CE1911 LABORATORY PROJECT

SUMMARY

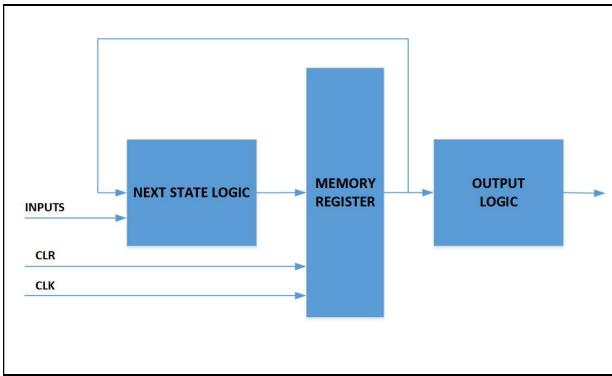
Sequential circuits use memory components to create computation behavior sequenced in time. Memory allows sequential circuits to remember a history of past inputs and use that history to help determine the next behavior. The history becomes a stored number called **state.** There are two types of sequential circuits: asynchronous sequential circuits and synchronous sequential circuits. **Finite state machines** are synchronous sequential circuits that use a clock signal to sample the next state into the storage memory. The next state number is a function of the current state number stored in memory as well as the input values.

$$NextState = F(CurrentState, Inputs)$$

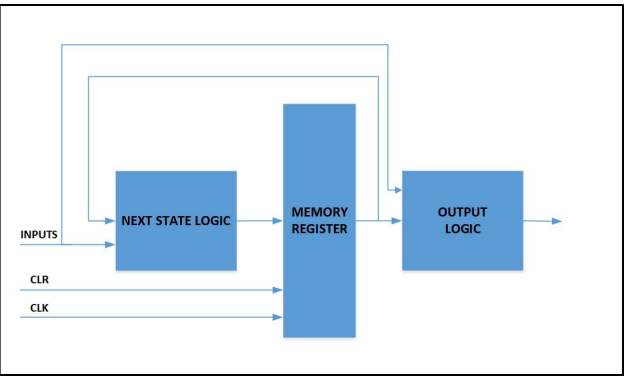
The outputs of a finite state machine are calculated using approaches from the mid-1950s pioneered by Edward Moore and George Mealy. A Moore Machine calculates outputs based only on the currently stored state. A Mealy Machine calculates outputs based on the stored state and the current inputs.

$$Outputs_{Moore} = F(Current\ State)$$

$$Outputs_{Mealy} = F(Current\ State, Inputs)$$



Moore Machine



Mealy Machine

Mealy Machine outputs can change value at any time if an input causes an output gate to change value. Moore Machine outputs can only change value at the clock edge when the state changes. For this reason, engineers consider Moore Machines safer and thus this type of machine is more common.

There are many standard state machines in widespread use in electrical and computer engineering. One of the most important state machines is the **counter**. Historically, counters have been part of circuity controlling traffic lights, digital clocks, appliance timing, automotive engine timing, and engine rotation tachometers to name just a few example systems. Today, many systems use software running on general-purpose computers and thus the use of stand-alone counters on circuit boards has diminished. However, engineers can still purchase stand-alone counters from the 7400 and CD4000 logic families. Examples include the CD4017, the 74161, and the 74191. In addition, counters are important components fabricated into the microarchitecture of microprocessors and microcontrollers. During this laboratory, you will design and implement a four-bit saturating up/down counter.

CE1911 LABORATORY PROJECT

PRELIMINARY LABORATORY READING

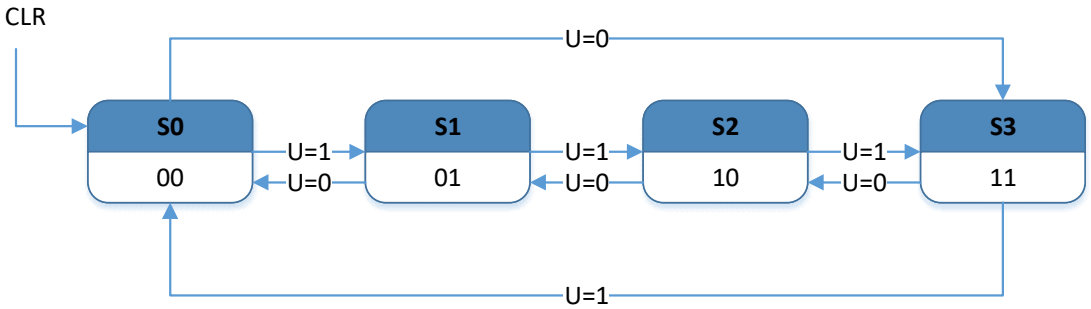
Counter state machines differ in their count patterns.

