07/20/2021 Fatma Erem Aksoy

2315075

CNG 232 LOGIC DESIGN - LAB 7 REPORT

Objective:

In this report, designing both a Datapath and a Controller FSM to achieve a Fibonacci Series Calculator will be discussed. The Control FSM used has two parts which are FSM and FSM_DECO. FSM here is a Moore Machine and FSM_DECO is a decoder. The aim of the Control FSM is to control data flows with the help of a sequential circuit.

6.6 PROJECT REPORT

6.6.1 STATE DIAGRAM OF THE FIBO_FSM

FIBO_FSM has two parts which are FSM and FIBO_DECO:

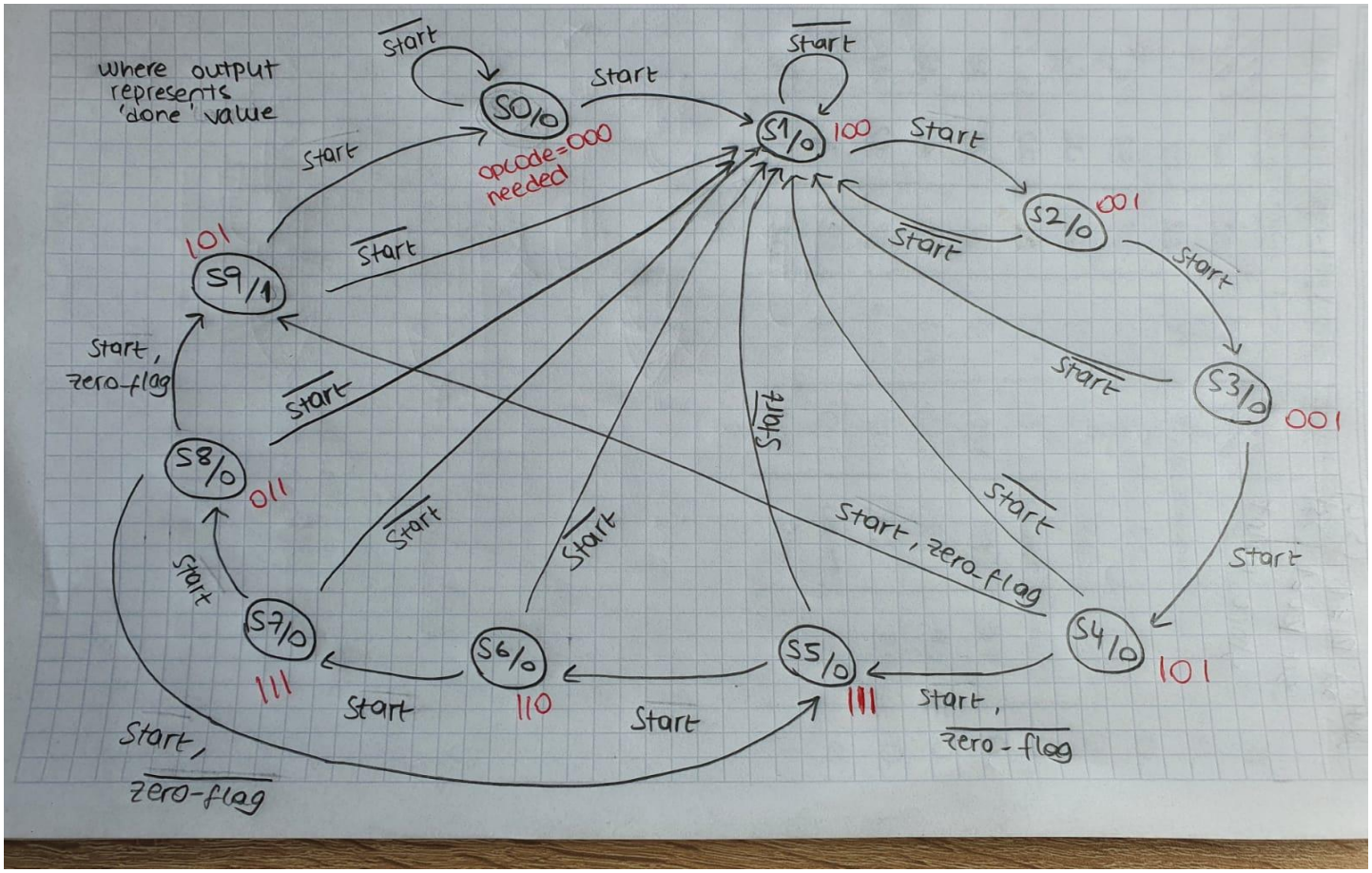
The state table and the diagram below belong to the control FSM. They show all the states, inputs, outputs, and the opcode needed. Start, zero_flag, reset are the inputs for the FSM while Done is an output. Also, opcode, operand1 and operand2 represent the outputs for the FSM to perform all the required control operations. As it can be understood from the states and the inputs given in the table; this FSM is a Moore Machine, meaning that the next state depends on the current state and the inputs.

