Date: October 26, 2024
To: Dr. Lei Huang
From: Arye Mindell
Subject: Sequential Logic Design

The objective of this experiment is to design and implement a three-bit synchronous up/down prime number counter with active-high enable. [1]

The counter cycles continuously at a rate specified by an external clock signal, with the count direction determined by a single input bit X_1, and the device enable controlled by a second input X_0. Outputs Y_0, Y_1, and Y_2 represent the 3-bit count. The counter's state is stored in two JK Flip-Flops, Q_1 and Q_0. [1]

The design implements a Moore model sequential circuit, as shown in the block diagram in Figure 1.

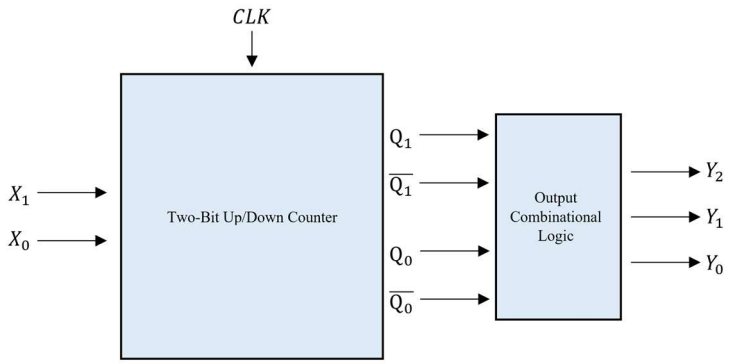

Figure 1: Prime-Counter Block Diagram

The circuit is separated into two modules: The first module, a two-bit up/down counter, manages state transitions based on the clock and input signals. The second module translates the current state into a three-bit binary representation of the corresponding prime number.

Figure 2 shows the specified state transitions and corresponding outputs. Each state represents a unique prime, as indicated by the three output bits.

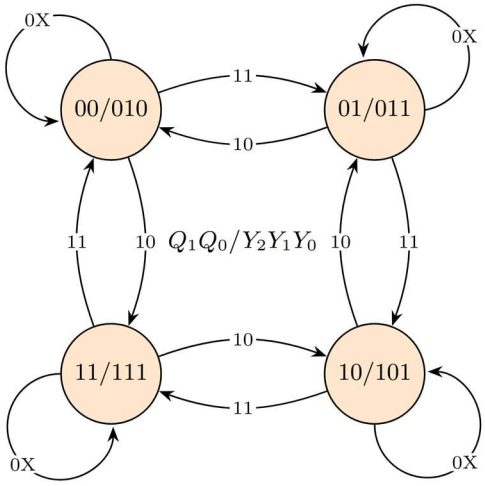

Figure 2: Prime-Counter State Diagram

Table 1(a) displays the output and next state as a function of the current state and inputs. Since the circuit employs a Moore model, the outputs are independent of the inputs, depending only on the current state.

Table 1: Prime Counter Logic Tables. (a) State Transitions and Outputs. (b) JK Flip-flop Excitation Behavior.

		X_1X_0				
State	Q_1Q_0	00	01	11	10	$Y_2Y_1Y_0$
2	00	00	11	01	00	010
3	01	01	00	10	01	011
7	11	11	10	00	11	101
5	10	10	01	11	10	111

(a)

Q	Q'	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

(b)

The circuit's state is stored in two JK flip-flops, with excitation behavior outlined in Table 1(b). Applying this behavior with the state transitions in Table 1(a) derives the K-maps in Figure 3.

