Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
Институт №3 «Системы управления, информатика и электроэнергетика»
Кафедра 307 «Цифровые технологии и информационные системы»

ОТЧЕТ

О выполнении задания по предмету

«Программирование ПЛИС»

«Секундомер»

Выполнили студенты:

Петров В. А. гр. М3О-312Б-20

Куприянов И. А. гр. М3О-321Б-20

Смехов Г. М. гр. М3О-312Б-20

Елисеев И. О. гр. М3О-321Б-20

Попов Д. С. Гр. М3О-321Б-20

Проверил:

Старший преподаватель каф. 307:

Коробков М.А.

Москва 2023 г.

Цель работы

Разработать программу для устройства с ПЛИС «Altera Cyclone IV E» с процессором EP4CE6E22C8, выполняющую функции секундомера: при нажатии на кнопку 1 на семисегментных индикаторах должен запуститься отсчёт от нуля, при нажатии на кнопку 2 отсчёт должен сбрасываться до нуля.

Блок-схема структуры разработанной программы

Структура программы представлена на блок-схеме ниже:

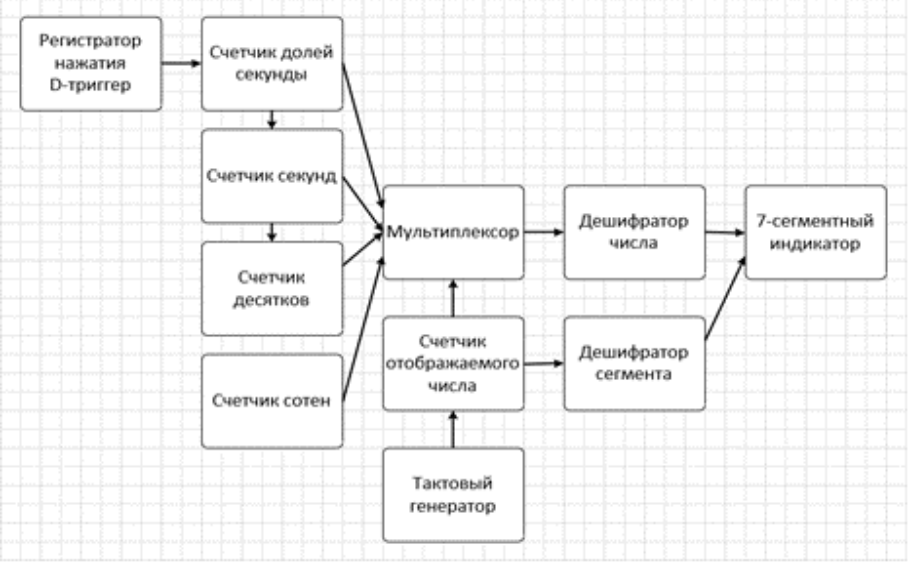

Рисунок 1 - блок-схема, иллюстрирующая работу программы

Теоретическая часть

При выполнении данной лабораторной работы была использована макетная плата RZ-EASYFPGA A2.2 Altera Cyclone IV FPGA, написание кода программы выполняется в среде программирования Quartus Prime, компиляция выполняется в ModelSim.

Выполнение отсчёта времени производится программно, исходя из частоты процессора в 50МГц.

Практическая часть

Был написан код программы, состоящий из следующих модулей:

1. Counter: модуль счетчика, увеличивающий значение на 1 каждый раз при поступлении сигнала тактирования. В программе использован универсальный счётчик, принцип его работы заключен в том, что в его свойствах прописывается число, до которого ему необходимо вести подсчёт.

   module counter # (parameter max = 9)

   (input logic clk, reset,

     output logic [3:0] out);

   где parameter max — до какого числа считает счетчик, clk — сигнал тактирования, reset — сигнал сброса, out — значение счетчика.

2. Prescaler: модуль делителя частоты, уменьшающий частоту тактирования входного сигнала до заданного значения. Так как частота работы процессора равна 50 МГц, это означает, что период тактового сигнала равен 20 нс. Значение prescaler_val может быть изменено, что позволяет установить на выходе необходимый для работы таймера период сигнала.

   module prescaler # (parameter prescaler_val = 50000000)

   (input logic clk, reset,

     output logic clk_out);

   где parameter prescaler_val = 50000000—параметр предделителя, clk — сигнал тактирования, reset — сигнал сброса, clk_out — выходной сигнал тактирования.

3. Multiplexer: модуль мультиплексора, выбирающий один из четырех входных сигналов в зависимости от значения выбора. Входные сигналы подключены к портам in0, in1, in2 и in3. Значение выбора подключено к портам sel0 и sel1. Значение выбора определяет, какой из входных сигналов будет выведен на порт out.

```
module multiplexer # (parameter size = 4)
(       input logic [size-1:0] in0, in1, in2, in3,
        input logic [1:0] select,
        output logic [size-1:0] out);
```

где parameter size — размер входных данных, in0, in1, in2, in3 — вход мультиплексора, select — номер выбранного входа, out—вход мультиплексора.

