ZedBoard Lab 6
LED and driver

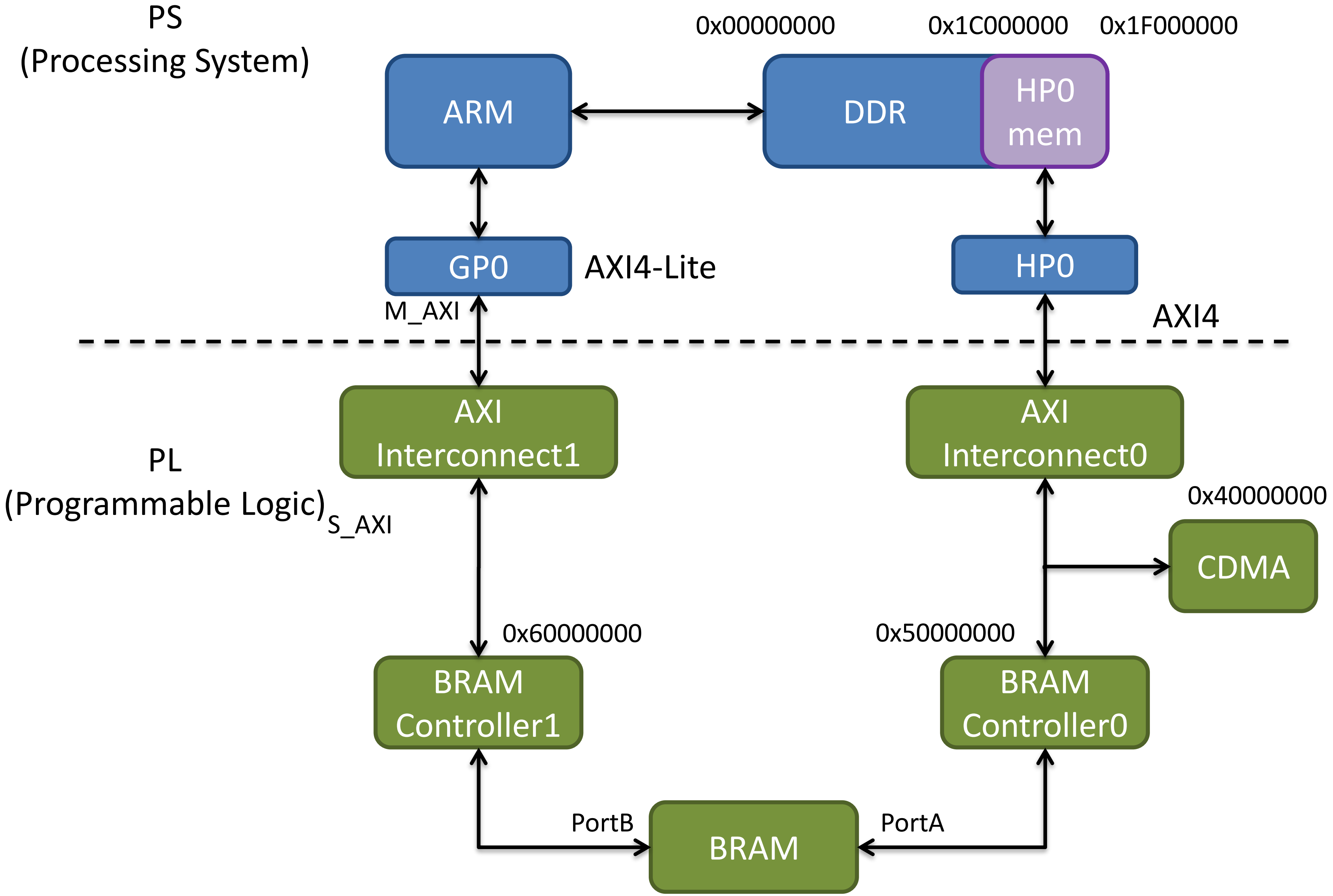
Chun-Chen Tu

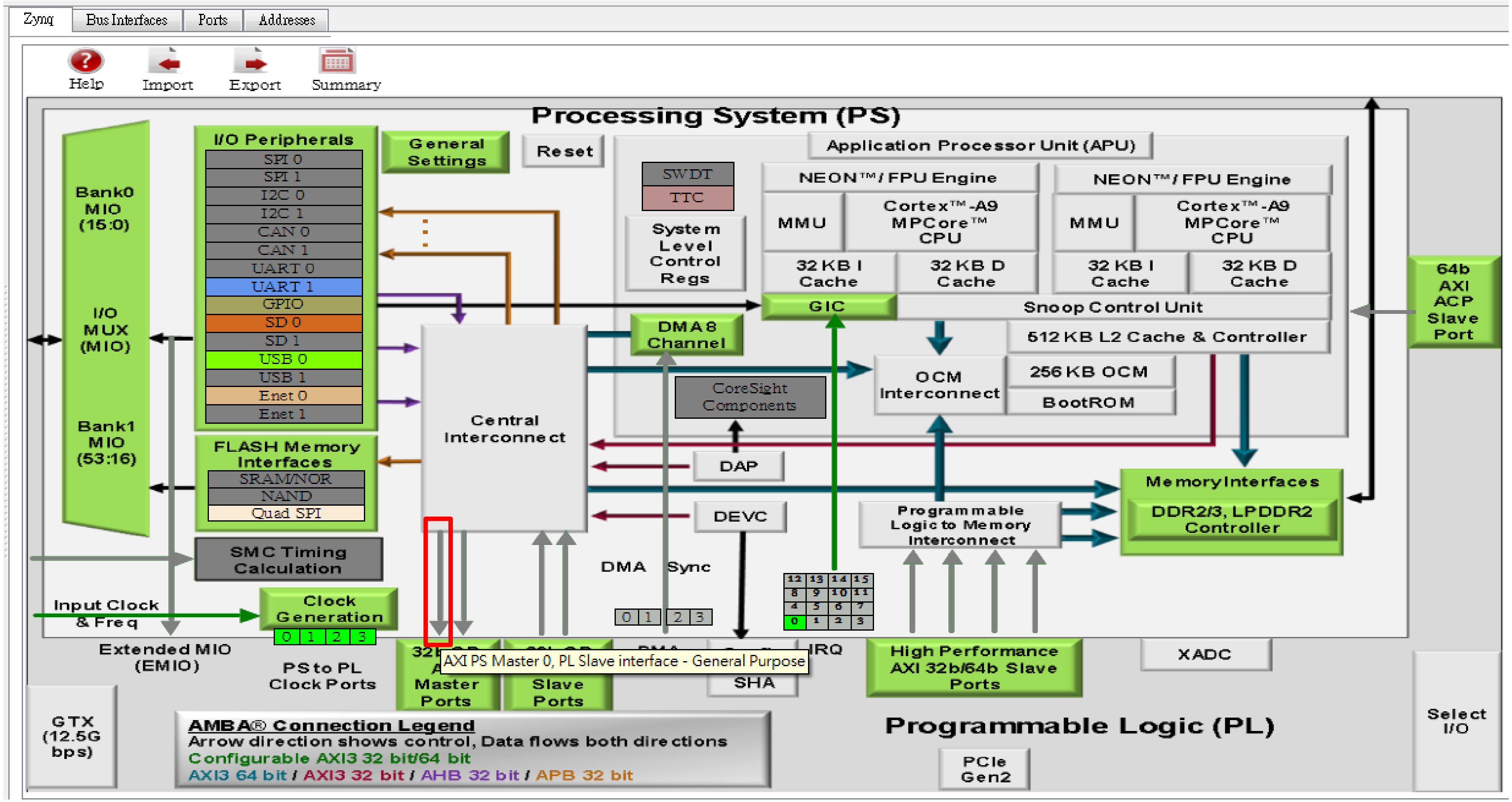
timtu@umich.edu

Features of design

- Data transmission using HP (High performance) channel.
- CDMA (Central Direct Memory Access) in charge of moving data.
  - Driver
  - Interrupt handling
- BRAM (Block RAM) control

System Architecture

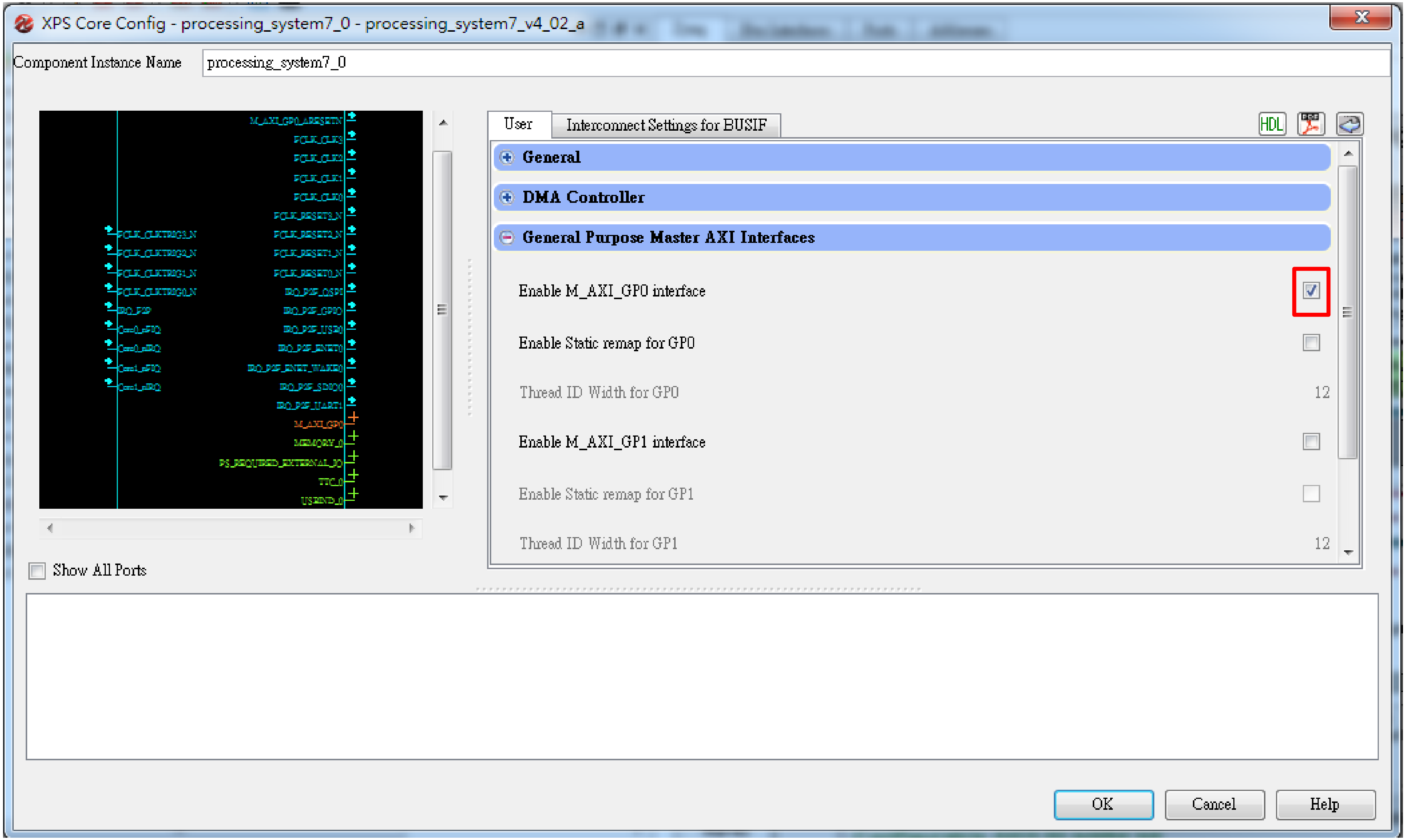




AXI PS Master 0, PL Slave interface - General Purpose

AMBA® Connection Legend
Arrow direction shows control, Data flows both directions
Configurable AXI3 32 bit/64 bit
AXI3 64 bit / AXI3 32 bit / AHB 32 bit / APB 32 bit

Enable the AXI GP 0 channel.



