DOI: 10.22184/1992-4178.2024.240.9.70.79

Пример реализации однотактного процессорного ядра RISC-V в САПР Altera Quartus II

А.В. Строгонов, д.т.н.[1], А. Винокуров, к.т.н.[2], А.И. Строгонов[3]

УДК 004.31 | ВАК 2.2.2

В России продолжается популяризация и развитие открытой архитектуры RISC-V. Сегодня разработкой проектов на базе открытой архитектуры RISC-V занимается ряд российских компаний и ведущих университетов. Например, микроконтроллер Hackee на базе ядра SCR1 (от российского разработчика Syntacore) был спроектирован совместно магистрантами НИУ «МИЭТ» и специалистами компании Yadro. «Микрон» представил микроконтроллер «МІК32 Амур» (К1948ВК018) на базе RISC-V и отладочную плату на его основе. Есть проекты и других российских компаний, действующих в рамках «Альянса RISC-V» [1]. Одним из направлений исследований в этой области является отработка прототипов процессоров на платформе ПЛИС. В статье рассмотрен пример реализации однотактного процессорного ядра RISC-V в базисе ПЛИС Cyclone V с применением САПР Altera Quartus II.

Для реализации процессора RISC-V в данной работе была выбрана однотактная микроархитектура, которая выполняет всю команду за один такт. Из-за того, что все действия выполняются за один такт, эта микроархитектура не требует никаких дополнительных регистров, недоступных для программиста. Основным преимуществом однотактной микроархитектуры является простой принцип ее работы.

В данной разработке в качестве основного языка был выбран VHDL, хотя процессор можно реализовать на двух языках: SystemVerilog и VHDL. Коды SystemVerilog/VHDL основных функциональных блоков RISC-V приведены в разделе 7.6.1 работы [2]. В работах [2] и [3] синтез кодов SystemVerilog/VHDL осуществлялся с помощью программы Synplify Premier от Synplicity.

При адаптации VHDL-кодов функциональных блоков процессорного ядра RISC-V возникли некоторые вопросы, связанные с синтезатором-компилятором QIS САПР Quartus II. Прояснить эти вопросы помогла оригинальная версия переводного учебника [3]. При разработке проекта в базисе ПЛИС Cyclone V были сохранены оригинальные коды SystemVerilog/VHDL и обозначения сигналов и шин.

Структурная и функциональная схемы однотактного процессорного ядра RISC-V (с поддержкой команды addi) показаны на рис. 1 и 2. На рис. 3 представлена тестовая

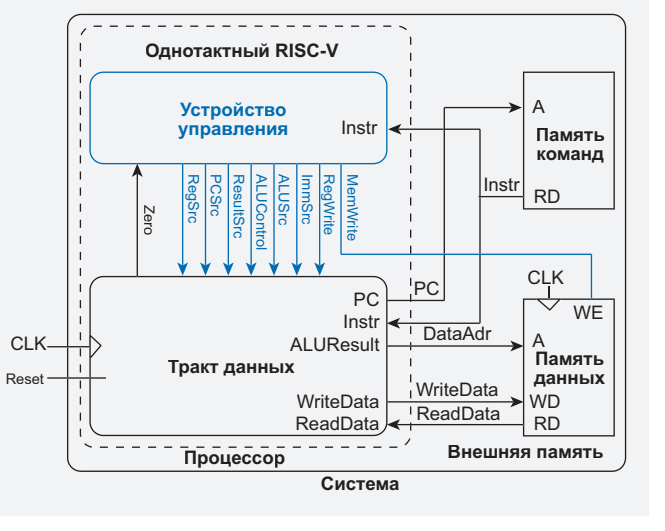


Рис. 1. Структурная схема однотактного процессорного ядра RISC-V

[1] Воронежский государственный технический университет, профессор кафедры твердотельной электроники, тел. +79102471470, andreistrogonov@mail.ru.

[2] Воронежский государственный технический университет, доцент кафедры твердотельной электроники.

[3] Воронежский государственный университет, факультет компьютерных наук, кафедра программирования и информационных технологий, ассистент.

программа (машинный код хранится в шестнадцатеричном файле riscvtest.txt), зашитая в ПЗУ (рис.7.64 и 7.65 из [2]). Тестируются следующие команды: add, sub, and, or, slt, addi, lw, sw, beq, jal. Если тест проходит успешно, то значение 25 (0×19) должно записаться по адресу 100 (0×64).

