Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Вычислитель SHA-256»

Студент __ИУ6-62Б_____ _____ ___С.В. Астахов_____
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ ___Т.А._Ким_____
(Подпись, дата) (И.О. Фамилия)

2022 г.

ЗАДАНИЕ

Реферат

Записка 33 стр., 11 рис., 6 источников, 5 прил.

СХЕМОТЕХНИКА, ЭВМ, SHA-256, ХЕШ-ФУНКЦИЯ, ПЛИС, FPGA, ВЫЧИСЛИТЕЛЬ, VERILOG, XILINX.

Объектом разработки является устройство, производящее расчет внутреннего цикла алгоритма хеширования SHA-256.

Цель работы – эскизный проект цифрового устройства ограниченной сложности.

При проектировании решены следующие задачи: анализ объекта разработки на функциональном уровне, разработка функциональной схемы устройства, выбор ПЛИС для реализации устройства, описание устройства на языке Verilog, синтез RTL-схемы устройства, разработка принципиальной схемы обвязки ПЛИС, расчет электрических параметров.

Результатом разработки является проект на языке Verilog, предназначенный для загрузки в ПЛИС, и набор конструкторской документации.

Содержание

Определения, обозначения, сокращения

ТЗ — техническое задание.

САПР — система автоматизированного проектирования.

ПЛИС — программируемая логическая интегральная схема.

ШД — шина данных.

ШУ — шина управления.

Мультиплексирование с разделением по времени — технология аналогового или цифрового мультиплексирования, в котором несколько сигналов или битовых потоков передаются одновременно как подканалы в одном коммуникационном канале.

Язык описания аппаратуры — специализированный компьютерный язык, используемый для описания структуры и поведения электронных схем, чаще всего цифровых логических схем.

RTL-схема — способ разработки (описания) синхронных цифровых интегральных схем, при применении которого работа схемы описывается в виде последовательностей логических операций, применяемых к цифровым сигналам (данным) при их передаче от одного регистра к другому (не описывается, из каких электронных компонентов или из каких логических вентилей состоит схема).

Файл пользовательских ограничений (UCF-файл, User Constraint File) — специализированный файл, используемый Xilinx ISE для задания параметров прошивки и конфигурации портов ввода-вывода ПЛИС.

Testbench — модуль на языке описания аппаратуры, используемый для задания входных сигналов при моделировании работы устройства.

Введение

В данной работе производится разработка проекта устройства, производящего расчет внутреннего цикла алгоритма хеширования SHA-256.

Устройство должно рассчитывать заданную часть алгоритма SHA-256 в соответствии со стандартом [1].

Так как для одной итерации алгоритма необходим большой объем данных, необходимо использовать мультиплексирование с разделение по времени. Также, в силу сложности устройства, целесообразно разрабатывать его, применяя язык описания аппаратуры [2].

Хеш-функции, в том числе SHA-256, применяются главным образом для вычисления контрольных сумм, работы с электронной подписью и построения уникальных идентификаторов для наборов данных. Широкое применение хеш-функций в современных информационных системах обуславливает актуальность разработки [3].

1 Анализ предметной области

Реализуемая данным устройством часть алгоритма SHA-256 может быть представлена в виде псевдокода, приведенного в листинге 1.

Листинг 1 – реализуемый алгоритм

```
k[0..63] :=
[0x428A2F98,
     // <...> еще 62 константы
0xC67178F2]

  // Инициализация вспомогательных переменных:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    f := h5
    g := h6
    h := h7

    // Основной цикл:
    для i от 0 до 63
       Σ0 := (a rotr 2) xor (a rotr 13) xor (a rotr 22)
       Ma := (a and b) xor (a and c) xor (b and c)
       t2 := Σ0 + Ma
       Σ1 := (e rotr 6) xor (e rotr 11) xor (e rotr 25)
       Ch := (e and f) xor ((not e) and g)
       t1 := h + Σ1 + Ch + k[i] + w[i]

        h := g
        g := f
        f := e
        e := d + t1
        d := c
        c := b
        b := a
        a := t1 + t2
```

На основе представленного псевдокода можно заключить, что для аппаратной реализации алгоритма понадобятся:

- вычислительный блок;

- блоки памяти по 8 32-битных регистров для переменных и буферизации вывода;

- блок управления;

- мультиплексирующий блок (для разделения информационных сообщений и значений служебных переменных);

- блок постоянной памяти для хранения констант.

При этом, стоит отметить, что согласно ТЗ, ШД должна иметь разрядность 32, поэтому необходимо использовать мультиплексирование с разделением по времени, чтобы инициализировать и считать все необходимые значения.

В силу большой разрядности переменных и сложности проектируемого устройства, было решено использовать для разработки язык описания аппаратуры. В качестве языка и САПРа для разработки были выбраны Verilog и Xilinx ISE соответственно. Эти средства обладают качественной и подробной документацией, широко распространены и имеют вариант распространения для некоммерческих целей.

2 Разработка электрической функциональной схемы

При анализе предметной области, было заключено, что устройство должно состоять из блока управления, блока памяти переменных, выходного буфера, мультиплексирующего блока и вычислительного блока.

Блок памяти переменных каждые 64 цикла исполнения алгоритма инициализируется извне, в остальных циклах он сохраняет значения полученные в предыдущем цикле. Поэтому данный блок должен иметь два информационных входа для соответствующих целей.

Мультиплексирующий блок в начале каждого раунда шифрования перенаправляет значения служебных переменных с информационного входа схемы в память. После этого данных блок перенаправляет значения информационных сообщений на вход вычислительного блока.

Вычислительный блок принимает на вход значения переменных из блока памяти переменных, значение очередной служебной константы и фрагмент информационного сообщения. Данные блок отвечает непосредственно за расчеты, описанные в стандарте SHA-256.

Выходной буфер принимает значения, полученные в ходе вычислений и передает их на выход устройства, а также в блок памяти переменных.

Блок памяти констант представляет из себя постоянную память, хранящую 64 служебных константы, которые необходимы для расчетов.

Блок управления на основе тактирующего сигнала генерирует необходимые сигналы выборки для блоков памяти, так как из-за большой разрядности все они имеет внутри себя схему выборки и для операций ввода-вывода используют мультиплексирование с разделением по времени.

Разработанная электрическая функциональная схема представлена на рисунке 1.