FIBO_DECO is a decoder which takes opcode, operand1 and operand2 as input and generates 6 outputs such as wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, and alu_opcode.

The state table for the control FSM:

State	Start / zero-flag / reset			opcode	operand1	operand2	done
	00-	01-	1--				
S0	S0	S0	S1	000	--	--	0
S1	S2	S2	S1	100	count	--	0
S2	S3	S3	S1	001	R1	--	0
S3	S4	S4	S1	001	R2	--	0
S4	S5	S9	S1	101	count	--	0
S5	S6	S6	S1	111	R3	R1	0
S6	S7	S7	S1	110	R1	R2	0
S7	S8	S8	S1	111	R2	R3	0
S8	S5	S9	S1	011	count	--	0
S9	S0	S0	S1	101	R1	--	1

The state diagram for the control FSM:



6.6.2 PARAMETERIZED VERILOG CODE

- module Memory(Clk, data, count); //memory will be used to store count
  parameter size = 4;
  input [size-1:0] data;
  input Clk;
  output reg [size-1:0] count;

  always @(posedge Clk)
  begin
  count <= data;
  end
  endmodule

- module FSM(Clk, Rst, Start, zero_flag, opcode, R1, R2, Done); //The first part of the FIBO_FSM

  parameter size = 3 ;
  input Clk, Rst, Start, zero_flag;
  output reg Done = 0;
  output reg [size-1:0] opcode; //opcode needed for the states
  output reg [1:0] R1, R2;
  reg [1:0] R3;
  reg next_state, state; //where state represents the current state

```
parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6, S7=7, S8=8, S9=9;


always @(posedge Clk or posedge Rst)    //State transitions
begin
 if(Rst)
   state = S0;
 else
   state <= next_state;
end


always @(state or Start)    //Next state assignments
begin
 if(Start)                  //the procedure will start when the start value is 1
   begin
     case(state)
       S0: begin
            next_state = S1;
            R1 = 00;
            R2 = 00;
            R3 = 00 ;
            opcode = 000;
          end
       S1: begin  R1 = 01; R2 = 01; next_state = S2; opcode = 100;  end
       S2: begin  next_state = S3; opcode = 001; end
       S3: begin  next_state = S4; opcode = 001; end
       S4: begin
                    opcode = 101;
            if (zero_flag==0)
              next_state = S5;
            else
              begin next_state = S0; Done = 1 ; end
          end
       S5: begin R3 <= R1; next_state = S6; opcode = 111; end
       S6: begin R1 <= R1 + R2; next_state = S7; opcode = 110; end
       S7: begin R2 <= R3; next_state = S8; opcode = 111; end
       S8: begin
            opcode = 011;
            if (zero_flag==0)       //checks if the count value is 0 to continue to proceed
              next_state = S5;
            else
              next_state = S9;
          end
       S9: begin next_state = S0; Done = 1 ; opcode = 101; end         //Done value will
                                                  //be assigned to 1 and the process will
                                       //end when the count value is 0 and zero_flag is 1
     endcase
   end
```

```
    else                    //go back to initial state (noop) if start value is 0
      next_state = S0;
   end

  endmodule
```

- ```
  module FSM_DECO(opcode, operand1, operand2, wrt_addr, wrt_en, load_data,
  rd_addr1, rd_addr2, alu_opcode);          //The second part of the FIBO_FSM
   parameter size = 4;
   input [size-2:0] opcode;
   input [1:0] operand1, operand2;
   output reg [size-2:0] alu_opcode;
   output reg [1:0] wrt_addr, rd_addr1, rd_addr2;
   output reg wrt_en, load_data;


   always @(opcode)
   begin
    alu_opcode = opcode;      //these values will be the same regardless of the
    rd_addr2 = operand2;      // opcode, so they are assigned at the beginning
    case(opcode)
     000: wrt_en = 0;
     001: begin
          wrt_addr = operand1;
          wrt_en = 1;
          load_data = 0;
         end
     010: begin
          rd_addr1 = operand1;
          wrt_addr = operand1;
          wrt_en = 1;
          load_data = 0;
         end
     011: begin
          rd_addr1 = operand1;
          wrt_addr = operand1;
          wrt_en = 1;
          load_data = 0;
         end
     100: begin
          wrt_addr = operand1;
          wrt_en = 1;
          load_data = 1;
         end
     101: begin
          rd_addr1 = operand1;
          wrt_en = 0;
         end
  ```
  ```

