An-Najah National University

Faculty of Engineering and IT

جامعة النجاح الوطنية

كلية الهندسة وتكنولوجيا المعلومات

Digital Circuit Design Lab2

Dr. Ashraf Armoush

Wednesday 2:00 pm – 5:00 pm

First Semester

Experiment Information		
Experiment Name: ASM (Fibonacci Number)	Experiment Number: #4	
Performed: 15 Oct 2020	Submitted: 16 Oct 2020	
Partner Students		
1- Taher Anaya	2- Hadi Jml	3 – Ameed Omar

Introduction:

In this experiment, we will implement an algorithmic state machine (ASM) on the Spartan-3E starter board to compute the n-th Fibonacci number using VHDL.

Objectives:

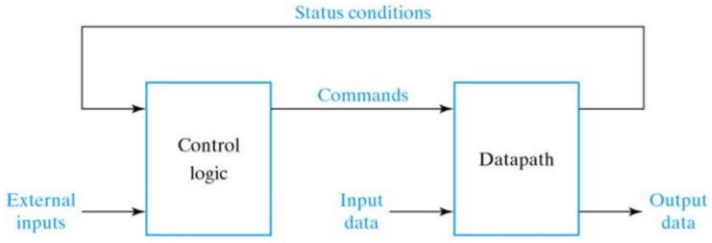
- Building ASM Chart for Fibonacci Number.
- Building VHDL Code from ASM Chart.
- Simulation VHDL code and synthesizing it on an FPGA.

Procedure:

This experiment's main objective is to build an ASM chart for the Fibonacci number system and then write a VHDL code to represent the ASM system with the appropriate states.

ASM Chart

Every digital system can be partitioned into two parts: data path circuits and control unit circuits. Data path circuits perform functions such as storing binary information (data) and transfer it from one system to the other system. Whereas control circuits determine the flow of operations of digital circuits.



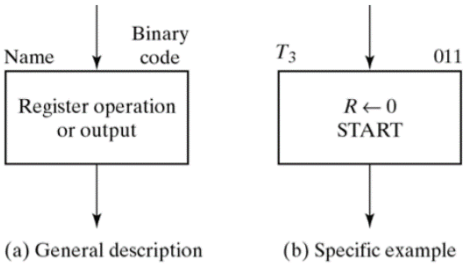
It is not easy to describe the behavior of large state machines using state diagrams. To overcome this difficulty, Algorithmic State Machine (ASM) charts can be used. ASM charts are similar to flow charts. They represent the flow of tasks to be performed by data path circuits and control circuits.

Procedure: *(cont.)*

The ASM chart is composed of 3 basic elements:

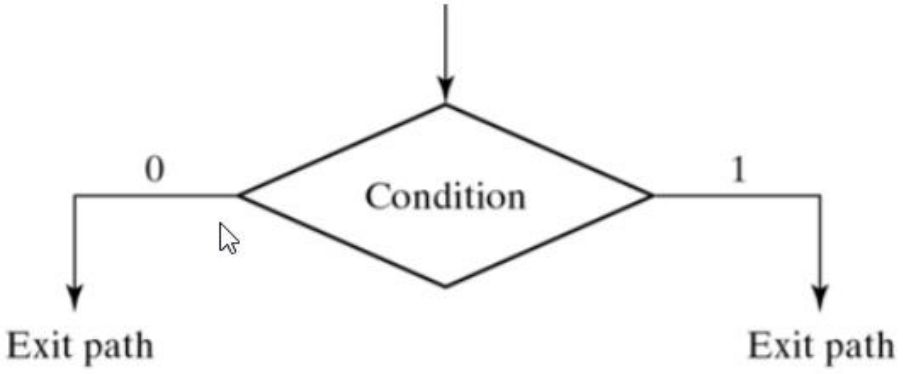
- ***State Box***

The state box is represented in a rectangular shape. Each state box represents one state of the sequential circuit. It is having one entry point and one exit point. The name of the state is placed above of state box. The unconditional outputs corresponding to that state can be placed inside the state box. Moore state machine outputs can also be placed inside the state box. The symbol of the state box is shown in the following figure.



