Prgetto di Reti Logiche

Yixin Liu (Codice Persona 10715474 - Matricola 940625)

August 5, 2022

1 Introduction

The project implemented a hardware module that interacts with a memory in VHDL. The module takes one byte at a time as input from the memory, then processes it in a way shown in figure 1 starting from the most significant bit of the input. After one byte is completely processed, it writes the 2-byte result on the memory. For example, let's say the input is 10100010, then the output should be 11010001 11001101. If there is any other input data, it continues to read and process them as described above.

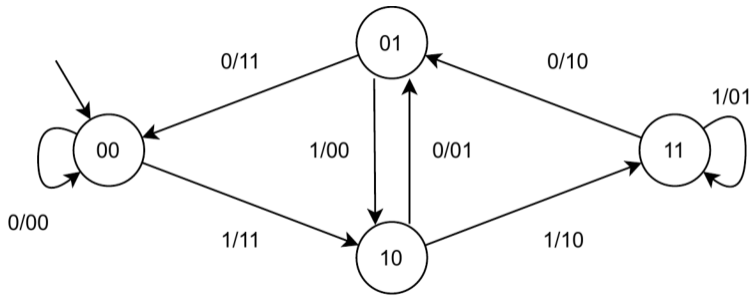

Figure 1: The finite state machine used to process inputs.

2 Architecture

2.1 Interface of component

Figure 2 shows how components interact with each other, in particular:

- i_clk is the CLOCK signal.
- i_rst is the RESET signal.
- i_start starts the machine when it equals to 1.
- i_data is the vector signal read from memory after a request of reading.
- o_address is the vector signal sends the intended reading/writing address to the memory. The address 0 contains the number of input bytes, the input bytes are restored starting from address 1 continuously. The results should be written on address 1000 and afterward without interruption.
- o_done is the signal that represents the end of elaboration and that all results have been written into memory.
- o_en is the ENABLE signal that enables the communication with memory.
- o_we is the WRITE ENABLE signal that has to be set to 1 when writing into memory.
- o_data is the data to be written into memory.

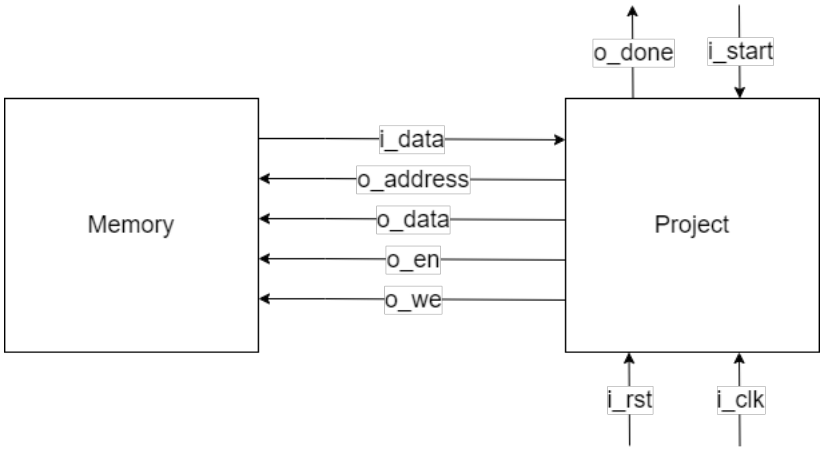

Figure 2: The interface between the memory and the implemented project.

2.2 Implementation

When i_start is set to 1, the machine starts elaboration from **Ready** state. Once the elaboration ended, it sets o_done to 1 and stays at **Done** state until i_start goes down to 0 which will take it back to **Ready** state. Furthermore, the i_rst signal will reset the machine to its default state.

2.2.1 States of the machine

- Ready: This is the initial state of the machine. It waits until i_start turns to 1, then it sets up output signals to read the number of words from address 0.
- Request_num_words: It waits for the response of memory.
- Get_num_words: It memorizes the information returned from memory into a register.
- Request_data: If num_words acquired in the Get_num_words stage is 0, it sets next state as **Done**. Otherwise, it manages to request 1 byte of data from memory.
- Wait_data: It waits for the response of memory.
- Pre_FSM: This is the last state before the machine enters the FSM described in Figure 1. It extracts the digit out of the byte and pass it as input to the FSM, sets up output mask. For example, if the machine is dealing with the most significant bit of the byte, then the output mask is 1100000000000000, if the machine is dealing with the least significant bit, the output mask is 0000000000000011. If all 8 bits of the byte have been processed, it goes to the state of write_output.
- A, B, C, D: These states perform in the same way as the FSM in Figure 1. They also set up output according to the output mask. In the end, they set up the machine to return to Pre_FSM state.
- Write_output: It writes the 1st part of the result into memory.
- Wait_writing: It writes the 2nd part of the result into memory. And head back to Request_data state.
- Done: This is the final state of the machine.