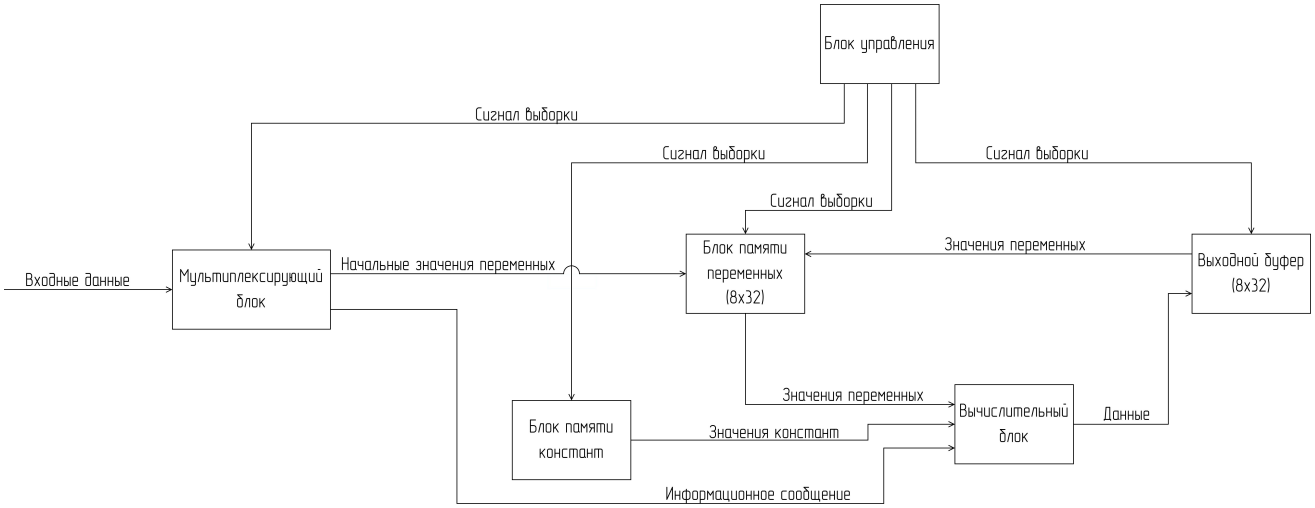


Рисунок 1 – функциональная схема устройства

3 Разработка описания устройства на языке Verilog

3.1 Разработка вычислительного блока

Вычислительный блок непосредственно реализует вычисления, описанные при анализе предметной области. Вычисления представляют собой расчет математических и побитовых логических операций.

Большинство из таких операций реализуется в языке Verilog с помощью операторов, однако модуль, реализующий побитовый сдвиг вправо понадобилось разработать самостоятельно. Так как сдвигать значения в необходимо на различное число бит, этот модуль был параметризован

посредством введения параметра N, содержащего число разрядов, на которые необходимо сдвинуть входное значение. Исходный код, описывающий данный модуль представлен в листинге 2.

Листинг 2 – модуль циклического сдвига вправо

```
`ifndef ROTR
`define ROTR

// OUT = NUM <<< N
module right_cyclic_shift #(parameter N=1)
     (num, out);

     input wire[31:0] num;
     output reg[31:0] out;

     integer i;

     always @* begin
          out = num;
          for(i=0; i<N; i=i+1) begin
               out[31:0] = {out[0], out[31:1]};
          end
     end

endmodule

`endif
```

Далее, с использованием данного модуля, были описаны другие модули, реализующие функции, необходимые для расчета SHA-256. В качестве примера, в листинге 3 приведен код, реализующий расчет функции Сигма-0:

Листинг 3 – модуль расчета функции Сигма-0

```
`include "right_cyclic_shift.v"

// sigma0 := (a rotr 2) xor (a rotr 13) xor (a rotr 22)
module func_sigma0(in_A, func);

     input wire[31:0] in_A;
     output wire[31:0] func;

     wire[31:0] A2, A13, A22;

     right_cyclic_shift #(2)
     A2_node( .out (A2), .num (in_A));

     right_cyclic_shift #(13)
     A13_node( .out (A13), .num (in_A));
```

```
    right_cyclic_shift #(22)
    A22_node( .out (A22), .num (in_A));

    assign func = A2 ^ A13 ^ A22;

endmodule
```

Все подобные модули были собраны в единый модуль вычислительного блока, исходный код которого представлен в листинге 4.

Листинг 4 – исходный код вычислительного блока

```
`include "func_t1.v"
`include "func_t2.v"
module logic_module
(in_A, in_B, in_C, in_D,
in_E, in_F, in_G, in_H,

in_Ki, in_Wi,

out_A, out_B, out_C, out_D,
out_E, out_F, out_G, out_H);

          input wire [31:0] in_A, in_B, in_C,
          in_D, in_E, in_F, in_G, in_H;

          input wire [31:0] in_Ki, in_Wi;

          output [31:0] out_A, out_E;
          output wire [31:0] out_B, out_C, out_D,
               out_F, out_G, out_H;

          wire[31:0] wire_t1, wire_t2;

          assign out_H = in_G;
          assign out_G = in_F;
          assign out_F = in_E;

          assign out_D = in_C;
          assign out_C = in_B;
          assign out_B = in_A;

          func_t1 t1(.in_E(in_E), .in_F(in_F),
```

```
        .in_G(in_G),
        .in_H(in_H), .in_Ki(in_Ki), .in_Wi(in_Wi),
        .func(wire_t1));

        func_t2 t2(.in_A(in_A), .in_B(in_B),
        .in_C(in_C), .func(wire_t2));

        assign out_E = in_D + wire_t1;
        assign out_A = wire_t1 + wire_t2;

endmodule
```

Далее, к блоку вычислений был подключен блок памяти констант. С целью проверки правильности работы разработанных блоков, было произведено моделирование их работы. Результаты моделирования совпали с результатами, полученными при запуске алгоритма SHA-256, реализованного на Python [4].

Полученные временные диаграммы показаны на рисунке 2. Где round_n — порядковый номер цикла алгоритма, in_Wi — фрагмент информационного сообщения, in_A, ..., in_H — входные значения переменных, out_A, ..., out_H — выходные значения переменных (результаты).

