Christopher Spadavecchia
Eli Shtindler
CPE 487

Lab 2

Project 1 - hex4count:

```
connect_hw_server: Time (s): cpu = 00:00:01 ; elapsed = 00:00:12 . Memory (MB): peak = 1151.664 ; gain = 16.656
open_hw_target
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292BCFBFBA
set_property PROGRAM.FILE {C:/Users/elism/Vivado Projects/hex4count/hex4count.runs/impl_1/hexcount.bit} [get_hw_devices xc7a100t_0]
current_hw_device [get_hw_devices xc7a100t_0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1435] Device xc7a100t (JTAG device index = 0) is not programmed (DONE status = 0).
create_hw_cfgmem -hw_device [get_hw_devices xc7a100t_0] -mem_dev [lindex [get_cfgmem_parts {s25fl128sxxxxxx0-spi-x1_x2_x4}] 0]
set_property PROBES.FILE {} [get_hw_devices xc7a100t_0]
set_property FULL_PROBES.FILE {} [get_hw_devices xc7a100t_0]
set_property PROGRAM.FILE {C:/Users/elism/Vivado Projects/hex4count/hex4count.runs/impl_1/hexcount.bit} [get_hw_devices xc7a100t_0]
program_hw_devices [get_hw_devices xc7a100t_0]
INFO: [Labtools 27-3164] End of startup status: HIGH
refresh_hw_device [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1434] Device xc7a100t (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
```

After uploading the code to the board was successful (the aftermath of the 'Program Device' step), the following lines were shown in the console.

Booting the program using QSPI

To accomplish this, we first relocated the blue jumper from JTAG to QSPI. Next, we performed three key steps in Vivado: generating the memory configuration file, adding the configuration memory device, and programming it. After restarting the board and waiting 10 seconds, the program successfully booted, and the four-digit hex count began.

Counter integrated with FSM

counter.vhd
Original:

```
ARCHITECTURE Behavioral OF counter IS
	SIGNAL cnt : STD_LOGIC_VECTOR (38 DOWNTO 0); -- 39-bit counter
BEGIN
	PROCESS (clk)
	BEGIN
		IF clk'EVENT AND clk = '1' THEN -- on rising edge of clock
			cnt <= cnt + 1; -- increment counter
		END IF;
	END PROCESS;
	count <= cnt (38 DOWNTO 23); -- 16 bits
	dig <= cnt (19 DOWNTO 17); -- 3 bits
END Behavioral;
```

Modified:

```
ARCHITECTURE Behavioral OF counter IS
    SIGNAL cnt : STD_LOGIC_VECTOR (38 DOWNTO 0); -- 39-bit counter

    component fsm is
        port (X, CLK, RESET: in std_logic;
        Y : out std_logic_vector(2 downto 0);
        Z : out std_logic);
    end component fsm;

    signal Y : STD_LOGIC_VECTOR(2 downto 0);
    signal Z : STD_LOGIC;
    signal dir : STD_LOGIC;

BEGIN
    fsm11100: entity work.fsm(fsmMealy11100) port map(X=>cnt(29), CLK=>cnt(23), RESET=>'0',
Y=>Y, Z=>Z);
    PROCESS(Z) -- swap the value of dir on rising edge of Z
    BEGIN
        IF (rising_edge(Z)) THEN
            dir <= not(dir);
        END IF;
    END PROCESS;

       PROCESS (clk)
       BEGIN
           IF clk'EVENT AND clk = '1' THEN -- on rising edge of clock
               IF dir = '0' THEN
                  cnt <= cnt + 1; -- increment counter
               ELSIF dir = '1' THEN
                  cnt <= cnt - 1; -- decrement counter
               END IF;
              END IF;
       END PROCESS;
    count <= cnt (38 DOWNTO 23); -- 16 bits
       dig <= cnt (19 DOWNTO 17); -- 3 bits
END Behavioral;
```

A new FSM component was created in counter.vhd based on the design from simulation assignment 3. Bit 29 of cnt was chosen as the input for the FSM because it changes infrequently enough to see the counter move in each direction for a decent amount of time. Bit 23 of cnt was selected as the clock signal because it is the first bit of the counter, so each time the counter visibly changes, the fsm has a state transition. A new process based on the output of the FSM, Z, was created. This process inverts the value of dir every time Z goes high. The dir signal was then used in the clk process to determine if the counter was going up or down.

designFSM.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
       port (X, CLK, RESET: in std_logic;
    Y : out std_logic_vector(2 downto 0);
    Z : out std_logic);
end fsm;

 architecture fsmMealy11100 of fsm is
       type state_type is (A, B, C, D, E);
    signal PS, NS : state_type := A;
    signal NZ : std_logic;
 begin
       clockAndReset: process(CLK, RESET)
     begin
       if (RESET = '1') then PS <= A; -- Set previous state to A when reset is 1
         elsif (rising_edge(CLK))
         then
             PS <= NS; -- Set previous state to next state on rising edge
             Z <= NZ;
         end if;
       end process clockAndReset;
 -- A: 1/0 -> B
 -- A: 0/0 -> A
 -- B: 1/0 -> C
 -- B: 0/0 -> A
 -- C: 1/0 -> D
 -- C: 0/0 -> A
 -- D: 1/0 -> D
 -- D: 0/0 -> E
 -- E: 1/0 -> B
 -- E: 0/1 -> A
     stateAndOutputLogic : process(PS, X)
     begin
     case PS is
       when A =>
              if (X = '1') then NZ<='0'; NS <= B;
            else NZ <= '0'; NS <= A;
            end if;
         when B =>
              if (X = '1') then NZ <= '0'; NS <= C;
            else NZ <= '0'; NS <= A;
            end if;
         when C =>
              if (X = '1') then NZ <= '0'; NS <= D;
            else NZ <= '0'; NS <= A;
            end if;
```

```
        when D =>
             if (X = '1') then NZ <= '0'; NS <= D;
            else NZ <= '0'; NS <= E;
            end if;
        when E =>
             if (X = '1') then NZ <= '0'; NS <= B;
            else NZ <= '1'; NS <= A;
            end if;
        end case;
    end process stateAndOutputLogic;

    with PS select
    Y <= "000" when A,
         "001" when B,
         "010" when C,
         "011" when D,
         "100" when E,
         "000" when others;
end fsmMealy11100;
```

A short answer on your thoughts as to why we have the "only flip between 0 and 1 one time" requirement and what might happen if we did not have that requirement.

Without the requirement to flip between 0 and 1 only once, some sequences may never be produced by the counter. This is because, in binary, certain patterns that switch between 0 and 1 multiple times do not follow a valid counting sequence. However, all binary sequences that flip between 0 and 1 only once will always follow a valid counting sequence. For example, the sequence 110101 does not appear in standard binary counting, whereas a sequence like 00011 (which flips only once) follows a valid counting pattern.