Hardware Assignment 3

CPE 233

Professor Gerfen

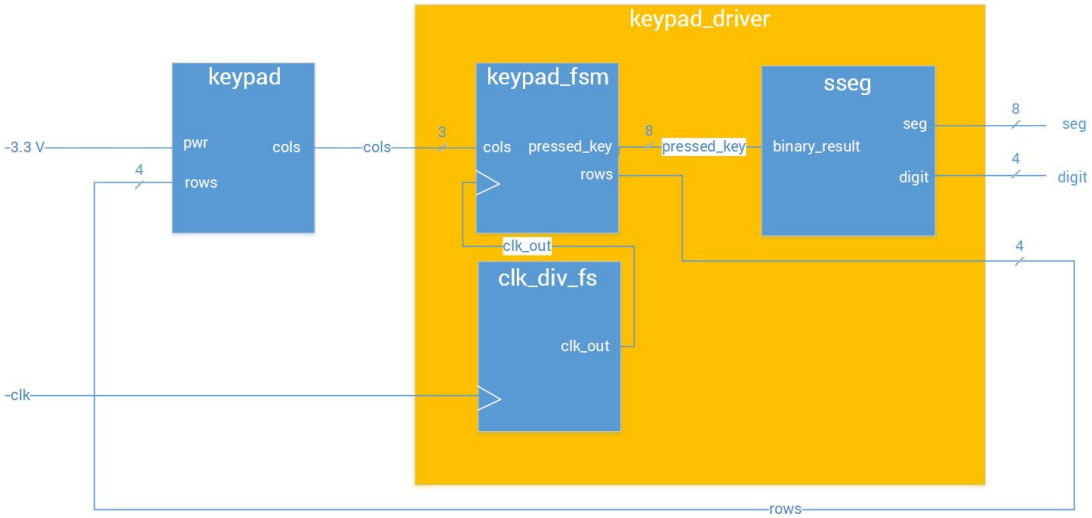
Russell Caletena, Josiah Pang, & Nathan Wang