- ***Decision Box***

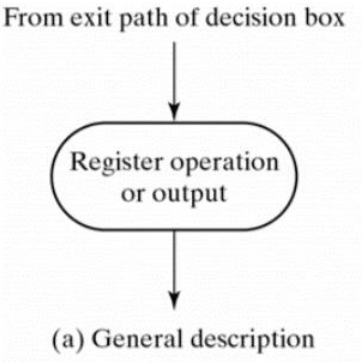
The decision box is represented in a diamond shape. It is having one entry point and two exit paths. The inputs or Boolean expressions can be placed inside the decision box, which is checked whether they are true or false. If the condition is true, then it will prefer the path1. Otherwise, it will choose path2. The symbol of the state box is shown in the following figure.



Procedure: *(cont.)*

- ***Conditional Box***

The conditional output box is represented in an oval shape with one entry point and one exit point similar to the state box. The conditional outputs can be placed inside the state box. In general, Mealy state machine outputs are represented inside the conditional output box. So, based on the requirement, we can use the above components properly for drawing ASM charts. The symbol of the state box is shown in the following figure.



Now let's talk a little more about the Fibonacci number.

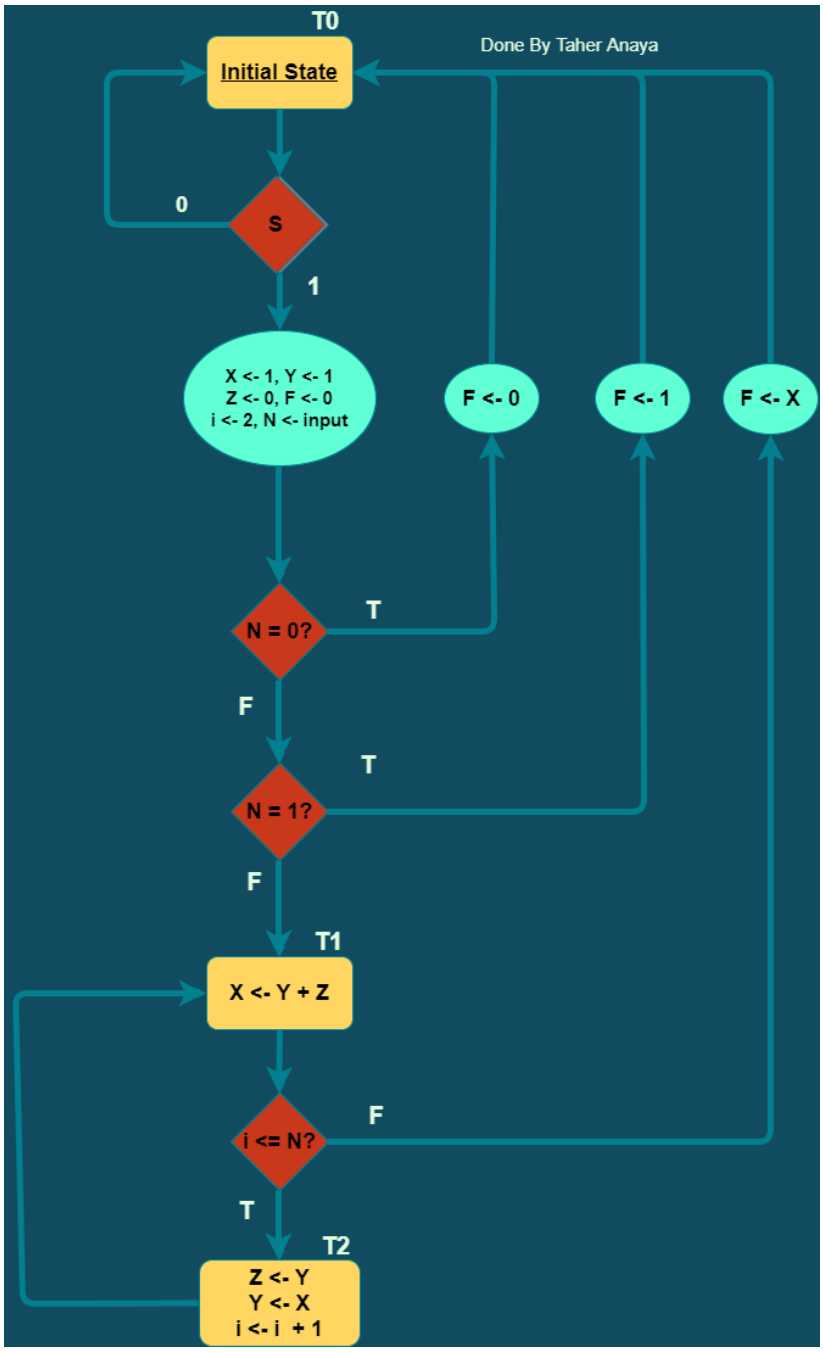
Fibonacci Number

In mathematics, the Fibonacci numbers commonly denoted Fn, form a sequence called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$f(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ f(n-1) + f(n-2), & n > 1 \end{cases}$$

Procedure: *(cont.)*

Now that we got to know the ASM and the Fibonacci, we can build an ASM chart for the Fibonacci system Like this:



Procedure: *(cont.)*

Now, we will implement the previous ASM chart on the Spartan-3E starter board to compute the n-th Fibonacci number in VHDL using three processes.

First, we want to build the ASM entity. Here's the entity declaration.

```
entity ASMFibonacci is
    Port ( S : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           N : in  STD_LOGIC_VECTOR (3 downto 0);
           F : out  STD_LOGIC_VECTOR (7 downto 0));
