Digital Systems Electronics Laboratory 05

Group 13

s295391 Giorgio Zoccatelli
s295567 Vittorio Macripò

Due Date:	April 29, 2024
Delivery date:	April 28, 2024
Instructor:	Professor Guido Masera

Politecnico di Torino
Accademic Year 2023/24

Contents

1 Introduction

The aim of this laboratory activity is to learn how to implement different FSM in VHDL. This task will require to learn different design techniques with respect to the previous lab activities.

2 One-Hot Finite state machine

2.1 Design Entry

The One Hot state machine, implemented in the file *one_hot_fsm.vhd* takes advantage of 9 flip-flops (forming a register) and of two different signal types:

1. **current_state** indicating the current state of the machine;
2. **future_state** indicating the future state of the machine.

These cases are encoded by a 9-bit binary sequence in which a single bit equal to '1' varies in order to distinguish one state from another.
The functioning of the machine consists in returning '1' as output ($LEDR_0$) following the detection of a sequence of at least 4 identical bits in input (SW_1) indistinctly whether they are 0s or 1s and overlapping is allowed.

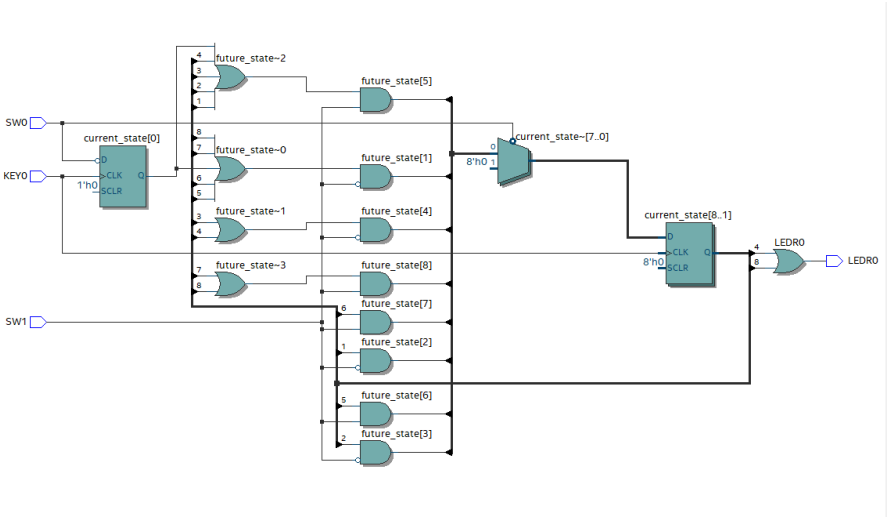


Figure 1: RTL analysis - Structure of One-Hot Fsm

2.2 Functional simulation

The description of the testbench is done in the file *one_hot_fsm_tb.vhd* using the clock signals *KEY0_tb*, reset *SW0_tb* and input *SW1_tb*.

2.3 Synthesis

In Figure 1, obtained through Modelsim, it can be observed that the results are consistent with the expectations. In fact, after receiving for 5 consecutive clock cycles an input value equal to '1', the output assumes the logical value '1' and then falls back to '0'.

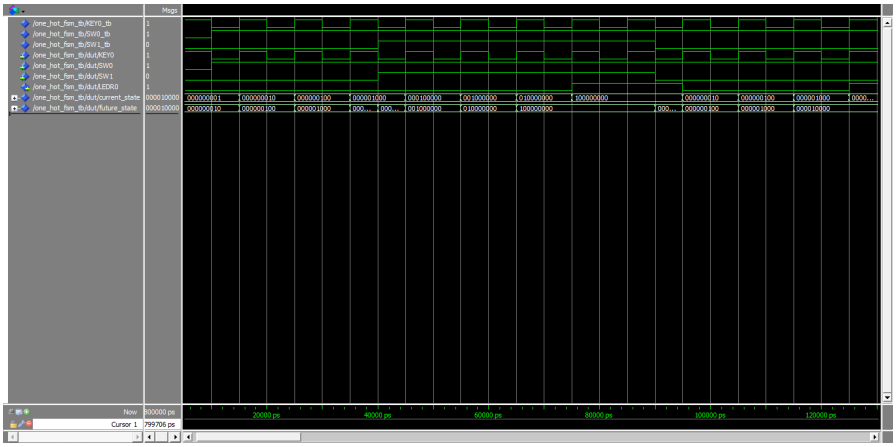


Figure 2: Waveform analysis - One-Hot Fsm

3 Modified One-Hot FSM

As seen in *version2.vhd* the working principle of the exercise is very similar to the previous one, the only substantial difference lies in the encoding of machine states that now have a '1' in the LSB, except for the machine reset state expressed as a 9-bit binary sequence of 0s.
As expected the results obtained by the testbench *version2_tb.vhd* on Modelsim are the same as the previous exercise, the only difference is the state code as described above.

