Banner on Seven Segment Displays

With Varying Speed and Scrolling Messages

Arsha Ali, Drew Correll, Nina Luong, Bruce McCallister
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: arshaali@oakland.edu, drewcorrell@oakland.edu, nluong@oakland.edu, bamccallister@oakland.edu

***Abstract*—the purpose of this project is to design and implement finite state machines with a datapath circuit that will primarily interface with all eight seven segment displays on the Nexys 4 DDR board. With the use of switches, the banner on the display will move in a forward or backward direction, or display one of two scrolling messages. Switches will also be used in order to choose one of four speeds at which the banner flows. Furthermore, the user can active the stop switch to freeze the current pattern across the seven segment displays. The major findings are that the banner on the seven segment displays is accomplished with the use of several hardware components controlled by finite state machines, including four speed counters, multiplexers, 7-bit registers, 56-bit parallel access shift registers, and a serializer consisting of another finite state machine, a decoder, and a multiplexer. The conclusions clearly indicate that a datapath circuit controlled by finite state machines is a powerful method in the design of digital systems. Recommendations to the user include thorough testing of all switches controlling speed and the banner pattern or message selection.**

I. Introduction

This report will outline the methodology, experimental setup, results, and conclusions from the undertaking of this project.

The motivation to create banners and messages on the seven segment displays stems from the desire to showcase the visual side of hardware. With visual effects, the audience can clearly see the outcome and proper functioning of the hardware components. By setting switches, the user can cause a forward snake banner pattern, a backward snake banner pattern, the scrolling message “HELLO”, or the scrolling message “ECE2700” to appear on the seven segment displays. For each pattern, the user will be able to use another set of switches to control one of four different speeds that the banner moves at. A last switch is used to stop the banner pattern. The purpose of this project is achieved with the use of the software Xilinx Vivado written in VHDL.

Topics discussed in class that are present in this project include the functionalities of several hardware components and their implementation in VHDL, such as N-bit registers, N-bit parallel access shift registers, multiplexers, and finite state machines. These components, along with counters, a serializer, and basic logic gates, make up the datapath circuit. The datapath circuit is effectively controlled by a finite state machine for each pattern, as well as a fifth finite state machine for the serializer output.

A topic not directly covered in class, but vital to the project, included the implementation of the serializer. Another topic that is vital to the project was the implementation of counters with different maximum counts that correspond to time delays used as speeds for the banner. With the design of proper state diagrams and the entire circuit, the desired results are achieved with precision and accuracy.

The applications of the project include soothing visual displays and communication. With the use of switches, the user is able to select the speed of each of the four patterns/messages to their desire. The user can also stop any pattern to freeze it on the seven segment displays by activating the stop switch.

.

II. Methodology

An architectural structure of how all of the individual components should be laid out was based off of an example in Unit 7 from ECE 2700 class [1]. After structurally organizing each component, an architectural diagram was created in order to limit the amount of complications which could arise.

A. Counters

Four different counters were created to allow the user to choose one of four speeds that a pattern could progress at. The inputs to the counters include a 100MHz clock from the Nexys, enable, resetn, and sclr. The enable bit was set to ‘1’ and the sclr bit was set to ‘0’ for all four counters in the top file. The 100MHz clock is produced by the Nexys. The resetn bit will reset the counters, which is controlled by the CPU_RESET push button on the Nexys. The outputs of each counter include the count Q and a bit z. The number of bits of Q depends on the maximum count that the counter can obtain. The bit z is set to 1 when the maximum count has been reached.

Since the Nexys has a 100MHz clock, this equals a period of 10ns. Equation 1 shown below was used to calculate the maximum count for a desired speed. S represents the desired speed, T represents the period of the

100MHz clock, and N represents the value of the counter that is to be obtained. The units of S and T are in seconds.

Equation 1:

$$N = S/T = S/10 * 10^{-9}$$

The four counters created correspond to speeds of 1.5 seconds, 1.0 second, 0.5 seconds, and 0.25 seconds. The N value for each is $75 * 10^6$, $50 * 10^6$, $25 * 10^6$, and $125 * 10^5$ respectively.

