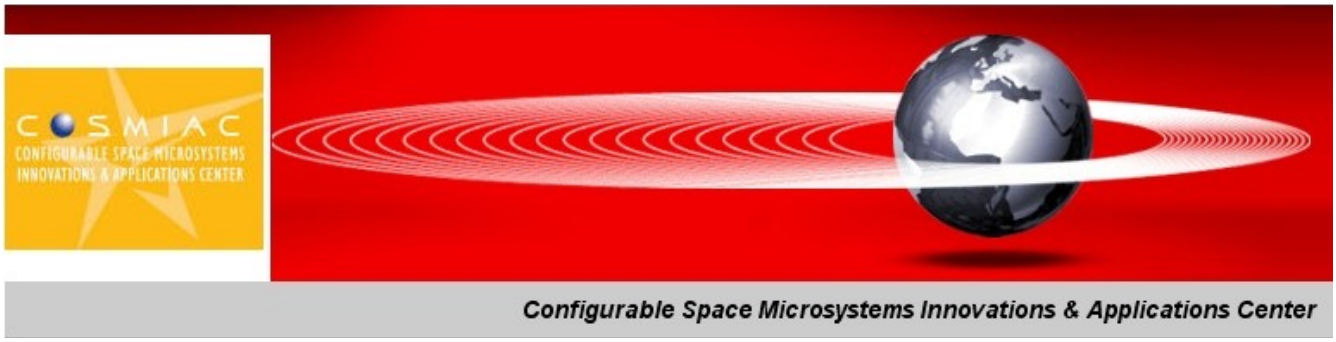

Lab 9 – Tutorial
Clock with Xilinx ISE 10.1 and Digilent Spartan 3E Starter

Introduction

A very important concept in digital design is that of the clock. A clock is used to synchronize systems in digital logic, and provides a convenient way to keep track of real time. Another equally important fact is the ability to translate information to a human readable form. In our case, the peripheral module seven segment display device PmodSSD [1] will be used. The Digilent Spartan 3E Starter board [2] has a 50 MHz oscillator that is used creatively to obtain other useful clock frequencies.

Objective

The objective is to implement a clock that utilizes an external seven segment display connected to the Digilent Spartan 3E Starter development board. The clock will be able to count up, in addition to down, and will display hundredths of seconds, seconds, and minutes. As always, there will be an option to reset the time back to zero. The clock will be synchronized with real time.

Process

1. Design a display controller capable of accepting a four bit input.
2. Design a counter capable of counting hundredths of a second up to minutes, both forwards and backwards.
3. Tie the counter to the display.

Implementation

1. The leds on the PmodSSD [1] are tied to power, so a digital high (‘1’) will be used for “ON”, and ‘0’ will be off. The seven segment display on the PmodSSD do not operate in the classical sense where each pin is controlled independently. Instead, both seven segments are controlled simultaneously. To enable a single segment of the display would activate the other. Each of the two C1 and C2 cathodes is controlled via a separate pin. The picture in Fig. 1 is included for convenience.

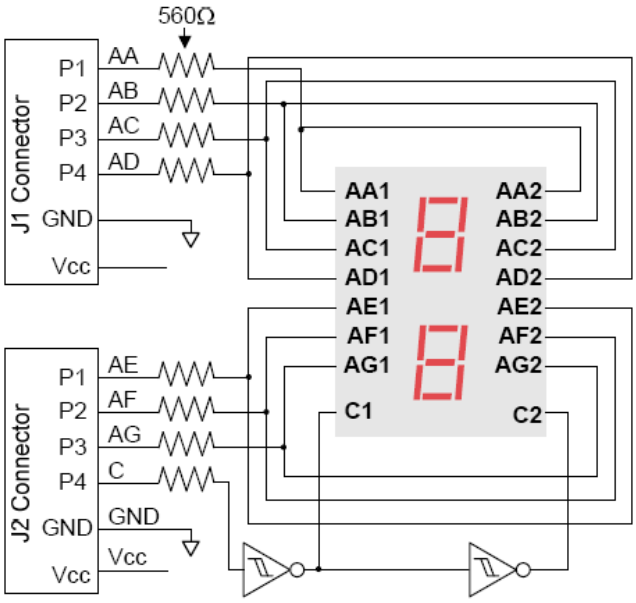

Fig. 1. Connection between the Digilent Spartan 3E Starter board and the peripheral module PmodSSD [1].