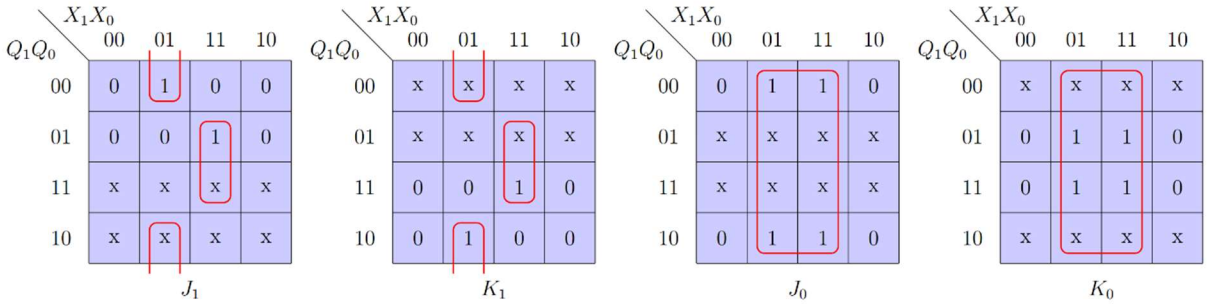

Figure 3: Flip-flop Excitation K-maps

Minimizing the K-maps in Figure 3 yields (1) and (2) which represent the excitation logic for each flip-flop input.

$$J_1 = K_1 = \left(\overline{Q_0}\,\overline{X_1}\,X_0\right) + (Q_0X_0X_1) \tag{1}$$

$$J_0 = K_0 = X_0 \tag{2}$$

Similarly, the output logic is determined from the K-maps of each output bit as a function of Q_1 and Q_0, as shown in Figure 4.

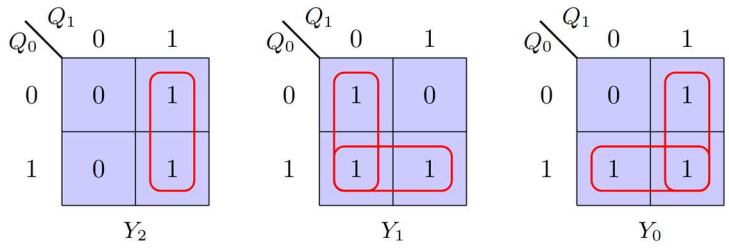

Figure 4: Output Logic K-maps

Minimizing the K-maps in Figure 4 yields (3) – (5) which describe the combinational logic of each output bit.

$$Y_2 = Q_0 \tag{3}$$

$$Y_1 = Q_0 + \overline{Q_1} \tag{4}$$

$$Y_0 = Q_0 + Q_1 \tag{5}$$

Figure 6 shows the simulation results which verify (1) – (5). The design and testbench code for this simulation are found in Attachment 1.

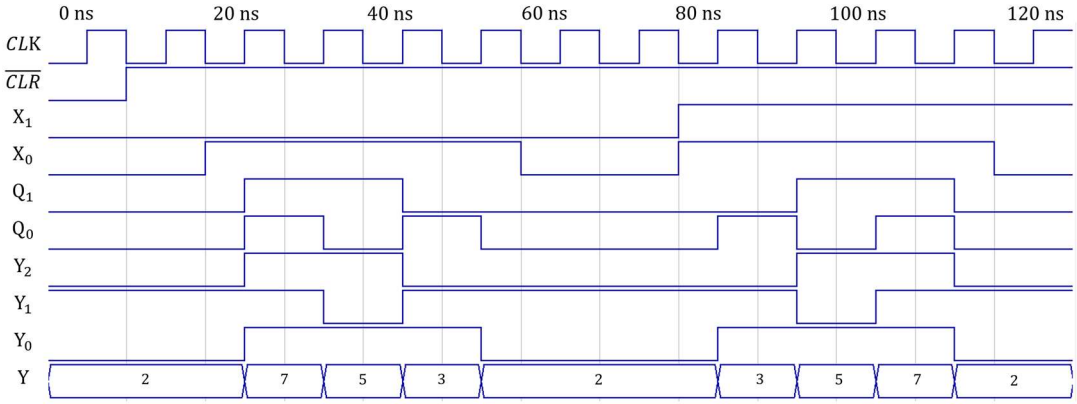

Figure 6: Prime-Counter Timing Diagram from Simulation

From 20 ns – 60 ns, X_0 is pulled high with X_1 low, which causes the output to decrement with each rising edge of the clock. From 60 ns – 80 ns both inputs are low, and the output remains unchanged. From 80-100ns, both inputs are pulled high, which causes the output to increment with each cycle. Finally, at 120 ns, X_0 is pulled low while X_1 remains high, disabling the count.