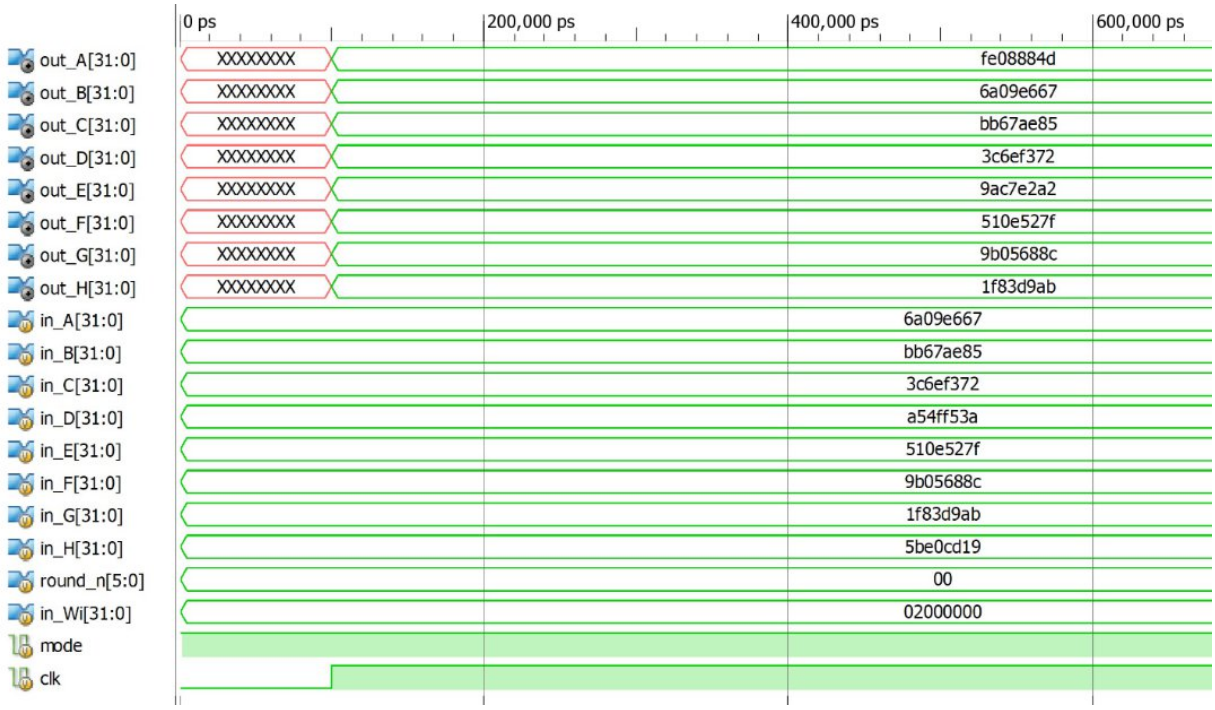


Рисунок 2 – временная диаграмма работы блока вычислений

3.2 Разработка блока памяти переменных

Так как в устройстве, согласно ТЗ, для ввода данных предусматривается 2 шины разрядностью 32 бита, необходимо было разработать блок памяти служебных переменных. Этот блок позволяет мультиплексировать по времени запись в память служебных переменных, для этой цели блок имеет встроенную схему выборки, на которую подается адрес вводимой константы (A - 0x01, B - 0x01, ..., H - 0x08). Поступление адреса не из приведенного выше диапазона (например, адрес 0x00) означает, что необходимо взять значения всех служебных переменных из выходного буфера. Исходный код блока памяти переменных представлен в листинге 5.

Листинг 5 – исходный код блока памяти переменных

```
module mem_controller(
     in_var, addr, clk,
     in_A, in_B, in_C, in_D,
     in_E, in_F, in_G, in_H,
     out_A, out_B, out_C, out_D,
     out_E, out_F, out_G, out_H,
     en
    );

     input wire [3:0] addr;
     input wire [31:0] in_var;
     input wire clk, en;

     input wire [31:0] in_A, in_B, in_C, in_D,
          in_E, in_F, in_G, in_H;
     output reg [31:0] out_A, out_B, out_C, out_D,
          out_E, out_F, out_G, out_H;

     always @(posedge clk) begin
          if (en == 1) begin
               case(addr)
               01: out_A <= in_var;
               02: out_B <= in_var;
               // <...>
               08: out_H <= in_var;
               default: begin
                         out_A <= in_A;
                         out_B <= in_B;
                         // <...>
                         out_H <= in_H;
                    end
               endcase
```

```
        end
    end


endmodule
```

3.3 Разработка блока выходного буфера

Результирующее значение хеш-функции SHA-256 формируется на основе значений служебных переменных. Так как выходные значения переменных необходимо мультиплексировать по времени, а также они нужны для вычисления следующих итераций, был разработан блок выходного буфера. Принцип его работы схож с принципом работы блока памяти служебных переменных, однако в случае выходного буфера, мультиплексируется по времени информационный выход блока, а ввод данных осуществляется параллельно. Исходный код рассматриваемого блока приведен в листинге 6.

Листинг 6 – исходный код выходного буфера

```
module output_buffer(
    out_var, addr, clk, en,
    in_A, in_B, in_C, in_D,
    in_E, in_F, in_G, in_H,
    out_A, out_B, out_C, out_D,
    out_E, out_F, out_G, out_H
   );

    input wire [3:0] addr;
    input wire clk;
    input wire en;
    output reg [31:0] out_var;

    output reg [31:0] out_A, out_B, out_C, out_D,
        out_E, out_F, out_G, out_H;

    reg [31:0] reg_A, reg_B, reg_C, reg_D,
        reg_E, reg_F, reg_G, reg_H;

    input wire [31:0] in_A, in_B, in_C, in_D,
        in_E, in_F, in_G, in_H;

    always @(posedge clk) begin
        if (en) begin
            $display("[v] Out mem A, B: %h %h",
                in_A, in_B);
```

```
                out_A <= in_A;
                out_B <= in_B;
                // <...>
                out_H <= in_H;

                reg_A <= in_A;
                reg_B <= in_B;
                // <...>
                reg_H <= in_H;
            end

            case(addr)
            01: out_var <= reg_A;
            02: out_var <= reg_B;
            // <...>
            08: out_var <= reg_H;
            endcase
        end

endmodule
```

После подключения блока вычислений, блока констант, блока памяти переменных и выходного буфера друг к другу, было произведено моделирование их совместной работы с целью проверки корректности их работы и уточнения поведения управляющих сигналов.

Полученные в ходе моделирования временные диаграммы приведены на рисунке 3.

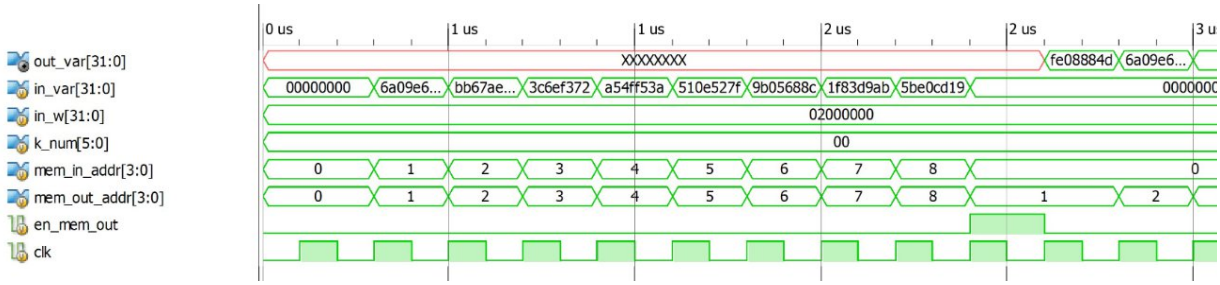


Рисунок 3 – временная диаграмма работы основных блоков

3.4 Разработка блока управления

Так как сочетания сигналов выборки полностью определяются последовательным номером исполняемого цикла алгоритма SHA-256, для обеспечения корректной работы блоков памяти и упрощения внешнего интерфейса устройства был разработан блок управления. Этот блок на основе тактового сигнала изменяет значение внутреннего счетчика. На основе

значения внутреннего счетчика генерируются управляющие сигналы, соответствующие управляющим сигналам на рисунке 3 (за исключением начального адреса выходного буфера, который не оказывает влияния на корректность работы устройства). Исходный код данного блока приведен в листинге 7.