NAME	BEHAVIOR	COUNT PATTERN
n-bit binary	standard binary number line	0 → 2^n-1
modulo-n	remainders of n	0 → n-1
binary coded decimal	binary of the decimal numbers	0 → 9
Gray code	only one bit changes per count	0, 1, 3, 2
ring	powers-of-2	001, 010, 100

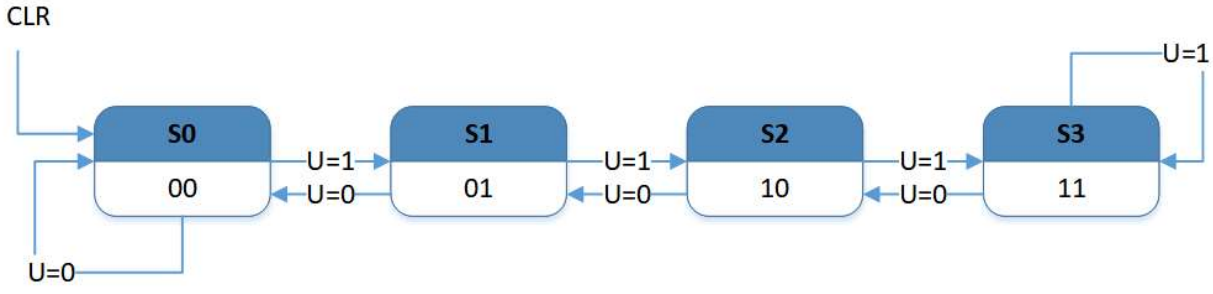
They also differ in their behavior at the end of their number line. **Standard counters** rolls back to the start of their number lines. **Saturating counters** lock at the end of the number line.

NAME	BEHAVIOR	COUNT PATTERN
2-bit standard binary	rolls over at the end of the number line	0 → 3, 0 → 3, …
2-bit saturating binary	locks at the end of the number line	0 → 3, 3, 3, …
standard modulo-5	rolls over at the end of the number line	0 → 4, 0 → 4, …
saturating modulo-5	locks at the end of the number line	0 → 4, 4, 4, …

Counters often have **direction control signals** that allow up-direction counting and down-direction counter.



2-bit standard up-down counter

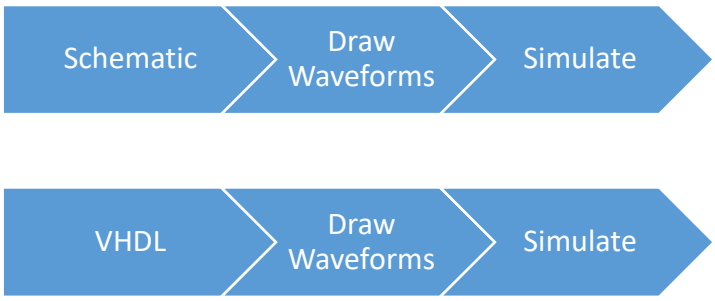


2-bit saturating up-down counter

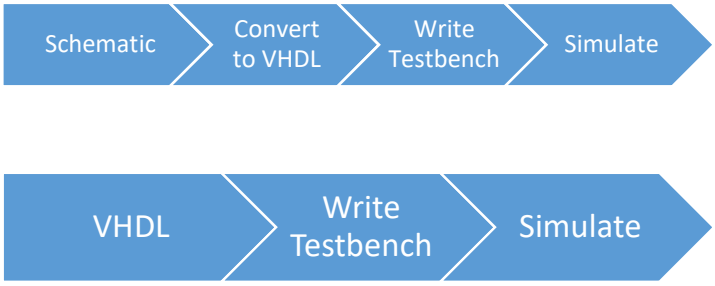
CE1911 LABORATORY PROJECT

NEW SIMULATION WORKFLOW

Simulation tools are an important part of modern engineering design workflow. These tools help to ensure that designs meet functional and timing requirements. Computer-aided design suites often include multiple ways to complete simulation. One technique uses input waveforms drawn by hand with a waveform editor. CE1901 used this approach. A second technique uses a hardware description language to describe the input voltage changes through time. Engineers call these hardware descriptions of test inputs **test benches**. CE1911 uses this approach.