4. System: основной модуль, объединяющий все остальные модули и реализует логику работы всей системы.

```
module system       (input logic clk, reset,
                     output logic[6:0] data_out,
                     output logic[3:0] digit_out,
                     output logic dp_out);
```

где clk — сигнал тактирования, reset — сигнал сброса, data_out — цифра на сегменте индикатора, digit_out — выбор сегмента индикатора, dp_out — сигнал точки.

5. Decoder_7seg: декодер 7-сегментного дисплея, принимающий два входных сигнала (цифру и значение) и выводит соответствующее значение на дисплей.

```
module decoder_7seg (input logic[3:0] data_in,
                     input logic[1:0] digit_in,
                     input logic dp_in,
                     output logic[6:0] data_out,
                     output logic[3:0] digit_out,
                     output logic dp_out);
```

где data_in — входные данные времени, которое необходимо вывести на соответствующий индикатор, digit_in — входные данные сегмента индикатора, на которое необходимо вывести число, dp_in—входной сигнал о необходимости отображения точки, data_out — выходные данные

времени, которое необходимо вывести на соответствующий индикатор, digit_out — выходные данные сегмента индикатора, на которое необходимо вывести число, dp_out—входной сигнал о необходимости отображения точки.

Тестирование

В начале необходимо протестировать модуль Prescaler, с целью определить правильность частоты тактирования для счетчиков. Для примера взят модуль Prescaler с параметром 50000 (после 50000 тактовых импульсов должен произойти фронт уровня выходного импульса, еще через 50000 таковых импульсов спад) Тактовая частота ПЛИС 50 МГц, значит период тактового сигнала 0,2 нс. Соответственно первый фронт выходного импульса должен быть через 5000 нс, а первый спад через 10000 нс.

Ниже представлен код теста на языке SystemVerilog:

```
module Testbench();
        logic clk =0, reset=1;
        logic clk_out_prescaler;
        prescaler #(50000) prescaler_0 (clk, reset, clk_out_prescaler);

        always    // no sensitivity list, so it always executes
          begin
            clk = 0; #50; clk = 1; #50;
          end

        initial
        begin
        reset=0;
    end
endmodule
```

Выполнено тестирование кода программы с помощью Modelsim, результаты показаны ниже:

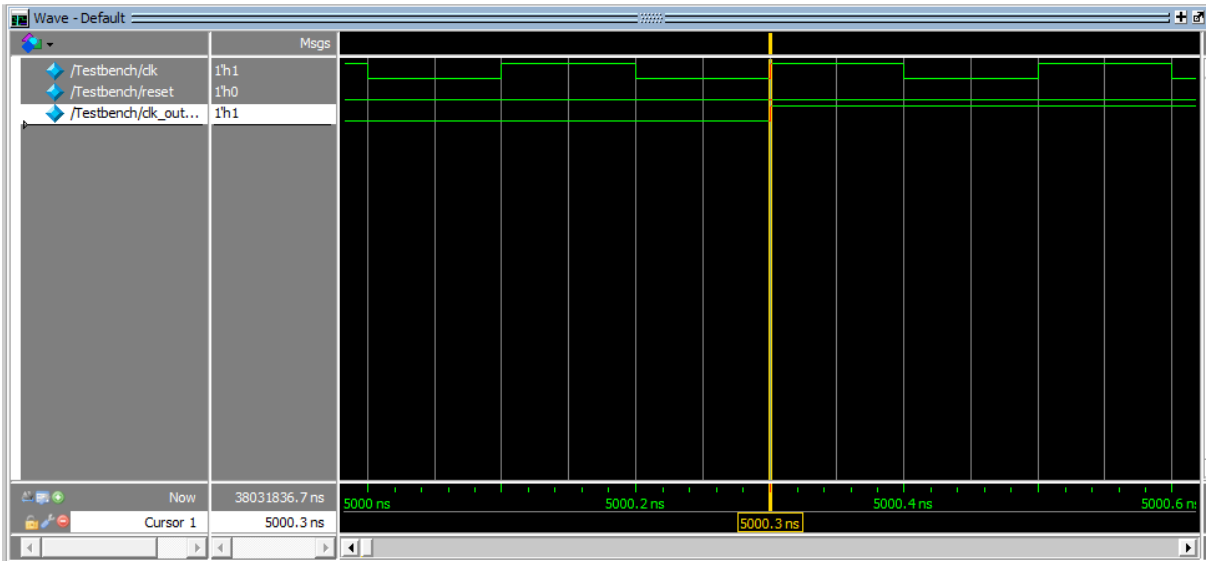

Рисунок 2 - тестирование кода программы в Modelsim

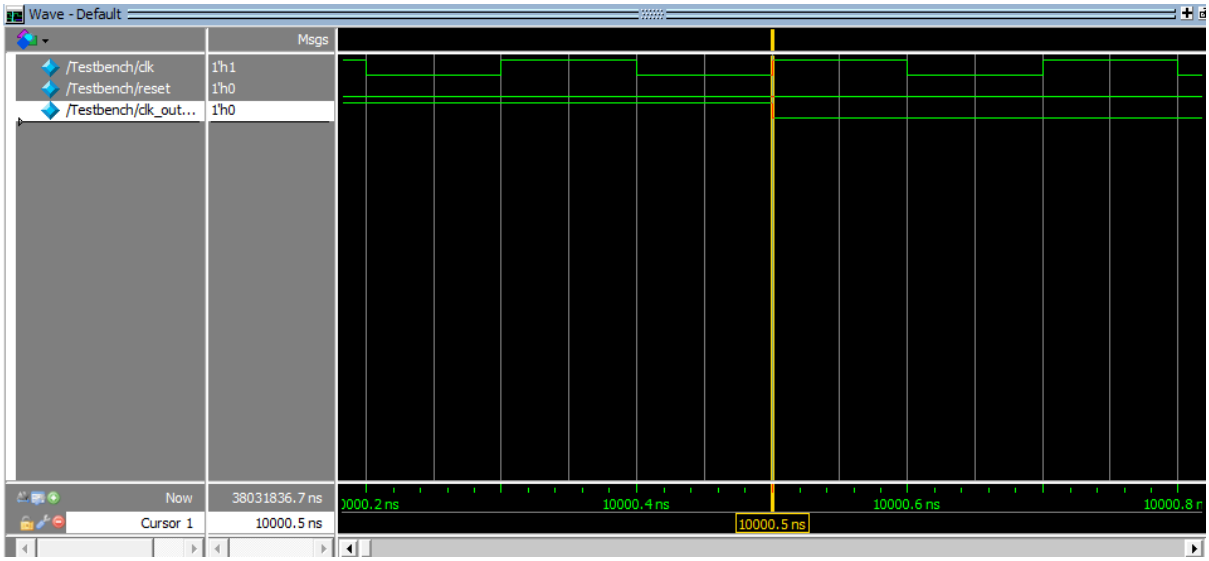

Рисунок 3 - *тестирование кода программы в Modelsim*

Проведено тестирование самого таймера. Для этого на языке SystemVerilog был написан тест для модуля system:

```
    module Testbench();
            logic clk =0, reset=1;
            logic[6:0] data_out;
            logic[3:0] digit_out;
            logic dp_out;

            system           sys           (clk,           reset,data_out,
digit_out,dp_out,clk_out_prescaler);

            always
              begin
                clk = 0; #100; clk = 1; #100;
              end

            initial
            begin
            reset = 0; #200; reset=1;
         end
    endmodule
```

Ожидается, что через каждые 10000 нс будет происходить битовый сдвиг влево переменной `digit_out`, через 1000000 нс после начала работы таймера, каждые 10000 нс будет также изменяться переменная `data_out`. Ожидаемые изменения переменной `digit_out` и `data_out` представлено в таблице ниже:

Таблица 1 – зависимость значений digit_out и data_out от времени

Время с начала запуска, мс	1	1,01	1,02	52	52,01	52,02
Значение `digit_out`	1110	1101	1011	1110	1101	1011
Значение `data_out`	1001111	0000001	0000001	0010010	0100100	0000001