Листинг 7 – исходный код блока управления

```
module control_block(
     clk, reset, in_mem_addr, k_num,
     out_mem_addr, en_mem_out,
     en_mem_in
    );

     input wire clk;
     input wire reset;
     output reg [3:0] in_mem_addr;
     output reg [3:0] out_mem_addr;
     output reg [5:0] k_num;
     output reg en_mem_out;
     reg [10:0] n;
     reg [10:0] n_buf;
     output wire en_mem_in;

     assign en_mem_in = 1;

     always @(posedge clk or posedge reset) begin
          if (reset == 1) begin
               n <= 0;
               n_buf <= 0;
          end
          else begin
               if (clk == 1) begin
                    n <= n + 1;
                    if (n%9 != 0)
                     n_buf <= n_buf + 1;
               end

               k_num = n/19;
               out_mem_addr <= n_buf%8 + 1;

               if (n%9 == 0)
                    en_mem_out <= 1;
               else
                    en_mem_out <= 0;

               if (n > 0)
               begin
                    if (n < 9)
```

```
                    in_mem_addr <= n;
               else
                    in_mem_addr <= 0;
          end
          else
               in_mem_addr <= 0;
     end
end

endmodule
```

Результаты моделирования работы блока управления приведены на рисунке 4. Управляющие сигналы соответствуют ожидаемым.

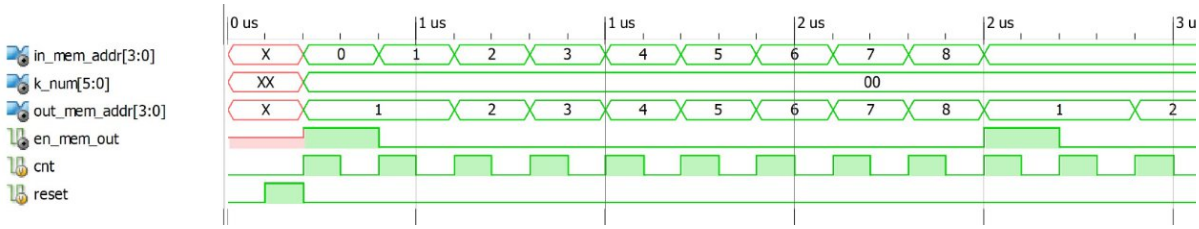


Рисунок 4 – временные диаграммы работы блока управления

3.5 Разработка описания устройства

После добавления мультиплексирующего блока, было разработано описание соединения блоков в проектируемом устройстве. Блоки памяти, вычислений, выходного буфера и констант были предварительно объединены в «главный блок». Описание проектируемого устройства приведено в листинге 8.

Листинг 8 – описание проектируемого устройства

```
`include "main_block.v"
`include "control_block.v"
`include "input_mux.v"

module asic2(
     clk, reset, in_data, out_var
    );

     input wire clk, reset;
     input wire [31:0] in_data;
     output wire [31:0] out_var;

     wire [31:0] mux_w, mux_var;

     wire [5:0] _k_num;
     wire [3:0] _in_mem_addr,
          _out_mem_addr;
```

```
    wire _en_mem_out;
    wire _en_mem_in;

    input_mux mux(.addr(_in_mem_addr),
    .in_data(in_data),
    .out_w(mux_w),
    .out_var(mux_var)
    );

    main_block core(.in_var(mux_var),
         .in_w(mux_w), .out_var(out_var),
         .k_num(_k_num),
         .mem_in_addr(_in_mem_addr),
         .mem_out_addr(_out_mem_addr),
         .en_mem_out(_en_mem_out),
         .en_mem_in(_en_mem_in),
         .clk(clk)
   );

    control_block dispatcher(
         .clk(clk), .reset(reset),
         .in_mem_addr(_in_mem_addr),
         .k_num(_k_num),
         .out_mem_addr(_out_mem_addr),
         .en_mem_out(_en_mem_out),
         .en_mem_in(_en_mem_in)
   );

endmodule
```

3.6 Синтез RTL-схемы

На основе приведенных ранее исходных кодов с помощью Xilinx ISE была сгенерирована RTL-схема. На рисунке 5 представлена RTL-схема, развернутая до уровня основных блоков устройства.