Верхний уровень иерархии проекта в САПР Quartus II, в отличие от работы [2], представлен не структурным файлом (обьект riscvsingle, пример 7.1 из [2]) на языке VHDL, а иерархической схемой в графическом редакторе (рис. 4). На рис. 5 в качестве примера показан тракт данных, соответствующий примеру 7.5 (обьект datapath).

При адаптации RISC-V в САПР Altera Quartus II возник ряд проблем. Все шины данных во всех функциональных блоках (в работе [2] они называются строительными блоками) должны быть 32-разрядными. В некоторых функциональных блоках требуется увеличение разрядности шин с 8 до 32 (регистр с сигналом сброса flopr, пример 7.8 [2]; шинный мультиплексор 2:1, пример 7.10 [2]; шинный мультиплексор 3:1, пример 7.11 [2]).

Синтезатор Quartus II не поддерживает пакет расширения IEEE.NUMERIC_STD_UNSIGNED.all от Synopsys или Mentor Graphics. Поэтому в данной работе использовали пакеты IEEE.NUMERIC_STD.all и IEEE.std_logic_unsigned.all.

Возникла также проблема с памятью команд (асинхронное ПЗУ). При

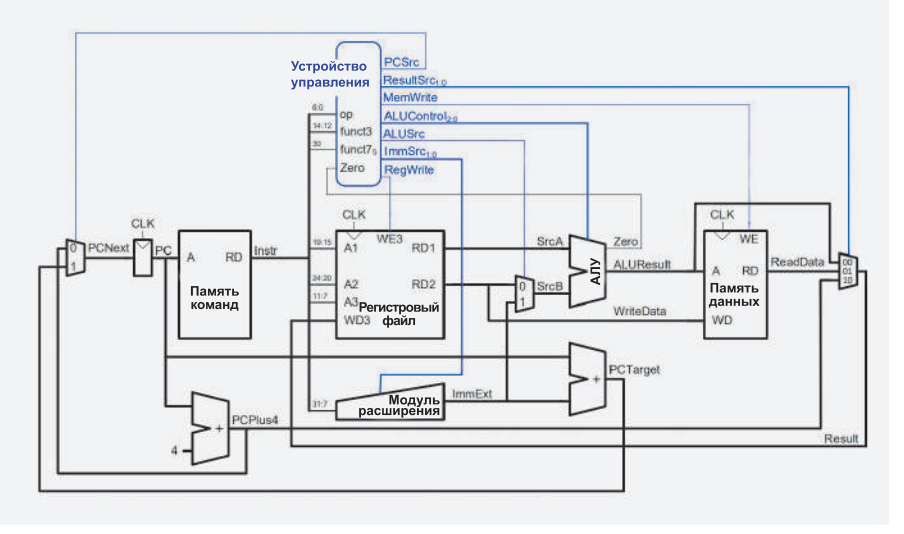


Рис. 2. Функциональная схема однотактного процессорного ядра RISC-V

```
#        RISC-V Assembly          Description              Address   Machine
main:    addi x2, x0, 5           # x2 = 5                 0         00500113
         addi x3, x0, 12          # x3 = 12                4         00C00193
         addi x7, x3, -9          # x7 = (12 - 9) = 3      8         FF718393
         or   x4, x7, x2          # x4 = (3 OR 5) = 7      C         0023E233
         and  x5, x3, x4          # x5 = (12 AND 7) = 4    10        0041F2B3
         add  x5, x5, x4          # x5 = (4 + 7) = 11      14        004282B3
         beq  x5, x7, end         # shouldn't be taken     18        02728863
         slt  x4, x3, x4          # x4 = (12 < 7) = 0      1C        0041A233
         beq  x4, x0, around      # should be taken        20        00020463
         addi x5, x0, 0           # shouldn't happen       24        00000293
around:  slt  x4, x7, x2          # x4 = (3 < 5)  = 1      28        0023A233
         add  x7, x4, x5          # x7 = (1 + 11) = 12     2C        005203B3
         sub  x7, x7, x2          # x7 = (12 - 5) = 7      30        402383B3
         sw   x7, 84(x3)          # [96] = 7               34        0471AA23
         lw   x2, 96(x0)          # x2 = [96] = 7          38        06002103
         add  x9, x2, x5          # x9 = (7 + 11) = 18     3C        005104B3
         jal  x3, end             # jump to end, x3 = 0x44 40        008001EF
         addi x2, x0, 1           # shouldn't happen       44        00100113
end:     add  x2, x2, x9          # x2 = (7 + 18)  = 25    48        00910133
         sw   x2, 0x20(x3)        # mem[100] = 25          4C        0221A023
done:    beq  x2, x2, done        # infinite loop          50        00210063
```