2. As one can see, the only way to display unique patterns on each of the two seven segment displays is to cycle each of the two displays by toggling the cathodes C1 and C2 in succession, all the while changing the seven segment display's configuration. For example, to display a '0' in the first position, while displaying '1' in the second position, one would cycle through a list at times t1 and t2, as shown in Table 1.

	t1	t2
C1	1	0
C2	0	1
AA	1	0
AB	1	1
AC	1	1
AD	1	0
AE	1	0
AF	1	0
AG	0	0

Table 1. List of the pin values required to show a 01 in the peripheral module PmodSSD [2],[3].

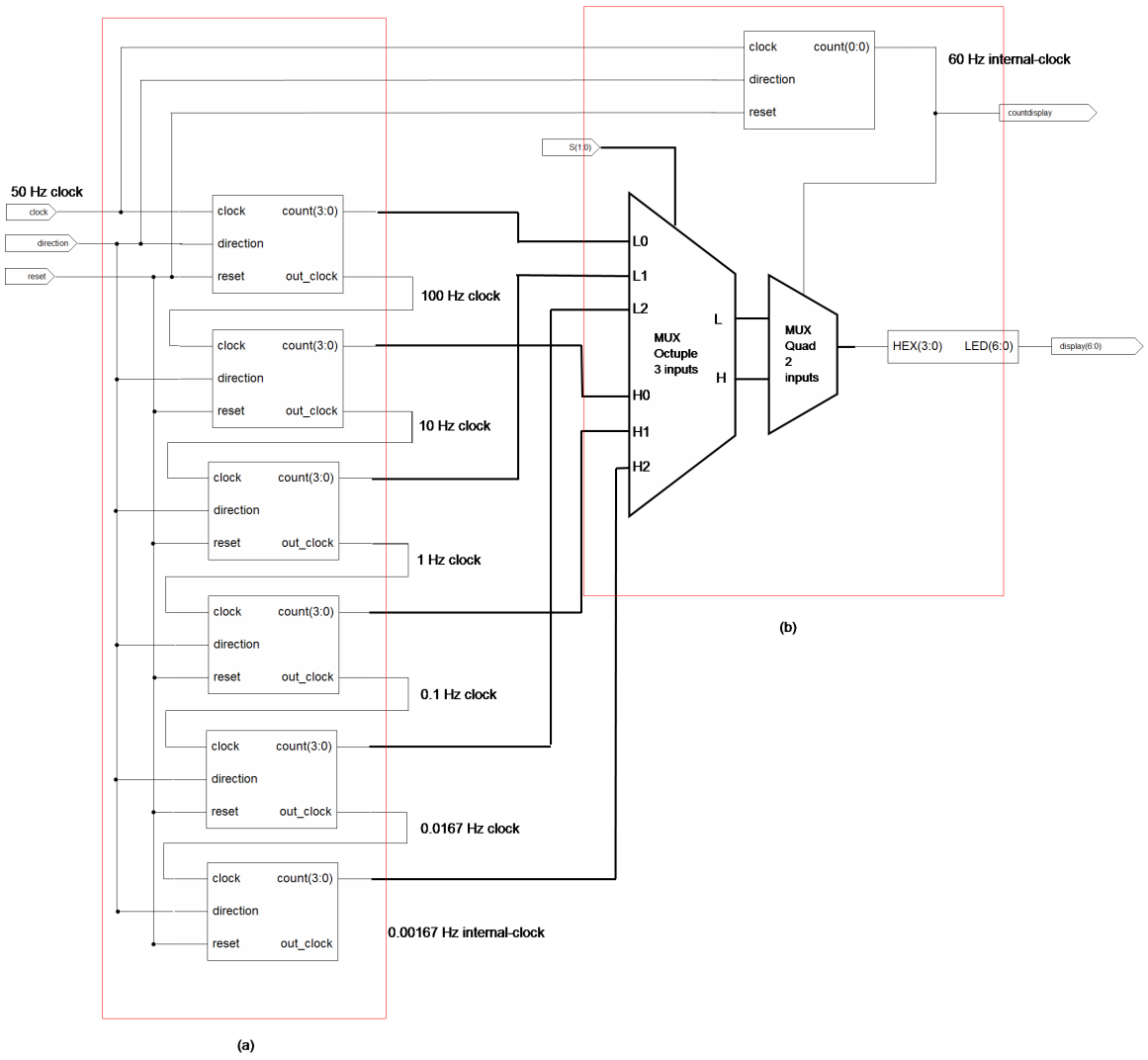

Fig. 2. General scheme of the required system.

3. The clocks will work in a tiered fashion as shown in the Fig. 2 part (a) enclosed by the red rectangle. That is, a 100 Hz clock will be divided from the original 50 MHz clock. The 100 Hz clock will be further divided to create the 10 Hz clock and so forth. 100 Hz will serve to update the hundredths of a second display, 10 Hz the tenths of a second, 1 Hz the seconds, 0.1 Hz the tens of a second, 0.0167 Hz the minutes and 0.00167 Hz the tens of a minute. Take a look to the following clock divider code to obtain 100 Hz from 50 MHz:

EXAMPLE CLOCK DIVIDER CODE:

```
entity divider is
GENERIC(N : INTEGER := 499999);
PORT (clk,reset : in std_logic;
      newclk : buffer std_logic);
end divider;

architecture Behavioral of divider is

SIGNAL step : INTEGER RANGE 0 to N;
```

```
begin
 process(clk,reset)
 begin
    if reset = '1' then
       newclk <= '0';
       step <= 0;
    elsif clk'event and clk = '1' then
       if step = N then
          step <= 0;
          newclk <= '1';
       else
          step <= step + 1;
          newclk <= '0';
       end if;
    end if;
 end process;
end Behavioral;
```

Note the *GENERIC* statement [4] in the second line. This is a powerful tool when we need several instances of an entity through the *port map* command. The line `GENERIC(N : INTEGER := 499999);` defines an integer variable `N` equal to 499,999, that determines the clock division by 500,000. Notice that if you need to divide the clock by a different value, it is only necessary to change the value of `N`. Furthermore, this value could be changed in a new instance as follows [3]:

```
hz100 : divcount generic map (N => 499999)
        port map (reset,direction,clock,count100,clock100);
```

In this example, we have an instance of the entity *divcount.vhd*, but before we proceed with the *port map* [3] command we add a line with the command *generic map* in order to assign a new value to the parameter `N`. It is worthy to say that you can use more than one parameter if it is necessary.

4. All of the counters will be commonly connected to a reset input in addition to an input which will control the direction all six clock dividers count. The counters will each have a four bit output leading to the display multiplexers that are shown in Fig. 2 part (b) enclosed by the red rectangle. Check the following example of a forward-backward counter code that counts integer values between 0 and 5 using the *GENERIC* command again:

EXAMPLE FORWARD-BACKWARD COUNTER CODE

```
entity counter is
GENERIC (N : INTEGER := 5);
PORT ( reset,clock,count_direction : in std_logic;
       count : buffer INTEGER RANGE 0 to N);
end counter;

architecture Behavioral of counter is

begin
process (clock, reset)
   begin
      if reset='1' then
         count <= 0;
      elsif clock='1' and clock'event then
```

```
        if count_direction='1' then
           if count = N then
              count <= 0;
           else
              count <= count + 1;
           end if;
        else
           if count=0 then
              count <= N;
           else
              count <= count - 1;
           end if;
        end if;
   end if;
end process;
end Behavioral;
```

5. The seven blocks in Fig. 2 part (a) that deliver the clocks and counts that synchronize the system are built joining the clock divider and counter in only one entity that will be called *divcount* using *component* and *port map* commands .