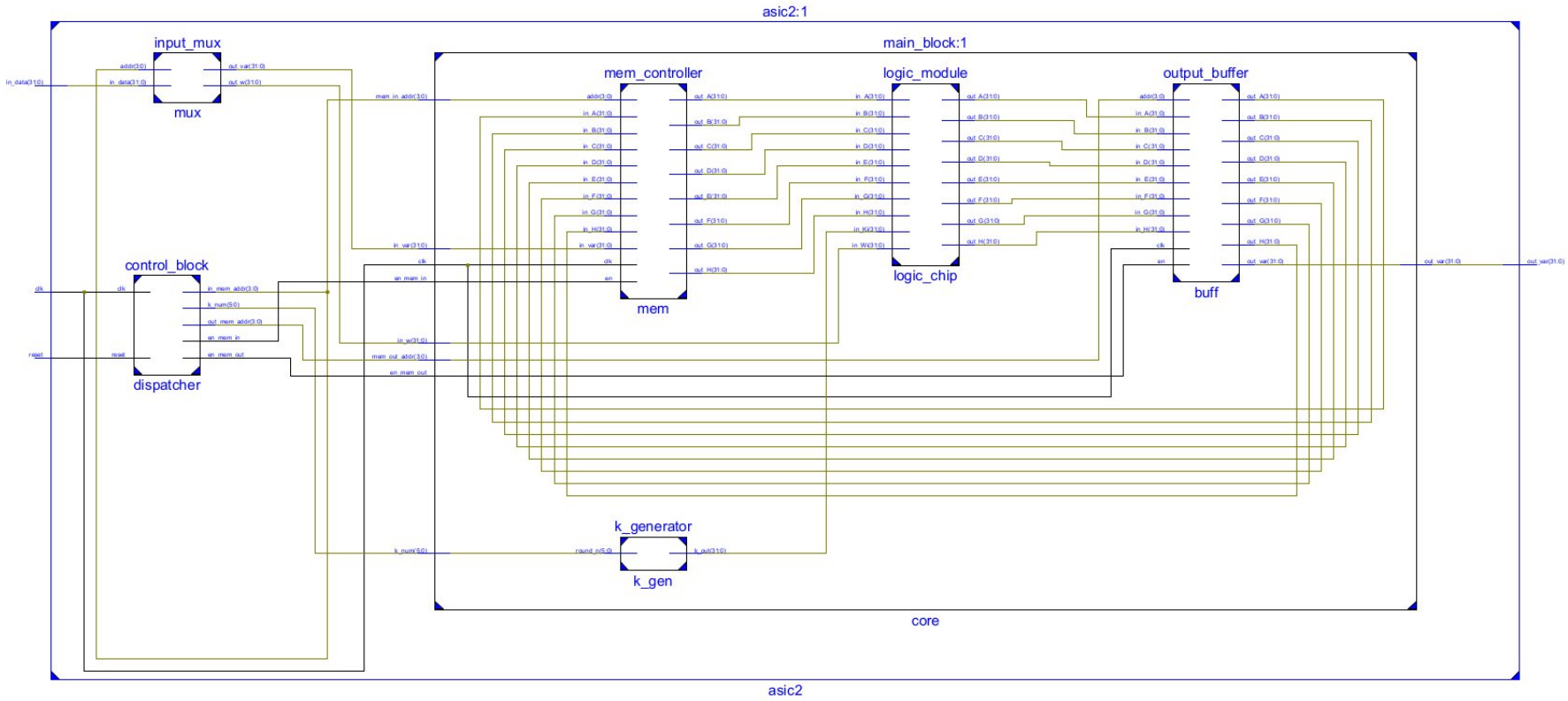


Рисунок 5 – RTL-схема верхнего уровня

На рисунке 6 показана более детальная RTL-схема вычислительного блока.

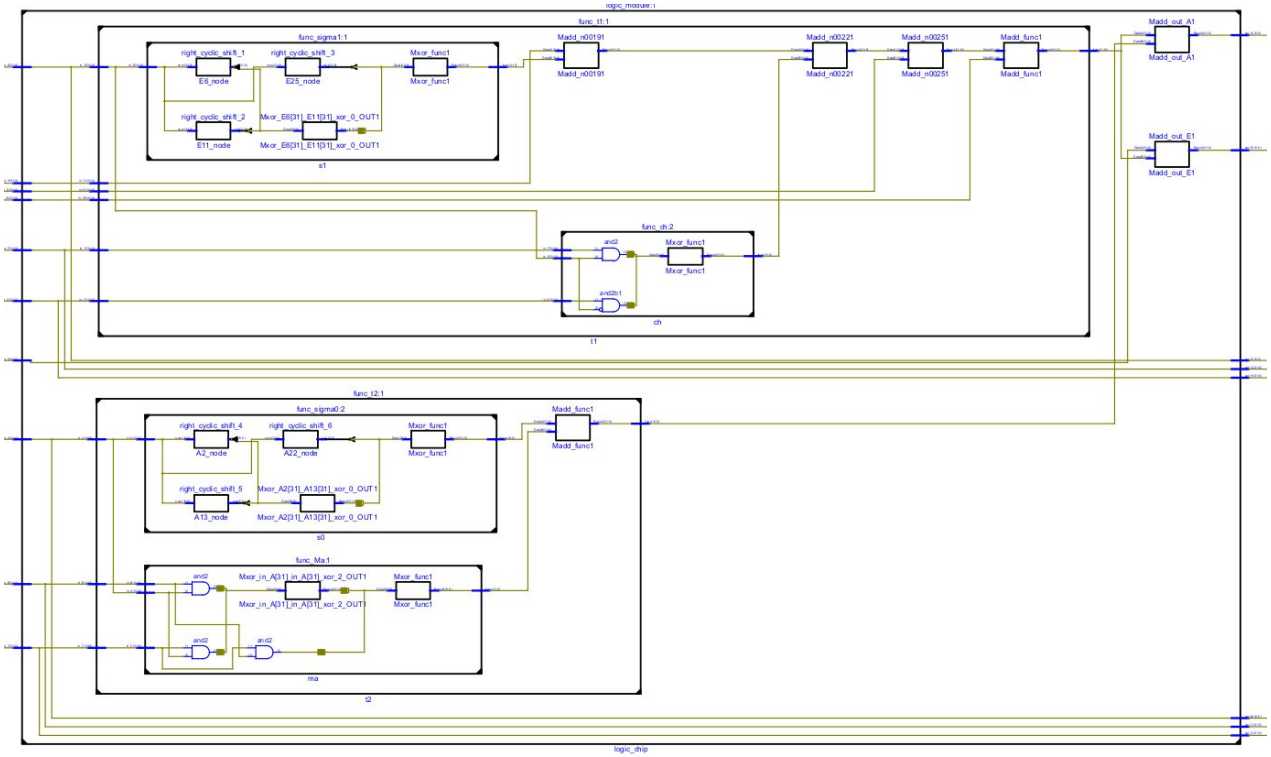


Рисунок 6 – RTL-схема вычислительного блока

На рисунках 7 и 8 приведены примеры RTL схем блоков, вычисляющих значение функций t1 и t2 соответственно. В состав блока, вычисляющего t1, среди прочего, входя блоки, вычисляющие функции Сигма-1 и Ch. В состав блока, вычисляющего t2, входят блоки, сдвигающие значения в шинах на заданное число разрядов.

На рисунке 9 представлен выходной буфер.