XPS Core Config - processing_system7_0 - processing_system7_v4_02_a

Component Instance Name: processing_system7_0

☐ Show All Ports

Enable M_AXI_GP0 interface ☑

Enable Static remap for GP0 ☐

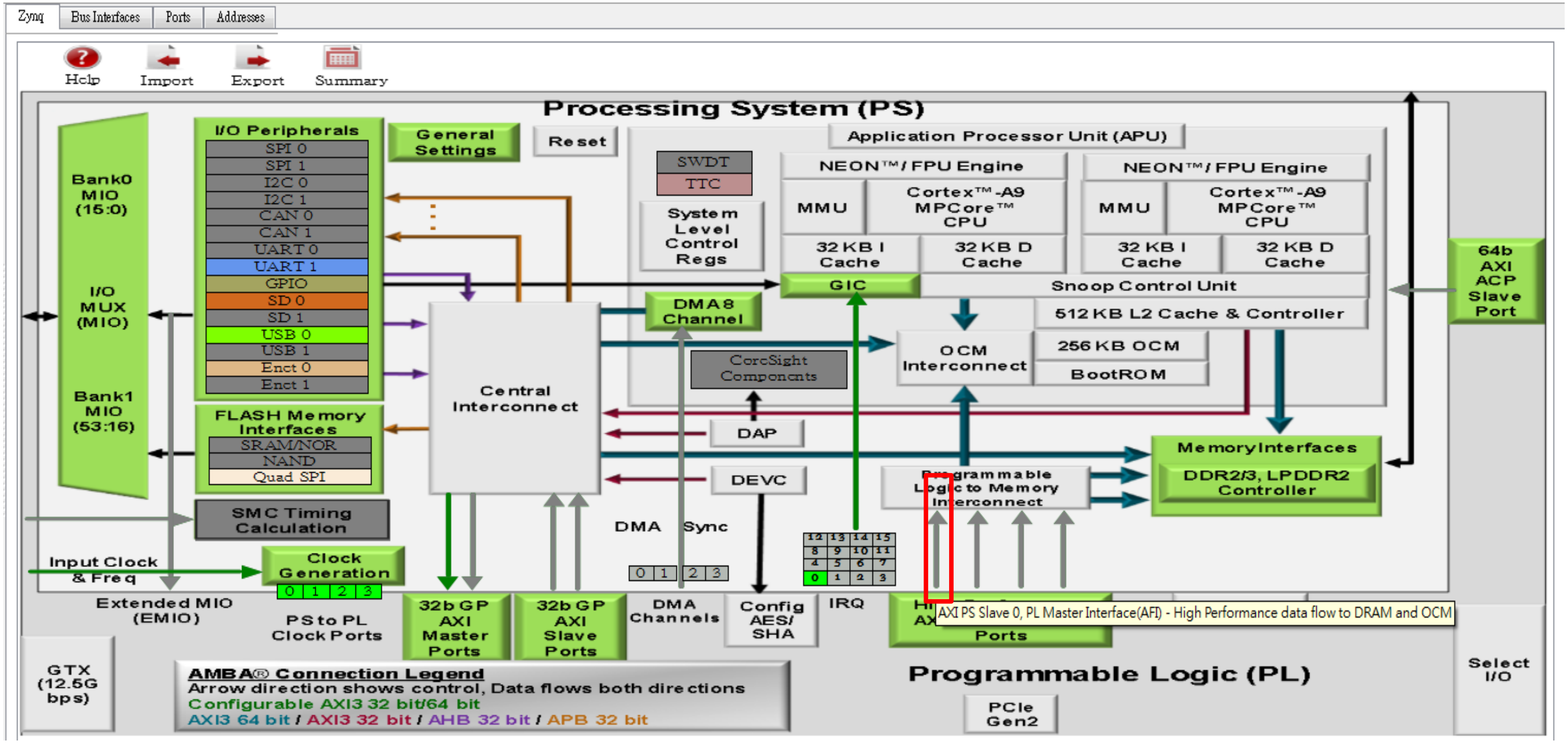
Thread ID Width for GP0 12

Enable M_AXI_GP1 interface ☐

Enable Static remap for GP1 ☐

Thread ID Width for GP1 12

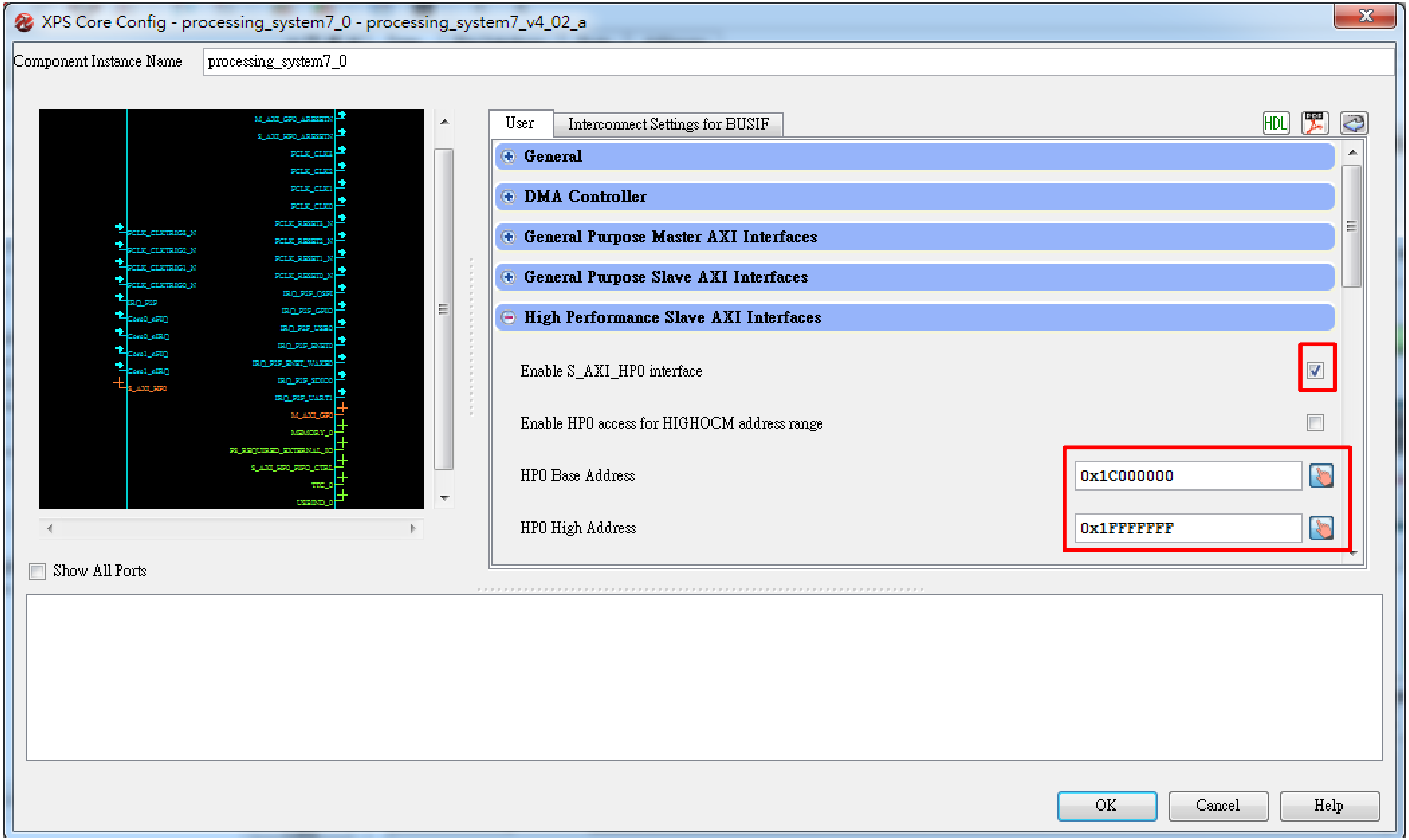
Check on the *M_AXI_GP0* box.



AXI PS Slave 0, PL Master Interface(AFI) - High Performance data flow to DRAM and OCM

AMBA® Connection Legend
Arrow direction shows control, Data flows both directions
Configurable AXI3 32 bit/64 bit
AXI3 64 bit / AXI3 32 bit / AHB 32 bit / APB 32 bit

Click on HP0 and enable it.



Component Instance Name: processing_system7_0

Enable S_AXI_HP0 interface ☑

Enable HP0 access for HIGHOCM address range ☐

HP0 Base Address: 0x1C000000

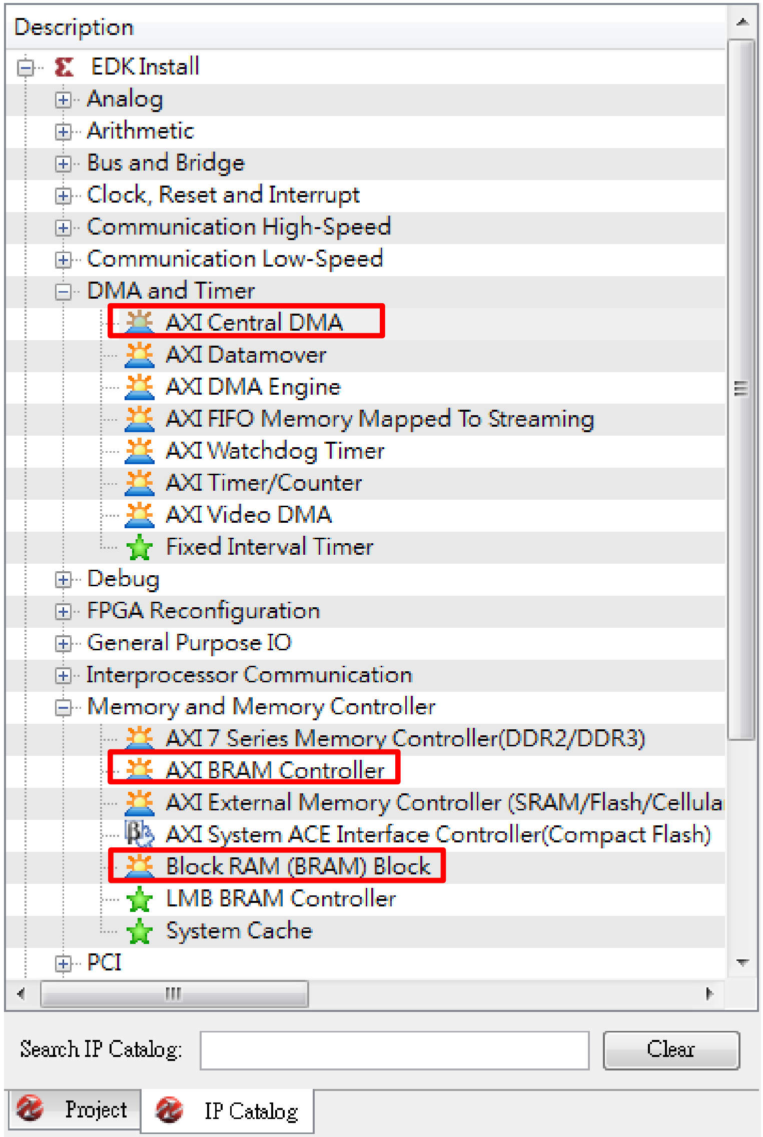
HP0 High Address: 0x1FFFFFFF

☐ Show All Ports

Click on Enable *S_AXI_HP0* interface.