   In Fig. 3. we can see the behavioral simulation of the *divcount* entity. It is worthy to mention that because of the computational cost, it isn't wise to simulate the behavior of the whole system, as was mentioned previously, the 100 Hz clock is obtained from the 50 MHz oscillator, therefore the simulation of only one period would imply 500,000 cycles of the 50 MHz clock, and we have to take in account that we don't need only one cycle of the hundredths of second. Instead, we need to simulate the minutes, this implies at least the simulation of 6,000 x 500,000 cycles, which means a high computational cost for a typical machine.

   However, as shown in the figure below, the behavioral simulation using ISE Simulator [4] of the *divcount* entity counts in an integer range 0 to 3 (forward and backward) and divides by 5 the frequency of the input clock (see output out_clock). Note that when the signal *direction* is low, counts down and viceversa.

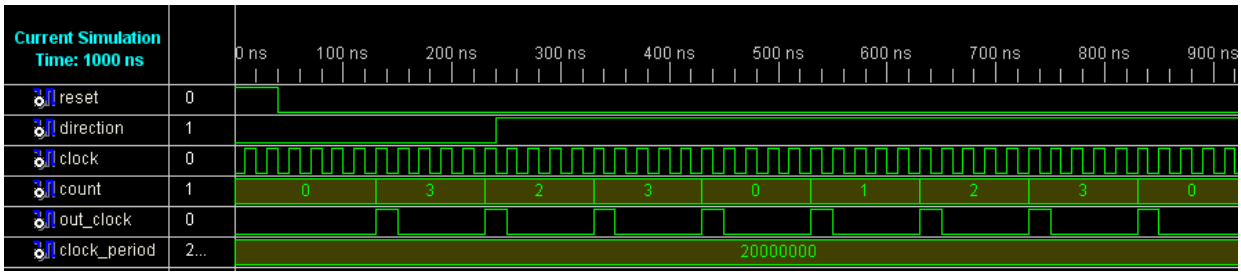

Fig. 3. Behavioral simulation of the *divcount* entity.

6. In the implementation of the display controller, one will have an octuple 3-inputs multiplexer in order to select to show either the hundredths of a second, the seconds or the minutes using the slide switches (see Fig. 8) as the select control. The previously mentioned values need to be represented using BCD code therefore we need four bits for the high and the low part of each value. The high and low parts are represented in Fig. 2 for the letters H and L respectively. The integers that are next to the letters at the input of the multiplexer are the indexes that select every couple of inputs. The following is an example of a quad 2 input multiplexer code [3]:

EXAMPLE QUADRUPLE 2 INPUTS MULTIPLEXER

```
entity mux4 is
PORT( A,B : in integer range 0 to 9;-- Multiplexer inputs A and B
      S : in integer range 0 to 1;-- Multiplexer selector S
      F : out integer range 0 to 9);-- Multiplexer output F
end mux4;

architecture Behavioral of mux4 is
begin

   with S Select
      F     <=    A when 0,
      B when others;

end Behavioral;
```

Hint: In order to convert it in an octuple 2-input multiplexer, you have to set the length of the inputs A, B and C to eight bits.

The behavioral simulation of the multiplexer using ISE Simulator [4] is shown in Fig. 4. where a and b are the input signals (4 bits), s is the selector and f the output (4 bits).

