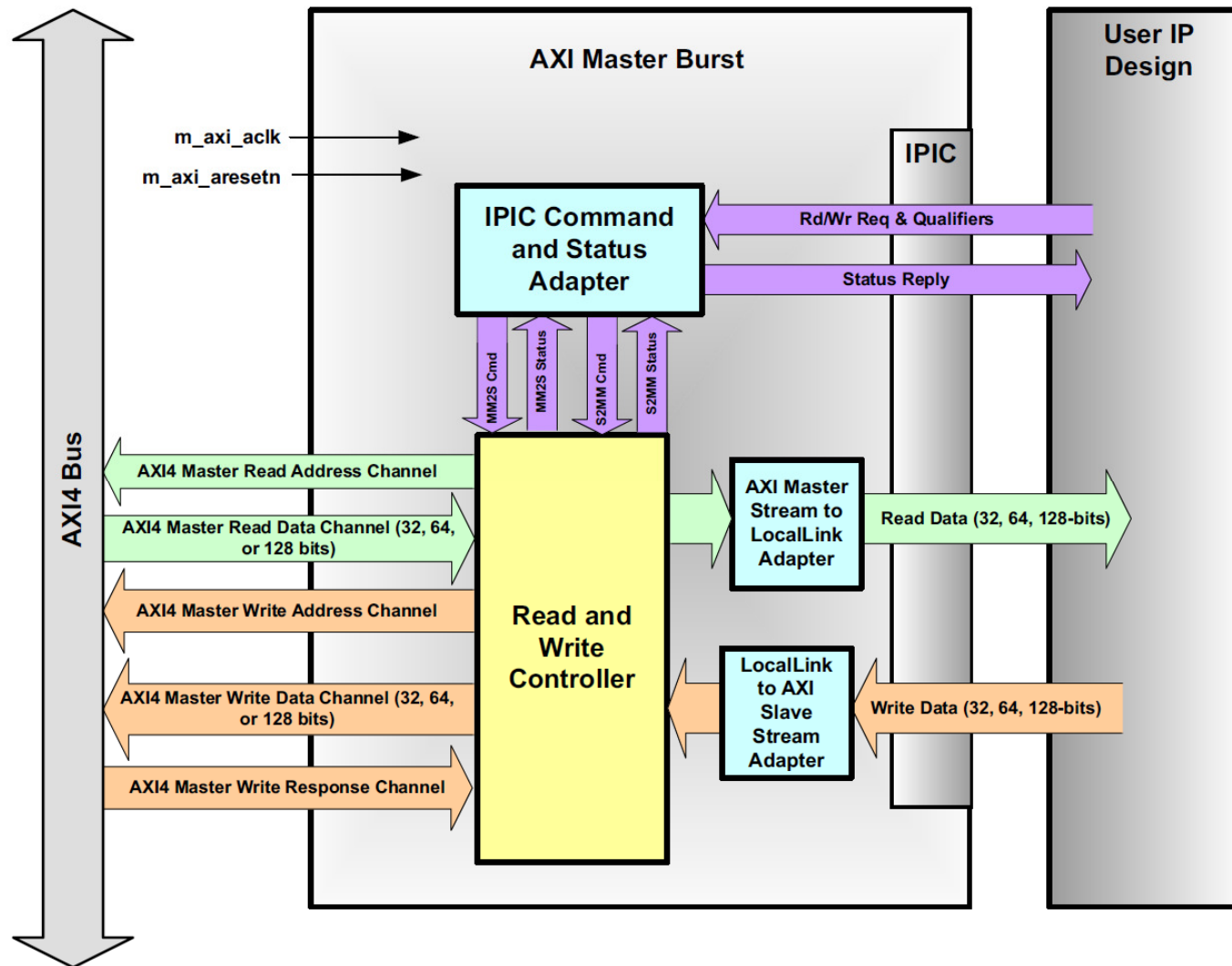
The signal does not drive any load pins in the design.
DRC detected 0 errors and 2 warnings. Please see the previously displayed individual error or warning messages for more details.
INFO:Security:54 - 'xc7z020' is a WebPack part.
WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for

