|  |  |
| --- | --- |
|  | **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.*

и

ЗАДАНИЕ

**Реферат**

Записка 21 стр., 7 рис., 3 табл., 4 источника, 2 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**

[Введение 6](#_Toc5)

[1 Анализ предметной области 7](#_Toc6)

[2 Разработка электрической функциональной схемы 8](#_Toc7)

[3. Разработка описания устройства на языке Verilog 9](#_Toc8)

[3.1 Разработка вычислительного блока 9](#_Toc9)

[3.2 Разработка блока памяти переменных 12](#_Toc10)

[3.3 Разработка блока выходного буфера 13](#_Toc11)

[**3.4 Разработка блока** **управления** 15](#_Toc12)

[4 Выбор и настройка выводов ПЛИС 18](#_Toc13)

[**4.1 Обоснование выбора модели ПЛИС (artex7? Spartan3?)** 18](#_Toc14)

[**4.2 Схема принципиальная**  18](#_Toc15)

[5 Проверка работоспособности и расчет характеристик схемы 18](#_Toc16)

[**5.1 Временные диаграммы (+ код test bench?)** 18](#_Toc17)

[**5.2 Расчет быстродействия** 18](#_Toc18)

[**5.3 Расчет потребления** 18](#_Toc19)

[Заключение 19](#_Toc20)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 20](#_Toc21)

**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).

# 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;

- 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 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.

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).

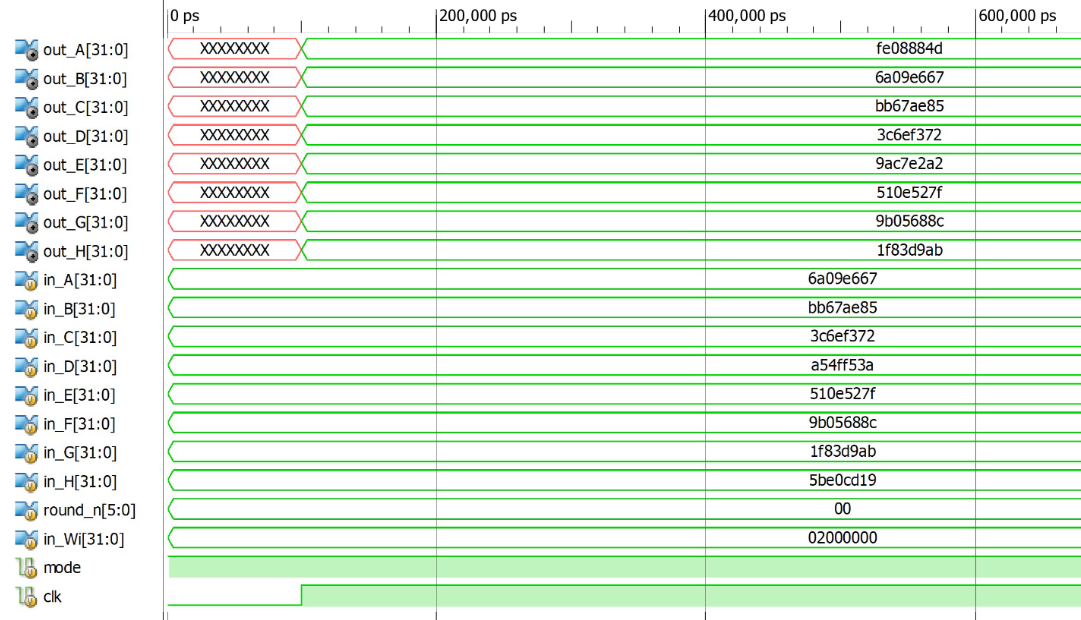


Рисунок 2 – временная диаграмма работы блока вычислений

## 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

);

input wire [3:0] addr;

input wire [31:0] in\_var;

input wire clk;

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

case(addr)

01: out\_A <= in\_var;

02: out\_B <= in\_var;

03: out\_C <= in\_var;

04: out\_D <= in\_var;

05: out\_E <= in\_var;

