Aarhus Universitet

ERTS - Group 2

Rapport

Assignment 1

Gruppemedlemmer:	**Studienumre:**
Daniel Tøttrup	201509520
Johan Vasegaard Jensen	201505665
Stinus Lykke Skovgaard	201401682

15. september 2019

Kapitel 1

ASSIGNMENT 3.1

Kapitel 2

ASSIGNMENT 3.2

Kapitel 3

ASSIGNMENT 3.3

Kapitel 4

EXERCISE 3.4

Create a cycle accurate communication model of a master and slave module that uses the Avalon Streaming Bus interface (ST). Simulate that a master are transmitting data to a slave module as illustrated in the figures 5-2 and 5-8. The slave should store received data from the master in a text file. Include in the model a situation where the data sink signals ready = '0'. The simulated result should be presented in the GTK wave viewer, so a VCD trace file must be created. It should be possible to configure the channel, error and data size define in a separate header file as illustrated in the below code snippet.

A total of of three .h files and 4 .cpp files have been made in this exercise. A Master.h, Slave.h and Top.h with corresponding .cpp files. A main.cpp has also been made.

The code below shows the Master.cpp code. The comments explain how the code works, but the essence of it is just to send an integer(1) and afterwards increment that integer for the next send cycle. That means the master will send the numbers 1...10. The master and slave are setup to use the Avalon Streaming Bus interface. The data transfer uses backpressure with a ready latency set to 1.

```
1  void Master::MasterThread(void)
2  {
3    //Arbitrary data to send.
4    dataToSend = 1;
5    while (1)
6    {
7      // Update ready state - Will not be updated until next clock
           cycle.
8      readyState = ready.read();
9
10     // Check if slave is ready to receive data
11     if (readyState)
12     {
13       // Indicate that data is being written
14       valid.write(true);
15
16       // Write data
17       data.write(dataToSend);
18
19       //Change data
```

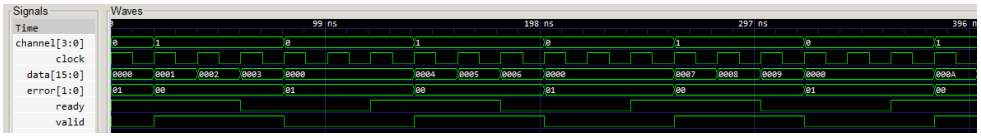
```
20        dataToSend += 1;
21
22        // Set channel to 1
23        channel.write(1);
24
25        // set error to 0
26        error.write(0);
27      }
28
29      else
30      {
31        // Indicate that no data is being written
32        valid.write(false);
33
34        // Write 0 to data (only to prettify in GTK-viewer).
35        data.write(0);
36
37        // Set channel to 0
38        channel.write(0);
39
40        // set error to 1
41        error.write(1);
42      }
43
44      // Wait until next clock cycle.
45      wait();
46    }
47  }
```

The slave.cpp can be seen below. Again the comments explains how the code works. Whenever the slave has received 10 integers from the master, it begins to print those numbers to a text file. The slave also changes it's ready state every three clock cycle.

```
1   void Slave::SlaveThread(void)
2   {
3     // Initial state of "ready".
4     ready.write(false);
5
6     // Simulate ready going low for 3 cycles, then high for 3 cycles,
          etc..
7     while (1)
8     {
9       // read new data if any valid data
10      if (valid.read())
11      {
12        test_data = data.read();
13        test_data_array[array_index] = test_data;
14        array_index++;
```

```
15        cout << "Slave received data: " << test_data << endl;
16
17        // Print the first 10 numbers recieved by the slave to a text
              file
18        if (test_data_array[9] != 0)
19        {
20          ofstream myfile("Slave_data.txt");
21          if (myfile.is_open())
22          {
23            for (int i = 0; i <= 9; i++)
24            {
25              myfile << test_data_array[i] << "\n";
26            }
27
28            myfile.close();
29          }
30          else cout << "Unable to open file";
31        }
32      }
33
34      // Make sure that slave is ready for 3 cycles, then not ready
            for 3 cycles.
35      if (state_counter < 3)
36      {
37        ready.write(true);
38      }
39      else
40      {
41        ready.write(false);
42      }
43      state_counter++;
44      state_counter = state_counter % 6;
45
46      wait(clk.posedge_event());
47    }
48  }
```

A trace file has been made, to ensure that the program is cycle accurate. A figure of this file can been seen below.



Figur 4.1: A screenshot of GTK viewer

According to the figure above we can confirm that the protocol is acting as it should.

Kapitel 5

EXERCISE 3.5

Implement a model that demonstrates a system design that transfer data at the TLM level refined to BCAM level. Use the sc_fifo to model communication at the TLM level and refine it to BCAM using adapters as inspiration study the example project SmartPitchDetector (InAdapter.h and OutAdapter.h). Here a master sends data to a slave using a sc_fifo and an adapter that converts to the bus cycle accurate interface on the receiving slave. Use the model from exercise 3.4 for the interface at the Avalon-ST sink interface for the slave as illustrated below

Kapitel 6

RESULTS