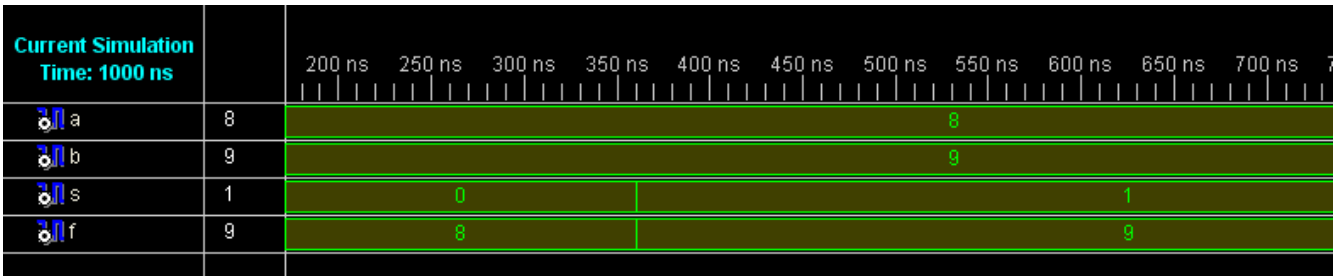

Fig. 4. Behavioral Simulation of the quad 2-inputs multiplexer.

7. The display controller has an additional quadruple 2-input multiplexer similar to the one shown in the previous code and its select input is connected to a clock divider which works at 60 Hz. This means that each digit is illuminated once every 16 ms but, because the eye can't perceive the blinking at this frequency, the digits appear continuously illuminated [1]. The 60 Hz signal is obtained through an instance of a clock divider shown previously that delivers a clock frequency of 60 Hz from the 50 MHz oscillator (dividing by 833,333), as you can note from the part (b) of Fig. 2.

8. The BCD to seven segment converter can be implemented in many ways. The following is another alternative to implement a BCD to seven segment code [3]:

EXAMPLE CODE BCD TO SEVEN SEGMENT CONVERTER

```
entity BCDseven is
   PORT( HEX : in integer range 0 to 9;
         LED : out std_logic_vector(6 downto 0));
end BCDseven;

architecture Behavioral of BCDseven is
   with HEX SELect
      LED<= "0111111" when 0,
            "0000110" when 1,
            "1011011" when 2,
            "1001111" when 3,
            "1100110" when 4,
            "1101101" when 5,
            "1111101" when 6,
            "0000111" when 7,
            "1111111" when 8,
            "1101111" when 9,
            "-------" when others;
end Behavioral;
```

Fig. 5 shows a simulation of the converter above using ISE Simulator.

