Assignment for CSCE 617

Due in 2 weeks

In this assignment, you will build a simple reconfigurable computing platform using Carbon SoC designer. It will give you a further understanding of what Carbon SoC designer can do. And you will get familiar with AMBA AXI protocol.

Before doing the assignment, you should have some basic knowledge about how to create a Carbon component from RTL files using Model Studio, how to run simulation in SoC designer and how to check registers status during simulation. You should learn these from the assignment 1 and 2.

Figure 1 shows the system you are going to build. There are three main components. We use AXI protocol for the communication. *AXIv2_Stub* and *AXIv2_Mem* are library components. You can imagine *AXIv2_Stub* is a commander. Both of them are built-in AXI compatible. You will write the logic for the Computing Unit using Verilog.

Figure 1 overview of the system

What the system behaves:

1. *AXIv2_Stub* issues instructions to Computing Unit.
2. Computing Unit will do some operations based on the received instructions.
3. Once the Computing Unit completes the specific operation, it will notify the *AXIv2_Stub* through Feedback Signal channel.
4. Once *AXIv2_Stub* receives the Feedback Signal, go to step 1.

AXIv2_Mem is just a memory component to store your operating data. At first, the memory will be initialized with some random data. Each time a group of two consecutive data will be handled in the computing unit. Therefore, the memory looks like Table 1. (In this example, the width of data is 8-bit).

Table 1 memory format

address	value	
0x00	A1	A2
0x02	B1	B2

0x04	C1	C2
0x06	D1	D2

We define two operations (add and multiply) within the Computing Unit(Table 2).

Table 2 types of operations

No.	Name	description
1	add	addition of two data
2	multiply	multiplication of two data

The command from AXIv2_Stub is composed by several values. The format of the instruction is shown on Table 3.

Table 3 command format

order	name	description
1	data address	the beginning address of operating data
2	number of data	the total number of data that will be operated, not the number of operation
3	operation type	1 is add, 2 is multiply

Summary of the work in this assignment:

- Write a script to drive *AXIv2_Stub* to issue commands and receive the feedback signal from the computing unit (The following tutorial will show you how to write scripts for *AXIv2_Stub*.)
- Use Verilog to implement the logic in the computing unit such that it can receive command data through AXI protocol from *AXIv2_Stub*, read the operating data with AXI protocol from *AXIv2_Mem*, do the operation and send the feedback signal once execution of the command is done.

MxScript Tutorial

MxScript is an interpreted language with a syntax that is similar to C. You can use MxScript to run Carbon simulation in batch mode. It also can be used to drive a library component, *AXIv2_Stub*. The following shows you how to use *AXIv2_Stub* with MxScript. For more information, please reference to /opt/SoCDesigner/doc/ Carbon_MxScript_Reference_Manual.pdf

Create a system that contains *AXIv2_Stub* and *AXIv2_Mem* in SoC Designer, like Figure 2. Connect *axi2_m* of *AXIv2_Stub* to *axi_s* of *AXIv2_Mem*

