МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра Вычислительной техники

Отчет по лабораторной работе №2

по дисциплине

«Проектирование и разработка систем на базе ПЛИС»

Тема работы:

«Проектирование синтезируемой модели конечного автомата и её верификация средствами САПР Xilinx ISE 14.x.»

Выполнили: студенты группы ИВБО-02-19

К. Ю. Денисов

Принял: ассистент

А. С. Боронников

Москва 2021

Содержание

1 Ход работы

1.1 Постановка задачи

Требуется описать конечный автомат, представляющий собой генератор фиксированной последовательности логических сигналов, в виде синтезируемой модели на языке Verilog HDL.

Автомат должен иметь интерфейс, представленный на рис 1.

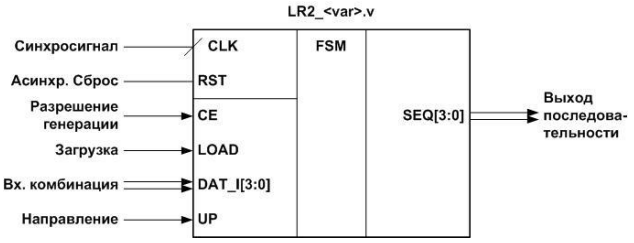

Рис. 1: Интерфейс цифрового автомата

Автомат является синхронным цифровым узлом, срабатывающим по восходящим фронтам синхросигнала *CLK*. Исключение составляет асинхронный вход сброса *RST*, принудительно устанавливающий регистр автомата в исходное состояние (определяется вариантом). Автомат должен реагировать на входные воздействия согласно таблице 1.

RST	CLK	LOAD	CE	UP	Действие
1	X	X	X	X	Асинхронный сброс SEQ <= Func(4'h0)
0	posedge	1	X	X	Загрузка SEQ <= Func(DAT_I)
0	posedge	0	1	0	Обратная генерация SEQ <= Func(i-1)
0	posedge	0	1	1	Прямая генерация SEQ <= Func(i+1)
0	posedge	0		X	Хранение SEQ <= SEQ

Таблица 1: Таблица функционирования автомата

Последовательность генерируемых сигналов определяется функцией Func(i), где i — 4-разрядный двоичный индекс, представляющий собой номер элемента последовательности.

Инкремент индекса соответствует прямой генерации последовательности. Декремент индекса соответствует обратной генерации последовательности.

Последовательность для каждого варианта выполнения работы определяется из таблицы вариантов следующим образом: индекс *i* задан входными

комбинациями от *F* до 0 в верхней строке таблицы, а выходные комбинации *Func(i)*, формируемые на выходах *SEQ[3:0]*, заданы строкой таблицы, соответствующей выбранному варианту. Допускается использовать различные варианты кодировки состояний автомата. Автомат может иметь организацию согласно абстрактным моделям Мили или Мура.

1.2 Индивидуальный вариант 149

Требуется описать конечный автомат, представляющий собой генератор фиксированной последовательности логических сигналов, в виде синтезируемой модели на языке Verilog HDL согласно данной таблице истинности и вектор-функции (см. таблицу 2).

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	4	4	8	3	0	7	2	2	D	7	C	5	2	A	C

Таблица 2: Вектор-функция

1.3 Структурная схема автомат

Построим структурную схему цифрового устройства. Используем делитель частоты для снижения частоты тактового генератора, фильтр дребезга для использования кнопок в качестве устройств ввода. См. рис. 2.