Рис. 3. Ассемблерный и машинный коды (крайний правый столбец – прошивка ПЗУ, текстовый файл riscvtest.txt)

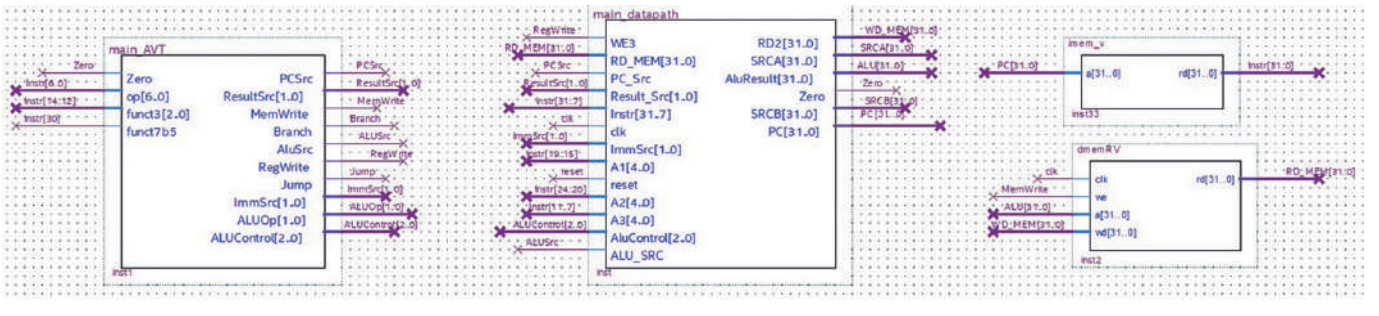


Рис. 4. Верхний уровень иерархии однотактного процессорного ядра RISC-V в САПР Quartus II