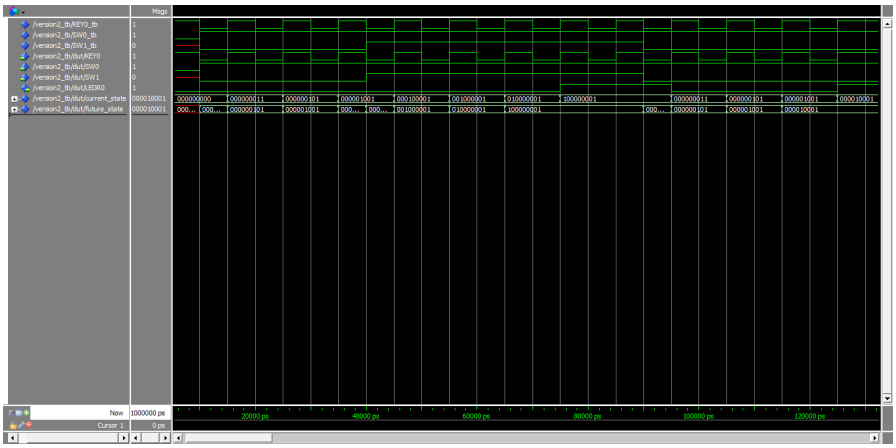


Figure 3: Waveform analysis - One-Hot Fsm version2

4 Two-process FSM

4.1 Design Entry

In the following exercise we will implement a FSM with the same task required as the previous exercises: the main difference is that we will use two difference processes in the same

FSM architecture description to implement both the one-hot codes given and the memory elements required. This modify will allow us to avoid creating a separate design for the flip-flops and the logic needed. A third process is added to set the output z that will drive $LEDR_0$. These steps are implemented in the design *fsm.vhd* while the top entity is described in the file *two_process_fsm.vhd*.

4.2 Functional Simulation

In the functional simulation we test our design by adding a testbench created to detect a correct sequence.

4.3 Synthesis

The first thing we can analyze is the wave shown in ModelSim which allow us to check if our design is working correctly as shown in Figure 4. Furthermore we can use two different tools on

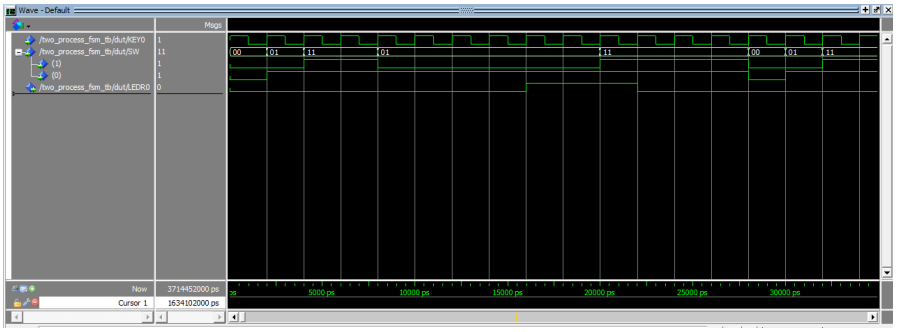
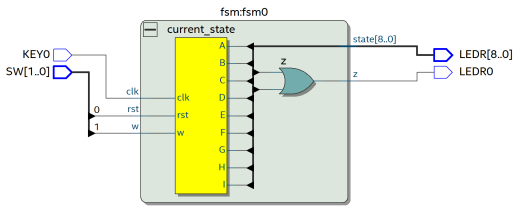
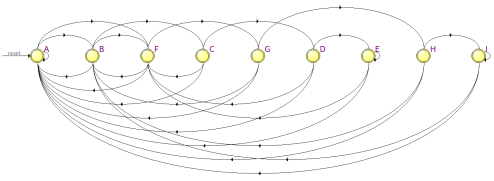


Figure 4: Waveform analysis - Two-process FSM

Quartus to analyze both the RTL Viewer and the State Machine Viewer to make a comparison with the tree diagram given in the exercise description.



(a) RTL Viewer



(b) State Machine Viewer

The last observation we can do is about the setting for the State Machine Processing. We selected both the One-Hot and the Minimal Bits: the difference is that in the first case the encoding will be exactly the same as the one given in the exercise description while in the second case the compiler will adopt another encoding using the least bits as possible. A quick comparison is given in the figures below.