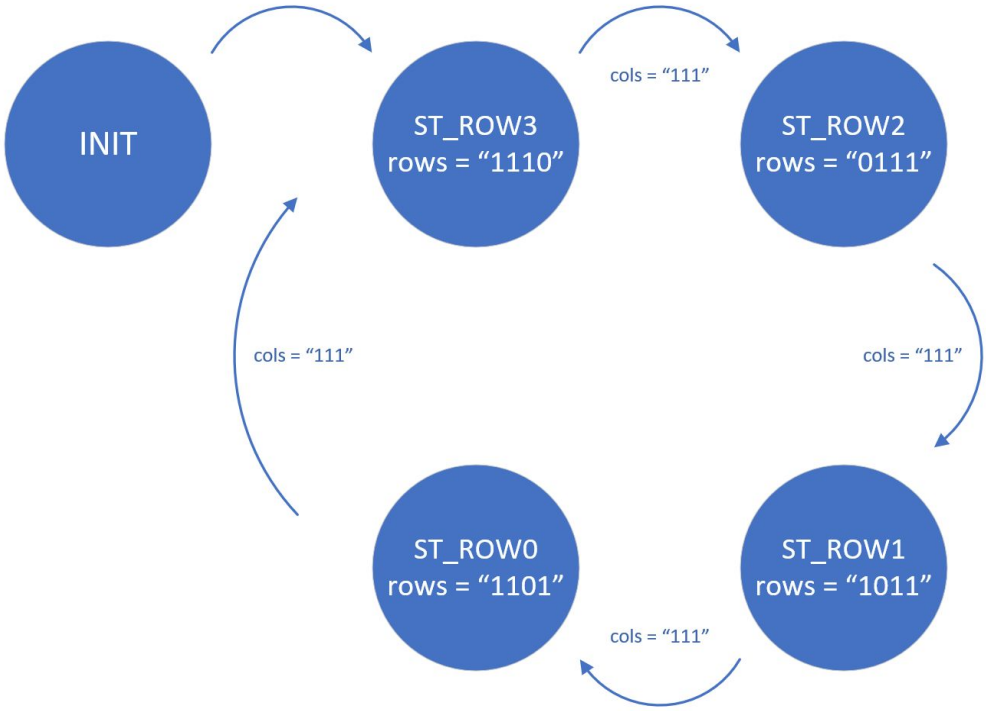
02/19/2018

YouTube Video Link

https://www.youtube.com/watch?v=-lZpaeYOX-8

Black Box Block Diagram

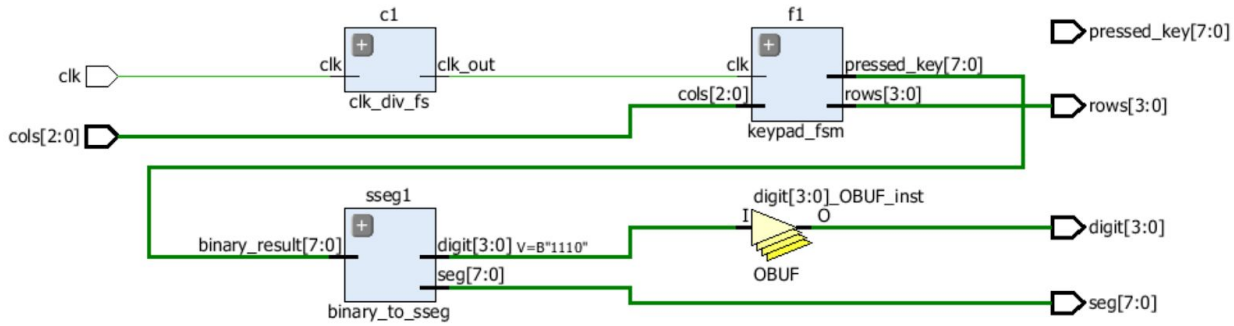




Behavior Description

This keypad driver has 12 buttons, 0-9, *, and #. Pressing each button displays its corresponding value on the seven segment display. The asterisk (*) displays a capital “A” while the pound (#) displays a lowercase “b.” Only one button can be pressed at a time. When the button is pressed, our FSM figures out exactly which button was pressed and outputs an 8-bit signal that corresponds to the seven segment input. As long as the button is pressed, the value will be output on the display.

Structural Design



Specification

Our keypad driver analyzes a single button press, and first translates that button press into a corresponding column value (active-low sequence). For instance, if the column input is recognized as “011”, that means that the button pressed is either values ‘1’, ‘4’, ‘7’, or ‘*’. If columns is “110”, then the button pressed is either ‘3’, ‘6’, ‘9’, or ‘#’. The driver runs to determine the keypad button press on the rising edge of a clock (synchronous). The column input is determined based on when one of the column lines has no voltage, since all three columns are connected to power (hence the columns being active-low). Once a column is recognized, the rows (output) are sequentially one at a time. At each row, the driver re-checks to see if one of the columns is triggered (meaning have a ‘0’ in its three-bit input). If columns results in “111”, the driver will scan the next row. Finally on a given row when the columns input has a ‘0’ in it, then you are use the columns input and rows output to determine the button pressed on the keypad, and output that value to the seven segment display. If the key pressed is ‘*’, then the seven segment display outputs an ‘A’. If the key pressed is ‘#’, then the seven segment display outputs a ‘b’. The driver will immediately display the key pressed on the seven segment display as long as a key is physically being depressed. If no button was pressed, nothing will output on the seven segment display.

Example Use Code

```
; CPE 233 Winter 2018
; Hardware Assignment 3
; Example Use Code
; This code takes in two user inputs (one digit each time from a standard keypad).
; Each time an input is taken in, two checks are done to ensure the button pressed
; is not asterisk or pound. If one or both inputs are asterisk or pound, then the
; program will output the assigned invalid value 0xFF. If neither are, then it will
; output their proper sum.
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;- Port Constants
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.equ input = 0x20 ; input takes a keypad value
.equ output = 0x40 ; outputs the sum or error
.equ asterisk = 0x10 ; value we assign to asterisk
.equ pound = 0x11 ; value we assign to pound key
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;- Main program
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.cseg
.org  0x01 ; memory location of instruction data
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
main:
        in r0, input ; takes in a single digit from keypad (one button press)
        cmp r0, asterisk ; sees if button pressed is asterisk
        breq invalid ; branches to output invalid value
        cmp r0, pound ; sees if button pressed is pound
        breq invalid ; branches to output invalid value
        in r1, input ; assign value from in_port to r0
        cmp r1, asterisk ; sees if button pressed is asterisk
        breq invalid ; branches to output invalid value
        cmp r1, pound ; sees if button pressed is pound
        breq invalid ; branches to output invalid value
        add r1, r0 ; sum the two value digits
        out r1, output ; outputs the sum
        brn main ; restarts for new calculation

invalid:
        mov r2, 0xFF ; Invalid display assigned value 0xFF
        out r2, output ; outputs invalid result
        brn main ; restarts for new calculation
```