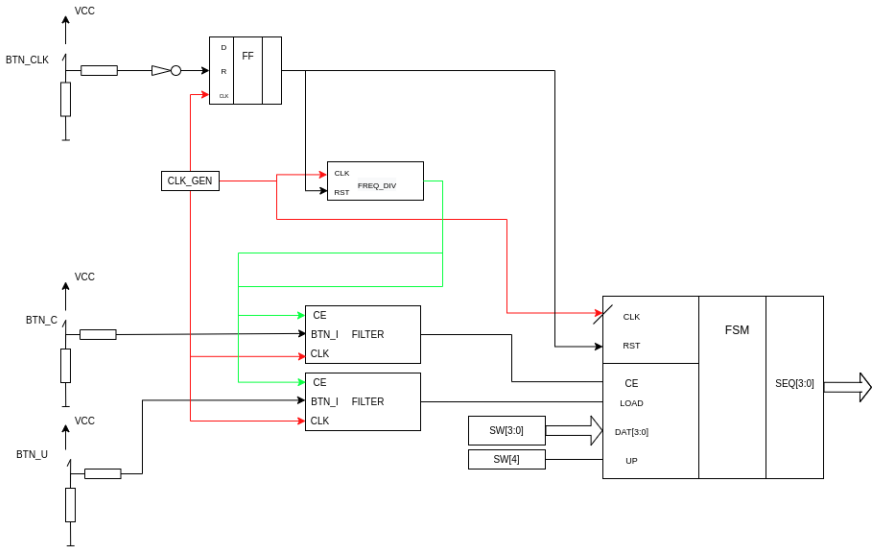

Рис. 2: Структурная схема устройства

1.4 Кодировка состояний автомата в двоичной и шестнадцатиричной системах

Опишем модуль behaviour.v, указав в нем состояния автомата, приведенные в шестнадцатеричной системе.

/home/denilai/Documents/repos/latex/scripts/behaviour.v

```
`timescale 1ns / 1ps

module beheviour(
  input[3:0]  X,
  output reg [3:0] Y
    );
   always@(X)
    case(X)
      4'h0: Y<=4'hc;
      4'h1: Y<=4'ha;
      4'h2: Y<=4'h2;
      4'h3: Y<=4'h5;
      4'h4: Y<=4'hc;
      4'h5: Y<=4'h7;
      4'h6: Y<=4'hd;
      4'h7: Y<=4'h2;
      4'h8: Y<=4'h2;
      4'h9: Y<=4'h7;
      4'ha: Y<=4'h0;
      4'hb: Y<=4'h3;
      4'hc: Y<=4'h8;
      4'hd: Y<=4'h4;
      4'he: Y<=4'h4;
      4'hf: Y<=4'h0;
    default: Y<=4'h0;

  endcase
endmodule
```

1.5 Граф состояний

Опишем граф перехода цифрового автомата согласно указанным режимам работы (переход в следующее или предыдущее состояние, загрузка состояния, хранение, сброс). См рис. 3.

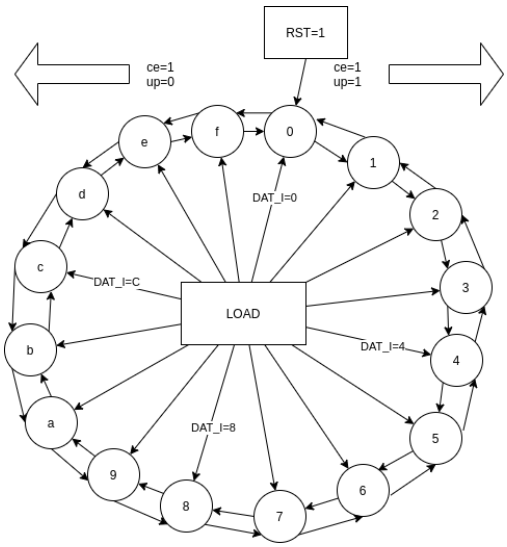

Рис. 3: Граф переходов

1.6 Создание проекта САПР Xilinx ISE

Приведем содержание verilog-модуля, описывающего цифровой автомат.

/home/denilai/Documents/repos/latex/scripts/fsm.v

```
`timescale 1ns / 1ps
//
module fsm(input        rst,
           input        clk,
           input        ce,
           input        load,
           input        up,
           input  [3:0] data,

           output [3:0] seq);

reg [3:0] state;

behaviour beh (.X(state),
               .Y(seq));

always @(posedge clk, posedge rst)
  begin
```

```
    if (rst)
      state <= 4'h0;
    else
      begin
        if (load)
          state <= data;
        if (ce && up)
          state <= state + 4'h1;
        if (ce && !up)
          state <= state - 4'h1;
        if (!ce && !load)
          state <= state;
      end
  end
endmodule
```

Приведем содержание verilog-модуля, описывающего делитель частоты.

