# AXI Interface

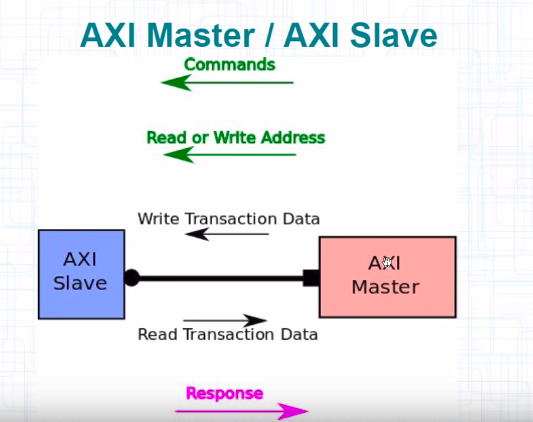
# AXA is a System on Chip(SoC) Bus which is a set of protocols that each module in the chip follows to communicate with each other.AXI has three basic concepts:

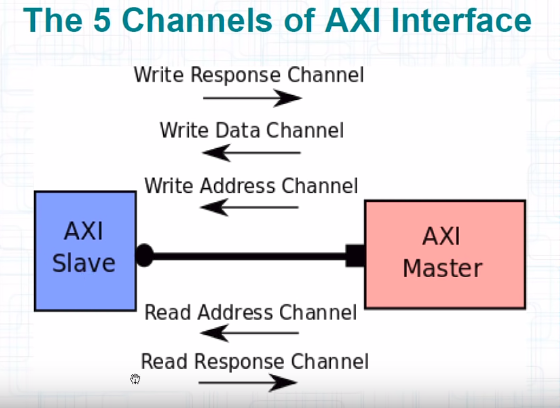
AXI Master, AXI Slave and AXI interconnect.

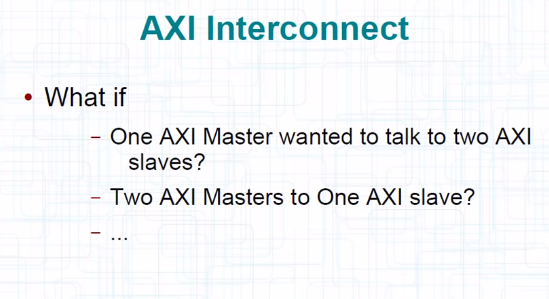
Transaction: Transfer of data from one point in the hardware to another point

Master: Initiates the transaction

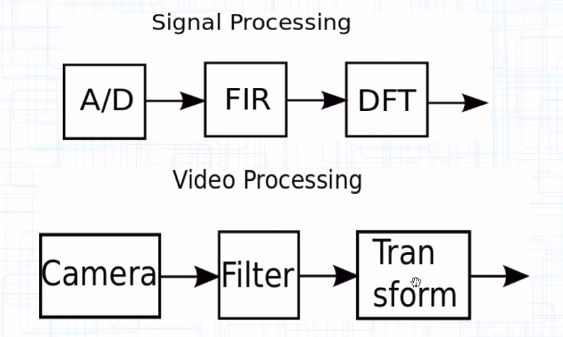
Slave: Responds to the initiated transaction