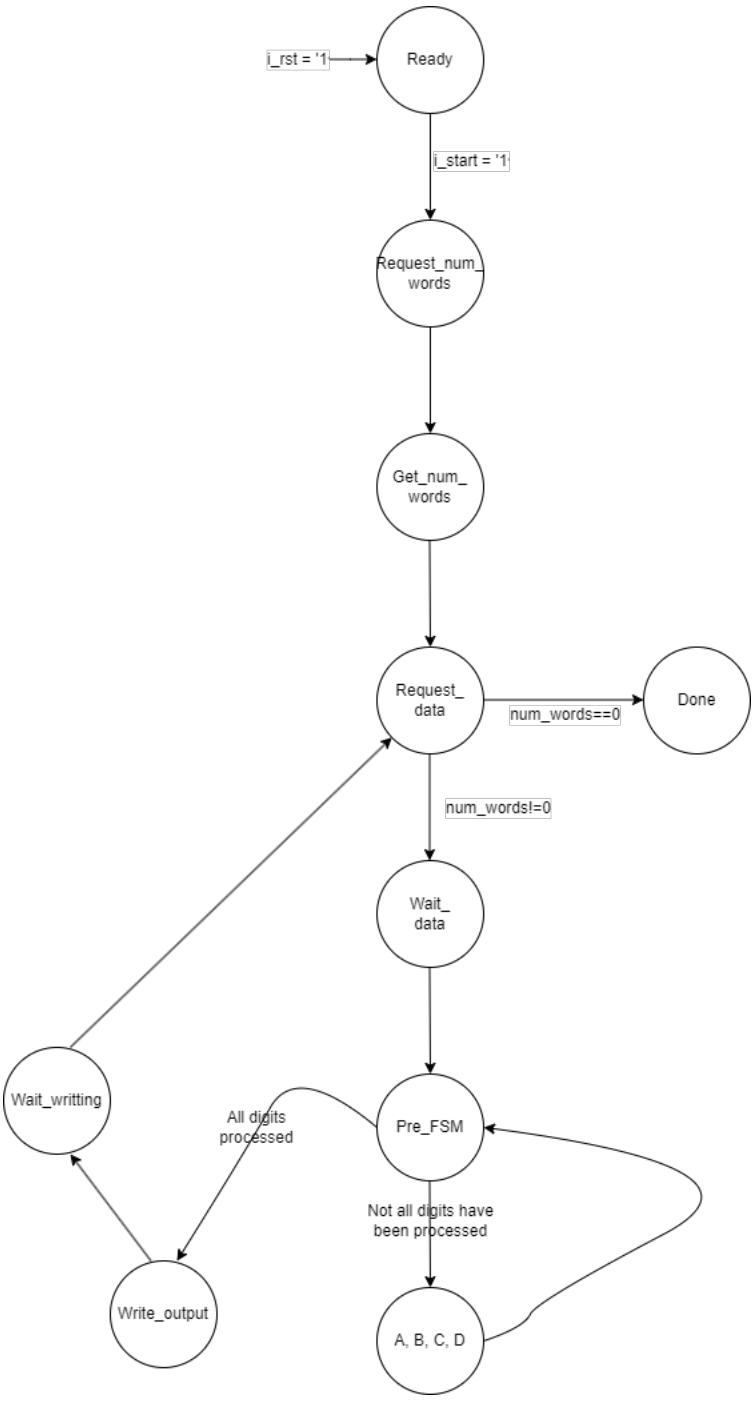

Figure 3: Simplified scheme of the machine

2.2.2 Processes

The implemented project has two processes.

1. The first process manages register transfer and asynchronous reset.
2. The second process represents the machine that analyzes input signals along with the current state, and determines where the system is going to evolve.

3 Experimental Results

3.1 Synthesis result

According to the synthesis report provided by Vivado, the design used 99 LUTs and 110 registers as flip flops, and 0 latches.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	96.161 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	49.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	183	Total Number of Endpoints:	183	Total Number of Endpoints:	111

All user specified timing constraints are met.

Figure 4: Asynchronous reset

3.2 Test results

To verify the correct behavior of the project, a test suite has been provided. The following behaviors have been tested in pre-synthesis and post-synthesis:

1. Given 0 byte as input, the machine did not write anything into memory.
2. Given 1 byte as input, the machine wrote the correct results into memory.
3. Given 2 bytes as input, the machine wrote the correct results into memory.
4. Given 2 bytes as input, the machine wrote the correct results into memory 2 times without resetting the machine.
5. Given 2 bytes as input, repeat the procedure 3 times, the machine wrote the correct results into 3 different memories.
6. Given 255 bytes as input, the machine wrote the correct results into memory.
7. Reset the machine 15 clock period after i_start turned 1, the machine wrote the correct results into memory.
8. Repeat test 6 three times, the machine wrote the correct results into 3 different memories.

Another test suite has been created to cover as many cases as possible:

1. Asynchronous reset: reset 10.5 clock period after i_start turned 1, the stage_reg changed to Ready the machine wrote the correct results into memory. Furthermore, the design timing summary is provided below.

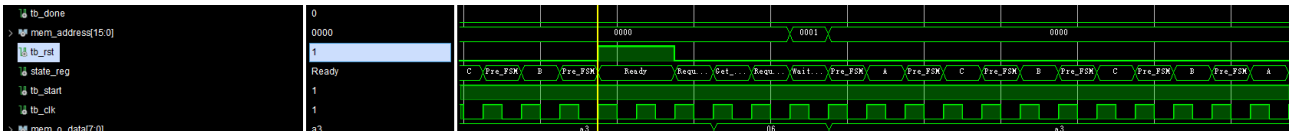

Figure 5: Design timing summary

2. Numerous tests with 1 - 255 bytes as input. The thought here is to verify the general correctness of the machine, all these tests passed positively.

4 Conclusion

The implemented project works as expected. It passed all provided tests and created tests on my part in behavioral simulation mode and post-synthesis simulation mode of Vivado. The biggest challenge when doing the project was writing VHDL code with more VHDL "smell". Writing the code in a non-typical VHDL way will result in additional meaningless LTUs and registers after the synthesis stage thus raising costs unnecessarily. Although after several versions of optimization, the use of LTUs and registers have been reduced a lot, I believe the project can still be optimized by refactoring the code with a more typical VHDL style.