CE1901 ModelSim-Altera Simulation Workflow



CE1911 ModelSim-Altera Simulation Workflow

ModelSim requires that engineers convert all schematic designs to textual hardware descriptions before simulation. The Quartus waveform editor completes this process automatically when it starts a simulation. Engineers must complete this step when using VHDL test benches.

CE1911 LABORATORY PROJECT

DELIVERABLES DUE AT THE END OF THE LABORATORY PERIOD

1. **Demonstrate** a 4-bit saturating up-down counter implemented as a Moore finite state machine and configured on the DE10-Lite. The **next-state logic block** and the **output logic block** are VHDL with-select dataflow architectures. The **top-level entity** is a schematic blueprint interconnecting the next state logic, memory register, and output logic.
2. **Submit** a laboratory report using the preferred method of your instructor. **Include** these sections:
    a. **Abstract**: no more than two pages describing the circuits and the work you have completed. This section should use text to convey to the reader the steps you took to complete the Moore Machine design and summarize your results.
    b. **Circuits**: a section that includes the Quartus schematic diagram, Quartus generated RTL diagram, and any text you wish to include that demonstrates your understanding of the work. This section could include truth tables, equations, etc. based on what you used in design.
    c. **Simulations**: a section that documents the method of testing for the circuit. In other words, given the inputs, how did you choose the number line you applied as voltages? Did it follow a truth table? Was it random? Did you set arbitrary values? Why did you choose your method? You must provide the ModelSim simulation waveforms with hand-written, electronic pen written, or typed text that demonstrates how you know the simulation verifies the circuit operates correctly.
    d. **Conclusion**: A short section that describes how the laboratory reinforced your learning of the course material. Include any questions you still have so that the instructor can provide additional help if needed.
    e. **Report submission deadline:** 5 p.m. one day after your laboratory period.

DESIGN AND SIMULATION INSTRUCTIONS

1. **Create** a Quartus VHDL project called **counter**. The project must include these files.

FILENAME	FORMAT	NEW OR REUSE	CIRCUIT
reg4.vhd	behavioral VHDL	new	4-bit register
nsl.vhd	with-select VHDL	new	next state logic
seg7.vhd	with-select VHDL	reuse from CE1901	output logic
counter.vhd	structural VHDL	new	4-bit saturating up-down counter Moore FSM

    a. **Use** Reference Figures 1-4 to guide your work.
    b. **Complete** Analysis and Synthesis on top-level design **counter** to ensure there are no errors.
    c. **Correct** any errors and **recompile** if needed.

2. **Add** a new VHDL file to the project. **Enter** the text in Figure 5 and **save** as **testbench_counter.vhd**.
3. **Rebuild** the design using **Start Analysis and Synthesis**.
4. **Configure** the simulation tool.

CE1911 LABORATORY PROJECT

Assignments → Settings → EDA Tool Settings → Simulation	
Tool Name	**ModelSim-Altera**
Format for Output Netlist	**VHDL**
NativeLink Settings	**Compile Test Bench**
Test Benches…	
New…	
Test Bench Name (must match test bench entity)	**testbench_counter**
Top Level Module in Test Bench	**testbench_counter**
End Simulation At (determined by VHDL time delays)	**5000ns**
Filename (browse, select, add)	**testbench_counter.vhd**
Tools → Options → EDA Tool Options	
Model-Sim Altera	**C:\intelFPGA_lite\18.0\modelsim_ase\win32aloem**

5. **Tools → Run Simulation Tool RTL Simulation**
   a. **ModelSim** will open and run the VHDL test bench to control the simulation. ModelSim will likely remain behind Quartus.
   b. **Verify** the circuit operates correctly. ModelSim will draw the waveforms in the output window. **Click** in the waveforms. **Use** keyboard keys **I** and **O** to zoom **in** and **out**.
   c. **Correct** any circuit errors by editing VHDL and re-simulating.