The output Q is not relevant for the purpose of this project. The output z from each of the four counters is inputted to a multiplexer, where the speed that passes is selected by the speed switches set by the user. This signal is then used as the enable input for all four pattern finite state machines.

B. *Forward Pattern Finite State Machine*

The inputs to the forward pattern finite state machine include resetn, enable, and the 100MHz clock. The enable of this finite state machine is the output z bit chosen by the user from the counters. This finite state machine controls the enable, sclr, and data for eight 7-bit registers. The enable and sclr directly connect to each register. The output from the finite state machine is actually a 2-bit selector that controls the output of a multiplexer, which is then sent as the data input to all of the registers.

The finite state machine consists of 24 states. Three consecutive states will continue to output the same 8 bits for sclr. The 8 bits are broken down into eight 1-bit sclr, which is then sent to each of the eight registers. During these three consecutive states, the output of the multiplexer will change.

By setting the sclr bit and enable bit for all registers equal to 1 except for one register, the output of all registers is cleared while the one activated register cycles through three different data inputs. The 7-bit data inputs correspond to the LED pattern for each anode before moving to the next anode. The finite state machine progresses by checking the enable input. When the input is 1, the next state happens. Otherwise, the current state remains. When the last LED pattern has been reached on the last anode, the finite state machine returns to the beginning and repeats.

C. *Backward Pattern Finite State Machine*

The finite state machine for the backward pattern employs the same logic as that of the finite state machine for the forward pattern. The true value behind these finite state machines is the ability to clear the output of all registers except for one register by manipulating the enable and sclr bits of the registers.

The difference in the finite state machine for the backward pattern is the sclr output bits. Since the pattern is desired to start at the last anode and move to the first, the sclr bits are set backwards compared to the forward finite state machine. This allows the last register, corresponding to the last anode, the capture three different LED patterns before moving to the next anode to the left.

D. *LEDs Pattern Multiplexer*

There are three different LED patterns that will appear on each anode before moving to the next anode. The 7 bits for each of the three patterns are predetermined and are the inputs to the LEDs pattern multiplexer. The pattern of the LEDs that is outputted is controlled by the output value of the selector from the finite state machine. This 7-bit pattern for the LEDs are then sent to the data input for all eight 7-bit registers for the forward pattern.

A different multiplexer is used for the backwards pattern, with three different LED patterns. The output of this multiplexer is also sent to all eight of the 7-bit registers for the backwards pattern.

E. *7-bit Registers*

For each of the forward and backward patterns, the finite state machines control eight 7-bit registers each. The inputs are resetn, data, enable, sclr, and the 100MHz clock. Since the output of the finite state machines only let the sclr value of one register to be set to 0, that is the only register where the output is changing according to the LEDs pattern multiplexer.

With the passing of three states, the next register's sclr value will be set to 0, and the three LEDs pattern will then be outputted from that register.

The 7-bit output from all eight registers are concatenated together to form a string of 56 bits. These 56 bits from the forward registers and the backwards registers are the input to the pattern selector multiplexer.

Physically, each register corresponds to one anode of the seven segment displays. This method allows for one anode to be activated at a time and cycle through three different LED patterns before moving to the next anode.

F. *"HELLO" Finite State Machine*

The inputs to the finite state machine are enable from the output of the chosen counter, the 100MHz clock, and resetn. The outputs are s_l and enable for the parallel access shift register.

The state diagram consists only of two states. In state 1, the enable to the shift register and s_l are both set to 1. After, the diagram moves immediately to state 2, where it will remain for the remainder of time. In state 2, when the enable to the finite state machine is 0, the output enable and s_l to the shift register is implied. When the enable is 1, the enable to the shift register is set to 1 and s_l is set to 0.

G. *"ECE2700" Finite State Machine*

This finite state machine is identical to the “HELLO” finite state machine. The inputs, outputs, state transitions, and outputs are all the same. The outputs however go to a different 56-bit parallel access shift register.

H. 56-bit Parallel Access Shift Registers

The inputs are resetn, 56 bits of data, s_l, enable, din, and the 100MHz clock. The output is 56 bits for Q. The 56 bits of data correspond to the 7 bits for the LEDs for all eight anodes. The messages of “HELLO” and “ECE2700” are predetermined in this way, where the LED bits for anodes not in use are all set to 1.