And set the address to be 0x1C000000 to 0x1FFFFFF

(This is DDR address from 448MB to 512MB)

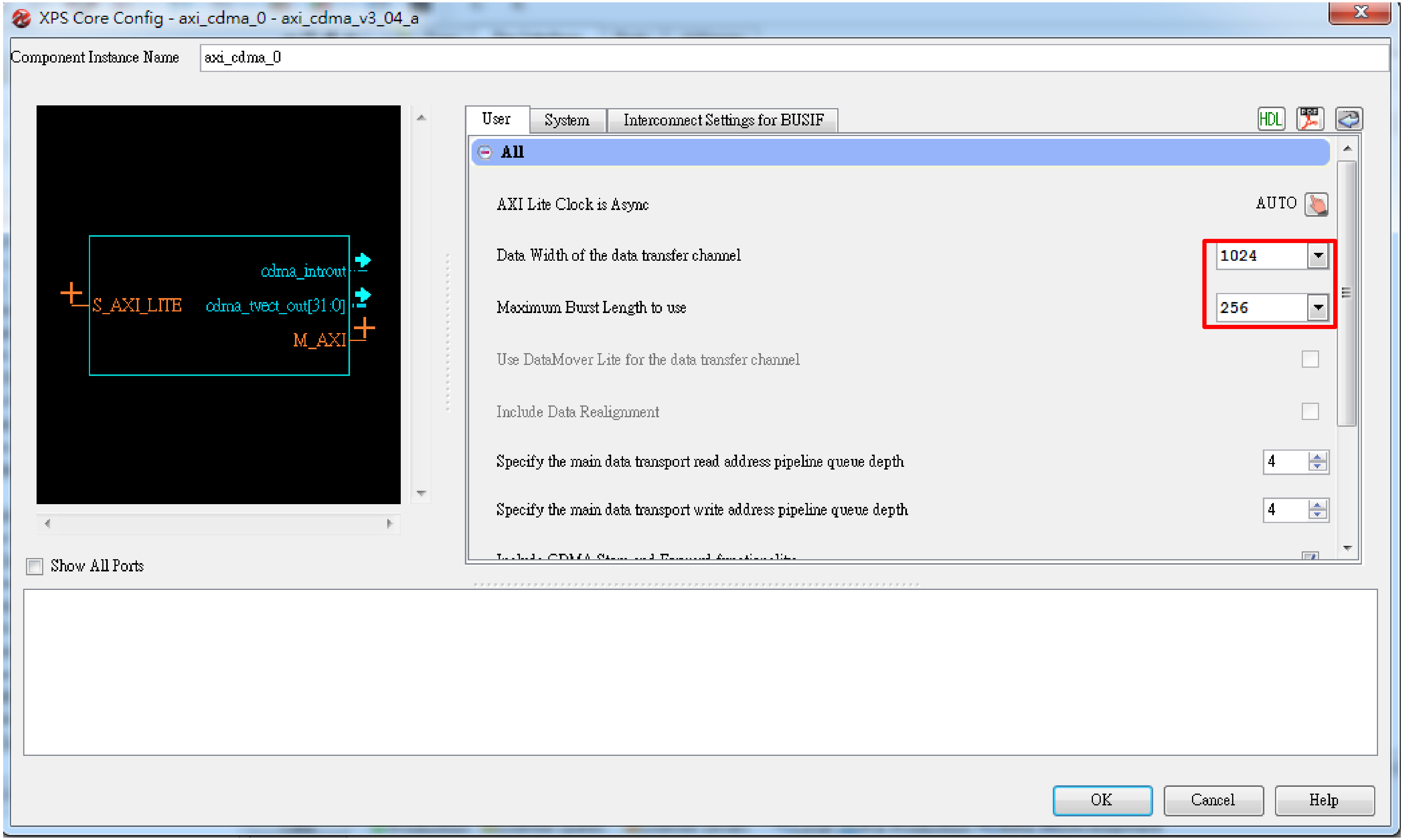


Add an AXI Central DMA design.

And two AXI BRAM Controller.

(Note: this will create two BRAM automatically, delete one).

Sometimes you need to add BRAM yourself.



XPS Core Config - axi_cdma_0 - axi_cdma_v3_04_a

Component Instance Name: axi_cdma_0

AXI Lite Clock is Async: AUTO

Data Width of the data transfer channel: 1024

Maximum Burst Length to use: 256

Use DataMover Lite for the data transfer channel □

Include Data Realignment □

Specify the main data transport read address pipeline queue depth: 4

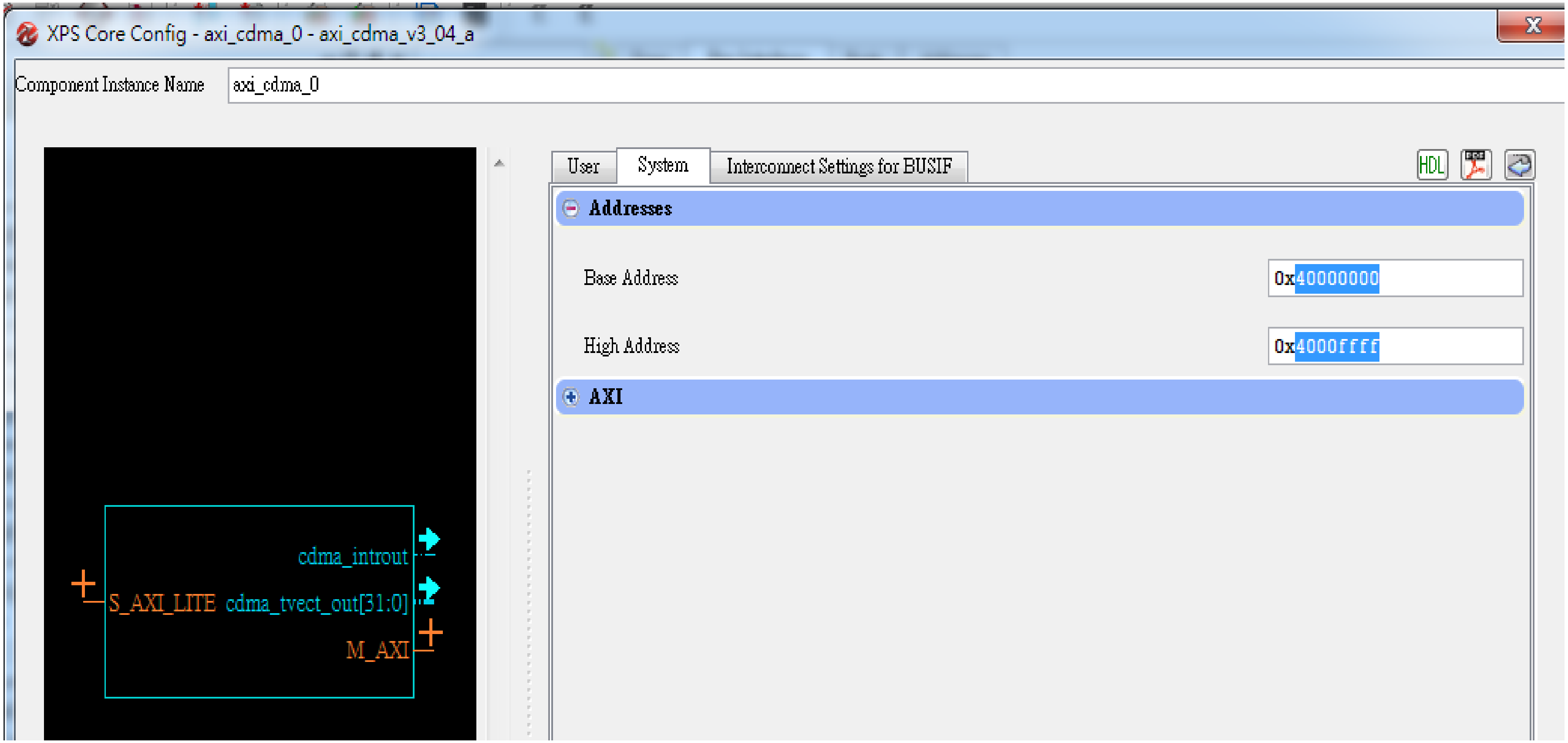
Specify the main data transport write address pipeline queue depth: 4

Include CDMA Store and Forward functionality ☑

CDMA setting:

Set the *Data Width* to 1024

and *Burst Length* to 256



XPS Core Config - axi_cdma_0 - axi_cdma_v3_04_a

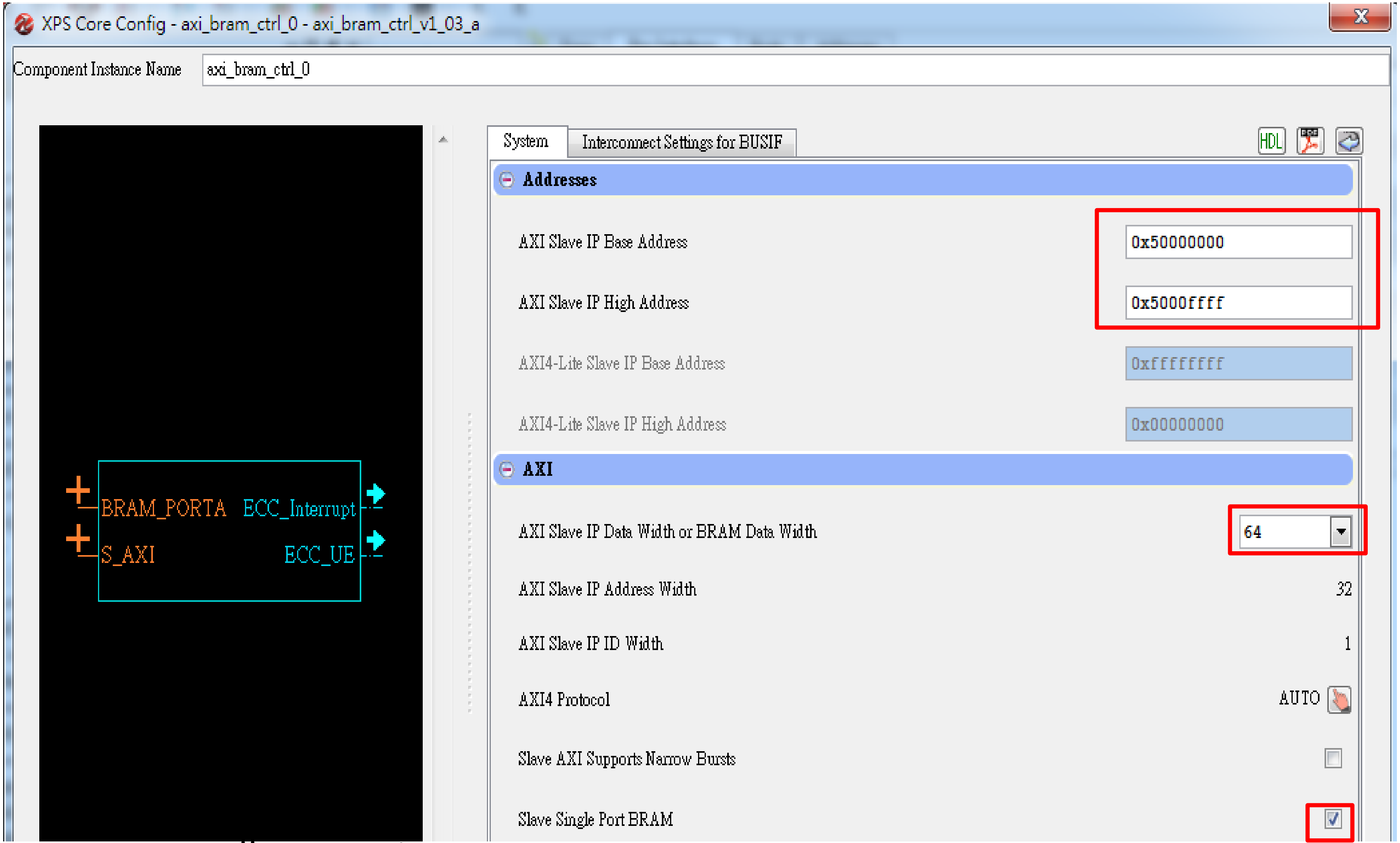
Component Instance Name: axi_cdma_0

Addresses

Base Address: 0x40000000

High Address: 0x4000ffff

Assign 0x40000000 – 0x4000ffff to CDMA


XPS Core Config - axi_bram_ctrl_0 - axi_bram_ctrl_v1_03_a
System
Interconnect Settings for BUSIF
HDL
BRAM_PORTA
ECC_Interrupt
S_AXI
ECC_UE

Component Instance Name: axi_bram_ctrl_0

Addresses

AXI Slave IP Base Address: 0x50000000

AXI Slave IP High Address: 0x5000ffff

AXI4-Lite Slave IP Base Address: 0xffffffff

AXI4-Lite Slave IP High Address: 0x00000000

AXI

AXI Slave IP Data Width or BRAM Data Width: 64

AXI Slave IP Address Width: 32

AXI Slave IP ID Width: 1

AXI4 Protocol: AUTO

Slave AXI Supports Narrow Bursts: □

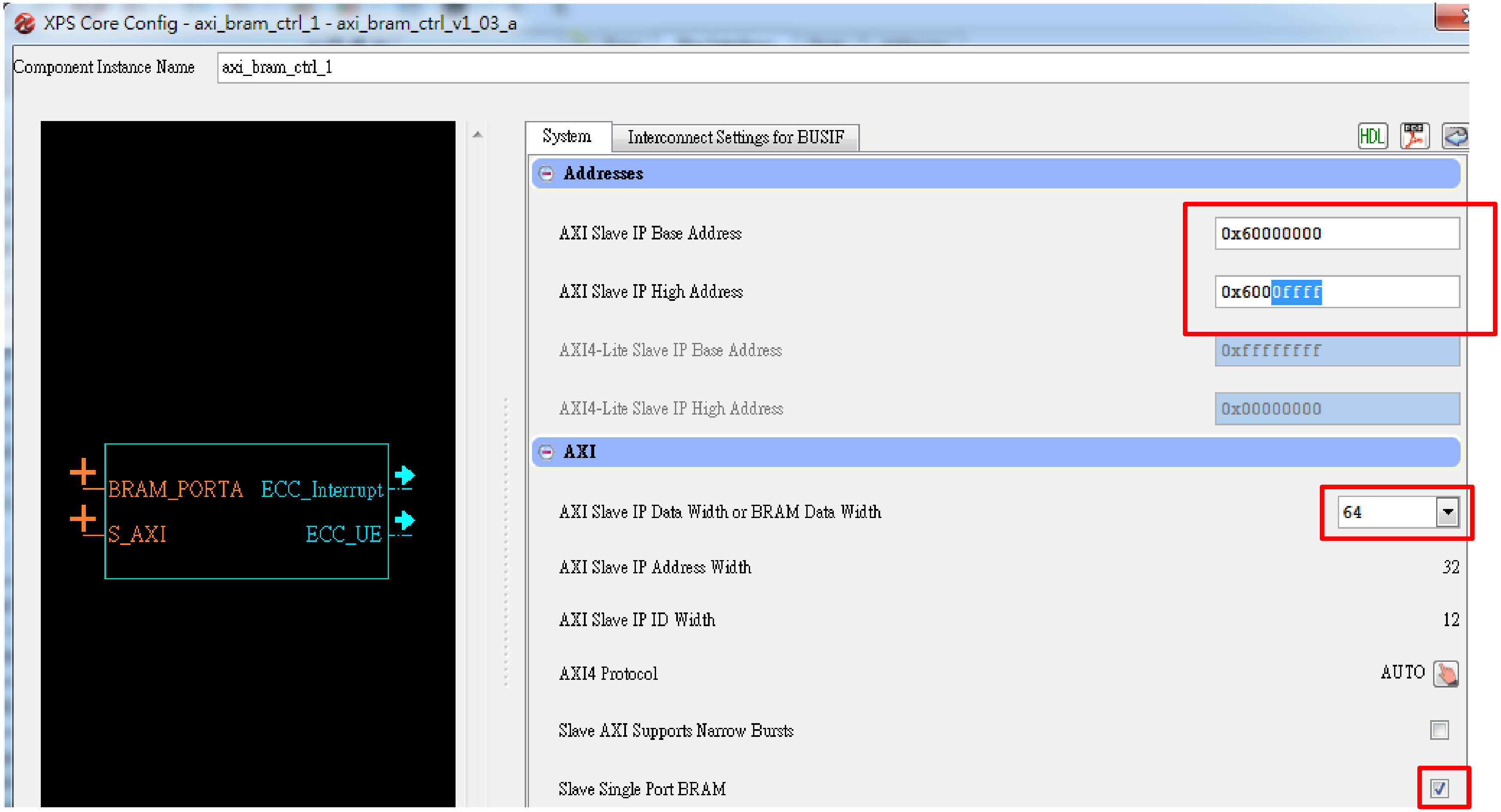
Slave Single Port BRAM: ☑

BRAM controller 0 setting:

Set the address from 0x50000000 to 0x5000FFFF (64KB for the BRAM)

Change to *Data Width* to 64

Click the *Slave Single Port BRAM*



XPS Core Config - axi_bram_ctrl_1 - axi_bram_ctrl_v1_03_a

Component Instance Name: axi_bram_ctrl_1

Addresses

AXI Slave IP Base Address: 0x60000000

AXI Slave IP High Address: 0x6000ffff

AXI4-Lite Slave IP Base Address: 0xffffffff

AXI4-Lite Slave IP High Address: 0x00000000

AXI

AXI Slave IP Data Width or BRAM Data Width: 64

AXI Slave IP Address Width: 32

AXI Slave IP ID Width: 12

AXI4 Protocol: AUTO

Slave AXI Supports Narrow Bursts: □

Slave Single Port BRAM: ☑

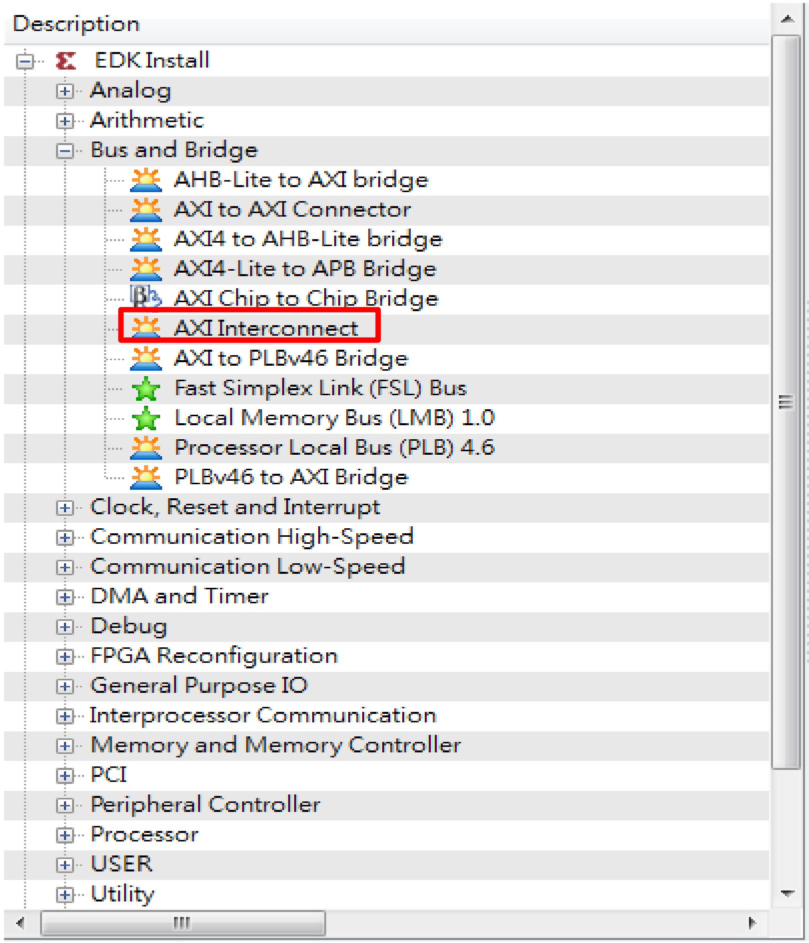
BRAM controller 1 setting:

Set the address from 0x60000000 to 0x6000FFFF

(Note: The address is different from the previous)