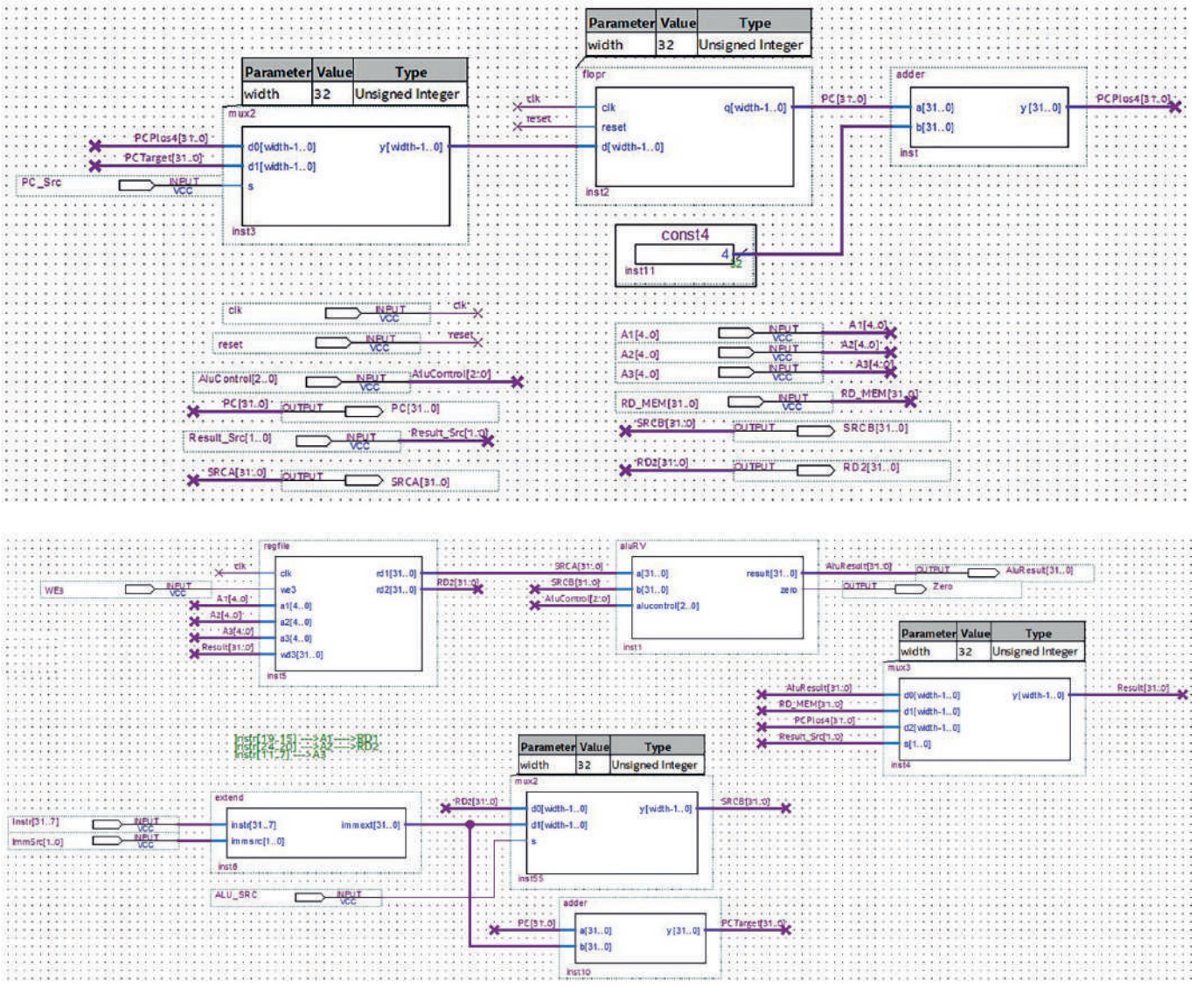


Рис. 5. Тракт данных однотактного процессорного ядра RISC-V в САПР Quartus II

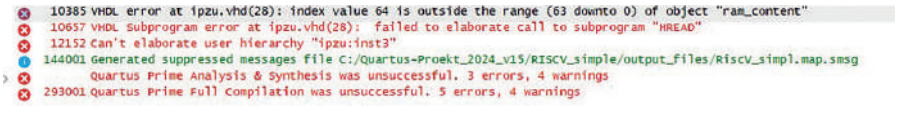


Рис. 6. Ошибка компиляции VHDL-кода ПЗУ из примера 7.14 работы [1]

компиляции исходного VHDL-кода ПЗУ (пример 7.14) возникает ошибка компиляции (рис. 6). Причина неудачной инициализации ОЗУ в проекте заключается в том, что операции ввода-вывода файлов в VHDL не поддерживаются синтезатором в Quartus II. Пример 1 (рис. 7) демонстрирует «подправленный» (сделан по шаблону XST [4, 5]) VHDL-код, который все же проходит безошибочную компиляцию с загрузкой из текстового файла riscvtest.txt, однако mif-файл конфигурации ПЗУ создается пустым (с нулевым содержимым) (пример 2, рис. 8). Это говорит о том, что VHDL-код ПЗУ с загрузкой из файла является не синтезируемым в САПР Quartus II, но может быть использован, например, в САПР Xilinx.

Воспользоваться встроенной мегафункцией ПЗУ ROM: 1PORT не получилось по причине ограничения разрядности адресной шины и того, что ПЗУ работает в синхронном режиме.

В руководстве пользователя [4] указано, что Quartus II поддерживает системные команды $readmem и $readmemh в Verilog для инициализации памяти с помощью текстового файла. Поэтому было принято решение использовать оригинальный SystemVerilog-код ПЗУ (пример 7.14). На рис. 4 показан проект процессорного

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use std.textio.all;
entity rom_prog is
 port(
  addr : in  std_logic_vector(31 downto 0);
         dout : out std_logic_vector(31 downto 0));
end entity;
architecture syn of rom_prog is
 type RomType is array(0 to 63) of bit_vector(31 downto 0);
 impure function InitRomFromFile (RomFileName : in string) return RomType is
  FILE RomFile : text is in RomFileName;
  variable RomFileLine : line;
  variable ROM : RomType;
 begin
  for I in RomType'range loop
   readline (RomFile, RomFileLine);
   hread (RomFileLine, ROM(I));
  end loop;
  return ROM;
  end function;

  signal ROM : RomType := InitRomFromFile("riscvtest.txt");
begin
  process (addr)
  begin
    dout <= to_stdlogicvector(ROM(conv_integer(addr)));
  end process;
  end syn;
```

Рис. 7. Пример 1: VHDL-код ПЗУ, который проходит безошибочную компиляцию в САПР Quartus II, но с генерацией «пустого» mif-файла