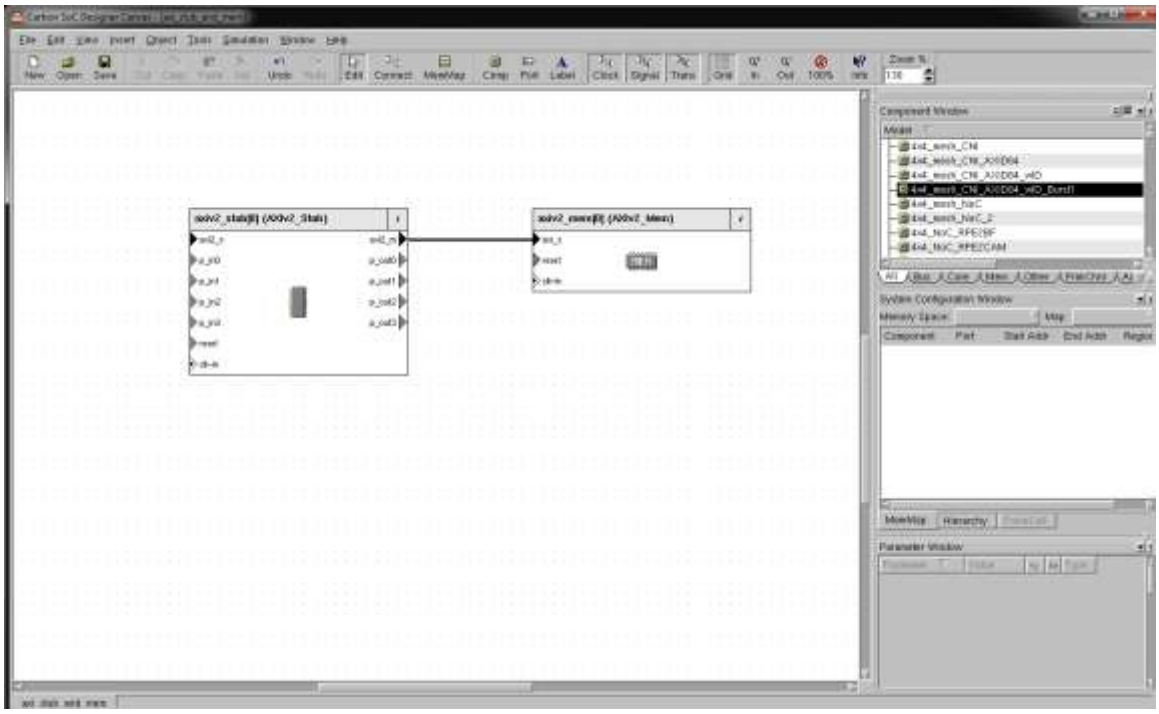

Figure 2 simple system

We want, first, the *AXIv2_Stub* writes data to *AXIv2_Mem*, then, *AXIv2_Stub* will read data from *AXIv2_Mem*. The two operations use AXI protocol. The width of data is 8-bit(1 byte).

Create a file named *stub.mxscr*. *.mxscr* is MxScript. Copy/type the code below to *stub.mxscr.*

```
#include "AXIv2_Stub_Macros.h" //please include the header file, no need to care where it is

/*-----------------write-------------------*/
message(MX_MSG_INFO,"Start to write memory");
int i=0;
int write_data=1;
```

```
int write_address = 0x00000000;
for(i=0;i<100;i++)          //do 100 times
{

        BURST4_WRITE(write_address,1,write_data,write_data+1,write_data+2,write_data+3);
        /*
                BURST4_WRITE(address, width of data(in terms of byte), value1, value2,...,
value4)
                You can also use BURST1_WRITE, BURST2_WRITE, ..., BURST16_WRITE.
        */

        write_address = write_address + 4;      //update the writing address
        write_data = write_data + 4;            //update the writing data
}

message(MX_MSG_INFO,"Writing memory is done!!");

/*-----------------read-------------------*/
int read_address = 0x00000010;
message(MX_MSG_INFO,"Start to read memory");

int read_data_1=0;
int read_data_2=0;
int read_data_3=0;
int read_data_4=0;

BURST4_READ(read_address,1,read_data_1,read_data_2,read_data_3,read_data_4);
/*
        BURST4_READ(address, width of data(in terms of byte), variable1, variable2,...,
variable4)
        You can also use BURST1_READ, BURST2_READ, ..., BURST16_READ.
*/

//display the read data
message(MX_MSG_INFO,"read_data_1: %d",read_data_1);
message(MX_MSG_INFO,"read_data_2: %d",read_data_2);
message(MX_MSG_INFO,"read_data_3: %d",read_data_3);
message(MX_MSG_INFO,"read_data_4: %d",read_data_4);
message(MX_MSG_INFO,"Reading memory is done!!");

stop();  //stop the simulation
```

Now run simulation for the system. Then it will ask you to select a file for AXIv2_Stub (Figure 3). Choose *stub.mxscr* you have just created and click **Proceed** (Figure 4).

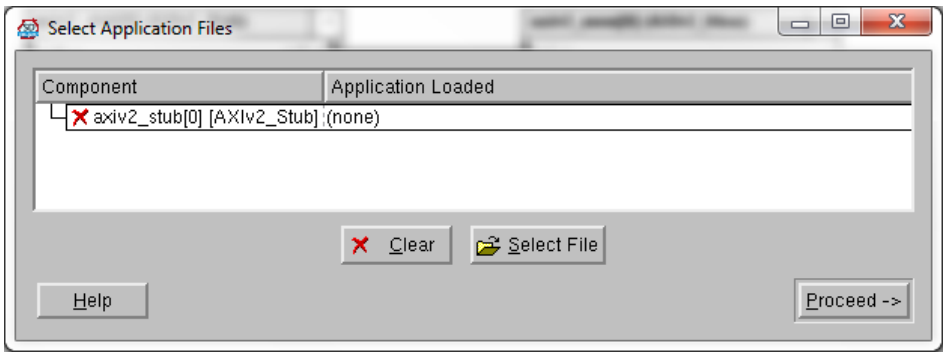

Figure 3 select a file for AXIv2_Stub

Figure 4 after select the mxscript file

On the *AXIv2_Mem*: **right click-> view memory**. Now you can see the content of AXIv2_Mem (Figure 5).

Figure 5 content of AXIv2_Mem

Click **Run** to execute the simulation. It stops automatically (Figure 6). We are able to see the content of AXIv2_Mem has been changed and turned red (Figure 7). At the message window (the bottom of Figure 6), we also can see the information we write in the mxscript file.