During the first state in the finite state machine, the s_l bit is set to 1, which loads these 56 bits. Afterward, the 56 bits only shift to the right by setting s_l to 0. The shifting is done 7 bits at a time, so that the LED pattern on each anode is moved to the right. The 7 least significant bits are the value of din, which get set to the 7 most significant bits. These 56 bits are then sent to the pattern selector multiplexer. By doing this, the “HELLO” and “ECE2700” messages scroll across the seven segment displays. When the messages reach the last seven segment display, they wrap around back to the first seven segment display.

I. Pattern Selector Multiplexer

The 56 bits from each of the four pattern finite state machines are the input to this multiplexer. The output is what will eventually reach the seven segment displays. The select line of this multiplexer is controlled by switches chosen by the user on the Nexys. When the select is “00,” the forward pattern will be outputted. When the select is “01,” “10,”, or “11,” the backward pattern, “HELLO” message, or “ECE2700” message will be the output respectively.

J. Serializer

The serializer circuit consists of a multiplexer and decoder controlled by the serializer finite state machine. The output z from a 1ms counter is used as the input enable to the finite state machine. The finite state machine also has the input of the 100MHz clock and resetn. The only output of this finite state machine is three bits that are the select of the multiplexer and the input to the 3-to-8 decoder.

The finite state machine consists of eight states. With resetn set to 0, the state is state 1. With the output z from the 1ms counter being 1, the next state is one greater than the current state. When the counter output z is 0, the current state remains. In each state, the three bits that are outputted are increased by a value of 1. For example, in state 1 the output is “000,” in state 2 the output is “001,” in state 3 the output is “010,” and so on.

The 56 bits that were outputted from the pattern selector multiplexer are broken down into eight grouping of 7-bits. Each 7-bits corresponds to the LEDs on each anode. The output of this multiplexer is the 7-bit pattern of the LEDs that appear on the anode enabled by the decoder.

The decoder takes the 3 bit input and decodes it to activate one anode. The 8 bit output from the decoder are sent to a NOT gate before being mapped to the physical anodes.

By cycling through each anode and the corresponding pattern of LEDs for that anode, each of the four patterns work flawlessly.

K. Stop Switch Bit

The stop bit is controlled by a switch on the Nexys. When the switch is set to 1, the pattern across the seven segment displays freezes. This is done by sending the input of this switch to a NOT gate. The output of this NOT gate and the output from the speed multiplexer are sent to an AND gate. This result is then sent to the enable inputs of all four pattern finite state machines.

When the stop switch is set to 1, the result of the AND gate is a 0. Thus, the pattern finite state machines are not enabled. If this is the case, the outputs of the finite state machines remain in the current state they are in. Thus, the pattern across the seven segment display stops moving.

When the stop switch is not activated, the patterns will move across the seven segment displays as the finite state machines progress into different states.

L. VHDL Code

The figures at the end of the report show sample codes of the finite state machines. The code for each finite state machine outlines the *Transitions* process of moving from one state to the next, as well an *Outputs* process that will eventually set the LEDs for each anode across the seven segment displays.

The counters were coded in VHDL by using sample code from Unit 7 VHDL [2]. The number of bits for Q was created using the math_real.log2 and math_real.ceil libraries.

As seen from the Figure 1 showing the *Transitions* process of the forward finite state machine, when resetn equals 0, the finite state machine is in state 1. All other transitions happen only on the rising edge of the input clock and based on the enable input. To move to state 2 from state 1, the enable to the finite state machine must be 1, otherwise it will remain at the current state. The enable bit is used to progress each state up to state 24. After state 24, the process returns back to state 1 and repeats. A similar *Transitions* process is used to write the VHDL code for the backwards finite state machine.

As seen from Figure 2 showing the *Outputs* process of the forward finite state machine, the selector, sclr, and enable to the register bits are set based on the state and the value of enable to the finite state machine. The sample code shows how the bits for the selector change, while the sclr bits remain the same for three consecutive states.