Change to *Data Width* to 64

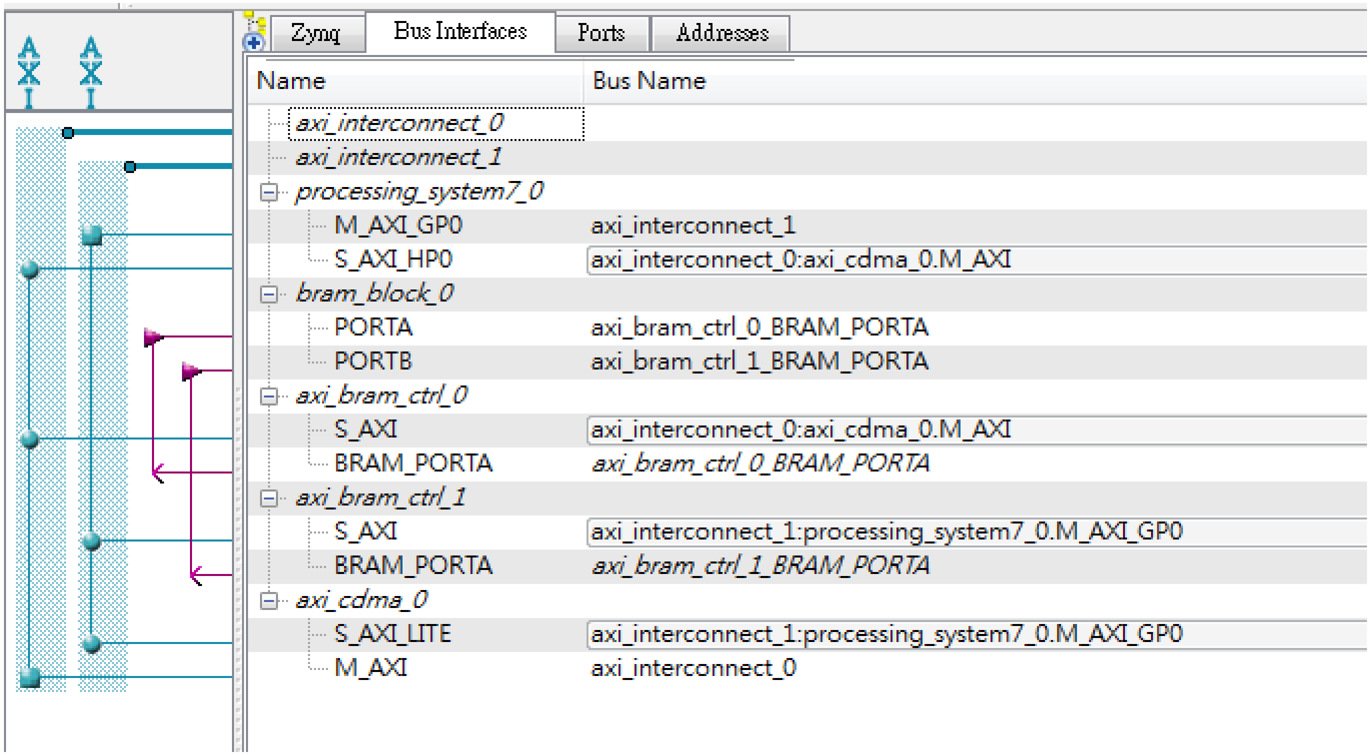
Click the *Slave Single Port BRAM*



Description

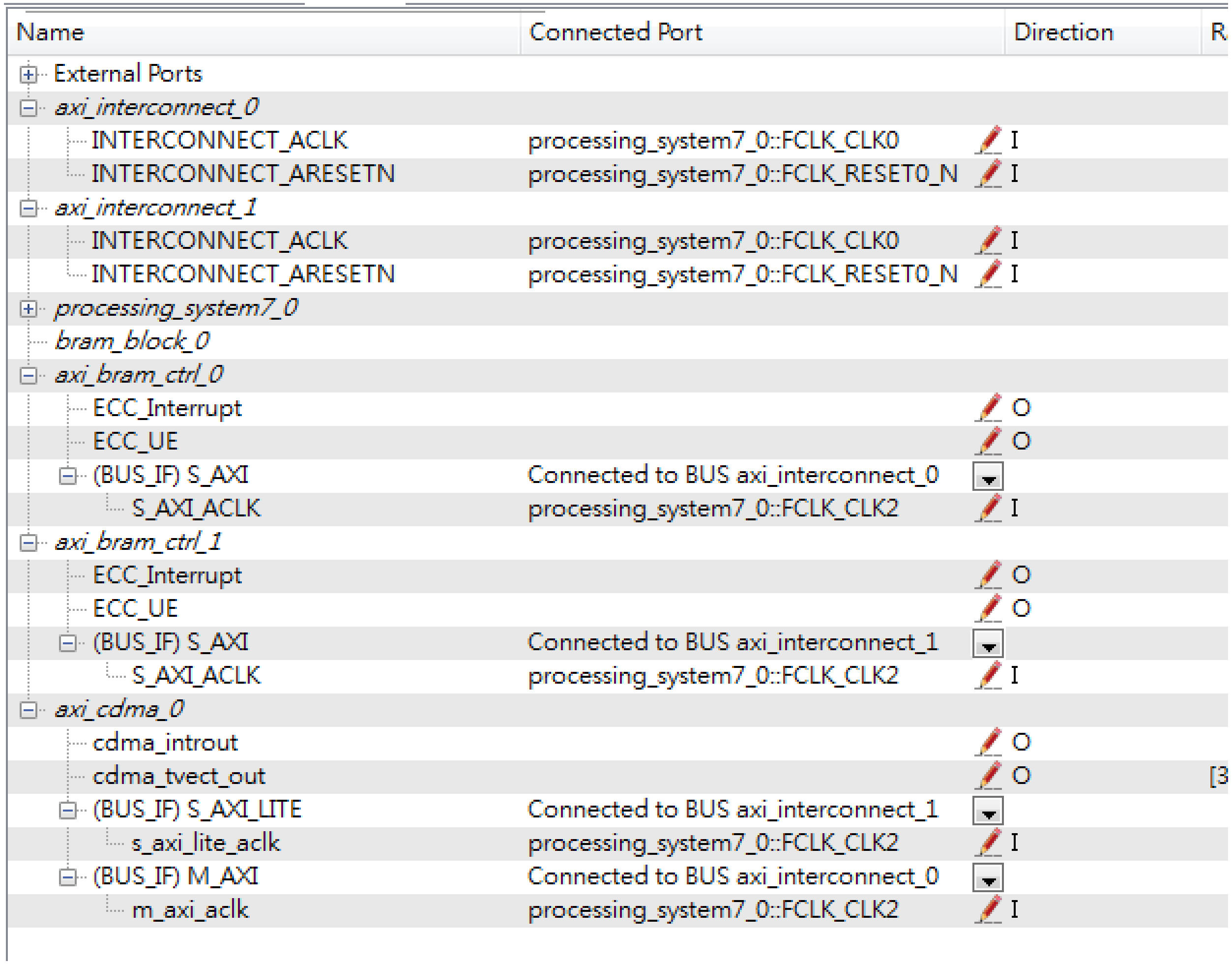
- EDK Install
  - Analog
  - Arithmetic
  - Bus and Bridge
    - AHB-Lite to AXI bridge
    - AXI to AXI Connector
    - AXI4 to AHB-Lite bridge
    - AXI4-Lite to APB Bridge
    - AXI Chip to Chip Bridge
    - AXI Interconnect
    - AXI to PLBv46 Bridge
    - Fast Simplex Link (FSL) Bus
    - Local Memory Bus (LMB) 1.0
    - Processor Local Bus (PLB) 4.6
    - PLBv46 to AXI Bridge
  - Clock, Reset and Interrupt
  - Communication High-Speed
  - Communication Low-Speed
  - DMA and Timer
  - Debug
  - FPGA Reconfiguration
  - General Purpose IO
  - Interprocessor Communication
  - Memory and Memory Controller
  - PCI
  - Peripheral Controller
  - Processor
  - USER
  - Utility

Create two AXI interconnect



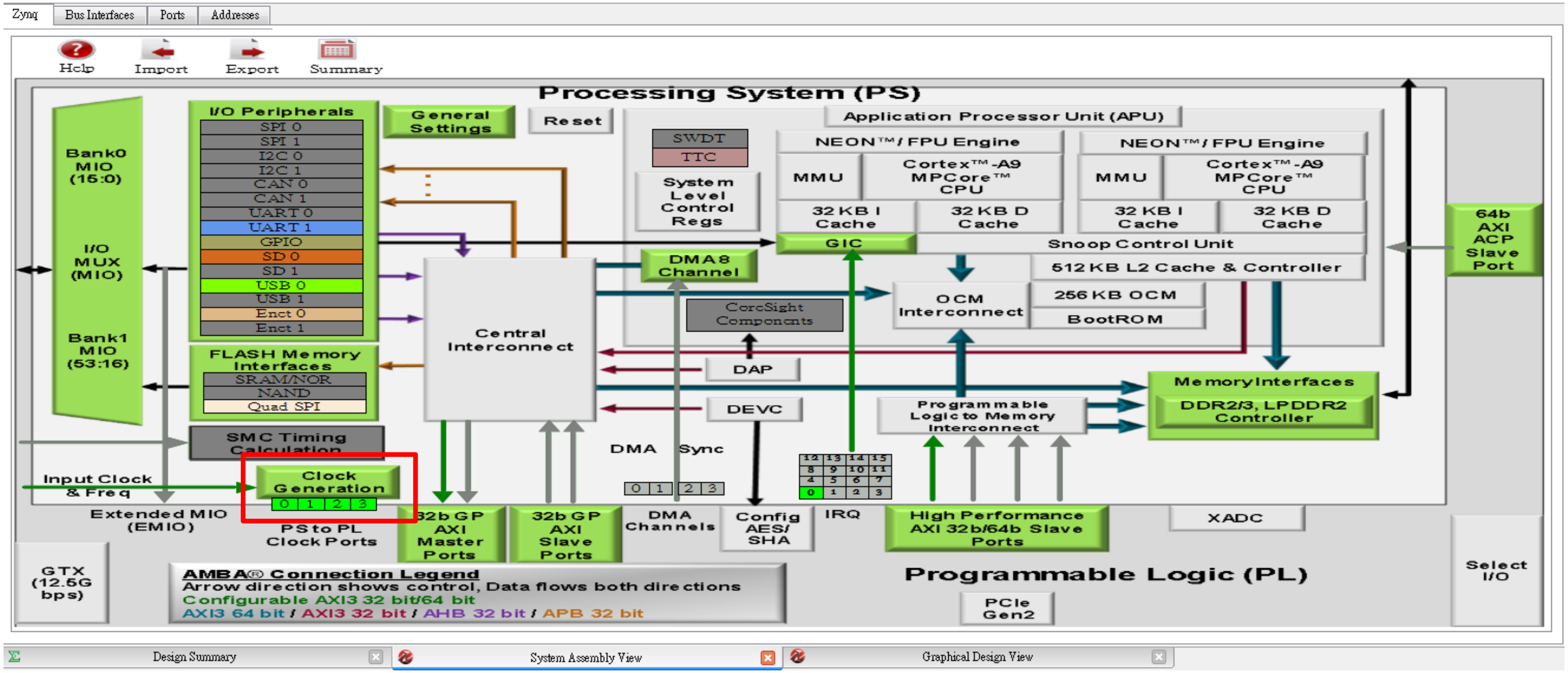
Name	Bus Name
axi_interconnect_0	
axi_interconnect_1	
processing_system7_0	
M_AXI_GP0	axi_interconnect_1
S_AXI_HP0	axi_interconnect_0:axi_cdma_0.M_AXI
bram_block_0	
PORTA	axi_bram_ctrl_0_BRAM_PORTA
PORTB	axi_bram_ctrl_1_BRAM_PORTA
axi_bram_ctrl_0	
S_AXI	axi_interconnect_0:axi_cdma_0.M_AXI
BRAM_PORTA	*axi_bram_ctrl_0_BRAM_PORTA*
axi_bram_ctrl_1	
S_AXI	axi_interconnect_1:processing_system7_0.M_AXI_GP0
BRAM_PORTA	*axi_bram_ctrl_1_BRAM_PORTA*
axi_cdma_0	
S_AXI_LITE	axi_interconnect_1:processing_system7_0.M_AXI_GP0
M_AXI	axi_interconnect_0

Connect the design like this.