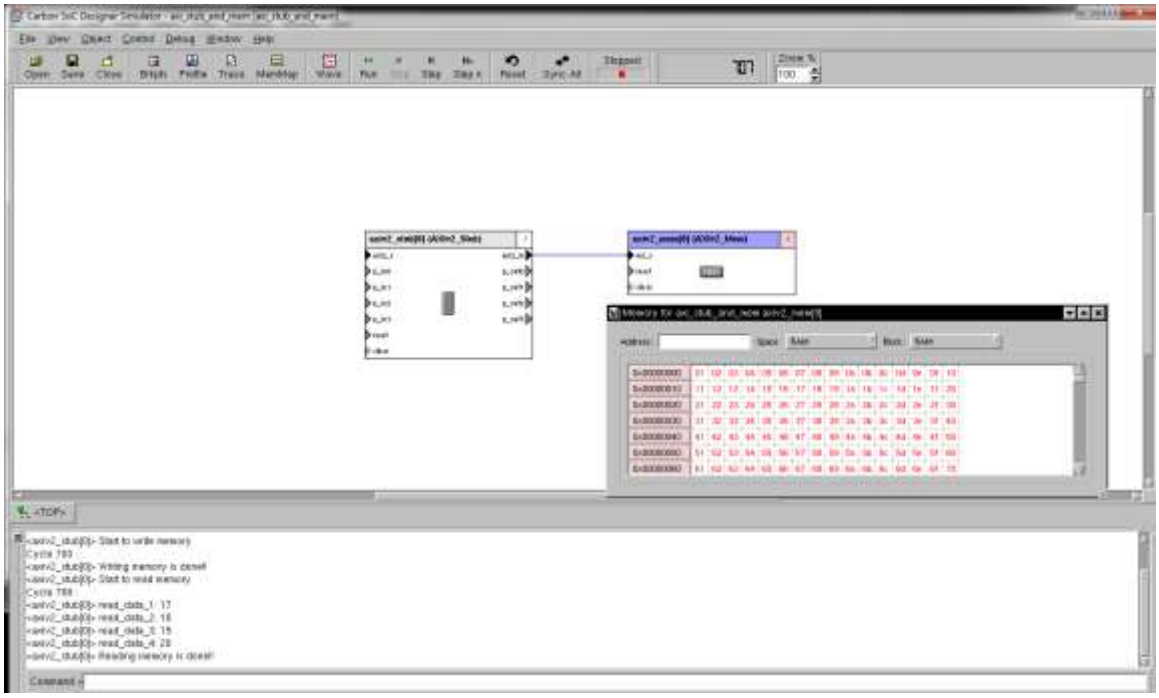

Figure 6 simulation is done

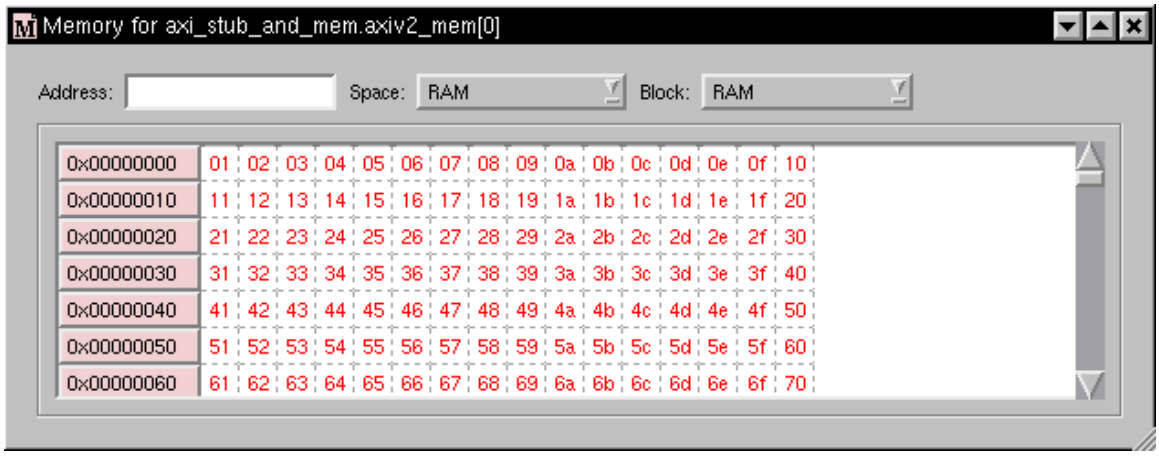

Figure 7 the content of AXIv2_Mem is changed

Contact: Deam Ieong (deam_ieong-0814@neo.tamu.edu)