As seen from Figure 3 showing the code for the "HELLO" and "ECE2700" finite state machines, the initial state is 1, and the state is 2 for the remainder of time. The *Outputs* process shows how the s_l bit and enable to the shift register is set when the enable to the finite state machine is 1. This allows for an initial loading of the data, and only shifting thereafter.

The VHDL code and components needed for the serializer were determined by analyzing a sample code [3]. This sample code helped design the correct components for the purposes of this project.

After the successful coding of each component, the top file interconnects these components using signals. The use of the port map and generic map functions are employed to do this. Port map is used to connect all the components together by creating the necessary signals. As seen by Figure 4, generic map is used to set the maximum count of the counters to create a specific speed.

III. Experimental Setup

The setup which was used in order to verify the functionality of the project was first to plan out as much step by step detail as possible of the program. This was done in order to limit the amount of errors which may occur. When programming, it was crucial that the state machine diagrams, the architecture of the project, and the logic was followed exactly as planned.

Each of the four designs which are displayed on the banner have their own finite state machine state diagram. An extensive state diagram was first created for each finite state machine to implement the pattern and flow of which LEDs would be illuminated on their corresponding anodes. Once the state diagrams were completed, an algorithmic state machine was designed, which would be implemented in each finite state machine.

The software Vivado 2017.4 was used during the creation of the program. This software allows the use of timing and functional simulations to verify the intended functioning of the project. The simulations were analyzed by creating a test bench of the top file. The test bench consisted of sample cases to the input bits, and the output parameters were determined from the simulation. This also allows to debug the code if the results are not as expected. When the code initially did not produce the expected results, the functional simulation was used to trace signals and correctly identify the source of error. By following the schematics as planned, it allowed for a successful program and code.

With some further design and implementation, the expected results include being able to use four different switches to set four different speeds for the banners. Also, another set of two switches will be used to choose one of four outputs to the seven segment displays: forward banner, backwards banner, "HELLO" message, or "ECE2700" message. The results are discussed in greater detail in the following section.

IV. Results

By using Vivado Functional and Timing Simulations, the results of the testbench were verified for accuracy. For example, when the selector is "00" the simulation will show one seven segment anode enabled, cycling through the three states dependent on the pattern of LEDs were chosen by the multiplexor controlled through the finite state machine. Similarly, the results for the backward pattern when the selector is "01" shows one seven segment anode enabled as the LEDs cycle through three states. For both of these cases, all bits of the LEDs on all the other anodes are set to 1, since they are of type common anode. A sample functional simulation showing the functioning of the forward pattern is included in Figure 5.

When the selector set to "10" for the scrolling "HELLO" message, the proper LEDs are enabled and the LED bits shift to the next anode and repeat. This is achieved by initially loading the data and only shifting 7 bits at a time thereafter. Similarly, when the selector is "11" for the scrolling "ECE2700" message, the LED bits shift to the next anode and then repeat. A sample functional simulation showing the shifting of 7 bits at a time for the "HELLO" message is included in Figure 6.

The same pattern will repeat for each of the eight 7 segment displays until the selector is manipulated to different bits.

When the stop switch is enabled, the pattern or message currently on the seven segment display freezes. When the stop bit is set back to 0, the pattern or messages continues from where it was frozen.

In addition, the switches used to control the speed propagation of the banner or scrolling messages is accurately achieved. With the speed switches set to "00," the pattern moves at a speed of 1.5 seconds. When the speed is set to "01," the pattern progresses at 1.0 seconds. For the cases of "10" and "11," the pattern moves at 0.5 seconds and 0.25 seconds respectively.

These results were achieved by correctly integrating various hardware components learnt in class. For the forward and backward snake patterns, the correct positioning of the LEDs on only a single anode was accomplished by enabling all the registers, and setting all but one registers synchronous clear value to 1. By doing this, the output of the registers corresponding to the anodes not currently in use were turned off by sending all bits of LEDs to 1.

For the "HELLO" and "ECE2700" scrolling messages, the use of a 56-bit parallel access shift registers were invaluable. At the beginning, the string of 56 bits corresponding to the message plus bits of 1 for empty anodes were loading to the shift register. After initial loading, the bits only shift 7 at a time. The 7 bits corresponding to the least significant bits are then sent as the 7 data input bits to become the next 7 most significant bits. By doing this, the messages are able to scroll or wrap around the seven segment displays.