```
        110: begin
            rd_addr1 = operand1;
            rd_addr2 = operand2;
            wrt_addr = operand1;
            wrt_en = 1;
            load_data = 0;
           end
        111: begin
            rd_addr1 = operand2;
            wrt_addr = operand1;
            wrt_en = 1;
            load_data = 0;
           end
      endcase
     end
    endmodule
```

- module Fibo_FSM(Clk, Rst, Start, zero_flag, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode, Done); //the combined FIBO_FSM with two parts
  parameter size = 3 ;
  input Clk, Rst, Start, zero_flag;
  output [size-1:0] alu_opcode;
  output [size-2:0] wrt_addr, rd_addr1, rd_addr2;
  output wrt_en, load_data, Done;
  wire [size-1:0] opcode;
  wire [size-2:0] R1, R2;

  FSM FSM (Clk, Rst, Start, zero_flag, opcode, R1, R2, Done);

  FSM_DECO FSM_DECO(opcode, R1, R2, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode);

  endmodule

- module FIBO_FSM_all(Clk, Rst, Start, Done); //the whole design with the datapath
  parameter size = 3 ;
  input Clk, Rst, Start;
  output Done;
  wire [size:0] data, count;
  wire [size-1:0] alu_opcode;
  wire [size-2:0] wrt_addr, rd_addr1, rd_addr2;
  wire zero_flag, wrt_en, load_data;