Shown in Figure 5, the final circuit design realizes the derived logic equations in modules as represented in Figure 1.

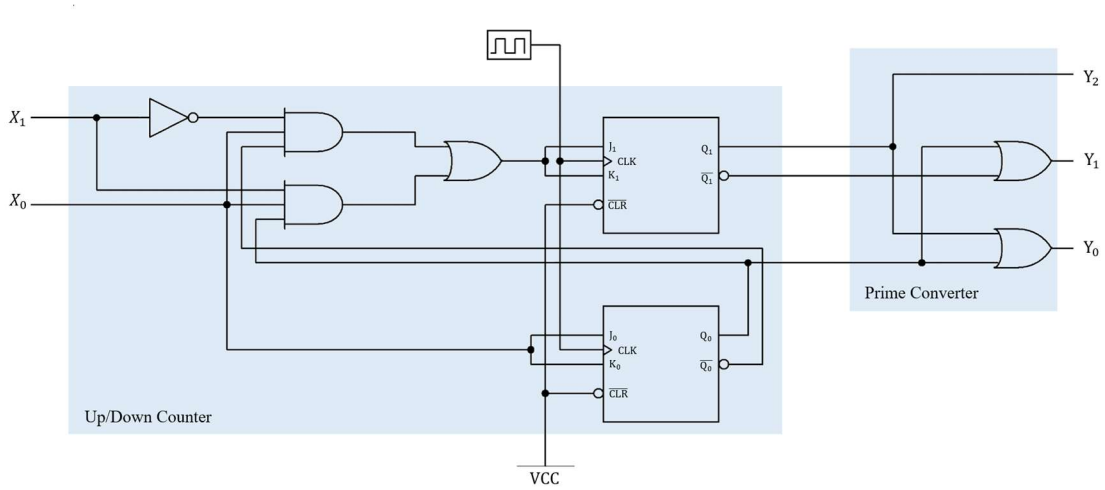

Figure 5: Prime-Counter Logic Diagram

The circuit is composed of 74LS32 OR gates, a 74LS04 Inverter, 74LS11 3-Input AND gates, and 7473 JK flip-flops. A built-in function generator inside the Tektronix 5-Series Mixed Signal Oscilloscope's drives the clock signal with a 1 Hz square wave. [1] The logic gates operate with a logic-high level of 5V, driven by a DC power supply. [2] The active-low CLR input of each flip-flop is pulled high.

Shown in Table 4, the procedure involves assembling and testing the modules sequentially. After each module is assembled, all cases in Table 1(a) are verified by pulling the inputs high or low as specified by the table and comparing the outputs indicated by LEDs to the expected output given by Figure 6.

Table 4: Experimental Procedure

Step	Action	Purpose	Materials	Expected Values
1	Assemble the Up/Down Counter module as pictured in Figure 5	Assemble Up/Down Counter	Logic Gates JK flip-flops Oscilloscope	
2	Connect LEDs at Q1 and Q0, verify proper state-cycling with each possible input combination	Test Up/Down Counter	LEDs Power Supply	See Figure 6, Q_0, Q_1
3	Assemble and connect Prime Converter module as pictured in Figure 5	Assemble Prime Converter	Logic Gates	
4	Connect LEDs at each output, verify output with each possible input combination	Test Design	LEDs Power Supply	See Figure 6, Y_2, Y_1, Y_0

Successful operation is confirmed if the LEDs display the expected sequence of prime numbers according to the input conditions, demonstrating accurate functionality of both the counter and prime conversion logic.

References:

[1] L. Huang, "Sequential Logic Design," *Memo to EECE 3100 Students*, Loyola Marymount University, Los Angeles, CA, Oct. 22, 2024. [Online]. Available: https://brightspace.lmu.edu/d2l/le/content/253458/viewContent/3109519/View.