/home/denilai/Documents/repos/latex/scripts/freq_div.v

```
`timescale 1ns / 1ps

// freq_div instatination example: freq_div #(10000) (.....)
module freq_div(input       rst,
           input       clk,

           output reg  co);
   reg [16:0] counter;
   parameter divisior = 17'd10000;

   always @(posedge clk, posedge rst) begin
    if (rst)
      begin
        counter <= 0;
        co <= 0;
      end
    else
        if (counter >= (divisior - 17'b1))
          begin
            counter <= 17'b0;
            co <= 1;
          end
        else
        begin
          co <= 0;
          counter <= counter + 17'b1;
        end
  end
endmodule
```

Приведем содержание verilog-модуля, описывающего фильтр дребезга.

/home/denilai/Documents/repos/latex/scripts/m_btn_filter.v

```
`timescale 1ns / 1ps
module M_BTN_FILTER_V10(

input       CLK,
input       CE,
input       BTN_IN,
input       RST,

//output      BTN_OUT,
output reg  BTN_CEO
  );
  parameter [3:0] CNTR_WIDTH = 4; // Internal Counter Width
  // Internal signals declaration:

  reg [CNTR_WIDTH - 1:0] FLTR_CNT;
  reg BTN_D, BTN_S1, BTN_S2;


  //----------------------------------------------// Main Counter:
  always @ (posedge CLK, posedge RST)
  if(RST)
    FLTR_CNT <= {CNTR_WIDTH{1'b0}};
  else
  if(!(BTN_S1 ^ BTN_S2)) // if BTN_S1 = BTN_S2
    FLTR_CNT <= {CNTR_WIDTH{1'b0}}; // Return to Zero
  else if(CE) // else if Clock Enable
    FLTR_CNT <= FLTR_CNT + 1; // Increment
  //----------------------------------------------// Input Synchronizer:
  always @ (posedge CLK, posedge RST)
    if(RST)
      begin
        BTN_D <= 1'b0;
        BTN_S1 <= 1'b0;
      end
    else
      begin
        BTN_D <= BTN_IN;
        BTN_S1 <= BTN_D;
      end
  //----------------------------------------------// Output Register:
  always @ (posedge CLK, posedge RST)
    if(RST)
      BTN_S2 <= 1'b0;
    else if(&(FLTR_CNT) & CE)
      BTN_S2 <= BTN_S1;
```

```
  //----------------------------------------------// Output Front Detector Clock
    Enable:
47  always @ (posedge CLK, posedge RST)
      if(RST)
49      BTN_CEO <= 1'b0;
      else
51      BTN_CEO <= &(FLTR_CNT) & CE & BTN_S1;
endmodule
53//----------------------------------------------assign BTN_OUT = BTN_S2;
//----------------------------------------------
```

Приведем содержание verilog-модуля, описывающего тестовое окружение, описывающее входные воздействия для данной модели.

/home/denilai/Documents/repos/latex/scripts/test–fsm.v

```
`timescale 1ns / 1ps
2
module test_fsm;
4
  // Inputs
6 reg       rst;
  reg       clk;
8 reg       ce;
  reg       load;
10 reg      up;
  reg [3:0] data;
12
  // Outputs
14 wire [3:0] seq;

16 // Instantiate the Unit Under Test (UUT)
  fsm uut (
18   .rst(rst),
    .clk(clk),
20   .ce(ce),
    .load(load),
22   .up(up),
    .data(data),
24   .seq(seq)
  );
26 always
    #5 clk=~clk;
28
  initial begin
30   // Initialize Inputs
    rst = 1;
32   clk = 0;
    ce = 0;
```

```
34      load = 0;
        up = 0;
36      data = 0;

38      // Wait 100 ns for global reset to finish
        #100;
40
        // count forward
42
        rst=0;
44      ce=1;
        up=1;
46      #165;

48      //count bashward

50      rst=0;
        ce=1;
52      up=0;
        #180;
54
        //one state (store mode)
56
        rst=0;
58      ce=0;
        up=0;
60      #100;


62
        rst=0;
64      ce=0;
        up=0;
66      load=1;

68      //load mode
        data=4'h0;
70      #20;
        data=4'h1;
72      #20;
        data=4'h2;
74      #20;
        data=4'h3;
76      #20;
        data=4'h4;
78      #20;
        data=4'h5;
80      #20;
        data=4'h6;
```

```
82      #20;
        data=4'h7;
84      #20;
        data=4'h8;
86      #20;
        data=4'h9;
88      #20;
        data=4'ha;
90      #20;
        data=4'hb;
92      #20;
        data=4'hc;
94      #20;
        data=4'hd;
96      #20;
        data=4'he;
98      #20;
        data=4'hf;
100     //#20;

102     $stop;

104     // Add stimulus here

106   end
endmodule
```