Console Warnings Errors

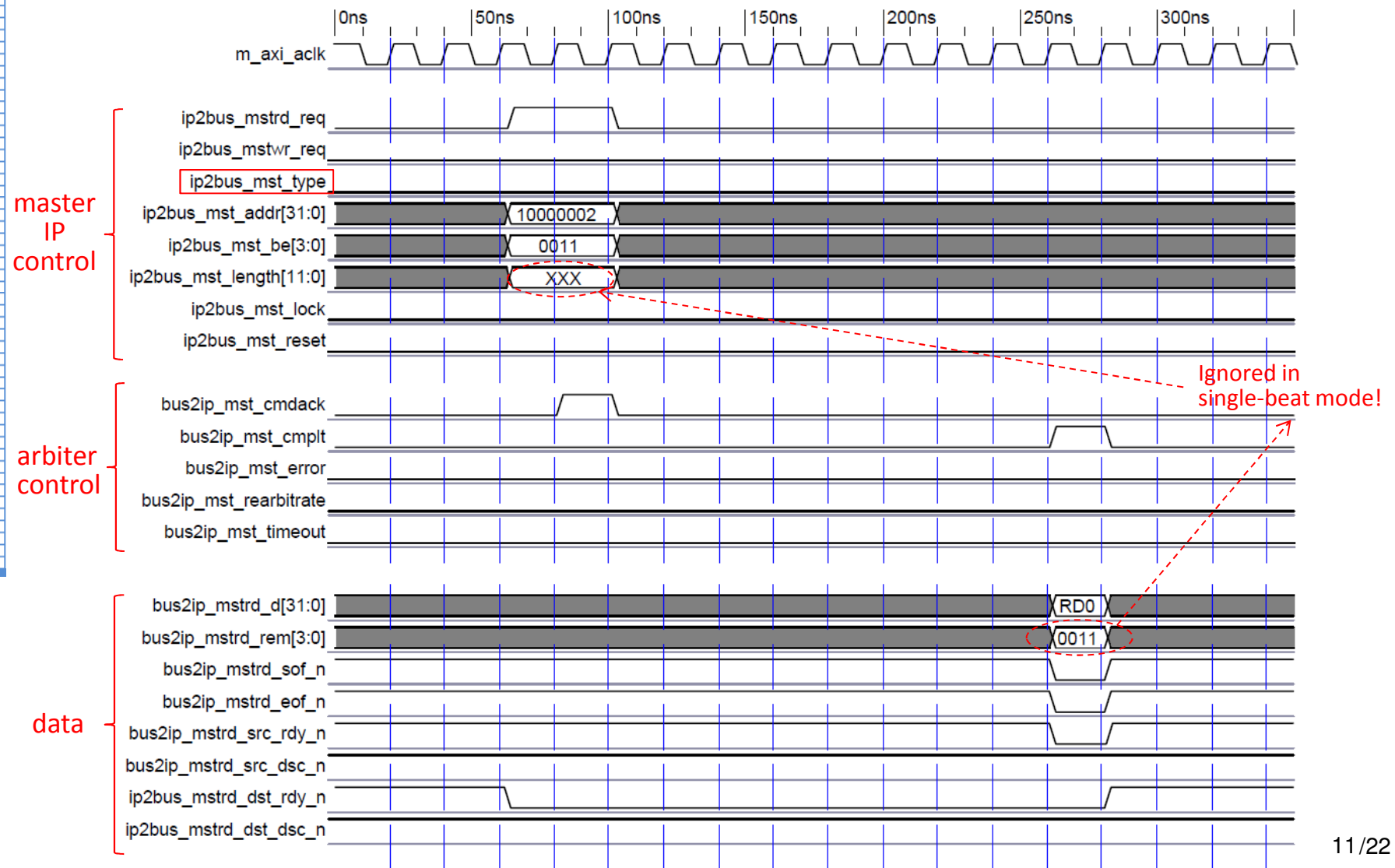
About the Lab 4 Package

- ❑ An ISE Design Suite project, `zynq_memcpy.zip`, is provided for your reference
 - The HW IP is similar to the `memcpy()` function
 - The source and destination addresses are 32-bit word-aligned and the length is the number of bytes to be transferred
 - To simplify the design, **length must be a multiple of 4!**
- ❑ Definitions of the interface registers
 - `reg0` – the register used to trigger HW and signal the completion
 - `reg1` – the destination address; the two LSBs are always zeros
 - `reg2` – the source address; the two LSBs are always zeros
 - `reg3` – the length of the transfer in bytes (must be a multiple of 4)