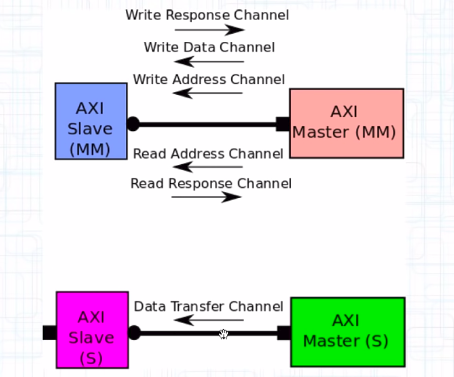


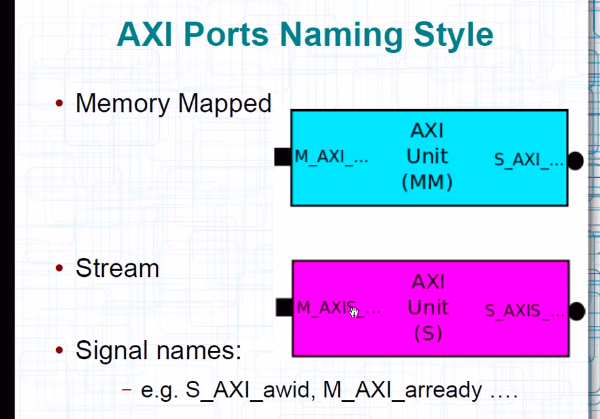


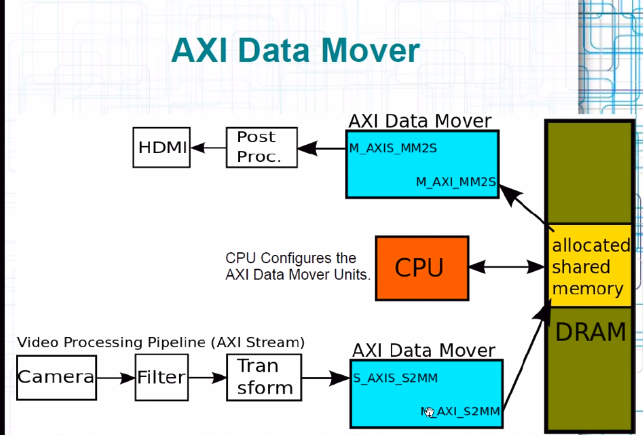
On your board there just can be flow of data and nothing else.In that case AXI stream interface comes into play.In this we don’t need an address.There is only one AXI channel,one way.



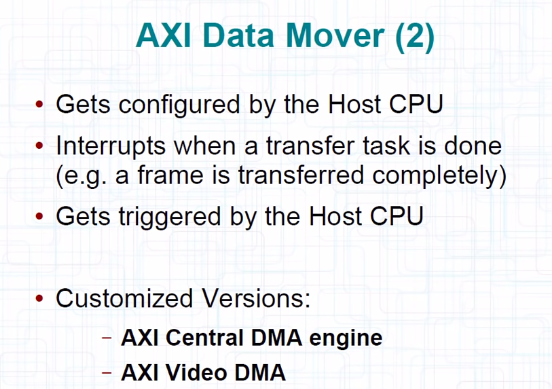
In the above picture, FIR and Filter acts like AXI stream interface.We don’t need to provide address to write or read data on it.It is just passing the data coming from A/D to DFT(in case of FIR).







Suppose we want to store the streamed data somewhere, but there is a problem.The “Transform” block doesn’t carry an address in which the data can be stored.Here comes the AXI Data Mover which converts the stream interface to Memory Mapped Interface.



# **Lesson 9 – Software development for ZYNQ using Xilinx SDK**

# Transfer data from ZYNQ PL to PS

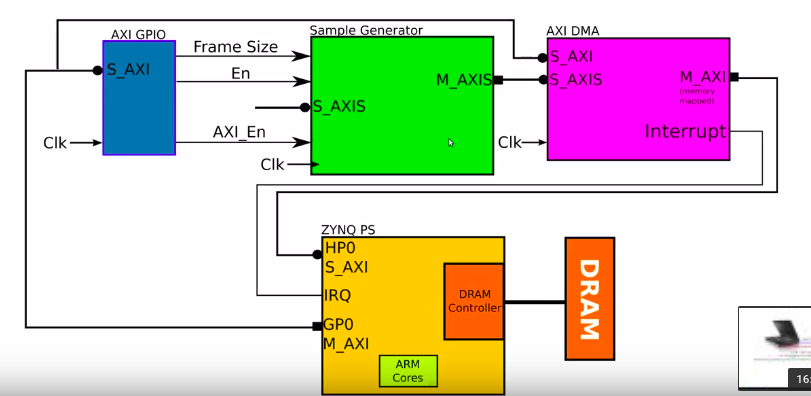
Operating System which you can run on ARM host:

1-Bare metal(standalone)

2-Linux

3-FreeBSD

4-Windows……



The hardware that we are going to implement in the PL

Sample Generator is using pure RTL flow to develop AXI stream interface covered in lec 7.AXI DMA works like a AXI Data Mover,i.e. Stores the AXI stream data into a particular place in memory.AXI GPIO is responsible for driving the signals to the Sample Generator.

He has used HP0 AXI Memory Mapped Slave Interface for transferring data from PL to PS.