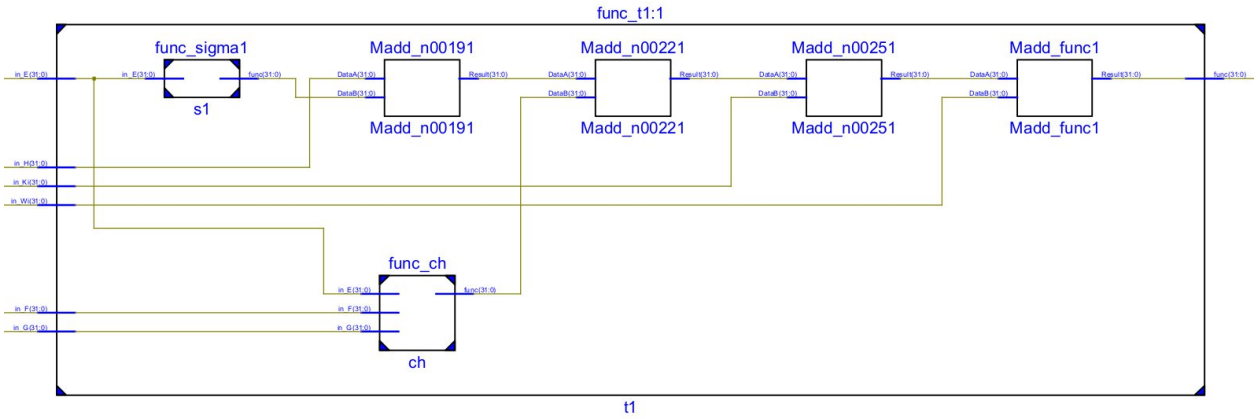


Рисунок 7 – блок вычисления функции t1

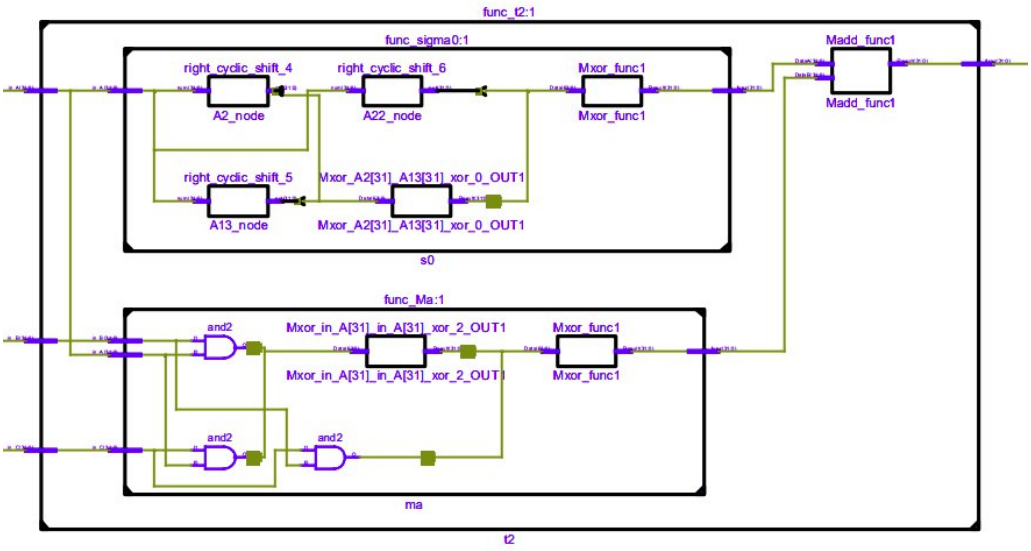


Рисунок 8 – блок вычисления функции t2

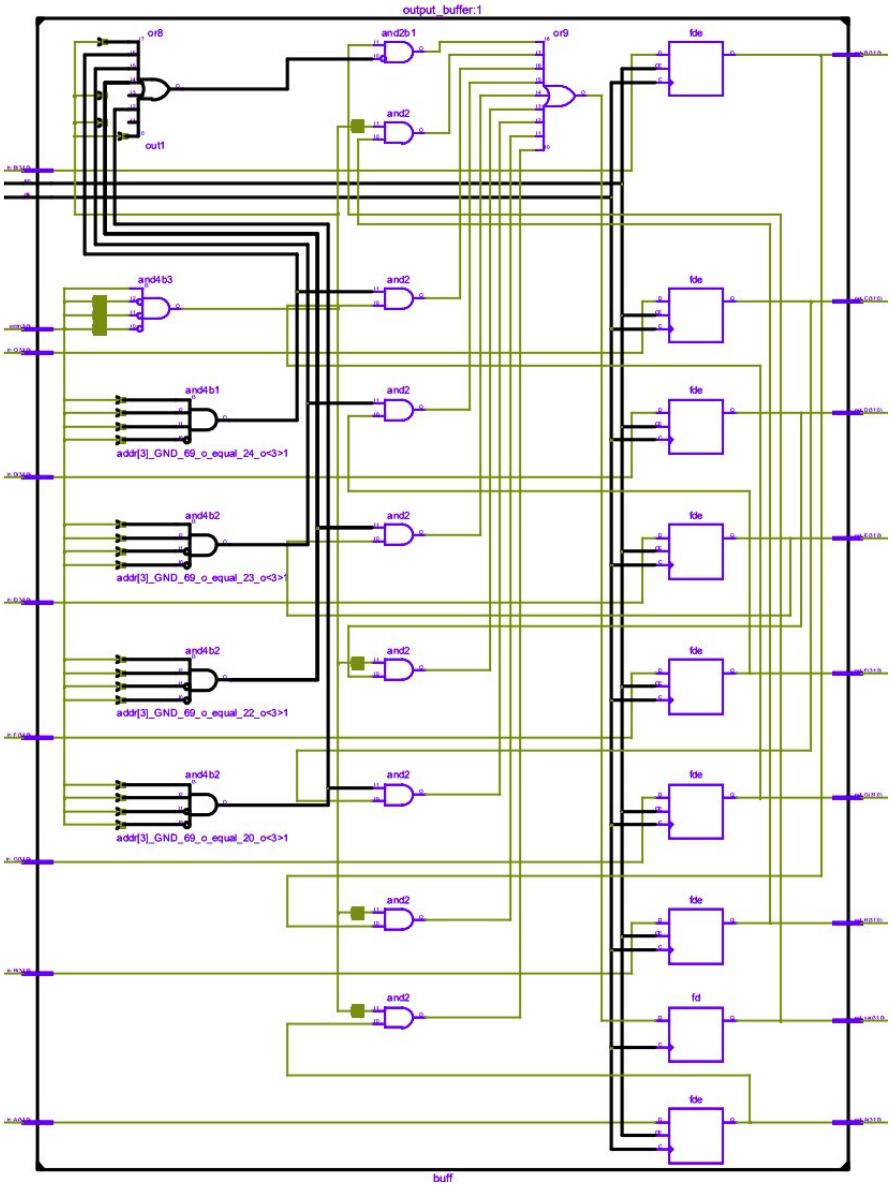


Рисунок 9 – блок выходного буфера

4 Назначение контактов микросхемы портам проекта

В качестве ПЛИС для прошивки проекта была выбрана ПЛИС Spartan-3 XC3S400. XC3S400 имеет 83 ввода-вывода общего назначения, что оптимально для решаемой задачи. Данная ПЛИС широко применяется и имеет качественную документацию [6].

С помощью утилиты PlanAhead было задано соответствие между контактами ПЛИС и портами проекта. Работа с утилитой PlanAhead показана на рисунке 10.