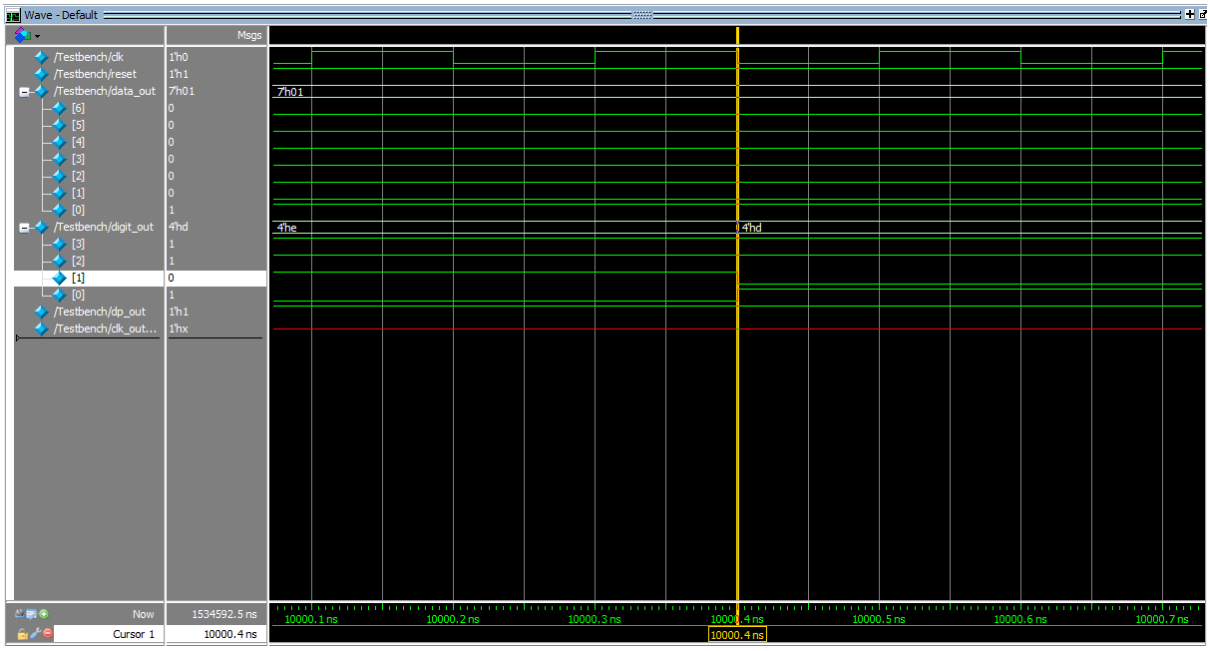

Рисунок 4 - - тестирование кода программы в Modelsim

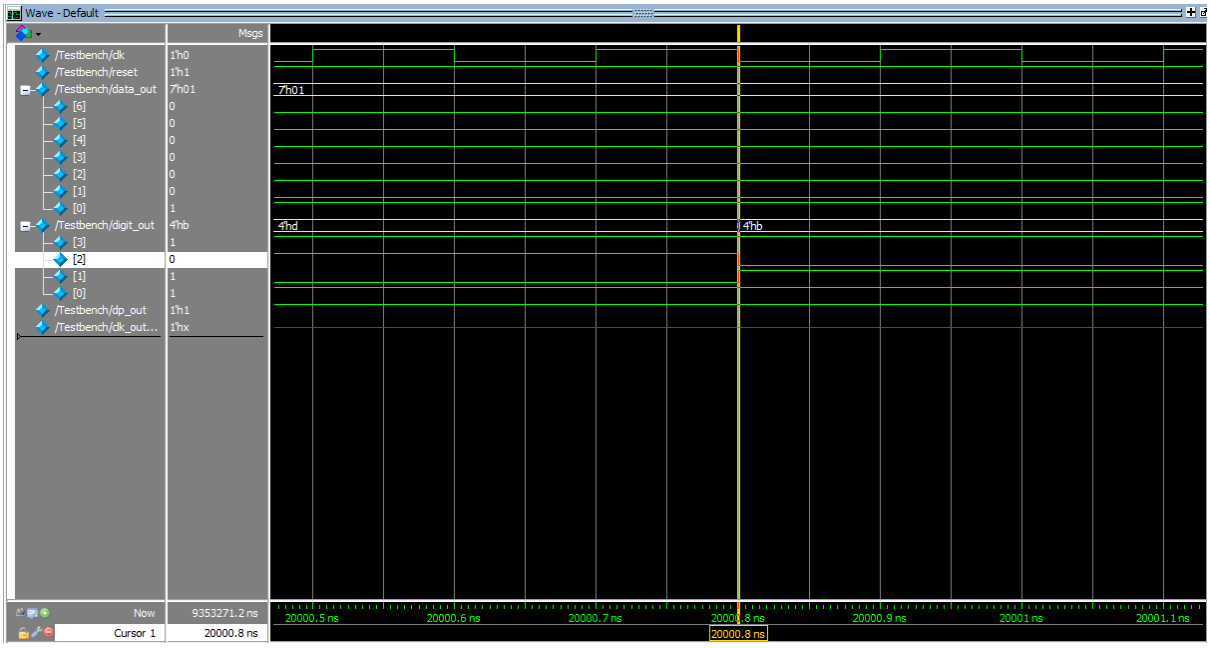

Рисунок 5 -- тестирование кода программы в Modelsim

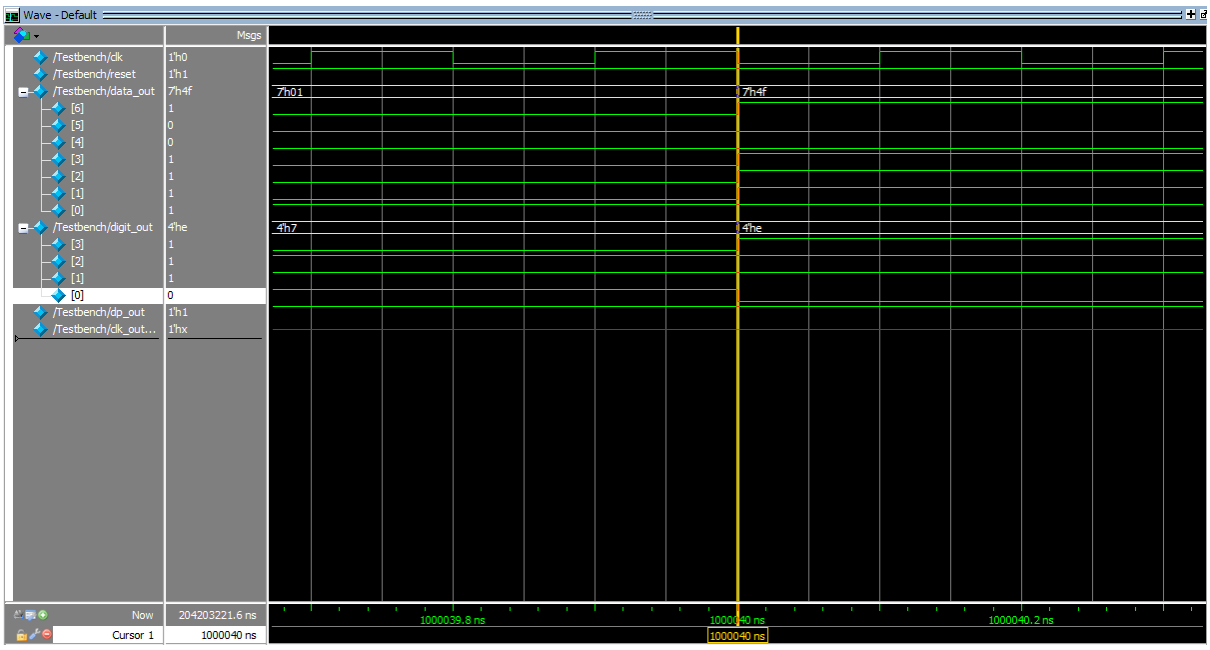

Рисунок 6 - тестирование кода программы в Modelsim