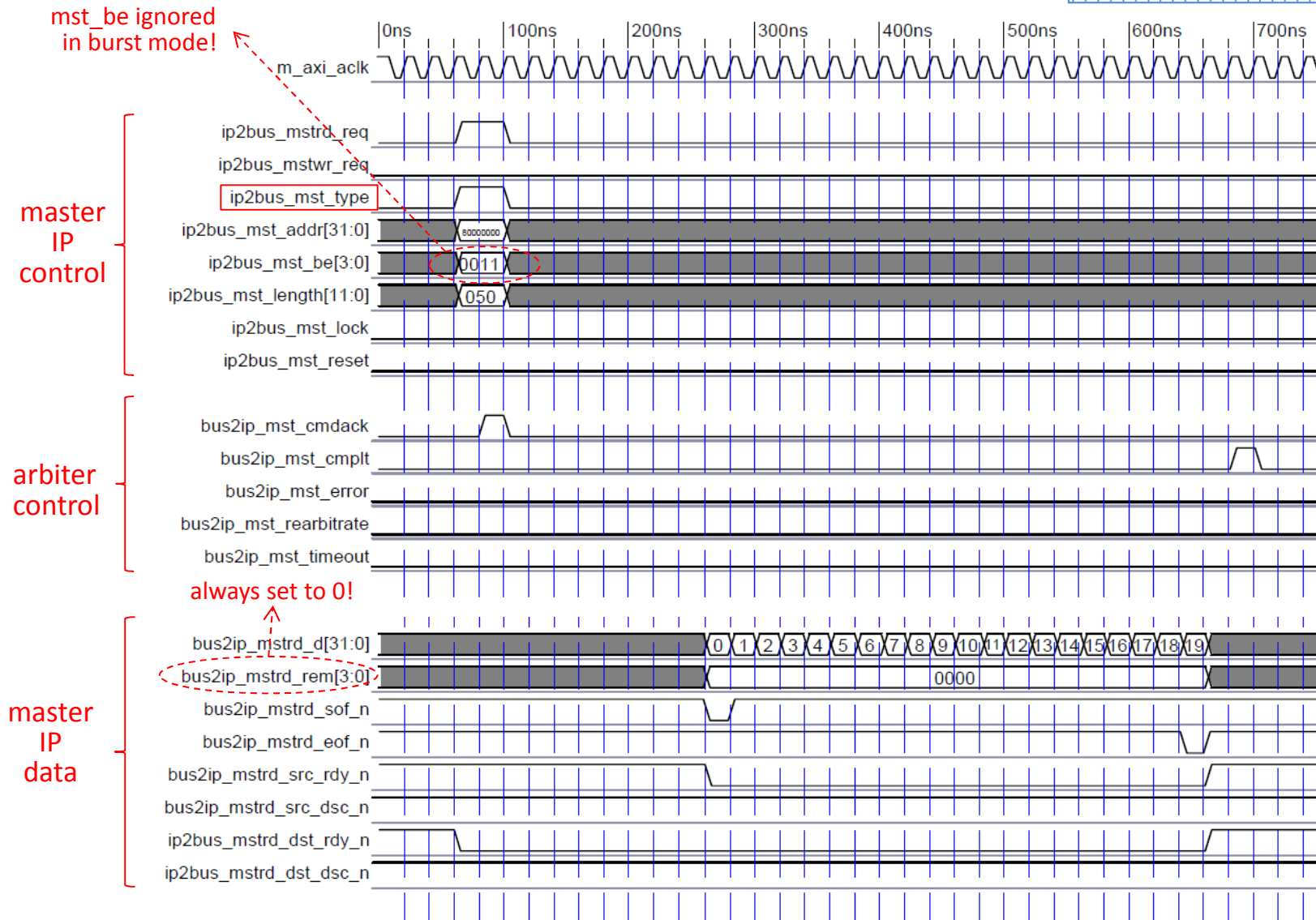
AXI Master Burst Block Diagram



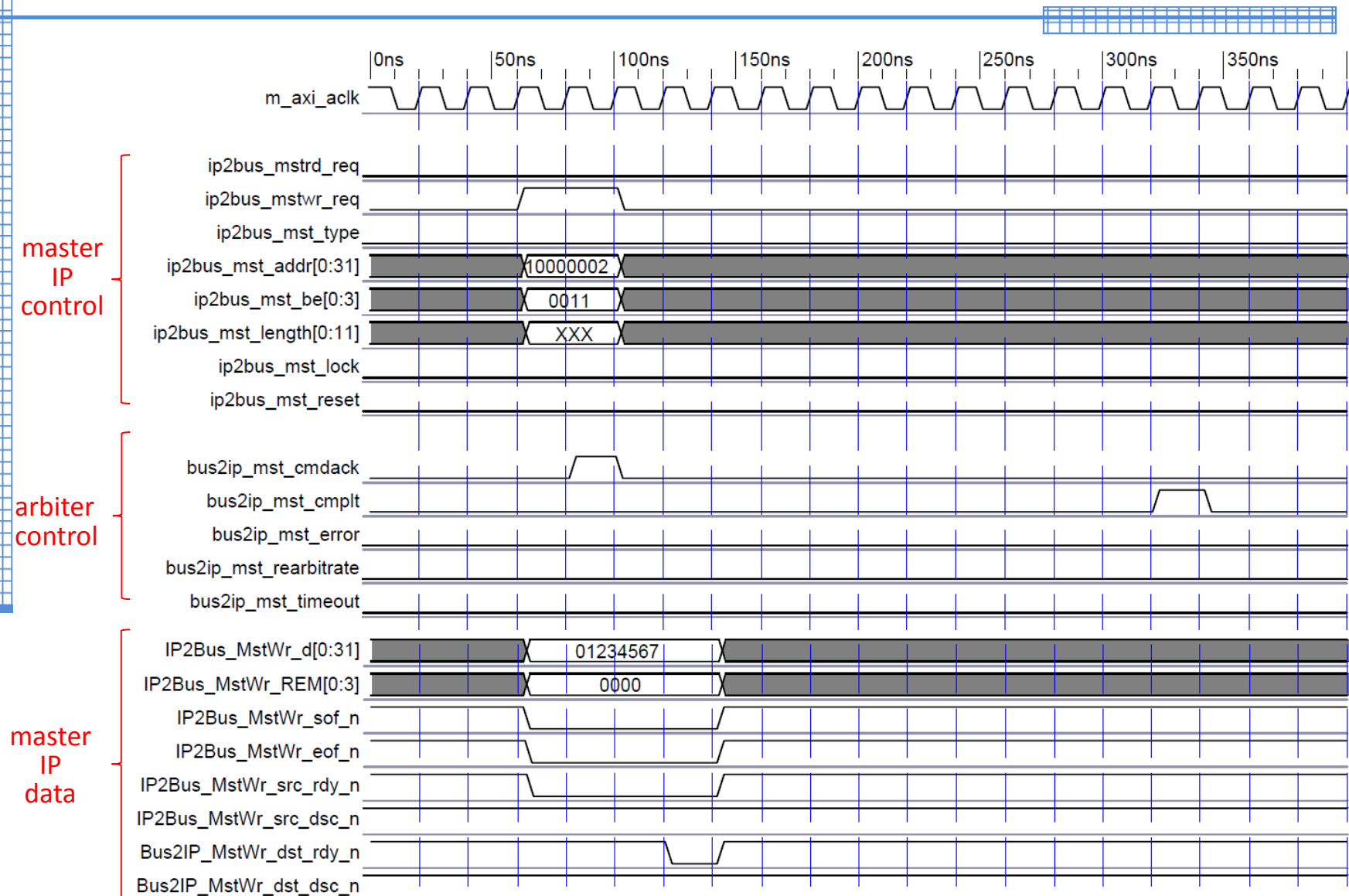
IPIF Single-Beat Read Operation



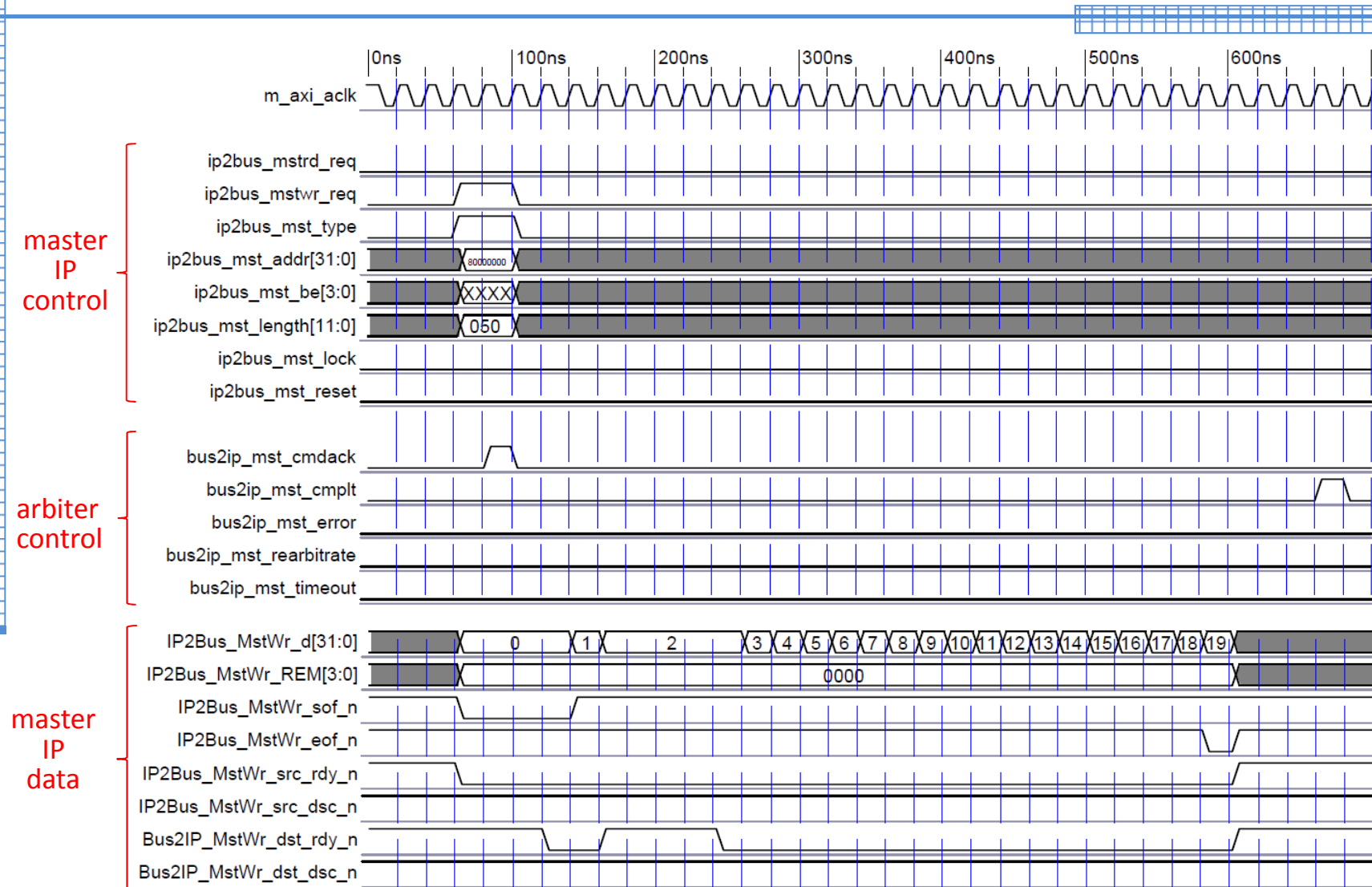
IPIF Burst Data Read Operation



IPIF Single-Beat Write Operation



IPIF Burst Data Write Operation



Cache Issues

- ❑ The memory accessed directly by the hardware cannot be cached by the CPU, or data inconsistency happens
- ❑ There are some functions in Xilinx BSP (in xil_cache.c) that allow you to control the cache behavior of the CPU, such as:
 - `void Xil_DCacheFlushRange(unsigned addr, unsigned size);`
 - `void Xil_DCacheDisable(void);`
- ❑ Cache flushes degrade the CPU performance, and should be used as few as possible