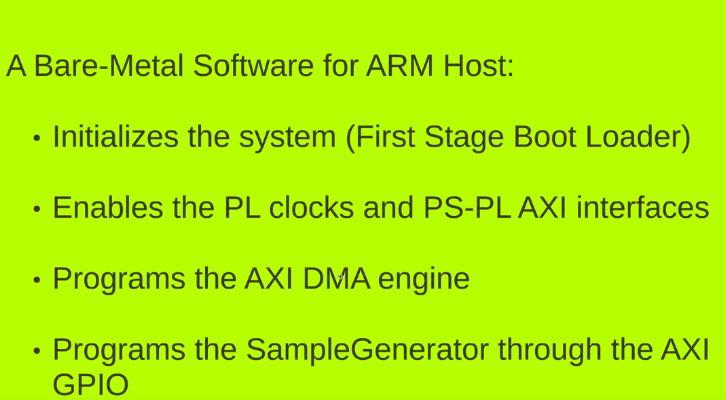
And GP0 AXI Memory Mapped Master Interface for transferring data from PS to PL.These interfaces are at the PS.

GP0 is used to configure AXI GPIO and AXI DMA.

We configure the AXI GPIO as output so that we can give any values to it.

We tell AXI DMA the address to which it has to store the data received from Sample Generator.

ARM Cores are used for these configuration.



# Creating the Base Hardware for exporting to Xilinx SDK

We develop the code in Vivado environment.

Add the repositories made in Lec 7(Sample Generator, AXI DMA, Zynq PS).

Blocks can be recustomised by double clicking on them.

After adding Zynq PS Click on Run Block Automation.Recustomise Zynq PS to add HP0 port.Whenever AXI DMA finishes it creates an interrupt signal that it has finished and sends to Zynq PS.So for reading that recustomise Zynq PS and add the interrupt input pin.We can add a constant block to give a constant signal to a pin.Select the no. of bits and and the value and you are done.Like in this example we keep the value of tkeep signal to be always “1111” which tells which bit of my data is valid.We add two AXI GPIO modules.One to drive ‘en’-enable pin and other to drive ‘Framesize[7:0]’ pin.

For connecting the AXI Memory mapped slave ports to the ARM Host we click on ‘Run Connection Automation’.

After all this we create an HDL wrapper.

Then click on ‘Generate Output Products’.

Then ‘Run Synthesis’

‘Run Implementation’

‘Generate Bitstream’

Wait for it.

‘Export hardware to SDK’

‘Launch SDK’ after exporting is done

Preparing the first stage boot loader

After you are done with launching SDK, you can open a C code that is the first code that the ARM host runs.It basically configure the registers in the PS.

Go to file, create a new Porject.This project will be used to start the ps7\_init() in the C code.We write this project name as fsbl(first stage boot loader).From list of templates select Zynq FSBL.

As you click ok the entire First Stage Boot Loader will be created by the Xilinx Software Develpment Kit(SDK) automatically.It is ready to be executed on your ARM Host.

The main.c file has all the conditions through which we load the Zynq device ,i.e. QPI Flash, SD card or JTAG.

To enable all the debug messages of fsbl write ‘#define FSBL\_DEBUG’.This shows what the fsbl is doing at each moment of time.As you save it the environment automatically compiles it.

Fsbl.elf is the name of the executable file.

Now go to the workspace where your elf file is.

Use this command ‘find . -name \*.elf’ to list all the elf files in the directory.

‘Cd ..’ is the command to go back to the previous directory.

Find the bitstream file using ‘find . -name \*.bit’

Xilinx machine debugger ‘xmd’ to run XMD in the directory where both files are present.

‘fpga’ to first program the PL

‘exec find . -name \*.bit’ this will tell you the name of the bitstream file with its directory.

‘fpga -f .(address of bit file)’

Now the PL is programmed.

‘connect arm hw’ we are connected to the hardware on the ARM system which alllowss us to debug it.Basically in order to get the control of CPU completely.

‘exec find . -name \*.elf’ to find the elf file

‘dow .(address of elf file)’ to download the elf file to the memory

We use another terminal to see the output of fsbl being executed.

Type ‘sudo picocom --b 115200 /dev/ttyACMO’

Type ‘run’ in the previous terminal.You can see the output in the new terminal.

FSBL was an program to initialise the PS.Now we create a new project in Xilinx SDK with name “hello\_world” and the template Hello World.

Now repeat the above steps.

‘exec find . -name \*.elf’

‘dow .(address of elf)’

You will get a message

“Debug Operation Not Supported on the Target, Current Processor State is not “Stopped””

This is because the processor is not stopped so you can’t write on the memory.