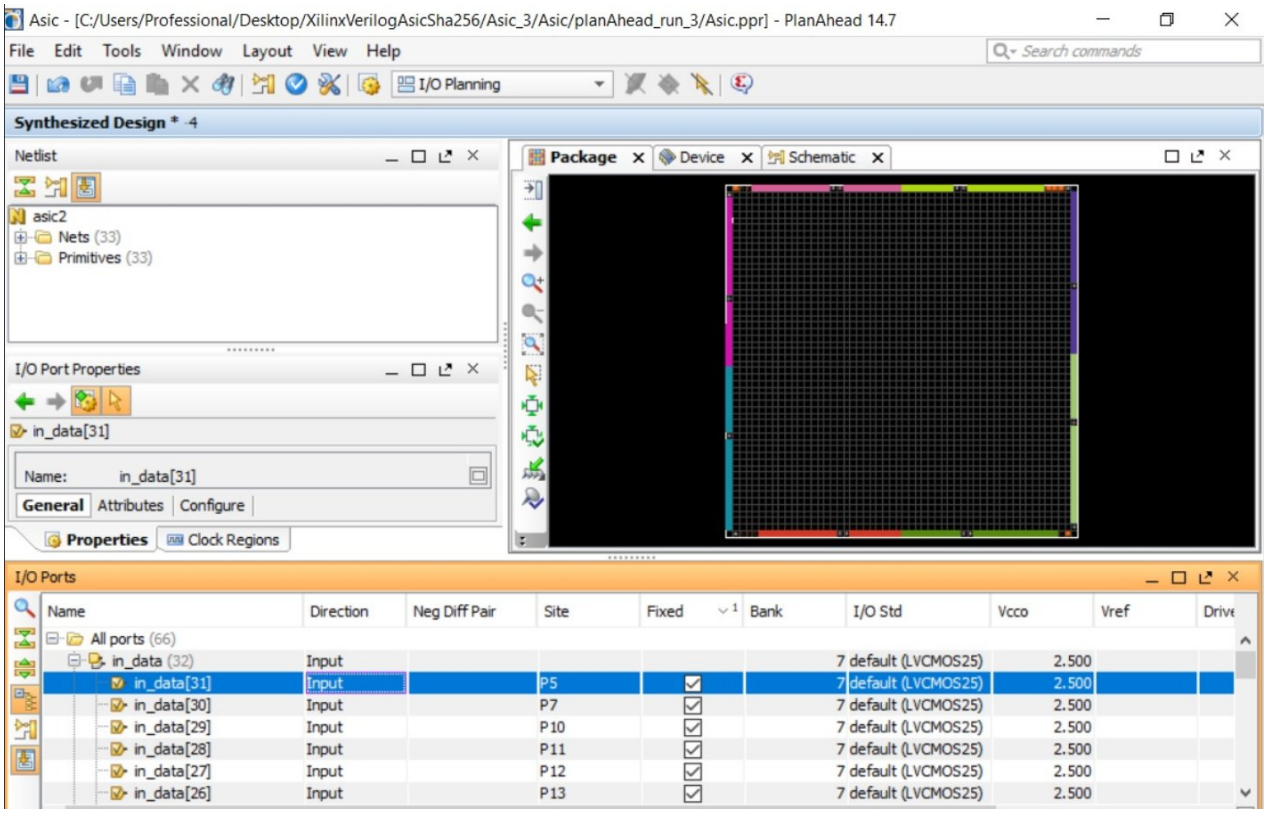


Рисунок 10 – работа с утилитой PlanAhead

Полученные в результате работы с утилитой PlanAhead соответствия портов проекта и контактов ПЛИС сохранены в виде UCF-файла. На основе этой информации разработана принципиальная схема (приложение В).

5 Проверка работоспособности и расчет характеристик схемы

5.1 Моделирование работы схемы

Xilinx ISE предоставляет встроенный инструмент ISE Simulator для моделирования работы схем. Для проведения моделирования был написан тестовый модуль на Verilog (так называемый «testbench»). Исходный код этого модуля приведен в листинге 8. Полученные в результате временные диаграммы представлены в приложении Г.

Листинг 8 — тестовый модуль

```
module asic3_tb;

	// Inputs
	reg clk;
	reg reset;
	reg [31:0] in_data;

	// Outputs
	wire [31:0] out_var;

	// Instantiate the Unit Under Test (UUT)
	asic2 uut (
		.clk(clk),
		.reset(reset),
		.in_data(in_data),
		.out_var(out_var)
	);

	initial begin

		clk = 0;
		reset = 0;
		in_data = 0;

		#100; reset = 1; #100; reset = 0;

		#100; clk=1; #100; clk=0; #100; clk=1; #100; clk=0;

		// A
		in_data = 32'h6a09e667;
		#100; clk=1; #100; clk=0;

		// B
		in_data = 32'hbb67ae85;
		#100; clk=1; #100; clk=0;

		//
		// <...> ввод C, D, E, F, G, H
```

```
        //


        // output
        in_data = 32'h02000000;

        #100; clk=1; #100; clk=0;
        #100; clk=1; #100; clk=0;

        //
        // <...> повтор тактов 6 раз
        //

    end

endmodule
```

5.2 Расчет параметров быстродействия и энергопотребления

Параметры быстродействия рассчитаны при синтезе схемы с помощью встроенной утилиты Xilinx ISE. Отчет о быстродействии приведен ниже:

```
--------------------------------------------------------------
Release 14.7 Trace  (nt64)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
-----------+------------+----------+------------+---------+
           |Max Setup to|  Process |Max Hold to | Process |
Source     | clk (edge) |   Corner | clk (edge) |  Corner |
-----------+------------+----------+------------+---------+
in_data<0> |    0.972(R)|      FAST|    1.194(R)|     SLOW|
in_data<1> |    0.999(R)|      FAST|    1.328(R)|     SLOW|
in_data<2> |    1.003(R)|      FAST|    0.809(R)|     SLOW|
in_data<3> |    0.916(R)|      FAST|    1.709(R)|     SLOW|
in_data<4> |    1.118(R)|      FAST|    1.750(R)|     SLOW|
in_data<5> |    0.829(R)|      FAST|    1.555(R)|     SLOW|
in_data<6> |    1.394(R)|      FAST|    0.753(R)|     SLOW|
in_data<7> |    1.597(R)|      SLOW|    1.043(R)|     SLOW|
in_data<8> |    1.403(R)|      FAST|    1.241(R)|     SLOW|
in_data<9> |    1.845(R)|      SLOW|    1.334(R)|     SLOW|
in_data<10>|    1.491(R)|      SLOW|    1.333(R)|     SLOW|
in_data<11>|    1.078(R)|      FAST|    1.446(R)|     SLOW|
in_data<12>|    1.088(R)|      FAST|    1.348(R)|     SLOW|
in_data<13>|    1.276(R)|      FAST|    1.463(R)|     SLOW|
in_data<14>|    0.996(R)|      FAST|    1.887(R)|     SLOW|
in_data<15>|    0.840(R)|      FAST|    2.124(R)|     SLOW|
in_data<16>|    0.831(R)|      FAST|    1.999(R)|     SLOW|
in_data<17>|    0.258(R)|      FAST|    1.996(R)|     SLOW|
in_data<18>|    1.001(R)|      FAST|    1.388(R)|     SLOW|
```

in_data<19>	0.419(R)	FAST	1.956(R)	SLOW
in_data<20>	0.678(R)	FAST	1.624(R)	SLOW
in_data<21>	0.252(R)	FAST	2.058(R)	SLOW
in_data<22>	0.611(R)	FAST	1.822(R)	SLOW
in_data<23>	0.275(R)	FAST	2.311(R)	SLOW
in_data<24>	0.490(R)	FAST	2.117(R)	SLOW
in_data<25>	0.364(R)	FAST	2.081(R)	SLOW
in_data<26>	0.362(R)	FAST	2.085(R)	SLOW
in_data<27>	0.666(R)	FAST	2.144(R)	SLOW
in_data<28>	0.323(R)	FAST	2.485(R)	SLOW
in_data<29>	0.535(R)	FAST	2.192(R)	SLOW
in_data<30>	0.476(R)	FAST	2.382(R)	SLOW
in_data<31>	0.489(R)	FAST	2.200(R)	SLOW
reset	1.564(R)	FAST	1.493(R)	SLOW