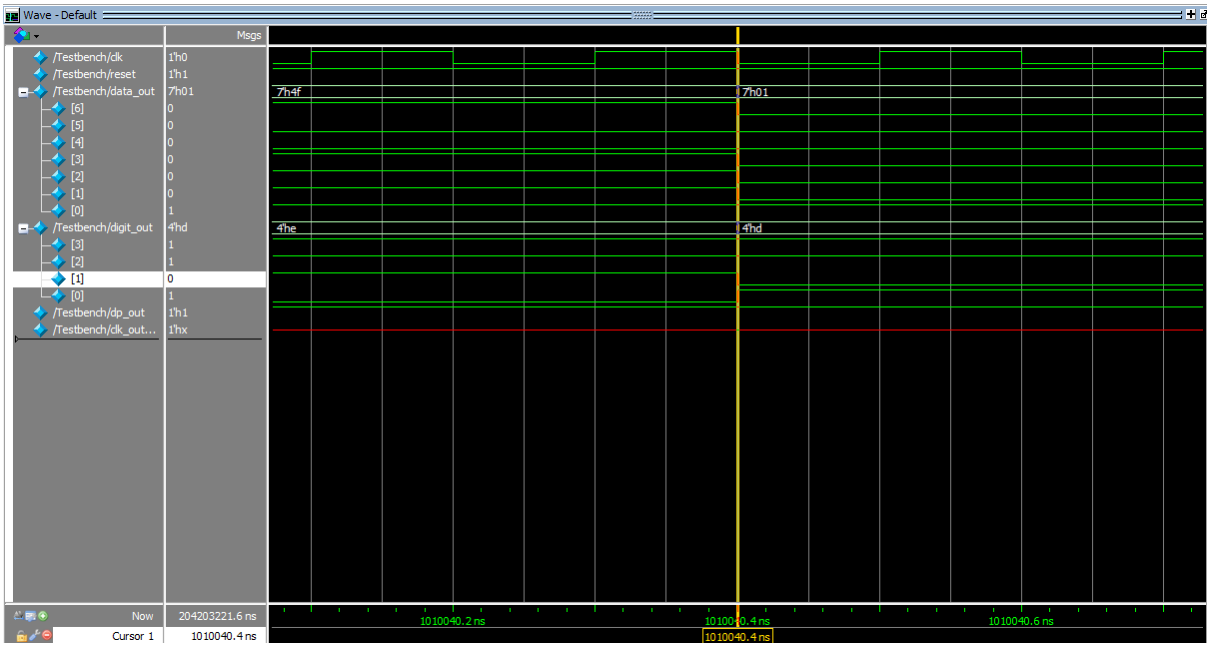

Рисунок 7 - тестирование кода программы в Modelsim

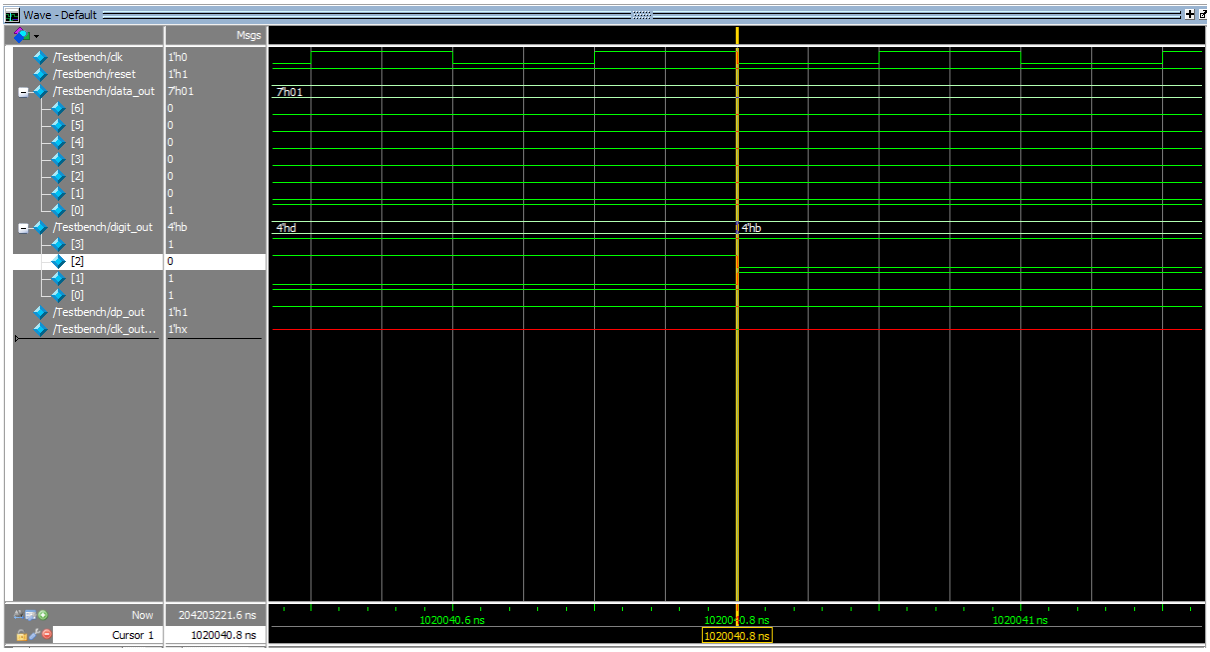

Рисунок 8 - тестирование кода программы в Modelsim

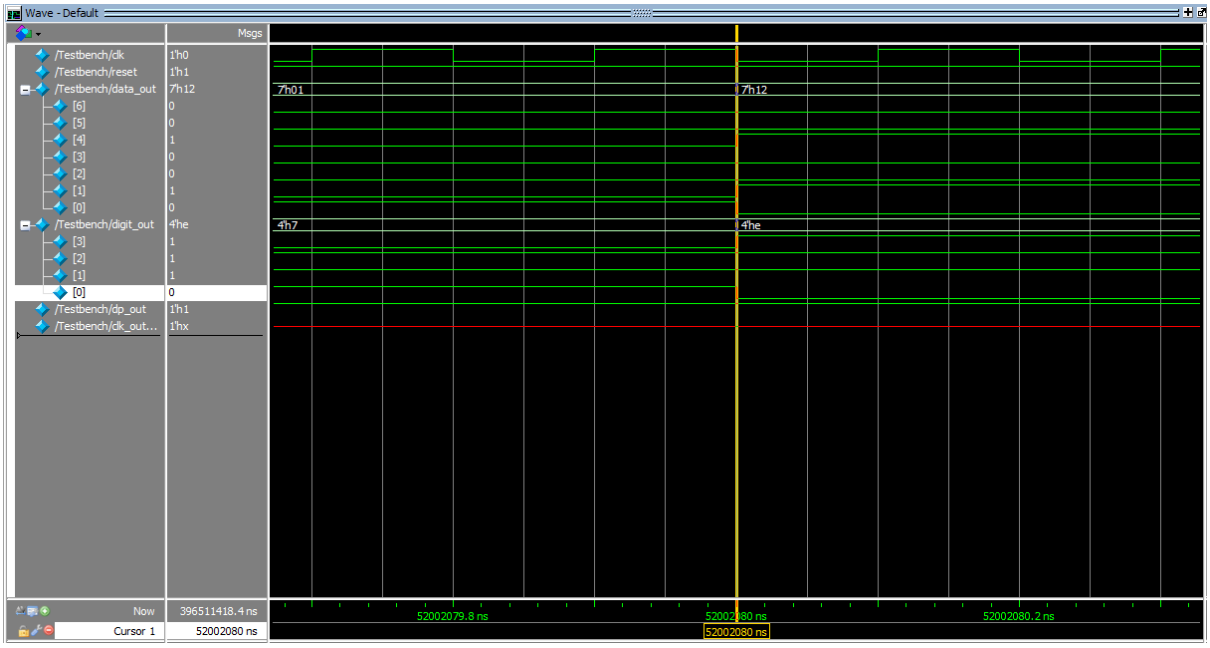

Рисунок 9 - тестирование кода программы в Modelsim