Приведем содержание verilog-модуля верхнего уровня

/home/denilai/Documents/repos/latex/scripts/top2.v

```
`timescale 1ns / 1ps
2
module top(
4   input           clk,
    input  [4:0]    SW,
6   input           CPU_RESET,
    input           BTNC,
8   input           BTNU,
    output [3:0]    LED
10    );


12
    reg             RST_I;
14  wire            RST;
    wire            CO;
16  wire            BTNC_CEO;
    wire            BTNU_CEO;
18
```

```
    always @ (posedge clk, negedge CPU_RESET)
      begin
       if(~CPU_RESET)
         RST_I <= 1'b1;
       else
         RST_I <= 1'b0;
     end
    assign RST=RST_I;
    freq_div #(10000) FREQ_CO (
    .rst (RST),
    .clk (clk),
    .co  (CO)
    );

    M_BTN_FILTER_V10 b_f_u(
    .CLK      (clk),
     .CE       (CO),
     .BTN_IN   (BTNU),
     .RST      (RST_I),
    .BTN_CEO  (BTNU_CEO)
    );

    M_BTN_FILTER_V10 b_f_c(
    .CLK      (clk),
     .CE       (co),
     .BTN_IN   (BTNC),
     .RST      (RST_I),
    .BTN_CEO  (BTNC_CEO)
    );


    fsm FSM1(
    .rst    (RST_I),
    .clk    (clk),
     .ce     (BTNC_CEO),
     .load   (BTNU_CEO),
     .up     (SW[4]),
     .data   (SW[3:0]),
    .seq    (LED)
    );
endmodule
```

1.7 Тестирование и отладка средствами симулятора iSim

После компоновки проекта, подключения модуля верхнего уровня, проведем верификацию спроектированных моделей с помощью симулятора iSim

из состава САПР Xilinx ISE Design Suite. Результаты тестирования можно видеть на рис. 4 и 5.