Clock clk to Pad

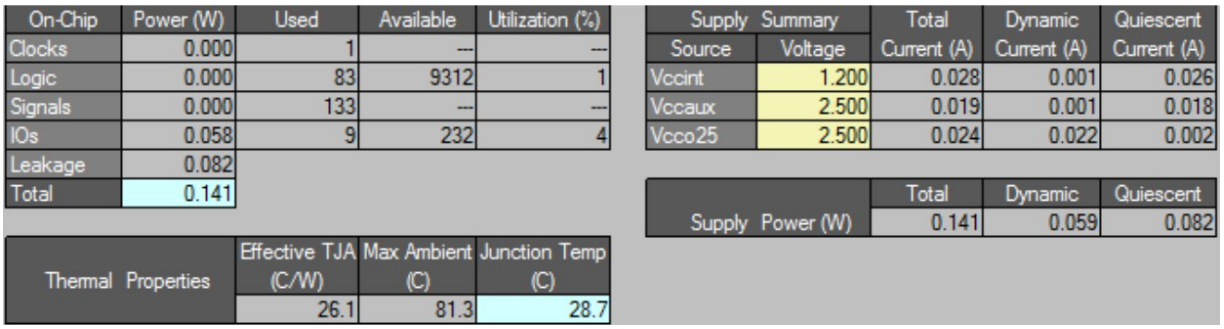
Destination	Max (slowest) clk (edge) to PAD	Process Corner	Min (fastest) clk (edge) to PAD	Process Corner
out_var<0>	8.776(R)	SLOW	3.796(R)	FAST
out_var<1>	9.413(R)	SLOW	4.239(R)	FAST
out_var<2>	9.217(R)	SLOW	4.061(R)	FAST
out_var<3>	8.690(R)	SLOW	3.731(R)	FAST
out_var<4>	9.300(R)	SLOW	4.173(R)	FAST
out_var<5>	8.778(R)	SLOW	3.811(R)	FAST
out_var<6>	8.739(R)	SLOW	3.792(R)	FAST
out_var<7>	8.552(R)	SLOW	3.680(R)	FAST
out_var<8>	8.625(R)	SLOW	3.793(R)	FAST
out_var<9>	8.552(R)	SLOW	3.672(R)	FAST
out_var<10>	8.464(R)	SLOW	3.628(R)	FAST
out_var<11>	8.730(R)	SLOW	3.860(R)	FAST
out_var<12>	8.631(R)	SLOW	3.732(R)	FAST
out_var<13>	8.598(R)	SLOW	3.797(R)	FAST
out_var<14>	8.666(R)	SLOW	3.741(R)	FAST
out_var<15>	8.514(R)	SLOW	3.718(R)	FAST
out_var<16>	8.869(R)	SLOW	3.959(R)	FAST
out_var<17>	8.453(R)	SLOW	3.643(R)	FAST
out_var<18>	8.665(R)	SLOW	3.763(R)	FAST
out_var<19>	8.527(R)	SLOW	3.656(R)	FAST
out_var<20>	8.865(R)	SLOW	3.903(R)	FAST
out_var<21>	8.809(R)	SLOW	3.870(R)	FAST
out_var<22>	8.668(R)	SLOW	3.766(R)	FAST
out_var<23>	8.541(R)	SLOW	3.740(R)	FAST
out_var<24>	9.105(R)	SLOW	4.054(R)	FAST
out_var<25>	8.652(R)	SLOW	3.734(R)	FAST
out_var<26>	8.470(R)	SLOW	3.635(R)	FAST
out_var<27>	8.775(R)	SLOW	3.841(R)	FAST
out_var<28>	8.506(R)	SLOW	3.660(R)	FAST
out_var<29>	8.550(R)	SLOW	3.668(R)	FAST

```
out_var<30>|      8.669(R)|     SLOW |      3.760(R)|     FAST|
out_var<31>|      8.734(R)|     SLOW |      3.820(R)|     FAST|
-----------+-------------+---------+-------------+---------+

Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |    7.838|    0.438|         |         |
---------------+---------+---------+---------+---------+
```

Задержки не превышают 10 нс, что позволяет схеме работать на частоте, указанной в ТЗ.

Параметры потребляемой мощности были рассчитаны с помощью утилиты xPower Analyser, входящей в состав САПР Xilinx ISE Design Suite. Результаты расчета приведены на рисунке 11.



On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.000	83	9312	1
Signals	0.000	133	---	---
IOs	0.058	9	232	4
Leakage	0.082			
Total	0.141			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	26.1	81.3	28.7

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.028	0.001	0.026
Vccaux	2.500	0.019	0.001	0.018
Vcco25	2.500	0.024	0.022	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.141	0.059	0.082

Рисунок 11 – расчет потребляемой мощности

Потребляемая мощность равна 0.141 Вт, что соответствует требованиям ТЗ.

Заключение

В ходе выполнения курсовой работы были получены функциональная и принципиальная схемы устройства, а также временные диаграммы моделирования его работы.

Устройство представляет собой вычислитель внутреннего цикла алгоритма хеширования SHA-256.

Разработка описания, моделирование и расчет характеристик устройства производились с использованием языка описания аппаратуры Verilog в среде Xilinx ISE (включая встроенные утилиты PlanAhead и iSim).

По результатам моделирования и расчетов был сделан вывод, что устройство работает согласно требованиям ТЗ.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Penny Pritzker, Willie E. May, Secure Hash Standard (SHS) / Penny Pritzker, Willie E. May. – Gaithersburg : FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, 2015. – 36 с.
2. В.В. Рубанов. Обзор методов описания встраиваемой аппаратуры и построения инструментария кросс-разработки. Труды Института системного программирования РАН, том 15, 2008, стр. 7-40.
3. Бегимбаева, О.А. АНАЛИЗ МЕТОДОВ И ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ ХЕШ-ФУНКЦИЙ / О.А. Бегимбаева, Е.Е. Усатова, С.Е. Нысанбаева. – Алматы : № 5 (2021): "Известия НАН РК. Серия физико-математическая", 2021. – 100-110 с.
4. Github: secworks - sha256 [Электронный ресурс]. – Режим доступа: https://github.com/secworks/sha256/blob/master/src/model/sha256.py. – Дата доступа: 05.03.2022.
5. Попов А.Ю. Проектирование цифровых устройств с использованием ПЛИС. – М.: изд-во МГТУ, 2009. – 79 с.
6. Xilinx Inc., Spartan-3 FPGA Family Data Sheet — San Jose: Inc. XILINX, 2013. — 272 с.

ПРИЛОЖЕНИЕ А

Техническое задание

Листов 4

ПРИЛОЖЕНИЕ Б

Функциональная схема

Листов 1

ПРИЛОЖЕНИЕ В

Принципиальная электрическая схема

Листов 1

ПРИЛОЖЕНИЕ Г

Временные диаграммы

Листов 1

ПРИЛОЖЕНИЕ Д

Перечень элементов

Листов 1