```
WIDTH=32;
DEPTH=64;
ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;
CONTENT BEGIN
20:  00000000000000000000000000000000;
     19:   00000000000000000000000000000000;
     18:   00000000000000000000000000000000;
     17:   00000000000000000000000000000000;
     16:   00000000000000000000000000000000;
     15:   00000000000000000000000000000000;
     14:   00000000000000000000000000000000;
     13:   00000000000000000000000000000000;
     12:   00000000000000000000000000000000;
     11:   00000000000000000000000000000000;
     10:   00000000000000000000000000000000;
     9:    00000000000000000000000000000000;
     8:    00000000000000000000000000000000;
     7:    00000000000000000000000000000000;
     6:    00000000000000000000000000000000;
     5:    00000000000000000000000000000000;
     4:    00000000000000000000000000000000;
     3:    00000000000000000000000000000000;
     2:    00000000000000000000000000000000;
     1:    00000000000000000000000000000000;
     0:    00000000000000000000000000000000;
END;
```

Рис. 8. Пример 2: фрагмент сгенерированного «пустого» mif-файла из VHDL-кода ПЗУ

```
--begin_signature
--imem_v
--end_signature
WIDTH=32;
DEPTH=64;
ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;
CONTENT BEGIN
20:  00000000001000010000000001100011;
     19:   00000010001000011010000000100011;
     18:   00000000100100010000000100110011;
     17:   00000000000100000000000100010011;
     16:   00000000100000000000000111101111;
     15:   00000000010100010000010010110011;
     14:   00000110000000000010000100000011;
     13:   00000100011100011010101000100011;
     12:   01000000001000111000001110110011;
     11:   00000000010100100000001110110011;
     10:   00000000001000111010001000110011;