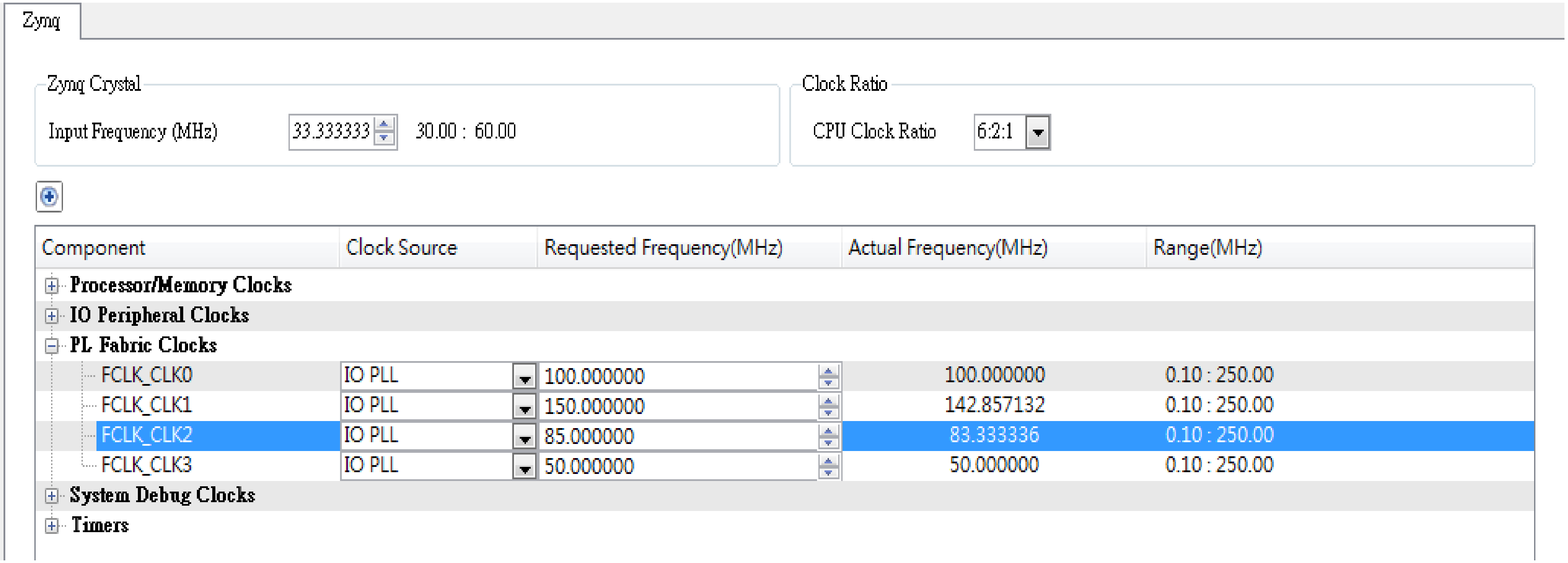


Name	Connected Port	Direction	R
⊞ External Ports			
⊟ *axi_interconnect_0*			
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I	
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I	
⊟ *axi_interconnect_1*			
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I	
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I	
⊞ *processing_system7_0*			
bram_block_0			
⊟ *axi_bram_ctrl_0*			
ECC_Interrupt		O	
ECC_UE		O	
⊟ (BUS_IF) S_AXI	Connected to BUS axi_interconnect_0		
S_AXI_ACLK	processing_system7_0::FCLK_CLK2	I	
⊟ *axi_bram_ctrl_1*			
ECC_Interrupt		O	
ECC_UE		O	
⊟ (BUS_IF) S_AXI	Connected to BUS axi_interconnect_1		
S_AXI_ACLK	processing_system7_0::FCLK_CLK2	I	
⊟ *axi_cdma_0*			
cdma_introut		O	
cdma_tvect_out		O	[3
⊟ (BUS_IF) S_AXI_LITE	Connected to BUS axi_interconnect_1		
s_axi_lite_aclk	processing_system7_0::FCLK_CLK2	I	
⊟ (BUS_IF) M_AXI	Connected to BUS axi_interconnect_0		
m_axi_aclk	processing_system7_0::FCLK_CLK2	I	

Connect clock and reset signals.



Click on *Clock Generation*.



Zynq Crystal

Input Frequency (MHz) 33.333333 30.00 : 60.00

Clock Ratio

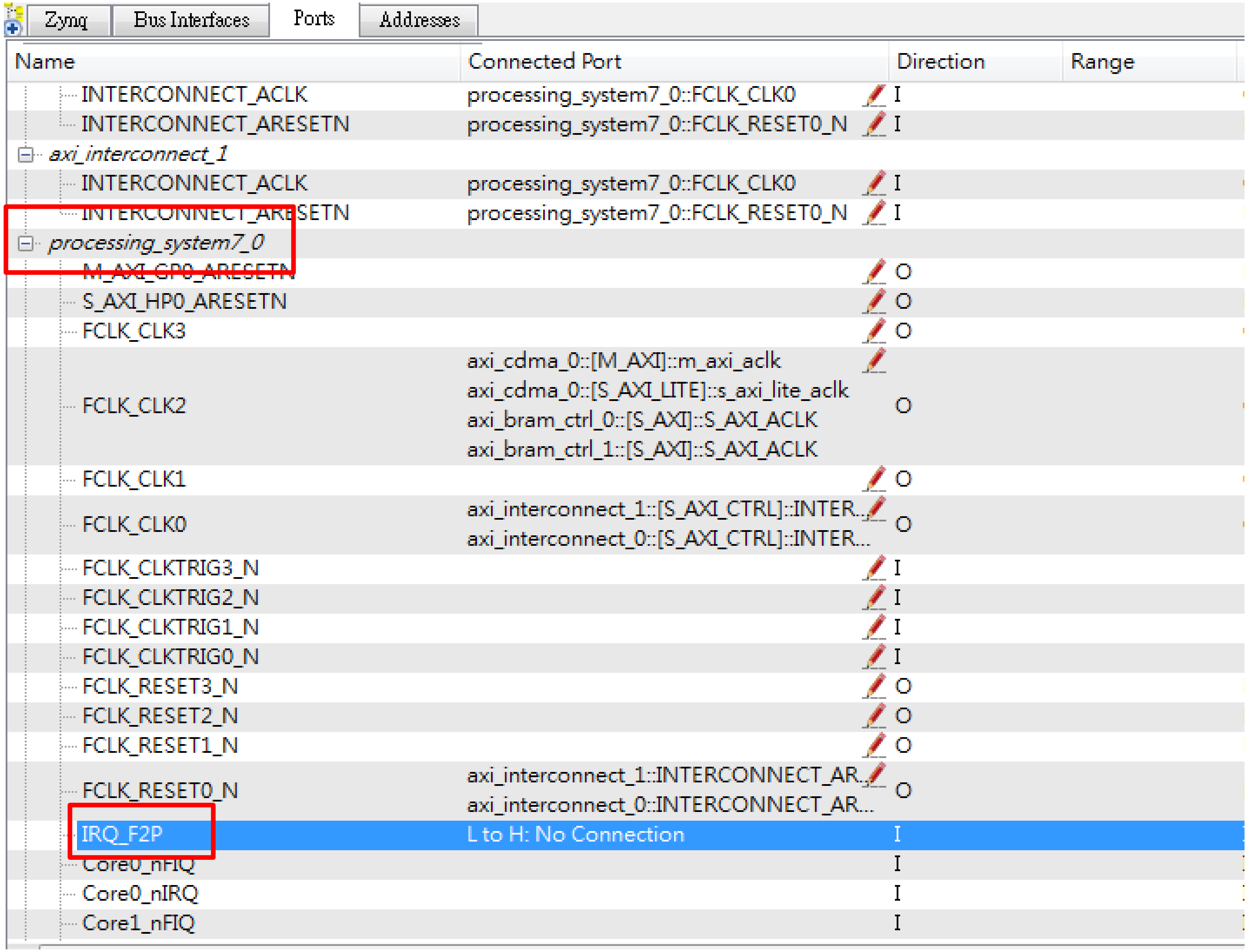
CPU Clock Ratio 6:2:1

Component	Clock Source	Requested Frequency(MHz)	Actual Frequency(MHz)	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
PL Fabric Clocks				
FCLK_CLK0	IO PLL	100.000000	100.000000	0.10 : 250.00
FCLK_CLK1	IO PLL	150.000000	142.857132	0.10 : 250.00
FCLK_CLK2	IO PLL	85.000000	83.333336	0.10 : 250.00
FCLK_CLK3	IO PLL	50.000000	50.000000	0.10 : 250.00
System Debug Clocks				
Timers				

Modify FCLK_CLK2 to be 85MHz

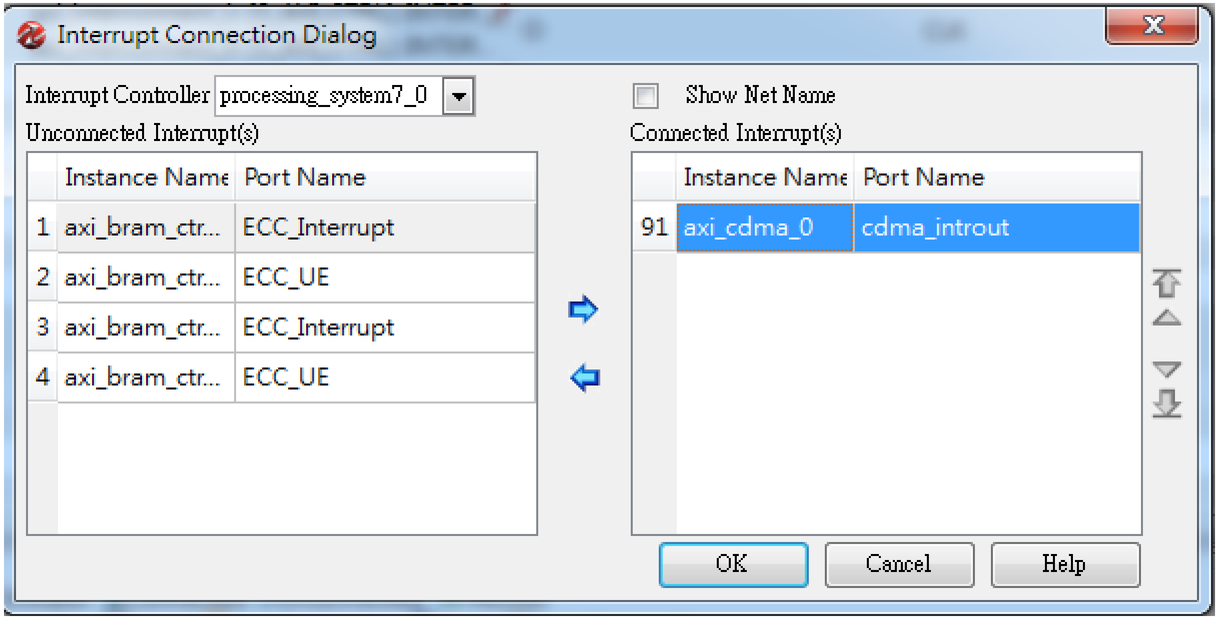
This is the clock we connect to BRAM controller and CDMA.