The Software Interface

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

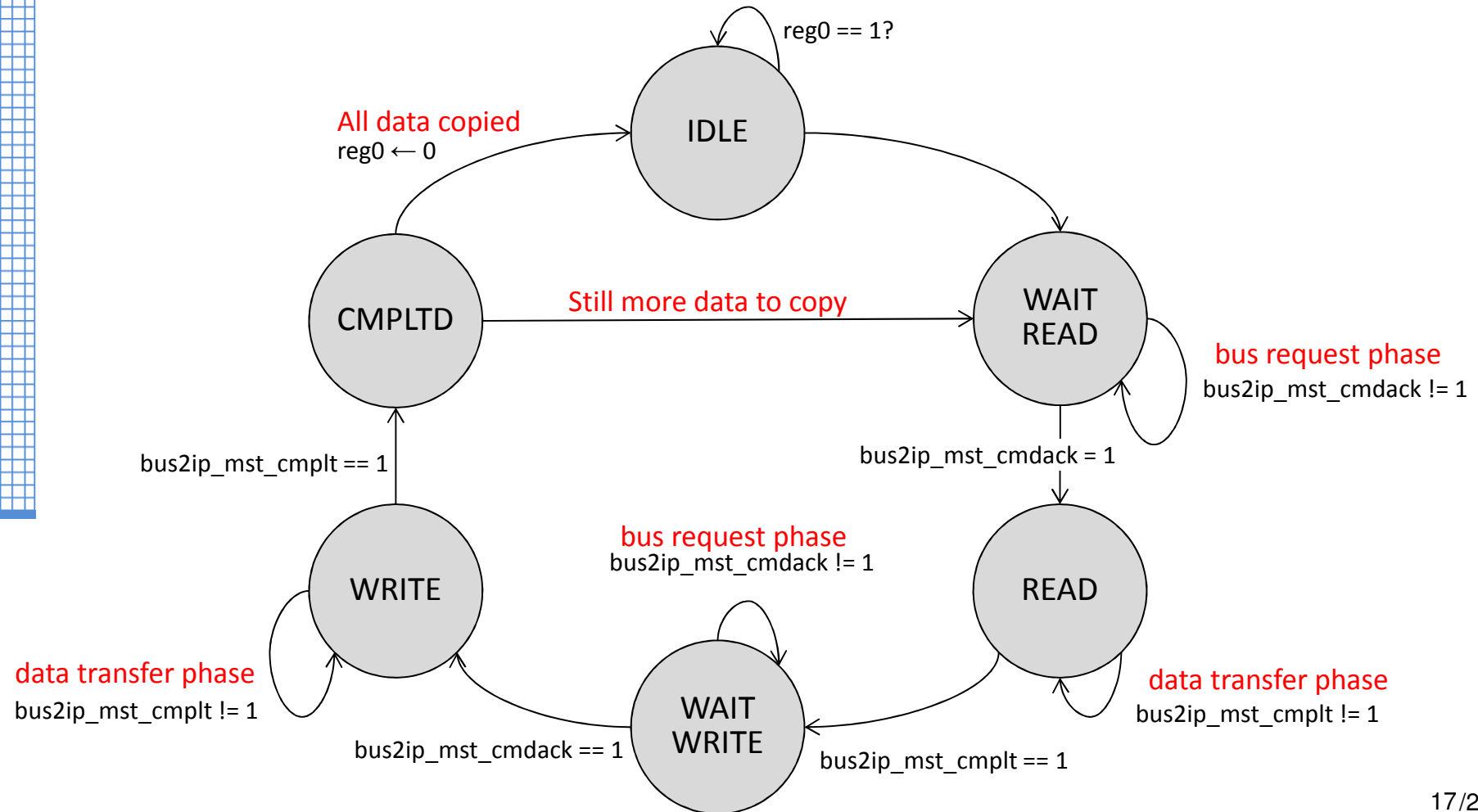
```
volatile int *hw_active = (int *) (XPAR_MEMCPY_0_BASEADDR + 0x0);
volatile int *dst_reg    = (int *) (XPAR_MEMCPY_0_BASEADDR + 0x4);
volatile int *src_reg    = (int *) (XPAR_MEMCPY_0_BASEADDR + 0x8);
volatile int *len_reg    = (int *) (XPAR_MEMCPY_0_BASEADDR + 0xc);

void hw_memcpy(void *dst, void *src, int len)
{
    /* flush the memory area from the CPU cache */
    Xil_DCacheFlushRange((unsigned int) src, len);
    Xil_DCacheFlushRange((unsigned int) dst, len);

    *dst_reg = (int) dst; // destination word address
    *src_reg = (int) src; // source word address
    *len_reg = len;       // transfer size in bytes
    *hw_active = 1;       // trigger the HW IP
    while (*hw_active);   // wait for the transfer to finish
}
```