Type ‘stop’ to stop the processor and then rewrite the dow command.

‘run’

You will get a message Hello World on the picocom terminal.

We can change the main.c code of Hello World to execute more things.

Transfer Data from PL to PS using AXI DMA

There is a function in main.c with name ps7\_init() which is responsible for initialising the shift registers of PS.

There is another important function named ps7\_post\_config() which is for manually initialising the registers if it didn’t happened automatically.

If you want to add a .c file to your project just copy and paste it to your folder project.

To include it in main.c type #include “ps7\_init.h” in the beginning.

To print something you can use print(Hello World\n\r); or the Xilinx version of print which is xil\_printf(“Hello World\n\r”);

To use the latter command you need to include xil\_io.h type #include <xil\_io.h>

To handle the interrupt signals generated by AXI DMA include the header file by typing #include “xscugic.h”

Now add the most important file that contains all the information of all hardware on the PL.

Type #include “xparameters.h”

Open this file xparameters.h, you will see many parameters.we are interested in the addresses which are defined by XPAR\_AXI\_DMA\_0\_BASEADDR(base address for AXI DMA) and XPAR\_AXI\_DMA\_0\_HIGHADDR(last address).

XPAR\_CPU\_CORTEXA9\_0\_CPU\_CLK\_FREQ\_HZ 666666687 is the parameter for the clock frequency of CPU.

Another important parameter is

XPAR\_FABRIC\_AXI\_DMA\_0\_S2MM\_INTROUT\_INTR 61 61 is the interrupt number.If you double click on Zynq PS you can see that interrupt line is connected to 61.

Now lets write the intitial configuration function for AXI DMA,

int InitializeAXIDMA(void){

unsigned int tmpval;

tmpval = Xil\_In32 (XPAR\_AXI\_DMA\_0\_BASEADDR + 0x30); // memory read

tmpval = tmpval | 0x1001; //modification in the read value , first bit is set 1 to start the S2MM //channel , 4th bit is set 1 to enable AXI DMA to generate interrupts

Xil\_Out32 (XPAR\_AXI\_DMA\_0\_BASEADDR + 0x30 , tmpval); // memory write

tmpval = Xil\_In32((XPAR\_AXI\_DMA\_0\_BASEADDR + 0x30);

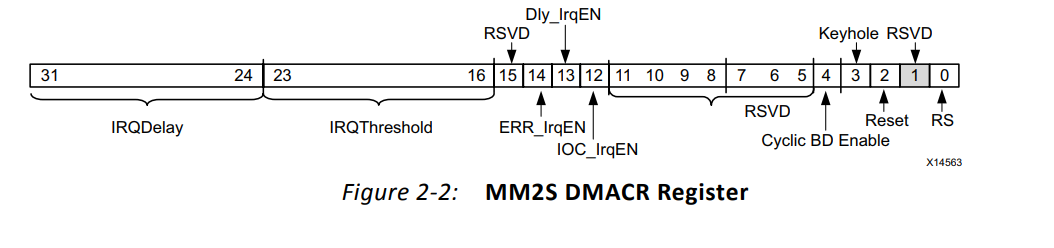
xil\_printf ( “value for dma control register : %x\n\r , tmpval); // printing the updated value of register to make sure it is what we wanted

Return 0;

}

To get the documentation on AXI DMA double click on it, it will show.

To see the address of the stream to memory mapped register S2MM\_DMACR.RS look at the table in the register space section in product specification section.To start the S2MM channel S2MM\_DMACR.RS=1.



Here is the link for the AXI DMA documentation:<https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf>

Now come back to the main program.

Now lets see the code to enable Sample Generator,

Int EnableSampleGenerator ( unsigned int numberofWords) {

//set the gpios direction as output

// the gpio is by default output, so this is not needed.

//set the value for Framesize

Xil\_Out32 (XPAR\_AXI\_GPIO\_0\_BASEADDR, numberofWords);

//enable Sample Generator

Xil\_Out32 (XPAR\_AXI\_GPIO\_1\_BASEADDR , 1);

Return 0;

}

Here is the link for AXI GPIO documentation:<https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf>

You can see the documentation on how you should configure the GPIO in order to read data.

Now how to initialise the interrupt control system,

Type IntitializeInterruptSystem (XPAR\_PS7\_SCUGIC\_0\_DEVICE\_ID);

IntitializeInterruptSystem( device ID) is a function in which you will also find a function InterruptHandler which basically tells what to do if the interrupt signal is high.