We need to modify the clock or there will be time violation.



Name	Connected Port	Direction	Range
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I	
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I	
axi_interconnect_1			
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I	
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I	
processing_system7_0			
M_AXI_GP0_ARESETN		O	
S_AXI_HP0_ARESETN		O	
FCLK_CLK3		O	
FCLK_CLK2	axi_cdma_0::[M_AXI]::m_axi_aclk axi_cdma_0::[S_AXI_LITE]::s_axi_lite_aclk axi_bram_ctrl_0::[S_AXI]::S_AXI_ACLK axi_bram_ctrl_1::[S_AXI]::S_AXI_ACLK	O	
FCLK_CLK1		O	
FCLK_CLK0	axi_interconnect_1::[S_AXI_CTRL]::INTER... axi_interconnect_0::[S_AXI_CTRL]::INTER...	O	
FCLK_CLKTRIG3_N		I	
FCLK_CLKTRIG2_N		I	
FCLK_CLKTRIG1_N		I	
FCLK_CLKTRIG0_N		I	
FCLK_RESET3_N		O	
FCLK_RESET2_N		O	
FCLK_RESET1_N		O	
FCLK_RESET0_N	axi_interconnect_1::INTERCONNECT_AR... axi_interconnect_0::INTERCONNECT_AR...	O	
IRQ_F2P	L to H: No Connection	I	
Core0_nFIQ		I	
Core0_nIRQ		I	
Core1_nFIQ		I	

Now we have to set the interrupt.



Interrupt Controller processing_system7_0

☐ Show Net Name

Unconnected Interrupt(s)

	Instance Name	Port Name
1	axi_bram_ctr...	ECC_Interrupt
2	axi_bram_ctr...	ECC_UE
3	axi_bram_ctr...	ECC_Interrupt
4	axi_bram_ctr...	ECC_UE

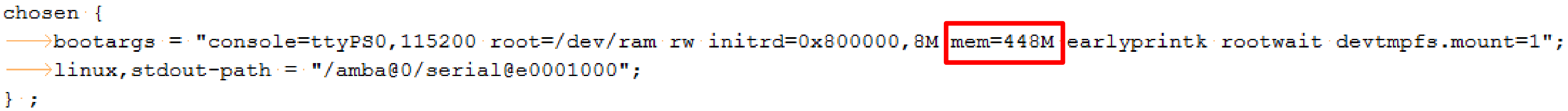
Connected Interrupt(s)

	Instance Name	Port Name
91	axi_cdma_0	cdma_introut

Move axi_cdma_0 to the other side.

Next

- Export to SDK
  - Create BOOT.BIN
  - Create devicetree.dtb and make the modification. Also to prevent the kernel from using HP0 memory we should modify bootargs.

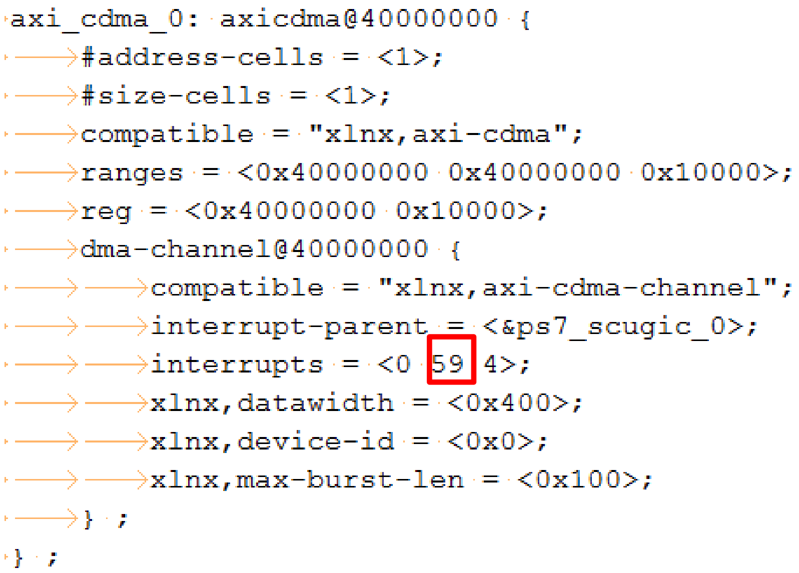


```
chosen {
	bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x800000,8M mem=448M earlyprintk rootwait devtmpfs.mount=1";
	linux,stdout-path = "/amba@0/serial@e0001000";
} ;
```

- Driver
  - Interrupts
  - Mutex (Mutual exclusion)
  - Linux kernel wait queue
  - Kernel/User memory, copy_from_user, copy_to_user
- Java user application.

Interrupts

- Polling v.s Interrupts
  - Polling: CPUs keep checking if something need to be handled.
  - Interrupt: Devices inform CPU.
- CDMA interrupt
  - OS recognized by IRQ number
  - But in OS, this should be IRQ_number + 32



```
axi_cdma_0: axicdma@40000000 {
	#address-cells = <1>;
	#size-cells = <1>;
	compatible = "xlnx,axi-cdma";
	ranges = <0x40000000 0x40000000 0x10000>;
	reg = <0x40000000 0x10000>;
	dma-channel@40000000 {
		compatible = "xlnx,axi-cdma-channel";
		interrupt-parent = <&ps7_scugic_0>;
		interrupts = <0 59 4>;
		xlnx,datawidth = <0x400>;
		xlnx,device-id = <0x0>;
		xlnx,max-burst-len = <0x100>;
	} ;
} ;
```

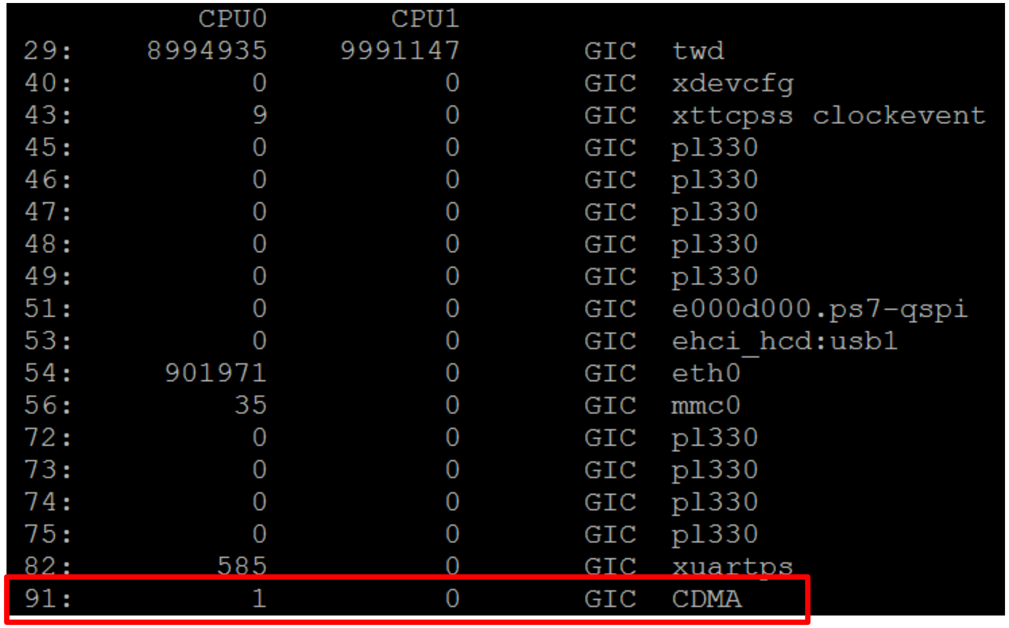
```
#define IRQ_CDMA                59
```

cdma_probe()

```
		PDEBUG("Set Interrupt\n");
		int ret;
		ret = request_irq(IRQ_CDMA+32, cdma_irq, IRQF_SHARED, "CDMA", cdma_dev);
```

irq handler

```
 static irqreturn_t cdma_irq(int irq, void *data)
 {
		struct cdma_dev *dev = data;
		iowrite8(0x01,dev->dev_virtaddr+0x6);
		dev->bytes_written += dev->count;
		dev->busy = 0;
		
        wake_up_interruptible(&cdma_wait);
		PDEBUG("IRQ\n");
		return IRQ_HANDLED;
 }
```