VHDL Source Code

Keypad Driver Wrapper

```
----------------------------------------------------------------------------
-- Company: CPE 233 Winter 2018
-- Engineer: Russell Caletena, Josiah Pang, & Nathan Wang
--
-- Create Date: 02/14/2018 10:00:31 PM
-- Design Name: Keypad Driver
-- Module Name: keypad_drvr - Behavioral
-- Project Name: HW Assignment 3
-- Target Devices: Basys 3
-- Description: Creates a keypad driver which reads in 12 values from a physical
--              keypad, ranging from 0-9, *, and #. Outputs to a 7-seg display.
--
--              This module is the wrapper for all the components.
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity keypad_drvr is
    Port (clk: in std_logic;
          cols: in std_logic_vector(2 downto 0);
          rows: out std_logic_vector(3 downto 0);
          digit: out std_logic_vector(3 downto 0);
          seg: out std_logic_vector(7 downto 0));

end keypad_drvr;

architecture Behavioral of keypad_drvr is

    -- Components used
    component clk_div_fs
    port (clk : in std_logic;
          clk_out : out std_logic);
    end component;

    component keypad_fsm
    port (clk: in std_logic;
```

```
        cols: in std_logic_vector(2 downto 0);
        rows: out std_logic_vector(3 downto 0);
        pressed_key: out std_logic_vector(7 downto 0));
    end component;

    component binary_to_sseg
    port (binary_result : in STD_LOGIC_VECTOR (7 downto 0);
        seg : out STD_LOGIC_VECTOR (7 downto 0);
        digit : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    -- Define signals
    signal sig_fsm_clk : std_logic;
    signal sig_temp_pressed_key: std_logic_vector(7 downto 0);

begin
    -- Port map
    c1: clk_div_fs
    port map(clk => clk,
            clk_out => sig_fsm_clk);

    f1: keypad_fsm
    port map(clk => sig_fsm_clk,
            cols => cols,
            rows => rows,
            pressed_key => sig_temp_pressed_key);

    sseg1 : binary_to_sseg
    port map(binary_result => sig_temp_pressed_key,
            seg => seg,
            digit => digit);

end Behavioral;
```

Keypad FSM

```
----------------------------------------------------------------------------------
-- Company: CPE 233 Winter 2018
-- Engineer: Russell Caletena, Josiah Pang, & Nathan Wang
--
-- Create Date: 02/18/2018 02:53:13 PM
-- Design Name: Keypad FSM
```

```
-- Module Name: keypad_fsm - Behavioral
-- Project Name: HW Assignment 3
-- Target Devices: Basys 3
-- Tool Versions:
-- Description: Defines the keypad FSM component for the keypad driver.
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity keypad_fsm is
  Port (clk: in std_logic;
        cols: in std_logic_vector(2 downto 0);
        rows: out std_logic_vector(3 downto 0);
        pressed_key: out std_logic_vector(7 downto 0));
end keypad_fsm;


architecture Behavioral of keypad_fsm is
    -- Create state types and signals for FSM
    type state_type is (st_init, st_row3, st_row2, st_row1, st_row0);
    signal ps, ns : state_type;
    signal tmp_clks : std_logic := '0';
    signal latched_pressed_key : std_logic_vector(7 downto 0);

begin

    -- Present and next state logic
    sync_process: process(clk)
    begin
        if (rising_edge(clk)) then
            ps <= ns;
        end if;
    end process sync_process;

    -- FSM logic
    comb_process: process (ps,ns)

    begin
        case ps is
            when st_init =>
                ns <= st_row3;
```

```
when st_row3 => -- Set row 0 to low, all others high (ACTUALLY ROW3)
   ns <= st_row2;
   rows <= "1110";
   if (cols = "011") then
      latched_pressed_key <= "00000001"; --assign 1 to pressed key
   elsif (cols = "101") then
      latched_pressed_key <= "00000010"; --assign 2 to pressed key
   elsif (cols = "110") then
      latched_pressed_key <= "00000011"; --assign 3 to pressed key
   else
      latched_pressed_key <= "11111111"; --assign nothing to pressed key
   end if;

when st_row2 => -- Set row 1 to low, all others high (ACTUALLY ROW2)
   ns <= st_row1;
    rows <= "0111";
   if (cols = "011") then
      latched_pressed_key <= "00000100"; --assign * to pressed key "00001010";
   elsif (cols = "101") then
      latched_pressed_key <= "00000101"; --assign 5 to pressed key "00000101";--
   elsif (cols = "110") then
      latched_pressed_key <= "00000110"; --assign 6 to pressed key "00000110";--
   else
      latched_pressed_key <= "11111111"; --assign nothing to pressed key
   end if;

when st_row1 => -- Set row 2 to low, all others high (ACTUALLY ROW1)
   ns <= st_row0;
   rows <= "1011";
   if (cols = "011") then
      latched_pressed_key <= "00000111"; --assign 7 to pressed key
   elsif (cols = "101") then
      latched_pressed_key <= "00001000"; --assign 8 to pressed key
   elsif (cols = "110") then
      latched_pressed_key <= "00001001"; --assign 9 to pressed key
   else
      latched_pressed_key <= "11111111"; --assign nothing to pressed key
   end if;

when st_row0 => -- Set row 2 to low, all others high (ACTUALLY ROW0)
   ns <= st_row3;
   rows <= "1101";
   if (cols = "011") then
```

```
                latched_pressed_key <= "00001010"; --assign 4 to pressed key "00000100";--
            elsif (cols = "101") then
                latched_pressed_key <= "00000000"; --assign 0 to pressed key "00001011";
            elsif (cols = "110") then
                latched_pressed_key <= "00001011"; --assign # to pressed key "00001100";--
            else
                latched_pressed_key <= "11111111"; --assign nothing to pressed key
            end if;

        end case;
    end process comb_process;

    pressed_key <= latched_pressed_key;

end Behavioral;
```