DE10-LITE CONFIGURATION INSTRUCTIONS

1. **Add** the VHDL **slowclk** component from Figure 6 to your project as a new VHDL file.
2. **Adjust** port maps in **counter.vhd** and include the slowclk component as shown in Figure 8. The connection to ground can be done in a port map as '0'.
3. **Do not** attempt to simulate. **Read** the VHDL comments in Figure 7.
4. **Complete** Analysis and Synthesis to rebuild the design. **Correct** any errors.
5. **Assign** pins for inputs and outputs using **Assignments → Pin Planner**.

SIGNAL NAME	I/O	DE10 PIN	DE10-LITE I/O DEVICE	REFERENCE IN DE10-LITE USER MANUAL
CLK	input	P11	clock circuit	Page 25
CLR	input	F15	slide switch 9	Page 27
LD	input	C11	slide switch 1	Page 27
U	input	C10	slide switch 0	Page 27
Q[3]	output	B10	led R3	Page 28
Q[2]	output	A10	led R2	Page 28
Q[1]	output	A9	led R1	Page 28
Q[0]	output	A8	led R0	Page 28
SEGMENT[7]	output	D15	hex0[7]	Page 29
SEGMENT[6]	output	C17	hex0[6]	Page 29
SEGMENT[5]	output	D17	hex0[5]	Page 29
SEGMENT[4]	output	E16	hex0[4]	Page 29
SEGMENT[3]	output	C16	hex0[3]	Page 29
SEGMENT[2]	output	C15	hex0[2]	Page 29
SEGMENT[1]	output	E15	hex0[1]	Page 29
SEGMENT[0]	output	C14	hex0[0]	Page 29

6. **Complete** Start Compilation.
7. **Configure** the DE10-Lite using **Tools → Programmer**.
8. **Test** your design. **Correct** any errors you find.

TEST PLAN FOR UNIT 4-BIT UP/DOWN SATURATING COUNTER						
STEP	**SIGNAL**	**SLIDE SWITCH**	**STARTING VALUE**	**TOGGLE TO**	**ENDING VALUE**	**VERIFY**
1	CLR	SW9	0	1	0	display 0
2	LD	SW1	0	1	1	display 0
3	U	SW0	0	1	1	up count
4	U	SW0	1	1	1	saturation at 15
5	U	SW0	1	0	0	down count
6	U	SW0	0	0	0	saturation at 0
7	U	SW0	0	1	1	up count
8	CLR	SW9	0	1	0	clear mid-count
9	LD	SW1	1	0	0	count stops
10	LD	SW1	0	1	1	count resumes

REFERENCE FIGURES

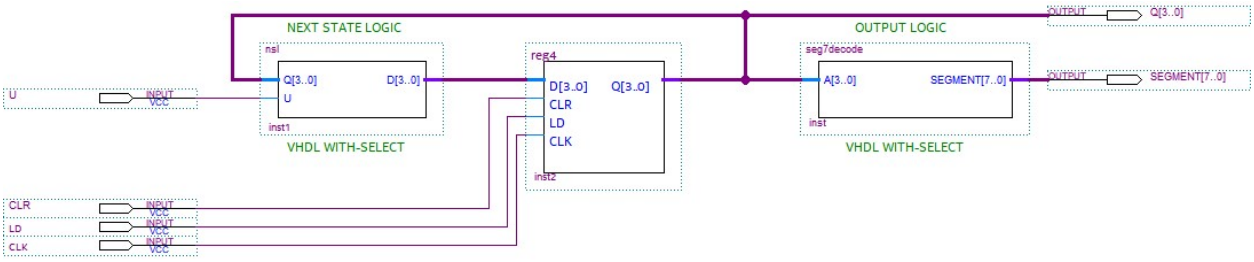


Figure 1: 4-bit Saturating Up-Down Counter Moore FSM

CE1911 LABORATORY PROJECT

INPUTS					DESIRED BEHAVIOR	OUTPUTS			
	CURRENT STATE					NEXT STATE			
U	Q3	Q2	Q1	Q0		D3	D2	D1	D0
0	0	0	0	0	count down				
0	0	0	0	1					
0	0	0	1	0					
0	0	0	1	1					
0	0	1	0	0					
0	0	1	0	1					
0	0	1	1	0					
0	0	1	1	1					
0	1	0	0	0					
0	1	0	0	1					
0	1	0	1	0					
0	1	0	1	1					
0	1	1	0	0					
0	1	1	0	1					
0	1	1	1	0					
0	1	1	1	1					
1	0	0	0	0	count up				
1	0	0	0	1					
1	0	0	1	0					
1	0	0	1	1					
1	0	1	0	0					
1	0	1	0	1					
1	0	1	1	0					
1	0	1	1	1					
1	1	0	0	0					
1	1	0	0	1					
1	1	0	1	0					
1	1	0	1	1					
1	1	1	0	0					
1	1	1	0	1					
1	1	1	1	0					
1	1	1	1	1					