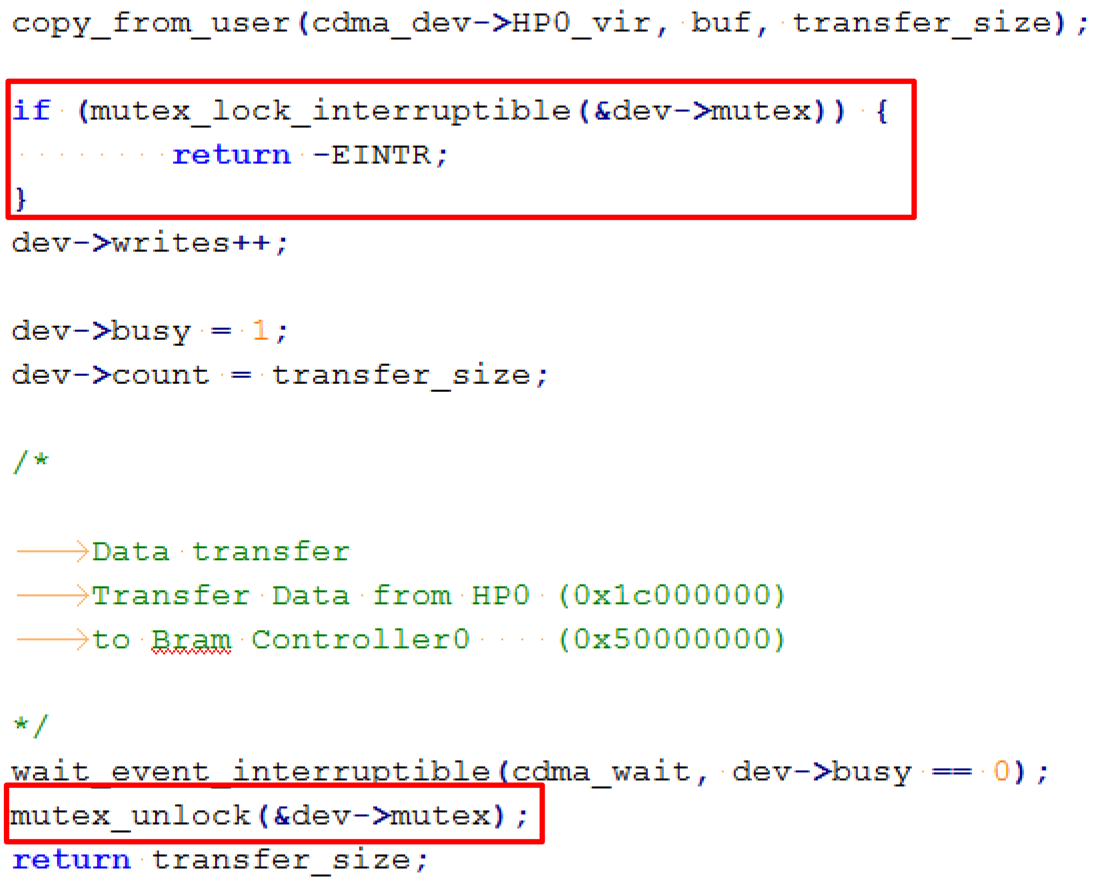


```
           CPU0       CPU1
 29:    8994935    9991147       GIC  twd
 40:          0          0       GIC  xdevcfg
 43:          9          0       GIC  xttcpss clockevent
 45:          0          0       GIC  pl330
 46:          0          0       GIC  pl330
 47:          0          0       GIC  pl330
 48:          0          0       GIC  pl330
 49:          0          0       GIC  pl330
 51:          0          0       GIC  e000d000.ps7-qspi
 53:          0          0       GIC  ehci_hcd:usb1
 54:     901971          0       GIC  eth0
 56:         35          0       GIC  mmc0
 72:          0          0       GIC  pl330
 73:          0          0       GIC  pl330
 74:          0          0       GIC  pl330
 75:          0          0       GIC  pl330
 82:        585          0       GIC  xuartps
 91:          1          0       GIC  CDMA
```

Mutex

- Why do we need mutex?
  - Under environment of multithread.
  - Prevent more than one thread to access resource.
  - mutex v.s semaphore
- Easy concepts
  - Lock mutex -> execute critical section -> unlock mutex

cdma_write()

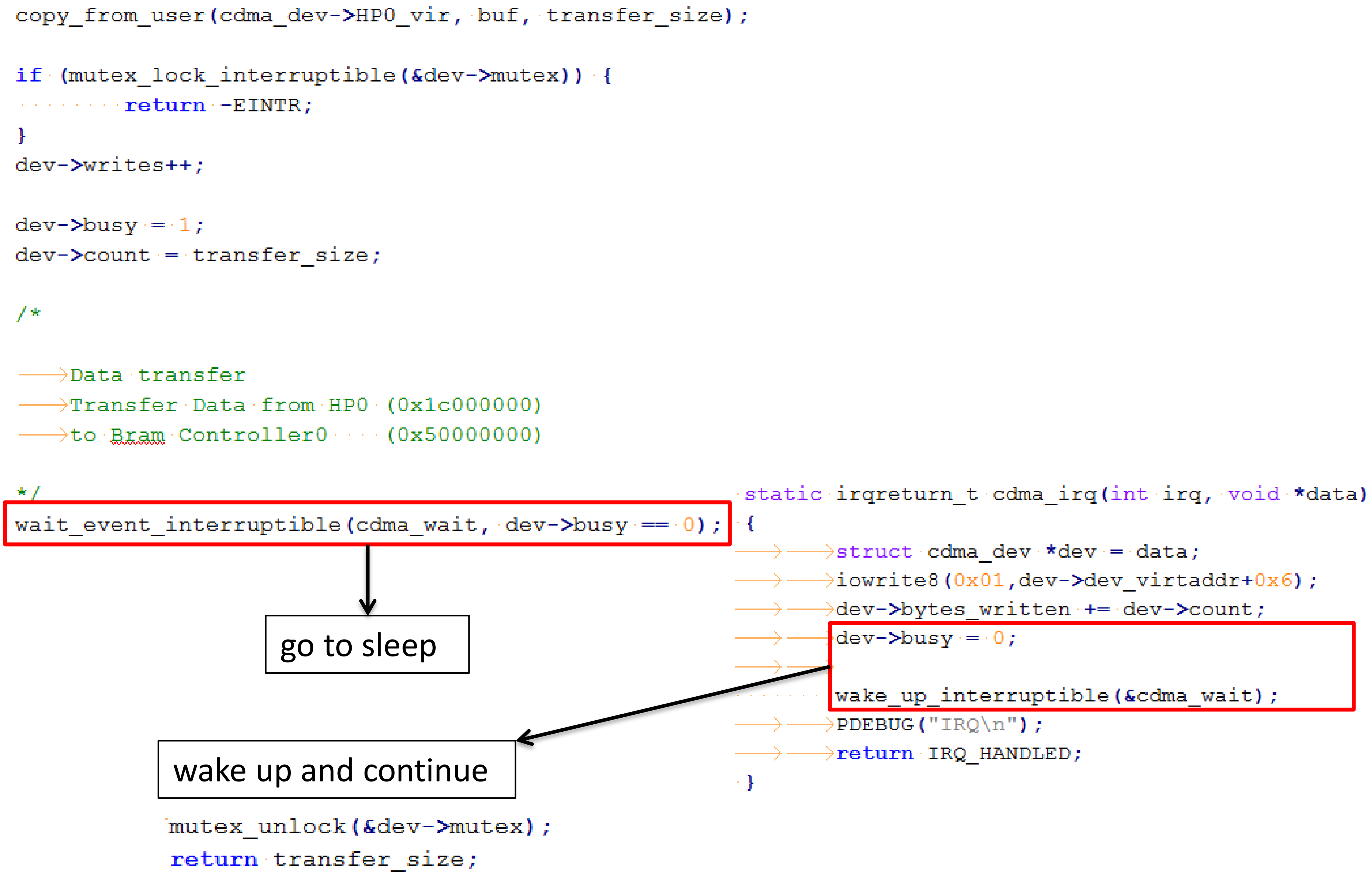


```
copy_from_user(cdma_dev->HP0_vir, buf, transfer_size);

if (mutex_lock_interruptible(&dev->mutex)) {
        return -EINTR;
}
dev->writes++;

dev->busy = 1;
dev->count = transfer_size;

/*

	Data transfer
	Transfer Data from HP0 (0x1c000000)
	to Bram Controller0    (0x50000000)

*/
wait_event_interruptible(cdma_wait, dev->busy == 0);
mutex_unlock(&dev->mutex);
return transfer_size;
```

Linux Kernel wait queue

- We need to stop the process and wait for CDMA (or other devices) to complete their works.
- A while loop is not a good idea.
  - Occupying CPU resource
- Linux Kernel wait queue is designed to solve this situation.
  - Process go to sleep and release CPU resources to other processes.

cdma_write()



```c
copy_from_user(cdma_dev->HP0_vir, buf, transfer_size);

if (mutex_lock_interruptible(&dev->mutex)) {
        return -EINTR;
}
dev->writes++;

dev->busy = 1;
dev->count = transfer_size;

/*

	Data transfer
	Transfer Data from HP0 (0x1c000000)
	to Bram Controller0    (0x50000000)

*/
wait_event_interruptible(cdma_wait, dev->busy == 0);
```

go to sleep

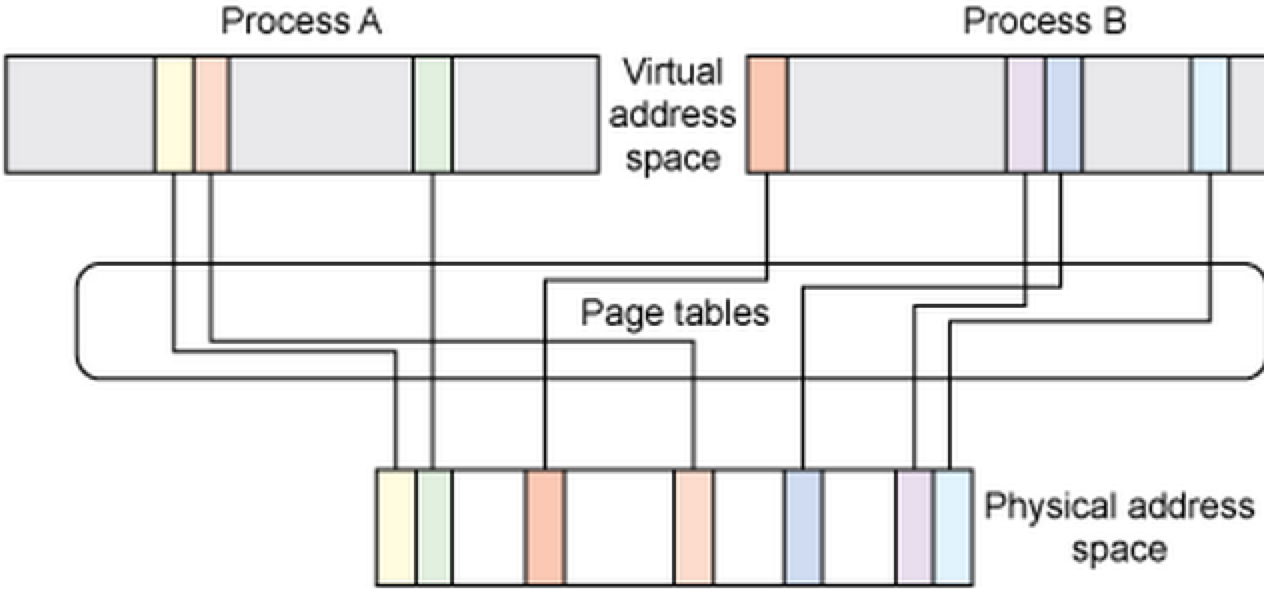
```c
static irqreturn_t cdma_irq(int irq, void *data)
{
		struct cdma_dev *dev = data;
		iowrite8(0x01,dev->dev_virtaddr+0x6);
		dev->bytes_written += dev->count;
		dev->busy = 0;
		
      wake_up_interruptible(&cdma_wait);
		PDEBUG("IRQ\n");
		return IRQ_HANDLED;
}
```

wake up and continue

```c
mutex_unlock(&dev->mutex);
return transfer_size;
```

copy_from_user, copy_to_user

- Why do we need this?
  - User space memory is fragmented to pages.
  - For security reason.



http://www.ibm.com/developerworks/library/l-kernel-memory-access/