After several rounds of debugging and slight modifications, the results were as expected. All results are explainable and accounted for by the methodology employed.

Conclusions

The main points that have been learned while doing this project is that finite state machines are powerful and useful hardware components used to control a datapath circuit. With the proper logic and connection of hardware components, desired results can be achieved and verified. It was also learned that a significant amount of detail and planning should go into the project before starting to code and implement the design. The code was much more easily written after the state diagrams and circuit were sketched. With the completion of this project, no issues remain to be resolved as the desired results were achieved. Further development with this project could include a more fluid snake pattern that appears to overlap between two anodes at a time. In addition, the RGB LEDs that are readily available on the Nexys 4 DDR board could be utilized to indicate the current set speed or pattern. Of course, more speeds and different banner patterns could also be created to add more variety to the project.

References

[1] Llamocca, Daniel. “Notes - Unit 7.” *Introductions to Digital Systems Design*, Jan. 2018, pp. 12–13., moodle.oakland.edu/pluginfile.php/4385644/mod_resource/content/5/Notes%20-%20Unit%207.pdf.

[2] Llamocca, Daniel. “Unit 7: Digital System Design.” *VHDL Coding for FPGAs,* slides 6-7., http://www.secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/Unit%207.pdf

[3] Llamocca, Daniel. “Serializer: Four 7-segment displays.” *VHDL Projects (VHDL files, testbench),* http://www.secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/ISE/Unit_7/serializer.vhd

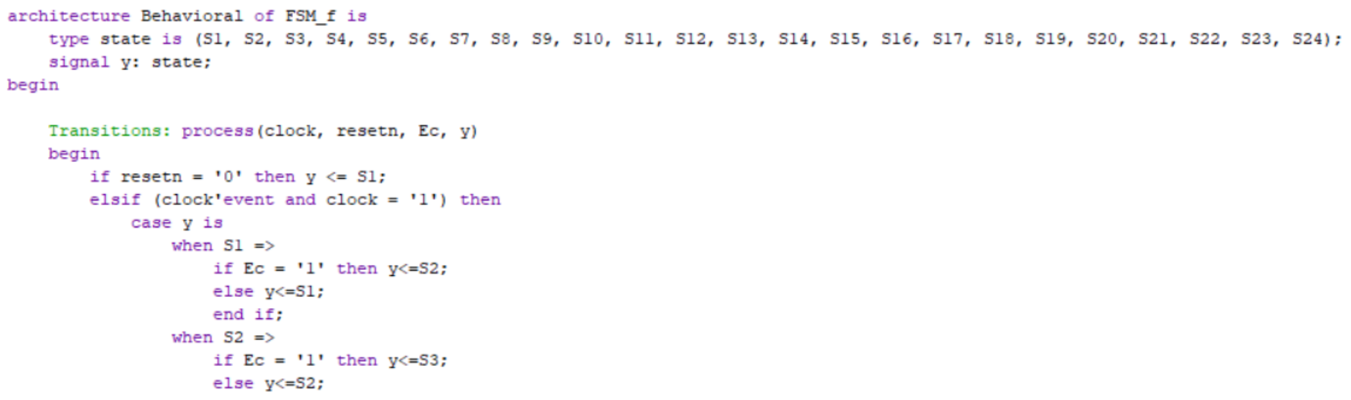
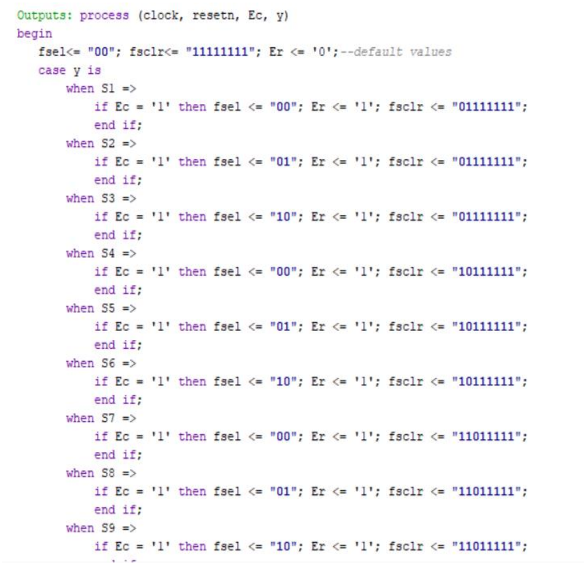
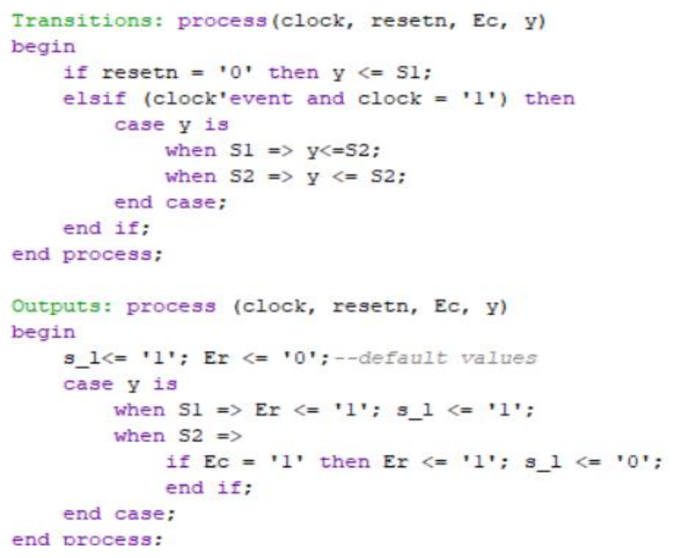

```
architecture Behavioral of FSM_f is
    type state is (S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S23, S24);
    signal y: state;
begin

    Transitions: process(clock, resetn, Ec, y)
    begin
        if resetn = '0' then y <= S1;
        elsif (clock'event and clock = '1') then
            case y is
                when S1 =>
                    if Ec = '1' then y<=S2;
                    else y<=S1;
                    end if;
                when S2 =>
                    if Ec = '1' then y<=S3;
                    else y<=S2;
```