  //Connecting memory - single register on the left
  Memory memory (Clk, data, count);

  //Connecting FIBO_FSM

Fibo_FSM Fibo_FSM(Clk, Rst, Start, zero_flag, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode, Done);

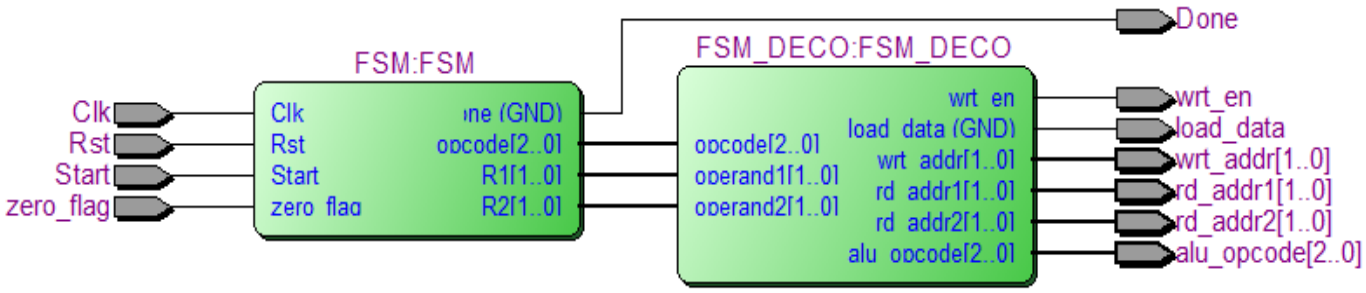
//Connecting datapath

Fibo_Datapath Datapath(Clk, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode, count, data, zero_flag);

endmodule

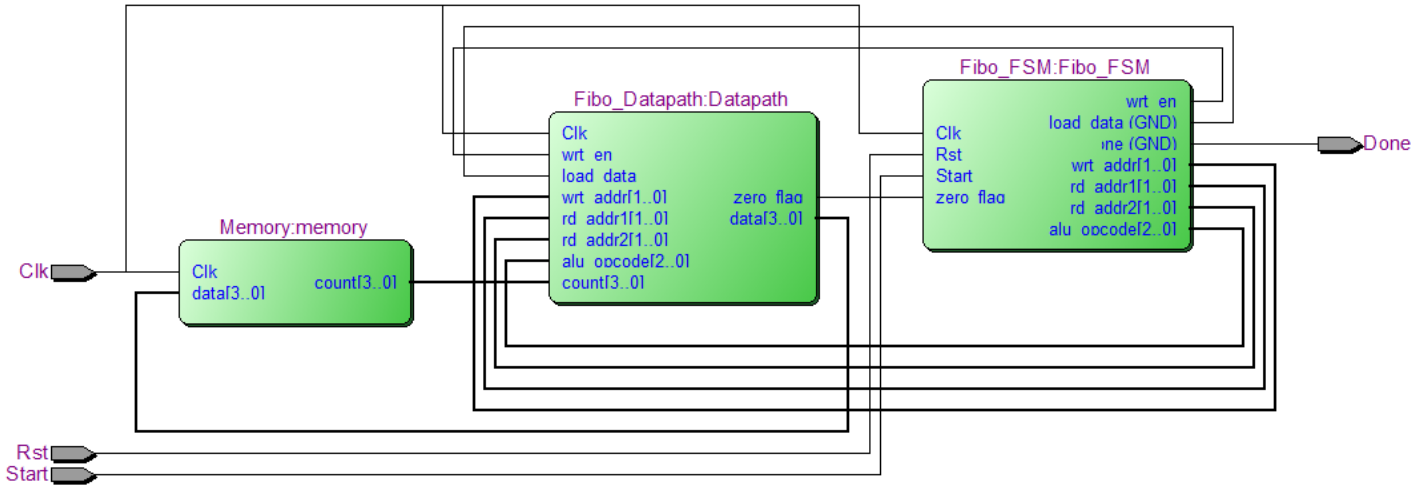
The schematic for the FIBO_FSM:

The schematic for the FIBO_FSM (combined FSM and FSM_DECO) is shown below has 4 external inputs which are Clock, Reset, Start and zero_flag, and also 7 outputs. One of those outputs, Done, checks if the process is over or not (according to the count value coming from the memory - datapath).



The schematic for the whole design (memory, FSM and datapath):

The schematic for the whole design includes a single memory (D-FF), an FSM (FIBO_FSM), and a datapath. They are synchronized with a clock.



6.6.3 TESTBENCH

- module Fibo_FSM_tb (); //testbench for the FIBO_FSM
  parameter size = 3 ;
  reg Clk, Rst, Start, zero_flag;
  wire [size-1:0] alu_opcode;
  wire [size-2:0] R1, R2, wrt_addr, rd_addr1, rd_addr2;
  wire wrt_en, load_data;
  wire Done;