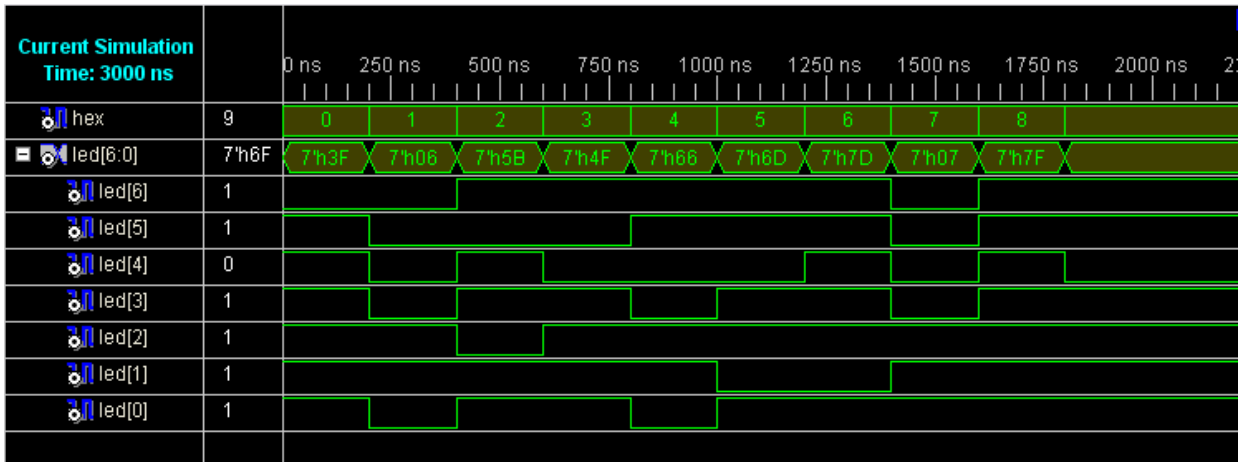

Fig. 5. Behavioral Simulation of the BCD 7 Segment Converter

Based on Fig. 1, the LSB and the MSB of the output LED in the code above correspond to the leds AA and AG.

9. Lastly, the two systems (the *divcount* and the display controller) need to be connected through the use of *port maps* and *component* commands. This process was explained in the Tutorial No. 3. The part (a) of Fig. 2 shows the *divcount* system and the part (b) the Display Controller. A possible RTL diagram obtained using Xilinx ISE 10.1 is shown in Fig. 6. The signals *count100*, *count10* and *count1* correspond to the hundredths of a second, tenths of a second and second respectively. And the signals *countd1*, *countd0167* and *countd00167* correspond to the tents of a second, the minutes and the tents of a minute respectively. *countdisplay* is the output that controls the cathodes C1 and C2 (see Fig. 1) and *display* is the seven segment output.

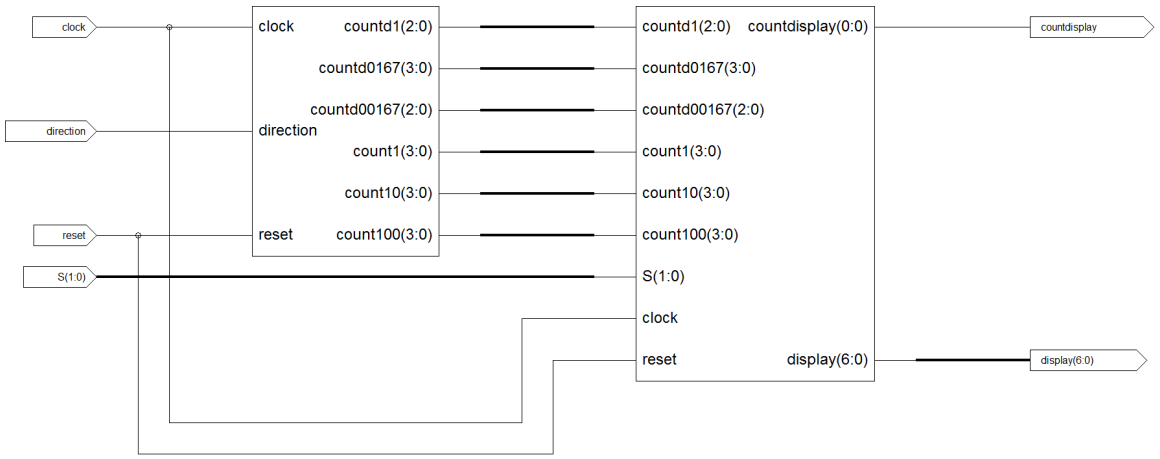

Fig. 6. RTL diagram of the whole system. Note the *divcount* system (left) and the Display Controller (right).

10. This will allows the counter outputs to connect directly to the display, forming the clock. The reason that the two systems were developed separately is that it greatly simplifies troubleshooting, makes the code more understandable, and allows the code to be easily reused. So now, the display controller can easily be reused in another project.