end ASMFibonacci;
```

Then, we wrote three processes. The first one is the move to the next state. The second one is the calculate the next state. Finally, the third one is the calculation outputs.

Signals Declaration and First Process:

This process is triggered by the clock's positive edge to assign the next state to the current state. It also contains an asynchronous reset, which returns the system to state zero, waiting for the start(S) signal.

```
architecture Behavioral of ASMFibonacci is

    -- All Signals --
   type stateType is (T0, T1, T2);
   Signal currentState, nextState: stateType;
   Signal i: integer range 0 to 15 := 0;
   Signal X, Y, Z: STD_LOGIC_VECTOR (7 downto 0);

begin

   -- First Process --
   p1: process (CLK, Reset)
   begin
      if (Reset = '1') then
         currentState <= T0;
      elsif (CLK'event and CLK = '1') then
         currentState <= nextState;
      end if;
   end process;
```

Procedure: *(cont.)*

Second Process:

This process follows the ASM chart to decide which state will come next based on the current state and the external inputs. It updates that at every-positive edge clock. It keeps the state that we should move to in the next cycle in (nextState) variable.

```
-- Second Process --
p2: process (S, currentState)
begin
   case currentState is
   when T0 =>
      if (S = '0' or (S = '1' and (N = "0000" or N = "0001"))) then
         nextState <= T0;
      else
         nextState <= T1;
      end if;
   when T1 =>
      if (i <= N) then
         nextState <= T2;
      else
         nextState <= T0;
      end if;
   when T2 =>
      nextState <= T1;
   end case;
end process;
```

Procedure: *(cont.)*

Third Process:

The positive edge of the clock triggers this process. It changes the outputs depending on the current state.

