**CSCE 5843: Reconfigurable Computing**

**Lab-02**

Topic:

Data Encryption Standard (DES) Hardware Acceleration Module with Block RAM, DDR DRAM and ARM core processor on ZCU104 FPGA Board.

Tools:

Vivado Design Suite 2022, Vitis IDE 2022, Tera Term

Introduction:

In this lab, we will learn how to establish communication between off-chip memory (DDR DRAM) and on-chip memory (BRAM) of the FPGA. First, the plaintexts will be stored in DRAM. Then they will be sent to the BRAM using direct memory access (DMA) module. After storing all plaintexts in the BRAM, the DES module will start reading from the same BRAM and generate ciphertexts. The ciphertexts will be stored in another BRAM after they are generated. After all the ciphertexts are stored, they will be sent to the DRAM using DMA.

Experimental Procedures:

**Creating Custom IP**

1. Follow the instructions from Lab-01 and Lab-02 to generate custom IP. For this lab, choose one axi stream slave interface and one axi stream master interface because they will be connected to the DMA as shown in Fig. 1.

Graphical user interface, application

Description automatically generated

Figure 1

1. The generated verilog files needs to be modified so that data coming through the slave interface are stored in BRAM. Then those data are read by the DES module to generate output, which will be stored in another BRAM. Then these data will be read and sent out through the master interface.
2. Two BRAM modules (one for inputs, another for outputs) are instantiated along with two BRAM controllers. You can use the BRAM module from IP catalog just like Lab-02 or write your own verilog code to infer BRAMs. The sample code for BRAM and BRAM controller module is shown in Fig 2 and 3. Change according to your need.
3. The BRAMs are simple dual port BRAMs (one port to write, another port to read). BRAM controller-01 is used to enable the BRAM-01, generate the address and read input data for the DES module from the address.

Text

Description automatically generated Text

Description automatically generated

Figure 2

1. BRAM controller-02 is used to enable the BRAM-02, increment the address to store the output data generated from the DES module and give a done signal when storing is finished. This done signal enables the master streaming interface for sending out the data to the DRAM through the DMA.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Figure 3

1. The bit width of the inputs and outputs are changed to 64 for DES. The number of words indicates how many data we want to store and read. Here it is set to 1024. There is an extra port from streaming interface called ‘tstrb’ which can be removed because it is not necessary for now. Some internal wires and registers are created for communications among different modules.

The key is constant and kept stored in a register all the time. Thus, it is not fetched from external storage, but you can do that too.

**Integration with Processor in Vivado Design Suite**

1. The block diagram for the overall design should look like Fig. 2 & 3. The address space should look like Fig. 1 in the address editor window.

Text

Description automatically generated with medium confidence

*Figure 1*

Diagram

Description automatically generated

*Diagram, schematic

Description automatically generatedFigure 2*

*Figure 3*

1. The configuration of the Zynq processor IP is shown in Fig. 4, 5 & 6. The DDR controller and UART are already connected, so, there is no need to add them externally. We just need to make sure they are selected. Since the DDR of ZCU104 is located on the processor system side, we need to use two slave interfaces on the Zynq processor to connect the DMA. These two interfaces must be “HP-FPD” because only this interface is connected to the DDR controller.

Graphical user interface, application

Description automatically generated

*Figure 4*

Graphical user interface, text, application, email

Description automatically generated

Figure 5

1. The DMA configuration is shown in Fig. 7. We will be using simple DMA transfer. So, disable scatter gather mode. Scatter gather mode is good for performance, but difficult to configure. The bit widths are set to 64 because we will be sending and receiving 64-bit data.
2. Follow the same steps as before to generate the bitstream.

Graphical user interface, application, table

Description automatically generated

Figure 6

Graphical user interface, application

Description automatically generated

Figure 7

**Running Software in Vitis IDE**

1. Follow the same instructions from Lab-01 & Lab-02.
2. First the processor saves input data into cache if cache is active. Thus, the data in the source address needs to be flushed to the main memory so that DMA can read from there. After DMA fetches back the data, cache of the destination address needs to be invalidated so that processor can read valid data for display.
3. C code and header files are attached. Change them in your own way if necessary.
4. Check the linker script as the following figure to see if memory mapped region is selected as DDR.

Graphical user interface

Description automatically generated

**Display Result on Terminal**

1. Follow the same process from Lab-01 & 02.
2. The C code on vitis gives both software and hardware results. Few results are displayed on the terminal as follows to verify the functionality of the design.
3. You may notice that the design’s output has two invalid data at the beginning if you run it for all the inputs. Because of this, it fails to produce last two outputs correctly because reading from output BRAM terminates by then. Thus, it produces 1022 outputs correctly for 1024 inputs. Try to figure out in the verilog code how you can read data for two more clock cycles from the output BRAM.

 