11. Create a new source ***Implementation Constraints File***, and assign the pins [4] as follows in order to use the Digilent PmodSSD connected to the 6-pin accessory headers J1 and J2 [2] respectively for the proposed pin assignment.

```
NET "clock"  LOC = "C9"  ;
NET "countdisplay<0>"  LOC = "F7"  ;
NET "direction"  LOC = "N17"  ;
NET "display<0>"  LOC = "B4"  ;
NET "display<1>"  LOC = "A4"  ;
NET "display<2>"  LOC = "D5"  ;
NET "display<3>"  LOC = "C5"  ;
NET "display<4>"  LOC = "A6"  ;
NET "display<5>"  LOC = "B6"  ;
NET "display<6>"  LOC = "E7"  ;
NET "reset"  LOC = "K17"  ;
NET "S<0>"  LOC = "L13"  ;
NET "S<1>"  LOC = "L14"  ;
```

Trough this pin assignment, the reset is connected to the push-button BTN_SOUTH (pin K17), the direction control is connected to the slide switch SW3, and the selectors S<1> and S<0> to the slide switches SW1 (pin L14) and SW0 (L13) respectively [2]. Please check Fig. 7 and Fig. 8 to locate these controls.

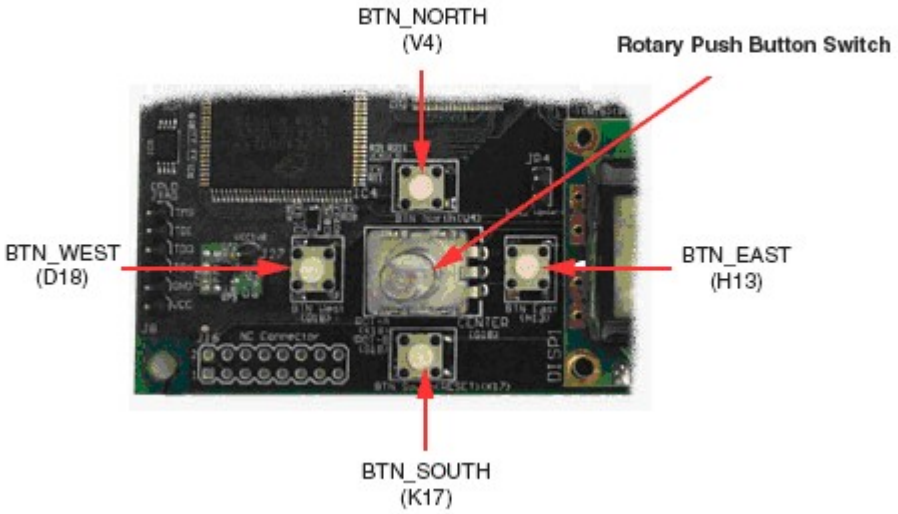

Fig. 7. Push-Buttons in the Digilent Spartan 3E Starter board [2]

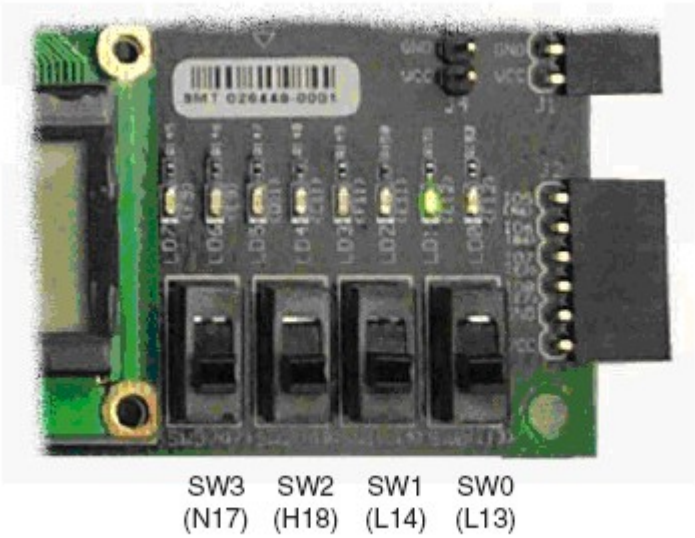

Fig. 8. Slide Switches in the Digilent Spartan 3E Starter board [2]

Fig. 9. shows the physical connections between the connector headers J1, J2 and the PmodSSD.