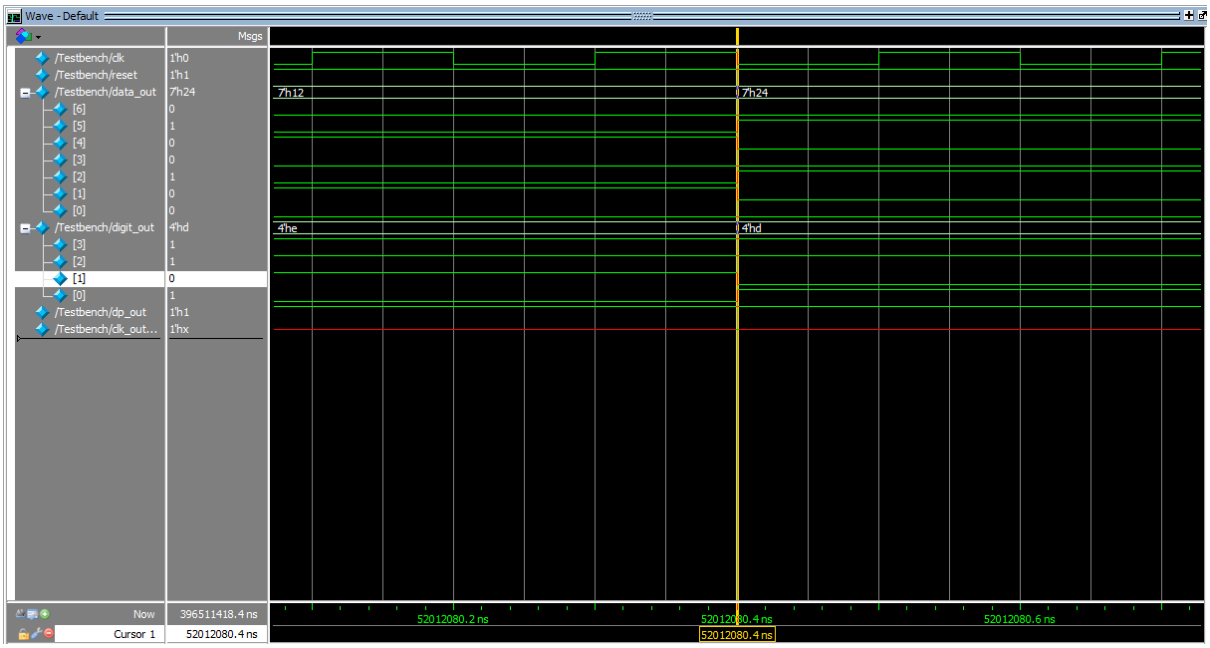

Рисунок 10 - тестирование кода программы в Modelsim

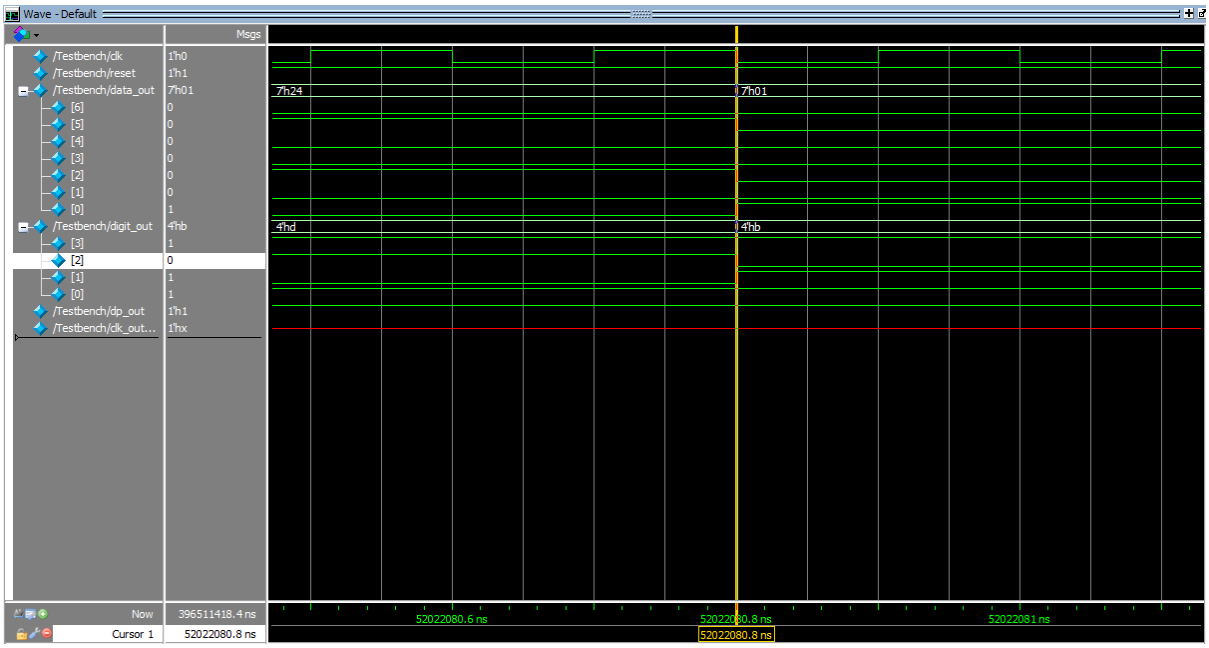

Рисунок 11 - тестирование кода программы в Modelsim

Фото готового устройства

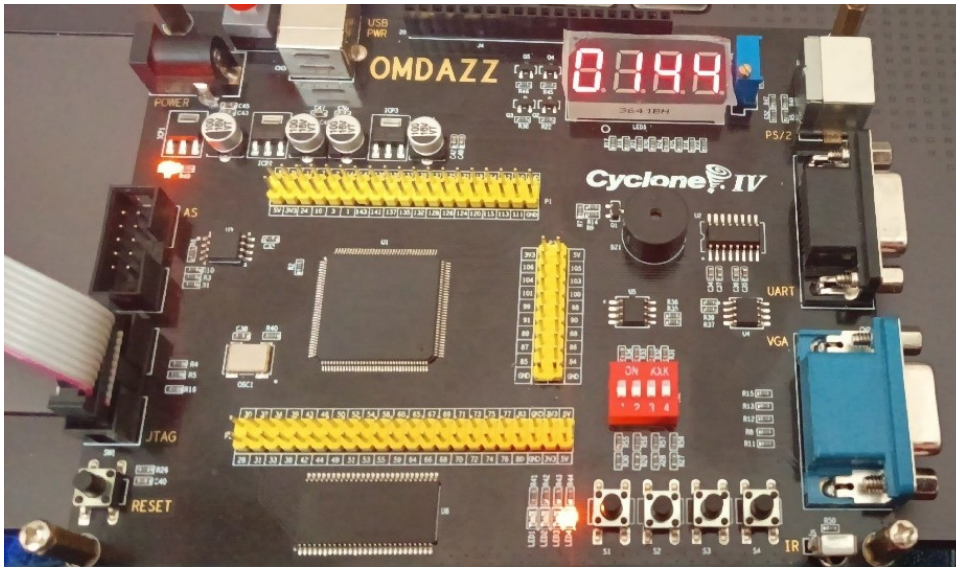

Рисунок 12 - фото устройства

Вывод

В результате выполнения лабораторной работы была написана программа для ПЛИС «Altera Cyclone IV», выполняющая функции секундомера. При нажатии на кнопку 1 на семисегментных индикаторах запускается отсчёт от нуля, при нажатии на кнопку 2 отсчёт сбрасывается до нуля.