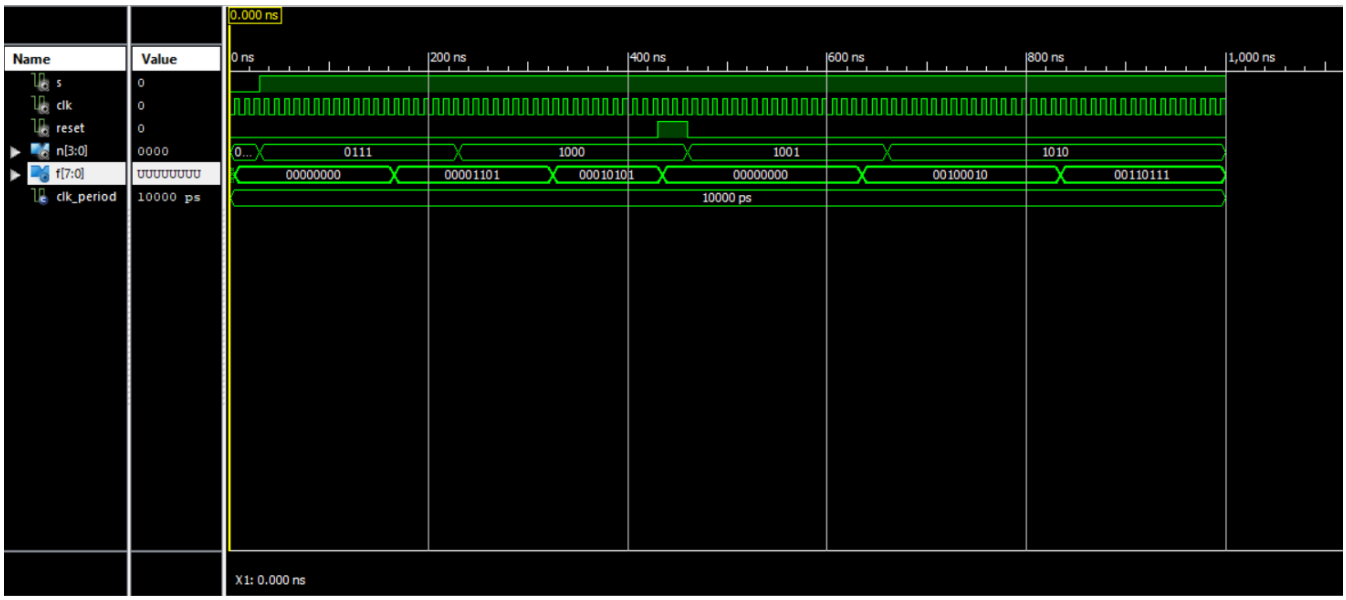
```
-- Third Process --
p3: process (CLK)
begin
   if (CLK'event and CLK = '1') then
      if ( Reset = '1') then
         F <= "00000000";
      end if;
      case currentState is
         when T0 =>
            if (S = '1') then
               X <= "00000001";
               Y <= "00000001";
               Z <= "00000000";
               i <= 2;
            end if;
            if (N = "0000") then
               F <= "00000000";
            elsif(N = "0001") then
               F <= "00000001";
            end if;
         when T1 =>
            X <= Y + Z;
            if (i > N) then
               F <= X;
            end if;
         when T2 =>
            Y <= X;
            Z <= Y;
            i <= i + 1;
      end case;
   end if;
end process;
```

Procedure: (cont.)

Now, we want to write a test bench module for the ASM code, as shown in the following figure:

```
-- Stimulus process
stim_proc: process
begin

   wait for CLK_period*3;

   S <= '1';
   N <= "0111";      wait for CLK_period*20;
   N <= "1000";      wait for CLK_period*20;

   Reset <= '1';     wait for CLK_period*3;
   Reset <= '0';
   S<= '1';

   N <= "1001";      wait for CLK_period*20;
   N <= "1010";      wait for CLK_period*20;

   wait;
end process;
```