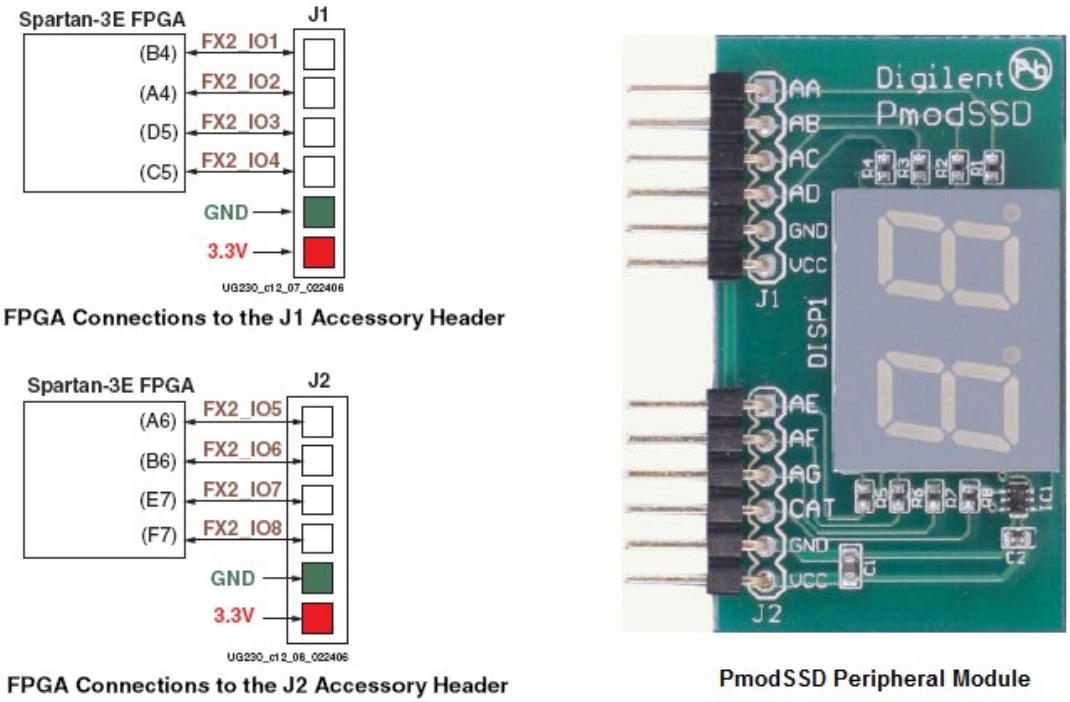

Fig. 9. Connections between the connectors J1, J2 and the PmodSSD [1],[2].

References

[1] *Digilent PmodSSD Peripheral Module Board Reference Manual.* Digilent Inc. Revision 06/07/05. pp. 1-2. Available: http://www.digilentinc.com/Data/Products/PMOD-SSD/Pmod%20SSD_rm.pdf

[2] *Spartan-3E Starter Kit Board User Guide.* Xilinx Inc. Revision UG230 March 9, 2006. pp. 15-122. Available: http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf

[3] S. Brown, "Fundamentals of Digital Logic with VHDL Design" Ed. Singapore: McGraw-Hill. 2000, pp 828.

[4] *ISE In-Deph Tutorial.* XILINX Inc. Copyright 1995-2007. pp. 36-39. Available: http://download.xilinx.com/direct/ise9_tutorials/ise9tut.pdf

About the author:

Jose Marcio Luna Castaneda
M.S. in Electrical Engineering Student
The University of New Mexico
Office: ECE Building Room 223, Phone: 277-1372
Areas of Interest: Multi-agents, robotics, hybrid systems, computational intelligence.
e-mail: jmarcio@ece.unm.edu

Updated By:

Brian Zufelt
Undergraduate student at the University of New Mexico