In the InterruptHandler function you will see a function StartDMATransfer which starts the transfer.

void StartDMATransfer ( unsigned int dstAddress, unsigned int len ) {

//write destination address to S2MM\_DA register

Xil\_Out32 ( XPAR\_AXI\_DMA\_0\_BASEADDR + 0x48, dstAddress );//destination address of the data

//write length to S2MM\_DA register

Xil\_Out32 ( XPAR\_AXI\_DMA\_0\_BASEADDR + 0x58, len );//length of data to write

}

Now the final line of code

getchar(); // it will start transfer when you press a key of keyboard

StartDMATransfer ( 0xa000000, 256 );

Now the program is complete and we can test it on board.

Load the program into the board and test.

For reading at different locations in the DRAM memory type mrd 0xa000000 in the terminal.

**Debugging Software using Xilinx SDK**

When you change the version of Vivado environment and you open your previous project it asks if you want to update the blocks.Go to IP status below and you can choose which block you want to update.To open the program through terminal change directory to vivado folder, type

vivado addressoffile (its a .xpr file)

Run Synthesis, Run implementation, Generate bitstream

Then go to SDK environment

type xsdk to run the SDK

Select a workspace

To get the disassembled code from the .elf file type

Arm-xilinx-eabi-objdump -d address\_of\_elf\_file > address\_of\_elf\_file.asm

To see this file type

vim address\_of\_asm\_file

You can actually see the function calls of your program and how does each line of code is executed from this file.

We used xmd command to load the datastream to the PL and also to load the elf file to the ARM host.We can do the same tasks using SDK also.

Go to Xilinx Tools

Go to Program FPGA

Load the location of the bitstream file ( .bit file)

Click on Program

Click on xmd Console on the top right

Click on the green play button on top middle to run the .elf file . This will execute this file on your ARM host.

You can debug the elf file also.

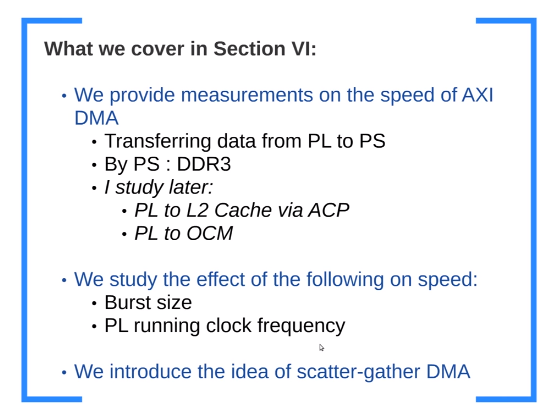
Click on the button left to the green play button to debug .elf file.

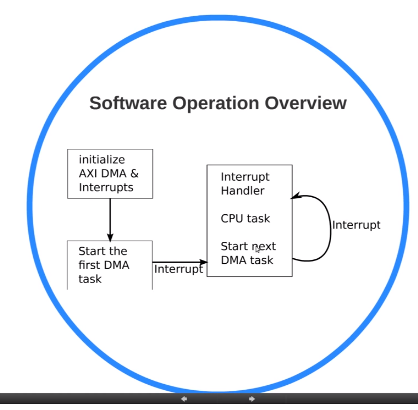
The layout of the SDK will change as we are debugging the program.

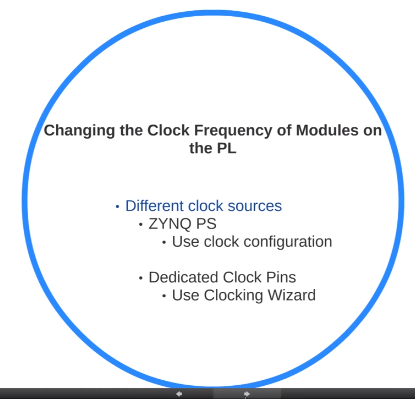
Important buttons are Step Over(F6) And Step Into(F5).

Write the breakpoints to the line which you want to debug.Use Step into and Step Over buttons to go deeper and leave the execution.

**AXI DMA Performance Measurement**







There are two parameters of AXI DMA that affect the performance of the data transfer from the PL to the PS.These are Width of Buffer Length Register and Max Burst Size.Double click on AXI DMA to see it.Burst size is the number of datawords that you read/write in each read/write transaction.