The Controller of the memcpy IP

- The FSM of the IP has six states:



Performance

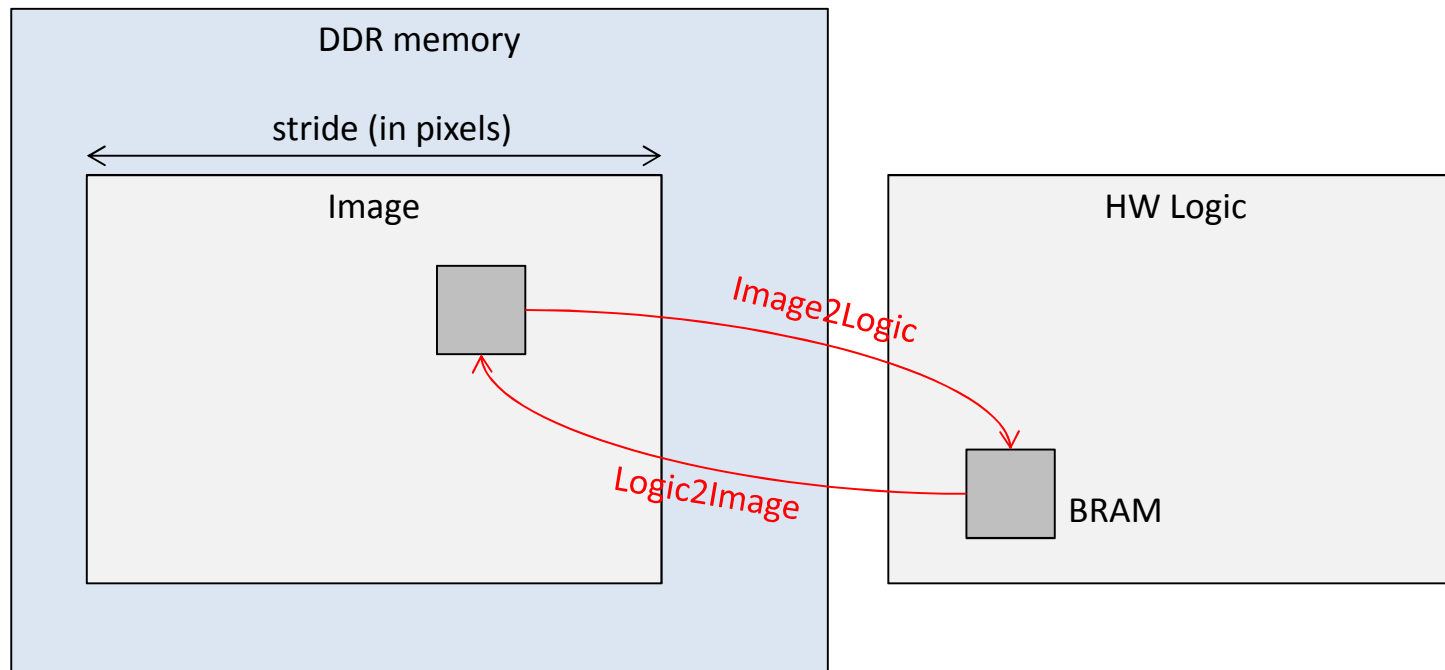
❑ DDR-to-DDR memory copy performance :

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

❑ Memory copy within cache: ~ 730 MBPS

Goal of Lab 4

- ❑ Your HW IP must have two 8×8 block-copy functions:
 - Image2Logic: copy an 8×8 image block to the on-chip BRAM
 - Logic2Image: copy an 8×8 BRAM block into the image