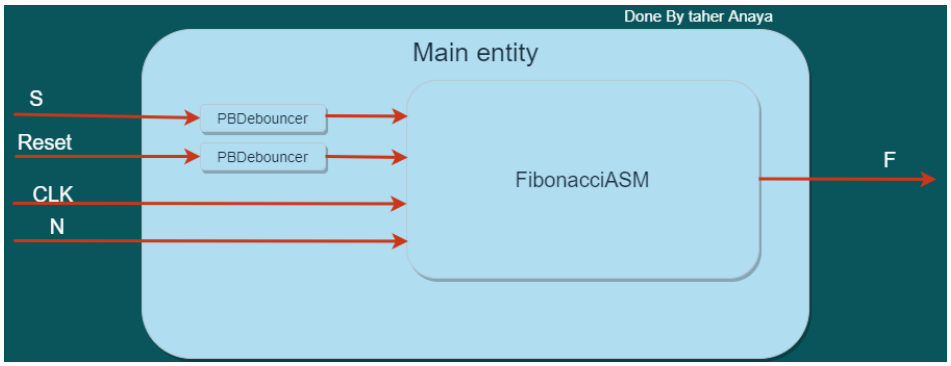
We want to simulate the test bench file and the result shown in the following figure to make sure that the design behaves correctly.



As we can see, no issues were found, and the result was exactly as expected.

Procedure: *(cont.)*

Now, we want to synthesize it, so we need to insert debounce in the main entity, as shown in the following figure:



From the previous picture, we can build the main entity, as shown in the following figure:

```
entity MainASM is
    Port ( S : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           N : in  STD_LOGIC_VECTOR (3 downto 0);
           F : out  STD_LOGIC_VECTOR (7 downto 0));
end MainASM;

architecture Behavioral of MainASM is

   component PBDebouncer
    Port ( PB : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           PB_Debounced : out  STD_LOGIC);
   end component;

   component ASMFibonacci
    Port ( S : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           N : in  STD_LOGIC_VECTOR (3 downto 0);
           F : out  STD_LOGIC_VECTOR (7 downto 0));
   end component;

   Signal SS, RR: STD_LOGIC;

begin

   PBD0: PBDebouncer port map (S, CLK, SS);
   PBD1: PBDebouncer port map (Reset, CLK, RR);
   AF: ASMFibonacci port map (SS, CLK, RR, N, F);

end Behavioral;
```

Note: PBDebouncer VHDL Code exists in previous reports.

Procedure: *(cont.)*

The final step is to synthesize the main entity on the FPGA, so we need to create a UCF file to map the inputs and outputs in VHDL to the switches, pushbuttons and LEDs in the FPGA, just like we did with the previous experiments.

Now, we write a user constraint file, as shown in the following figure:

```
# PlanAhead Generated IO constraints

NET "S" PULLDOWN;
NET "Reset" PULLDOWN;

# PlanAhead Generated physical constraints

NET "CLK" LOC = C9;
NET "S" LOC = V4;
NET "Reset" LOC = K17;
NET "N[0]" LOC = L13;
NET "N[1]" LOC = L14;
NET "N[2]" LOC = H18;
NET "N[3]" LOC = N17;
NET "F[0]" LOC = F12;
NET "F[1]" LOC = E12;
NET "F[2]" LOC = E11;
NET "F[3]" LOC = F11;
NET "F[4]" LOC = C11;
NET "F[5]" LOC = D11;
NET "F[6]" LOC = E9;
NET "F[7]" LOC = F9;
```

Conclusion:

The ASM is an algorithm that consists of a few steps, which is used to simplify a sequential digital system. An ASM chart resembles a conventional flow chart but the difference is that a traditional flow chart does not have timing relationships, but the ASM takes timing relationships into account. An ASM chart describes the sequence of events and the timing relationship between the states of a sequential controller and the events that occur while going from one state to the other. It is employed to design a sequential circuit with many external inputs. With many external inputs, it becomes complicated to use state tables to create the circuit.