06: out\_F <= in\_var;

07: out\_G <= in\_var;

08: out\_H <= in\_var;

default: begin

out\_A <= in\_A;

out\_B <= in\_B;

out\_C <= in\_C;

out\_D <= in\_D;

out\_E <= in\_E;

out\_F <= in\_F;

out\_G <= in\_G;

out\_H <= in\_H;

end

endcase

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\_C <= in\_C;

out\_D <= in\_D;

out\_E <= in\_E;

out\_F <= in\_F;

out\_G <= in\_G;

out\_H <= in\_H;

reg\_A <= in\_A;

reg\_B <= in\_B;

reg\_C <= in\_C;

reg\_D <= in\_D;

reg\_E <= in\_E;

reg\_F <= in\_F;

reg\_G <= in\_G;

reg\_H <= in\_H;

end

case(addr)

01: out\_var <= reg\_A;

02: out\_var <= reg\_B;

03: out\_var <= reg\_C;

04: out\_var <= reg\_D;

05: out\_var <= reg\_E;

06: out\_var <= reg\_F;

07: out\_var <= reg\_G;

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.

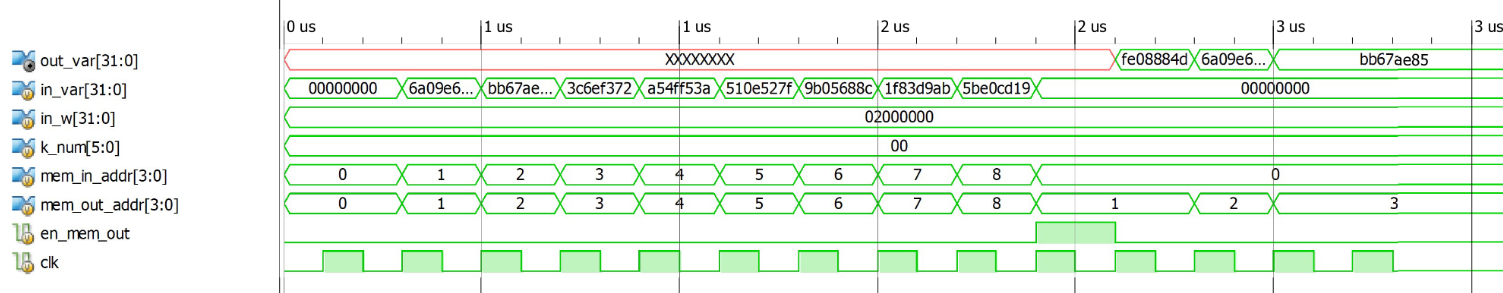


Рисунок 3 – временная диаграмма работы основных блоков

## **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

);

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;

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.