Binary to Seven Segment Display

```
----------------------------------------------------------------------------
-- Company: CPE 233 Winter 2018
-- Engineer: Russell Caletena, Josiah Pang, & Nathan Wang
--
-- Create Date: 02/20/2018 05:40:43 PM
-- Design Name: Binary to Seven Segment Display
-- Module Name: binary_to_sseg - Behavioral
-- Project Name: HW Assignment 3
-- Target Devices: Basys 3
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity binary_to_sseg is
  Port (binary_result: in std_logic_vector(7 downto 0);
        seg: out std_logic_vector(7 downto 0);
        digit: out std_logic_vector(3 downto 0));
end binary_to_sseg;

architecture Behavioral of binary_to_sseg is

begin

    -- Assigns seven segment output based on input segment value
    proc1: process (binary_result)
```

```
    begin
        digit <= "1110";
        case binary_result is
            when "00000000" => seg <= "11000000"; --0
            when "00000001" => seg <= "11111001"; --1
            when "00000010" => seg <= "10100100"; --2
            when "00000011" => seg <= "10110000"; --3
            when "00000100" => seg <= "10011001"; --4
            when "00000101" => seg <= "10010010"; --5
            when "00000110" => seg <= "10000010"; --6
            when "00000111" => seg <= "11111000"; --7
            when "00001000" => seg <= "10000000"; --8
            when "00001001" => seg <= "10010000"; --9
            when "00001010" => seg <= "10001000"; --A
            when "00001011" => seg <= "10000011"; --b
            when others => seg <= "11111111";
        end case;
    end process proc1;
end Behavioral;
```

Clock Divider

```
----------------------------------------------------------------------------
-- Company: Ratner Engineering
-- Engineer: bryan mealy
--
-- Create Date:    15:27:40 04/27/2007
-- Design Name:
-- Module Name:    CLK_DIV_FS
-- Project Name:
-- Description: This divides the input clock frequency into two slower
--              frequencies. The frequencies are set by the the MAX_COUNT
--              constant in the declarative region of the architecture.
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---------------------------------------------------------------------
-- Module to divide the clock
---------------------------------------------------------------------
```

```
entity clk_div_fs is
    Port (       clk : in std_logic;
            clk_out : out std_logic);
end clk_div_fs;

architecture my_clk_div of clk_div_fs is
   constant period : integer := 100000;  -- clock divider for 50% duty cycle
   signal tmp_clks : std_logic := '0';

begin

    my_div_slow: process (clk,tmp_clks)
    variable div_cnt : integer := 0;
    begin

    -- Clock Stuff
    if (rising_edge(clk)) then
      if (div_cnt >= period) then --2.5ms
        tmp_clks <= not tmp_clks;
        div_cnt := 0;
      else
        div_cnt := div_cnt + 1;
      end if;
    end if;
    clk_out <= tmp_clks;

    end process;

end my_clk_div;
```