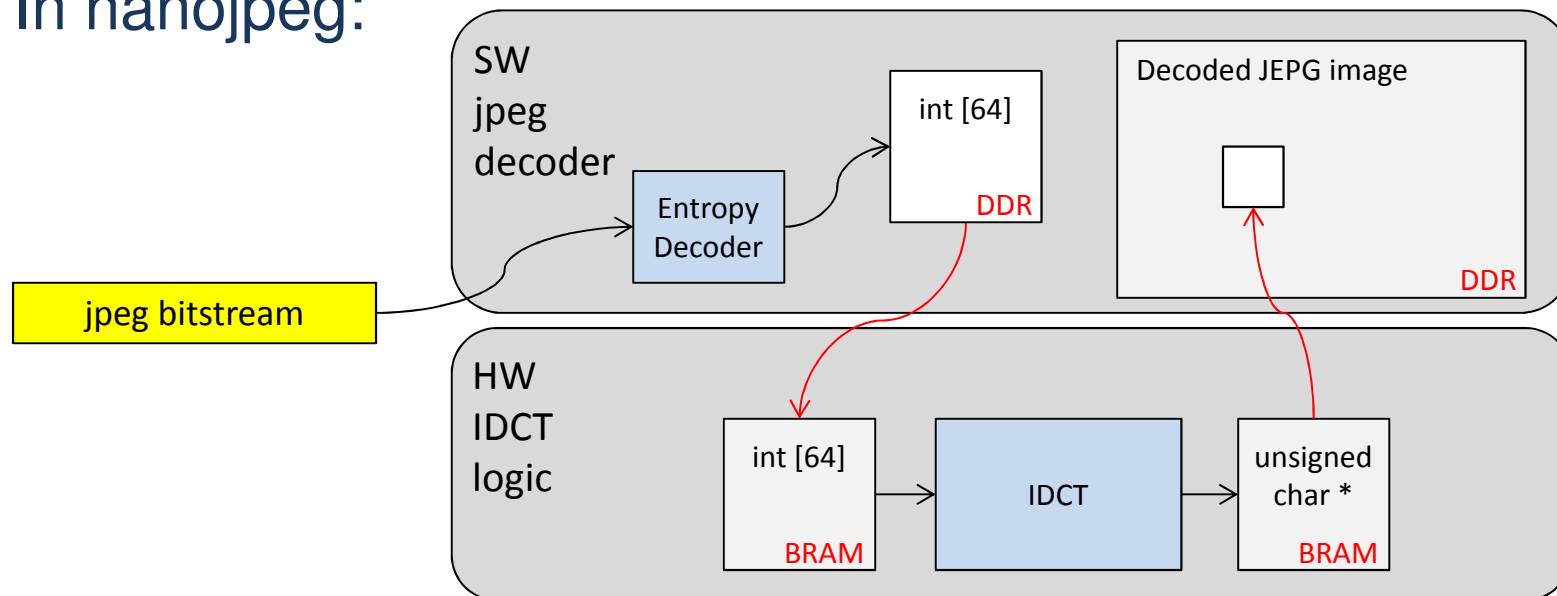
HW-SW Codesign Interface

□ The interface registers are defined as:

- `hw_active` – writing '1' to it executes Image2Logic, writing '2' to it executes Logic2Image, reading 0 from it means HW is done (idle)
- `block_ptr` – the upper-left pixel address of the image block
- `stride` – the width of the image that contains the block
- `bpp` – the number of bytes per pixel to copy, the value can be 1 or 4. A char 8×8 matrix has `bpp = 1`, an integer 8×8 matrix has `bpp = 4`.

Behavior of Your Logic

- ❑ Once triggered, your logic must copy a rectangular area from/to the DDR memory
- ❑ Burst copy only works for consecutive addresses
 - A block copy must be divided into several bursts, for example:
 - If stride = 8, bpp = 4, a single burst of 64 words can be used
 - If stride > 8, bpp = 1, eight bursts of 2 words can be used
- ❑ In nanojpeg:



Reference

- ❑ Xilinx LogiCORE AXI Master Burst IP data sheet:

http://www.xilinx.com/support/documentation/ip_documentation/axi_master_burst/v1_00_a/ds844_axi_master_burst.pdf