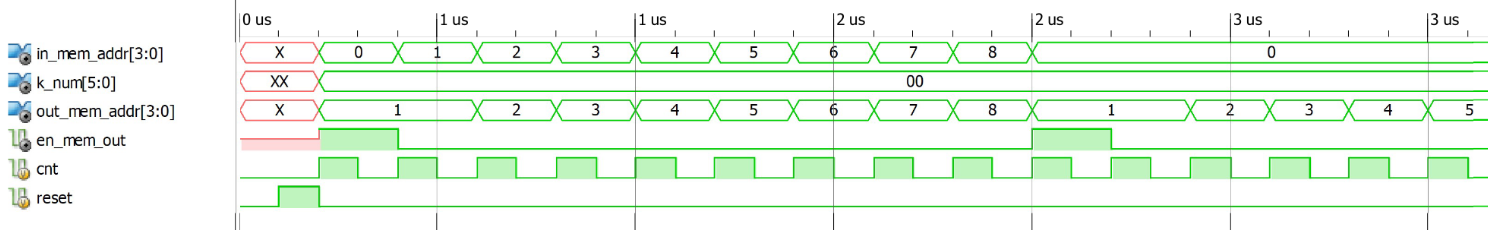


Рисунок 4 – временные диаграммы работы блока управления

**3.5 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.

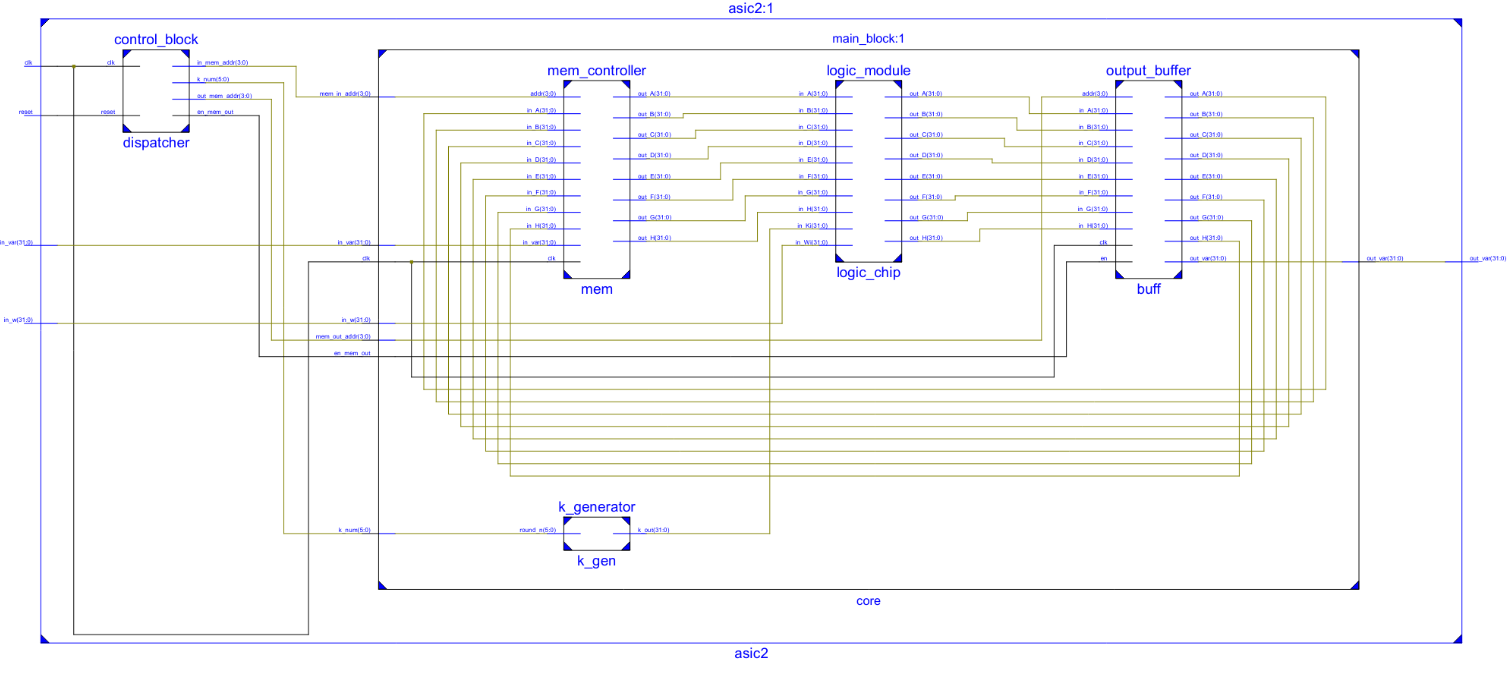


Рисунок 5 – RTL-схема верхнего уровня

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

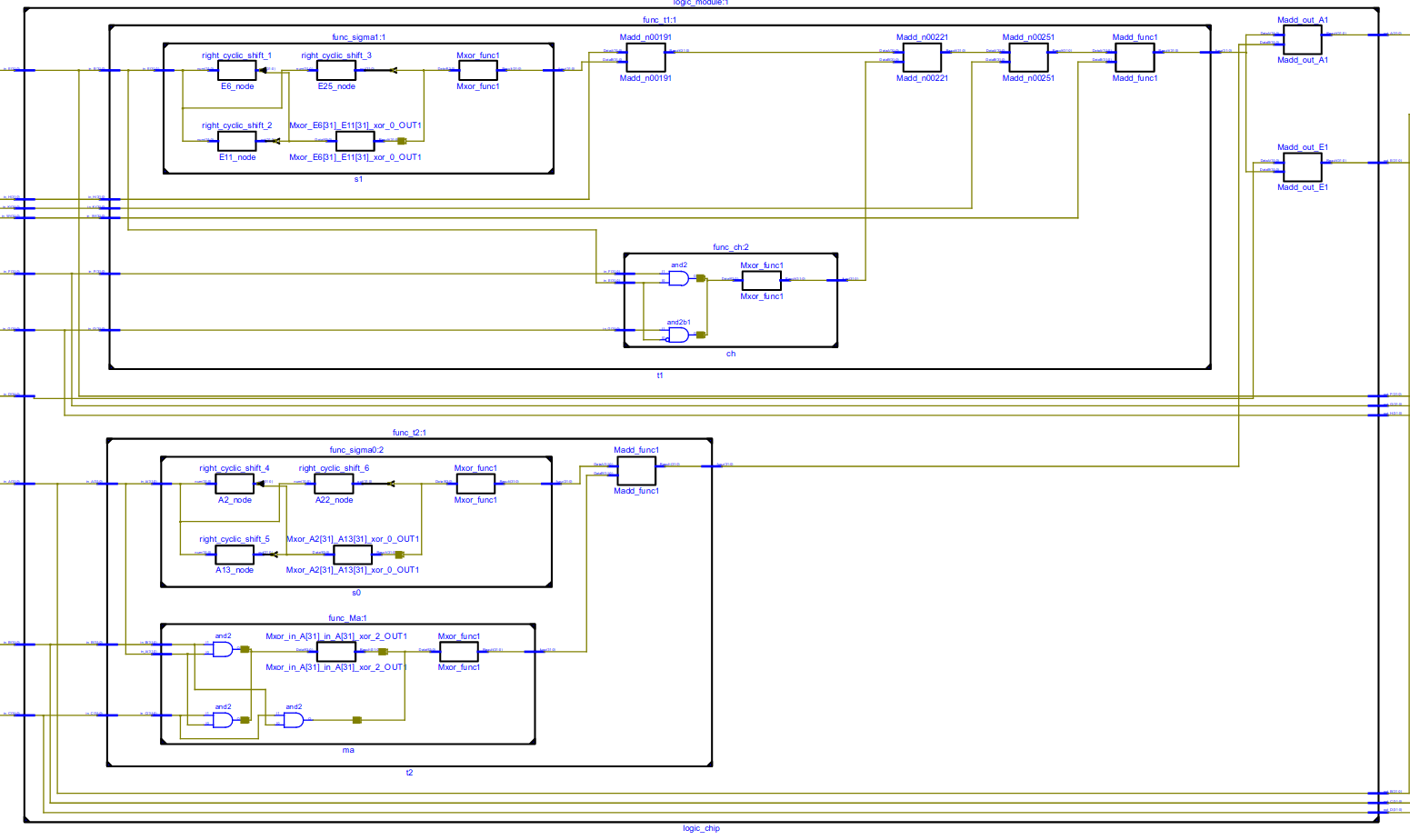


Рисунок 6 – RTL-схема вычислительного блока

Более детальная RTL-схема всего устройства, демонстрирующая, помимо прочего, регистры и схемы выборки памяти представлена в приложении Б.

# 4 Выбор и настройка выводов ПЛИС

## **4.1 Обоснование выбора модели ПЛИС (artex7? Spartan3?)**

## **4.2 Схема принципиальная**

# **5 Проверка работоспособности и расчет характеристик схемы**

## **5.1 Временные диаграммы (+ код test bench?)**

## **5.2 Расчет быстродействия**

## **5.3 Расчет параметров энергопотребления**

# Заключение

# 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.