[2] Texas Instruments, "SN74LS04 Hex Inverter," Datasheet, [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74ls04.pdf?ts=1729977523913

Attachments:

1. VHDL Simulation Test Code

PrimeCounter.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity PrimeCounter is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        X1 : in STD_LOGIC;  -- Up/Down control
        X0 : in STD_LOGIC;  -- Enable
        Y2, Y1, Y0 : out STD_LOGIC  -- Output 3-bit prime representation
    );
end PrimeCounter;

architecture Behavioral of PrimeCounter is
    signal Q1, Q0 : STD_LOGIC := '0';  -- Flip-flop states

    -- Define internal signals for JK flip-flop inputs
    signal J1, K1, J0, K0 : STD_LOGIC;
begin

    -- Excitation Logic
    J1 <= (not Q0 and not X1 and X0) or (Q0 and X0 and X1);
    K1 <= (not Q0 and not X1 and X0) or (Q0 and X0 and X1);
    J0 <= X0;
    K0 <= X0;

    -- JK Flip-Flop Process
    process (clk, reset, Q0, Q1)
    begin

        if reset = '0' then
            Q1 <= '0';
            Q0 <= '0';
        elsif rising_edge(clk) then
            -- Q1 JK flip-flop
            if J1 = '1' and K1 = '0' then
                Q1 <= '1';
            elsif J1 = '0' and K1 = '1' then
                Q1 <= '0';
            elsif J1 = '1' and K1 = '1' then
                Q1 <= not Q1;
            end if;

            -- Q0 JK flip-flop
            if J0 = '1' and K0 = '0' then
                Q0 <= '1';
            elsif J0 = '0' and K0 = '1' then
                Q0 <= '0';
            elsif J0 = '1' and K0 = '1' then
                Q0 <= not Q0;
            end if;
        end if;
    Y2 <= Q1;           -- Most Significant Bit
    Y1 <= not Q1 or Q0;     -- Middle Bit
    Y0 <= Q0 or Q1;           -- Least Significant Bit
    end process;
process(Q1, Q0)
begin

end process;
    -- Output Logic for 3-bit prime representation

end Behavioral;
```

PrimeCounter_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PrimeCounter_tb is
end PrimeCounter_tb;
architecture Behavioral of PrimeCounter_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component PrimeCounter
        Port (
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            X1 : in STD_LOGIC;
            X0 : in STD_LOGIC;
            Y2, Y1, Y0 : out STD_LOGIC
        );
    end component;
    -- Inputs
    signal clk : STD_LOGIC := '0';
    signal reset : STD_LOGIC := '0';
    signal X1 : STD_LOGIC := '0';  -- Up/Down control
    signal X0 : STD_LOGIC := '0';  -- Enable
    -- Outputs
    signal Y2, Y1, Y0 : STD_LOGIC;
    signal Y : STD_LOGIC_VECTOR(2 downto 0);  -- 3-bit vector for binary output
    -- Clock period
    constant clk_period : time := 10 ns;
begin
    -- Instantiate the Unit Under Test (UUT)
    uut: PrimeCounter
        Port map (
            clk => clk,
            reset => reset,
            X1 => X1,
            X0 => X0,
            Y2 => Y2,
            Y1 => Y1,
            Y0 => Y0
        );
    Y <= Y2 & Y1 & Y0;
    -- Clock process definitions
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;
    -- Testbench process to test different cases
stim_proc: process
begin
    -- Reset
    reset <= '0';
    wait for clk_period;
    reset <= '1';
    wait for clk_period;
    -- Enable and Count Up
    X0 <= '1'; -- Enable
    X1 <= '0'; -- Count up
    wait for 4*clk_period;
    -- Disable
    X0 <= '0'; -- Disable
    wait for 2*clk_period;
    -- Enable and Count Down
    X0 <= '1'; -- Enable
    X1 <= '1'; -- Count down
    wait for 4*clk_period;
    -- Disable
    X0 <= '0'; -- Disable
    wait for 2*clk_period;
    -- End Simulation
    wait;
end process;

end Behavioral;
```