Figure 2: Complete this truth table to guide your **next state logic** design

CE1911 LABORATORY PROJECT

INPUTS				OUTPUTS							
Q3	Q2	Q1	Q0	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0
0	0	0	0								
0	0	0	1								
0	0	1	0								
0	0	1	1								
0	1	0	0								
0	1	0	1								
0	1	1	0								
0	1	1	1								
1	0	0	0								
1	0	0	1								
1	0	1	0								
1	0	1	1								
1	1	0	0								
1	1	0	1								
1	1	1	0								
1	1	1	1								
0	0	0	0								
0	0	0	1								
0	0	1	0								
0	0	1	1								
0	1	0	0								
0	1	0	1								
0	1	1	0								
0	1	1	1								
1	0	0	0								
1	0	0	1								
1	0	1	0								
1	0	1	1								
1	1	0	0								
1	1	0	1								
1	1	1	0								
1	1	1	1								

Figure 3: Complete this truth table to guide your **output logic** design

Design the output logic to write hexadecimal characters onto a seven-segment display.

```
library ieee;
use ieee.std_logic_1164.all;

entity reg4 is
port(CLK, CLR, LD: in std_logic;
     D: in std_logic_vector(3 downto 0);
     Q: out std_logic_vector(3 downto 0));
end entity reg4;

architecture behavioral of reg4 is
begin

  update: process(CLK, CLR, LD)
  begin
    if rising_edge(CLK) then
      if CLR = '1' then Q <= X"0";
      elsif LD = '1' then Q <= D;
      end if;
    end if;
  end process update;

end architecture behavioral;
```

Figure 4: VHDL File reg4.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity testbench_counter is
end entity testbench_counter;

architecture dataflow of testbench_counter is

  signal CLK, U, CLR, LD: std_logic;            -- control signals
  signal Q: std_logic_vector(3 downto 0);       -- memory output
  signal SEGMENT: std_logic_vector(7 downto 0); -- data inputs

begin

  -- place the unit under test
  UUT: entity work.counter port map(U=>U,CLK=>CLK,LD=>LD,CLR=>CLR,
                                    Q=>Q,SEGMENT=>SEGMENT);

  -- write the clock process to generate the clock square wave
  clock: process
  begin
    CLK <= '0'; wait for 50ns;
     CLK <= '1'; wait for 50ns;
  end process clock;

  -- write the systematic process of testing: time delayed voltage changes
  test: process
  begin
     CLR <= '1', '0' after 200ns;   -- keep CLR active two clock periods
     U <= '1', '0' after 2200ns;    -- count up for 20 clocks + 2 clear
     LD <= '1';                     -- always sample D
     wait;
  end process test;

end architecture dataflow;
```

Figure 5: VHDL Test Bench

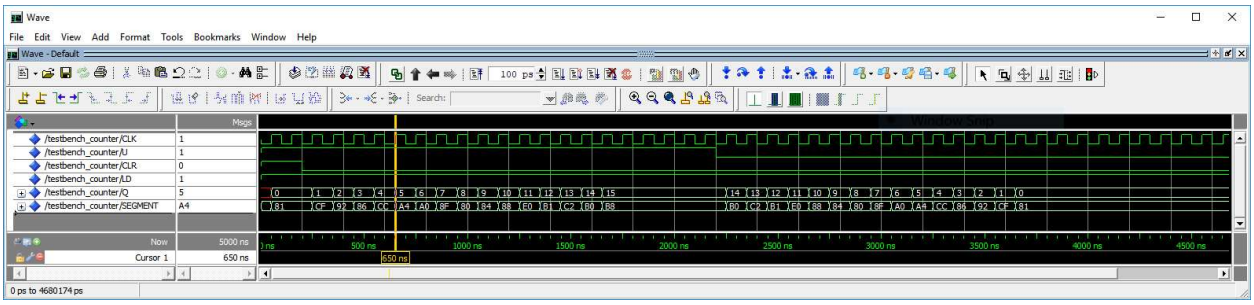


Figure 6: ModelSim-Altera simulation results

- The test bench creates a periodic clock waveform with a 100ns period.
- The test bench asserts the CLR control signal for two clock periods.
- The test bench asserts the U control signal for 2200 ns and de-asserts it for the rest of the simulation.
- The test bench asserts the LD control signal for the whole simulation; the register samples D each clock.
- The stored state Q shows the memory counting from 0 to 15 and saturating.
- The stored state Q shows the memory counting from 15 to 0 and saturating.