	Name	I	H	G	F	E	D	C	B	A
1	A	0	0	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	0	1	1
3	C	0	0	0	0	0	0	1	0	1
4	D	0	0	0	0	0	1	0	0	1
5	E	0	0	0	0	1	0	0	0	1
6	F	0	0	0	1	0	0	0	0	1
7	G	0	0	1	0	0	0	0	0	1
8	H	0	1	0	0	0	0	0	0	1
9	I	1	0	0	0	0	0	0	0	1

(a) One-Hot encoding

	Name	state_bit_3	state_bit_2	state_bit_1	state_bit_0
1	A	0	0	0	0
2	B	0	0	0	1
3	C	0	0	1	1
4	D	0	1	0	0
5	E	0	1	0	1
6	F	0	0	1	0
7	G	0	1	1	0
8	H	0	1	1	1
9	I	1	0	0	0

(b) Minimal Bits encoding

5 ”HELLO” FSM

5.1 Design Entry

The aim of this last exercise is to implement a design through the concept of FSM able to display the world HELLO using *HEX* displays and to make it shift from right to left automatically with a blank space. Please notice that the restriction of scrolling the world every second was not implemented because it would've have been difficult to implement a correct testbench: in fact if we use a clock period of 20 ns by selecting *CLOCK_50* we will need to count until 50 million just to reach one single shift. We can now analyze each design.

- We need two counters both to count for the screen refreshing (*counter_1sec.vhd*) and to count for the detecting of the double L repetition (*counter_1to5.vhd*).
- We need the files *display_converter.vhd*, *mux_2to1.vhd* and *regn.vhd* for the correct displaying of the letters.
- We need a design to implement the FSM logic described in *fsm.vhd*.
- We need a memory element to drive both the FSM and the register (*flipflop.vhd*).
- We need a top entity *hello_fsm.vhd*.

5.2 Functional simulation

After the completion of the design we can implement a testbench (*hello _fsm_tb.vhd*) to test the correctness on ModelSim. In practice we will just set the reset function with *KEY0*.

5.3 Synthesis

We can start to check through the Wave analyzer available on ModelSim whether our design is correct (Figure 7).

In addition to that we can have a look at the RTL Viewer and State Machine Viewer to have more informations about the circuit design and about the FSM logic as can be seen in Figure 7 and Figure 8.

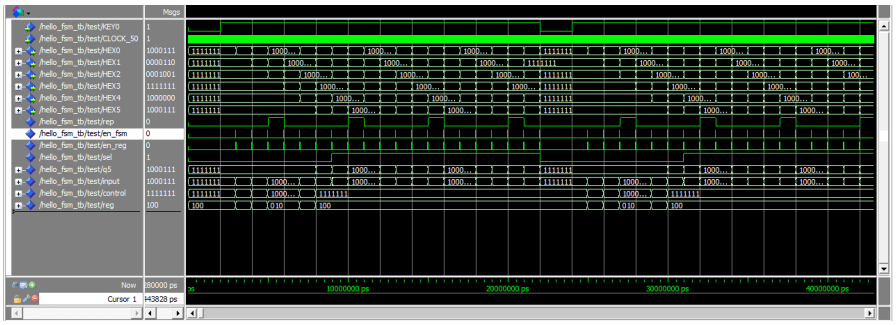


Figure 7: Waveform analysis - ”HELLO” FSM

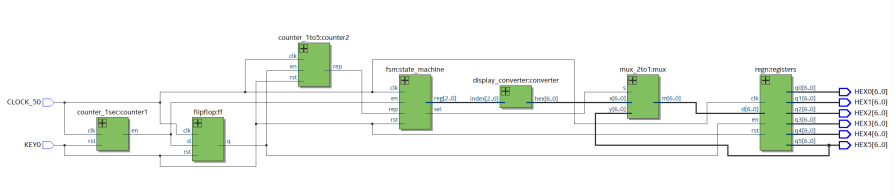


Figure 8: RTL Viewer

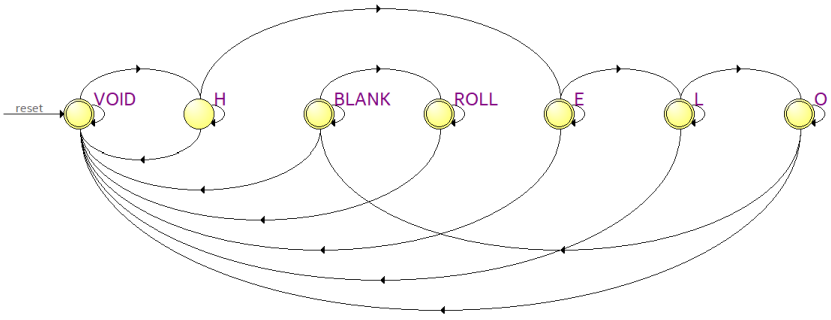


Figure 9: State Machine Viewer