Figure 1: Forward FSM *Transitions* Process.



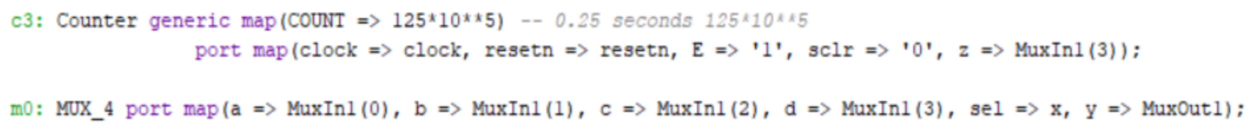
```
Outputs: process (clock, resetn, Ec, y)
begin
   fsel<= "00"; fsclr<= "11111111"; Er <= '0';--default values
   case y is
       when S1 =>
           if Ec = '1' then fsel <= "00"; Er <= '1'; fsclr <= "01111111";
           end if;
       when S2 =>
           if Ec = '1' then fsel <= "01"; Er <= '1'; fsclr <= "01111111";
           end if;
       when S3 =>
           if Ec = '1' then fsel <= "10"; Er <= '1'; fsclr <= "01111111";
           end if;
       when S4 =>
           if Ec = '1' then fsel <= "00"; Er <= '1'; fsclr <= "10111111";
           end if;
       when S5 =>
           if Ec = '1' then fsel <= "01"; Er <= '1'; fsclr <= "10111111";
           end if;
       when S6 =>
           if Ec = '1' then fsel <= "10"; Er <= '1'; fsclr <= "10111111";
           end if;
       when S7 =>
           if Ec = '1' then fsel <= "00"; Er <= '1'; fsclr <= "11011111";
           end if;
       when S8 =>
           if Ec = '1' then fsel <= "01"; Er <= '1'; fsclr <= "11011111";
           end if;
       when S9 =>
           if Ec = '1' then fsel <= "10"; Er <= '1'; fsclr <= "11011111";
```