```
-- ************************************************************
-- * FILENAME: slowclock.vhd                                  *
-- * PROVIDES:                                                *
-- * - High frequency clocks prevent the use of LEDs to see   *
-- *   output signals changing because human eyes only resolve *
-- *   blinking light between 0 and approximately 30Hz.       *
-- * - This file creates a slow 1 second clock by recognizing  *
-- *   that the 50MHz clock is pulsing 50E6 times per second.  *
-- *   A one second clock would pulse 1 time per second.      *
-- *   Thus, the one second pulse is high for 25E6 fast clocks *
-- *   and low for 25E6 fast clocks.                          *
-- ************************************************************
-- * TO USE:                                                  *
-- * - Add this component into the clock path of an FSM under  *
-- *   test. Connect the 50MHz clock to the input named CLK50  *
-- *   and connect output CLK1 to the machine clock input.    *
-- * - In Quartus, use structural VHDL to complete this port   *
-- *   mapping or use a schematic blueprint that drops both   *
-- *   this component and the machine component in as the top  *
-- *   level entity.                                          *
-- * - DO NOT ATTEMPT TO SIMULATE when this module is in place *
-- *   because it will take 50 million clock periods to get   *
-- *   one clock period on your machine. Your computer will   *
-- *   take forever to complete a simulation and consume much  *
-- *   hard drive space. Only simulate your machine before you *
-- *   add this component to the clock path.                  *
-- ************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity SLOWCLK is
port( RST, CLK50MHZ: in std_logic;
      CLK1HZ:  inout std_logic);
end entity SLOWCLK;

architecture BEHAVIORAL of SLOWCLK is
  signal COUNT: integer;
  -- 25E6 times high and 25E6 times low
  constant HALF: integer := 25000000;

begin
  UPDATE: process(RST,CLK50MHZ)
  begin
    if RST = '1' then COUNT <= 0; CLK1HZ <= '0';
    elsif rising_edge(CLK50MHZ) then COUNT <= COUNT + 1;
      if COUNT = HALF then CLK1HZ <= not CLK1HZ; COUNT <= 0;
      end if;
    end if;
  end process;

end architecture BEHAVIORAL;
```

Figure 7: A VHDL slow clock component

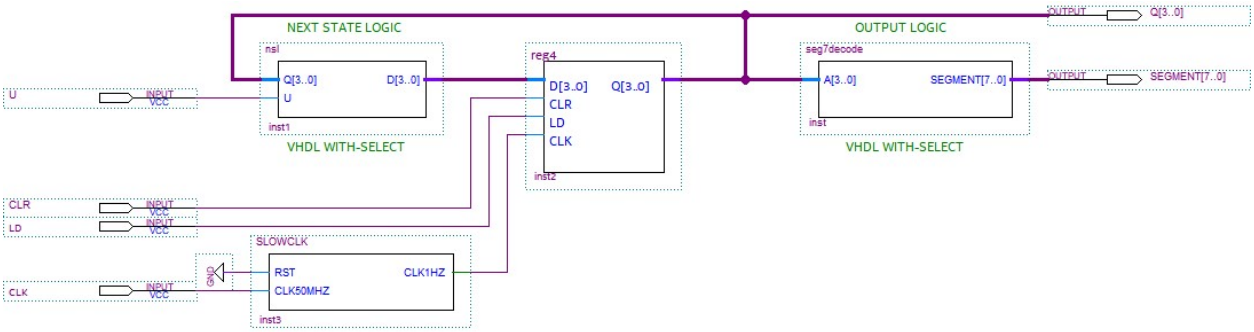


Figure 8: The counter FSM using a slower clock for human interaction