  Fibo_FSM Fibonacci_Seq(Clk, Rst, Start, zero_flag, wrt_addr, wrt_en, load_data, rd_addr1, rd_addr2, alu_opcode, Done);
  always #10 Clk = ~Clk;

  initial
  begin

  Clk=0; Rst=1; Start=0;
  #10 Rst=0; Start=0; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;
  #10 Rst=0; Start=1; zero_flag=0;

  end
  endmodule

- module FIBO_FSM_all_tb (); //the testbench for the whole design(FSM and datapath)
  reg Clk, Rst, Start;
  wire Done;

  FIBO_FSM_all Fibo_Design (Clk, Rst, Start, Done);

  always #5 Clk = ~Clk;

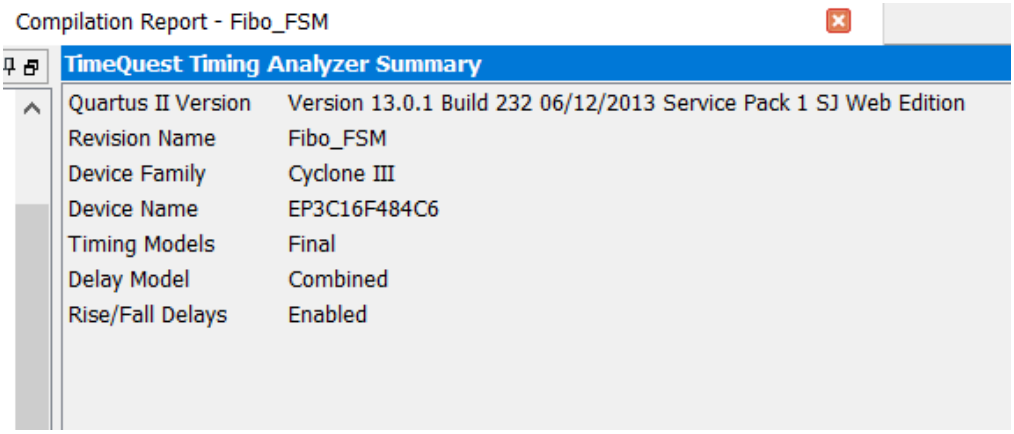
  initial
  begin

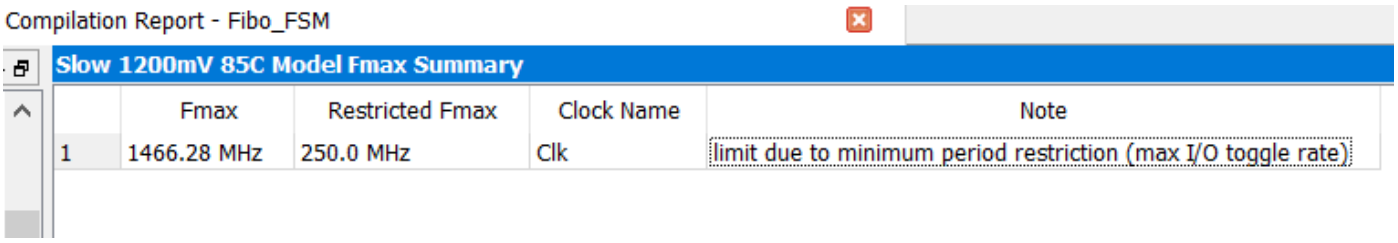
  Clk=0; Rst=1; Start=0;
  #10 Rst=0; Start=0;
  #10 Rst=0; Start=1;
  #10 Rst=0; Start=1;

```
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;
        #10 Rst=0; Start=1;

        end
endmodule
```

6.6.4 TIMING CONSTRAINTS





Conclusion:

To conclude, the circuit has an FSM which is a Moore Machine, and a decoder in its FIBO_FSM unit. Also has a datapath, which includes both sequential and combinational circuits in it. The design is synchronized with a clock meaning that the states change at the rising edge of the clock (in each clock change). Also, the fmax value for the design can be seen as 1466.28MHz from the timing analysis summary.