К недостаткам получившейся программы можно отнести тот факт, что используемые счётчики являются асинхронными и отсутствие связи между ними может привести к ошибке в отсчёте времени при неправильной настройке тактирования (порядок переключения миллисекунд, секунд и десятков секунд может быть нестабильным).

Приложение

Код программы:

```
module counter
# (parameter max = 9)
( input logic clk, reset,
                                output logic [3:0] out);

always_ff @(posedge clk)
begin
  if (reset)
        out <= 0;
  else
  begin
        if (out == max)
              out <= 0;
        else
              out <= out + 1;
  end
end
endmodule



module prescaler
# (parameter prescaler_val = 50000000)
( input logic clk, reset,
  output logic clk_out);

  logic [31:0] counter_val;
  logic [31:0] prescaler_div;
  always_ff @(posedge clk)
  begin
        if (reset)
        begin
              clk_out <= 0;
              counter_val <= 0;
              prescaler_div <= prescaler_val >> 1;
        end
        else
        begin
              if (counter_val == prescaler_div)
              begin
                    clk_out <= clk_out ^ 1;
                    counter_val <= 0;
              end
              else
                    counter_val <= counter_val + 1;
        end
  end
endmodule

module multiplexer
# (parameter size = 4)
 (      input logic [size-1:0] in0, in1, in2, in3,
  input logic [1:0] select,
  output logic [size-1:0] out);
```

```
      always_comb
      begin
      case (select)
      0: out = in0;
      1: out = in1;
      2: out = in2;
      3: out = in3;
      default: out = in0;
      endcase
      end
    endmodule

    module system (input logic clk, reset,
                            output logic[6:0] data_out,
                            output logic[3:0] digit_out,
                            output logic dp_out,
                            output logic clk_out_prescaler);
      logic[3:0] data;
      logic reset_out;
      logic clk_out;
      assign reset_out = ~reset;
      assign clk_out = ~clk;

    prescaler      #(50000)      prescaler_0      (clk_out,      reset_out,
clk_out_prescaler0);

      prescaler     #(5000000)     prescaler_1     (clk_out,     reset_out,
clk_out_prescaler1);

      prescaler     #(50000000)     prescaler_2     (clk_out,     reset_out,
clk_out_prescaler2);

      prescaler    #(500000000)    prescaler_3    (clk_out,    reset_out,
clk_out_prescaler3);

      prescaler   #(64'd5000000000)   prescaler_4   (clk_out,   reset_out,
clk_out_prescaler4);

      counter #(9) cnt_91(clk_out_prescaler1, reset_out, data1);

      counter #(9) cnt_92(clk_out_prescaler2, reset_out, data2);

      counter #(5) cnt_93(clk_out_prescaler3, reset_out, data3);

      counter #(9) cnt_94(clk_out_prescaler4, reset_out, data4);

      logic [1:0] digit;

      counter #(3) cnt_3(clk_out_prescaler0, reset_out, digit);

      multiplexer #(4) multi(data1, data2, data3, data4, digit, data);

      decoder_7seg dec (data, digit, 0, data_out, digit_out, dp_out);

    endmodule

    module decoder_7seg (input logic[3:0] data_in,
                                        input logic[1:0] digit_in,
```

```
                                        input logic dp_in,
                                        output logic[6:0] data_out,
                                        output logic[3:0] digit_out,
                                        output logic dp_out);
  always_comb
  begin
        dp_out = ~dp_in;
        case(data_in)
              4'h0: data_out = 7'b0000001 ;
              4'h1: data_out = 7'b1001111 ;
              4'h2: data_out = 7'b0010010 ;
              4'h3: data_out = 7'b0000110 ;
              4'h4: data_out = 7'b1001100 ;
              4'h5: data_out = 7'b0100100 ;
              4'h6: data_out = 7'b0100000 ;
              4'h7: data_out = 7'b0001111 ;
              4'h8: data_out = 7'b0000000 ;
              4'h9: data_out = 7'b0001100 ;
              4'ha: data_out = 7'b0001000 ;
              4'hb: data_out = 7'b1100000 ;
              4'hc: data_out = 7'b0110001 ;
              4'hd: data_out = 7'b1000010 ;
              4'he: data_out = 7'b0110000 ;
              4'hf: data_out = 7'b0111000 ;
              default: data_out = 7'b1111110 ;
        endcase
        case(digit_in)
              2'h0: digit_out = 4'b1110 ;
              2'h1: digit_out = 4'b1101 ;
              2'h2: digit_out = 4'b1011 ;
              2'h3: digit_out = 4'b0111 ;
              default: digit_out = 4'b0000 ;
        endcase
  end

endmodule
```