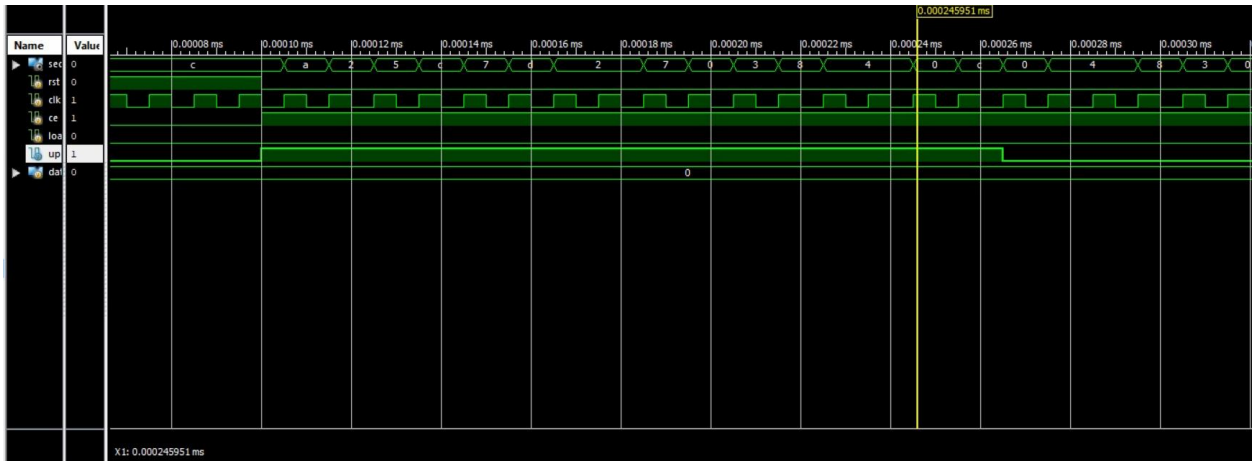

Рис. 4: Вывод iSim

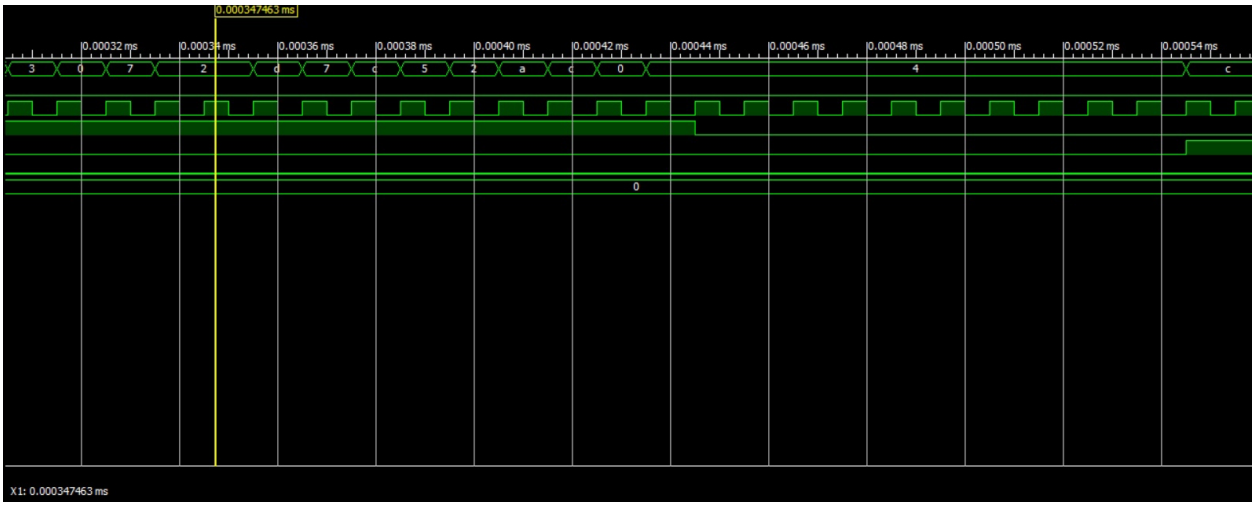

Рис. 5: Вывод iSim

Приведем структуру проекта. См. рис. 6.

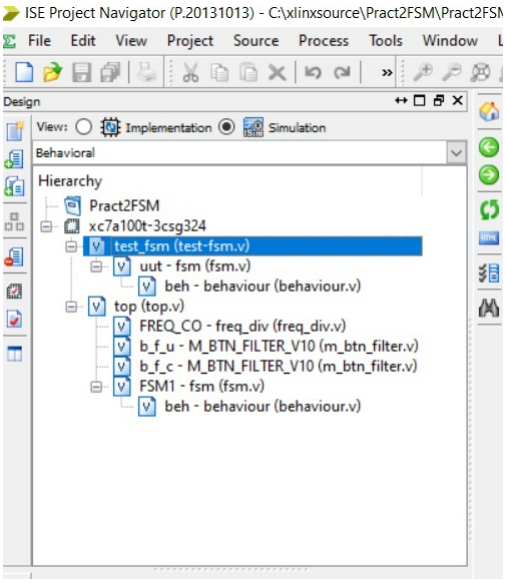

Рис. 6: Иерархия проекта

2 Вывод

В ходе данной практической работы нами были получены общие навыки работы с программным обеспечением Xilinx ISE Design Suite, изучены основы языка Verilog.

С помощью полученных знаний был спроектирован конечный автомат, представляющий собой генератор фиксированной последовательности логических сигналов, в виде синтезируемой модели.