Figure 2: Forward FSM *Outputs* Process.



```
Transitions: process(clock, resetn, Ec, y)
begin
    if resetn = '0' then y <= S1;
    elsif (clock'event and clock = '1') then
        case y is
            when S1 => y<=S2;
            when S2 => y <= S2;
        end case;
    end if;
end process;

Outputs: process (clock, resetn, Ec, y)
begin
    s_1<= '1'; Er <= '0';--default values
    case y is
        when S1 => Er <= '1'; s_1 <= '1';
        when S2 =>
            if Ec = '1' then Er <= '1'; s_1 <= '0';
            end if;
    end case;
end process;
```

Figure 3: "HELLO" FSM *Transitions* and *Outputs* Process.



```
c3: Counter generic map(COUNT => 125*10**5) -- 0.25 seconds 125*10**5
                port map(clock => clock, resetn => resetn, E => '1', sclr => '0', z => MuxIn1(3));

m0: MUX_4 port map(a => MuxIn1(0), b => MuxIn1(1), c => MuxIn1(2), d => MuxIn1(3), sel => x, y => MuxOut1);
```

Figure 4: Top File use of Port Map and Generic Map.

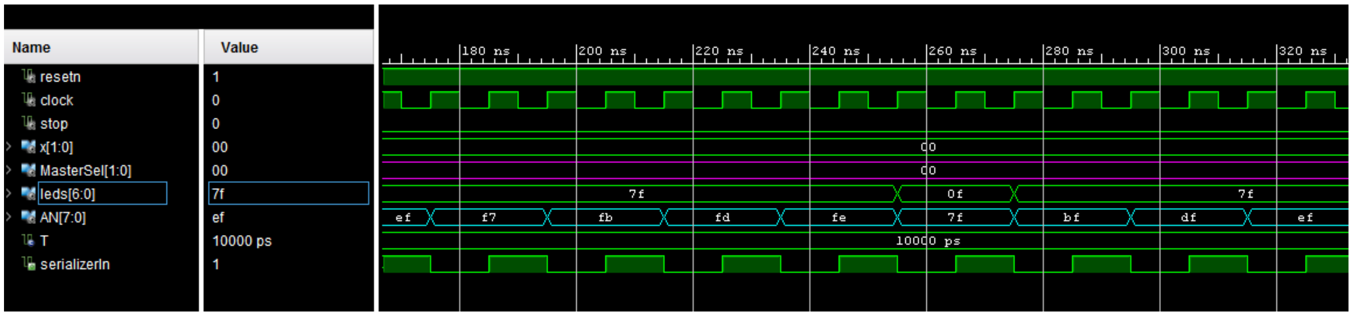


Figure 5: Forward Pattern Sample Functional Simulation.

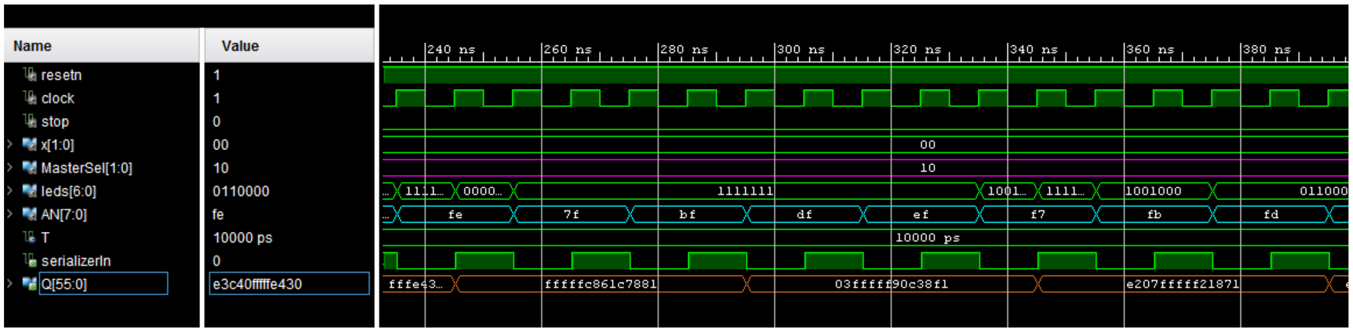


Figure 6: "HELLO" Message Sample Functional Simulation.