     9:    00000000000000000000001010010011;
     8:    00000000000000010000010001100011;
     7:    00000000010000011010001000110011;
     6:    00000100111001010001000001100011;
     5:    00000000010000101000001010110011;
     4:    00000000010000011111001010110011;
     3:    00000000001000111110001000110011;
     2:    11111111011100011000001110010011;
     1:    00000000110000000000000110010011;
     0:    00000000010100000000000100010011;
END;
```

Рис. 9. Пример 3: фрагмент сгенерированного mif-файла из SystemVerilog-кода ПЗУ

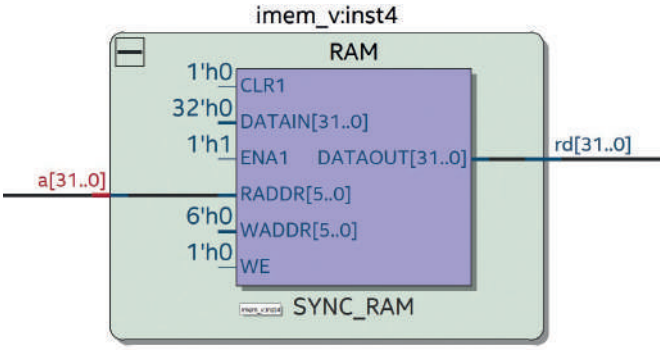


Рис. 10. RTL-представление из SystemVerilog-кода ПЗУ

ядра RISC-V в САПР Quartus II с использованием ПЗУ и ОЗУ на SystemVerilog.

Пример 3 (рис. 9) показывает, что САПР генерирует заполненный mif-файл (64 строки (слова) по 32 бита в каждой; система счисления содержимого DATA_RADIX представлена в двоичном формате BIN, а содержимого адресов – в беззнаковом десятичном UNS) из SystemVerilog-кода ПЗУ. Рис. 10 показывает, что асинхронное ПЗУ организуется на базе синхронного ОЗУ с 6-разрядной адресной шиной RADDR[5..0].

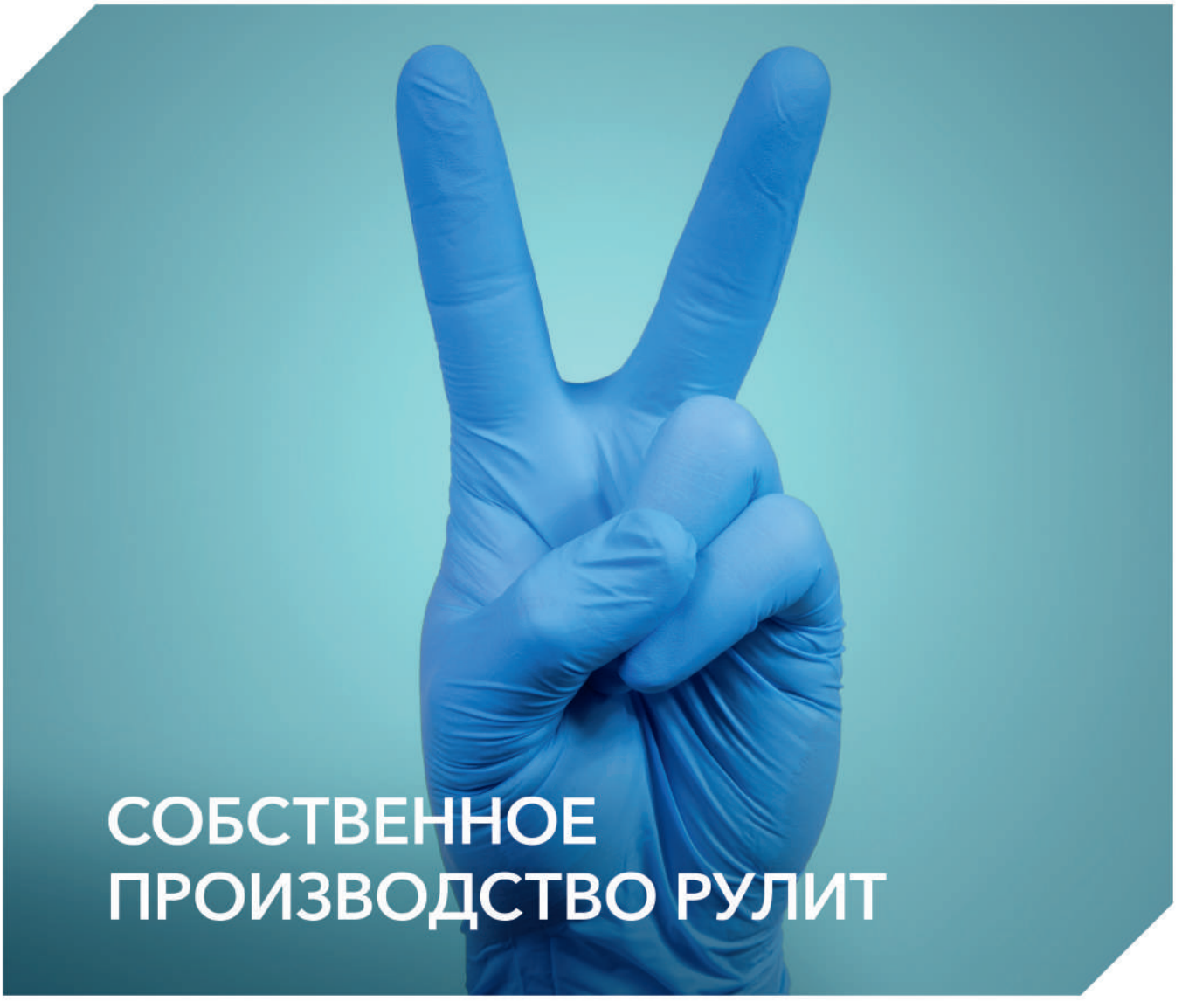
Тем не менее, можно организовать инициализацию ПЗУ ([5]) непосредственно из VHDL-кода (пример 4, рис. 11).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use std.textio.all;
entity rom_prog is
 port(
  addr : in  std_logic_vector(31 downto 0);
         dout : out std_logic_vector(31 downto 0));
end entity;
architecture syn of rom_prog is
 type RomType is array(0 to 80) of bit_vector(31 downto 0);
 impure function InitRomFromFile (RomFileName : in string) return RomType is
  FILE RomFile : text is in RomFileName;
  variable RomFileLine : line;
  variable ROM : RomType;
 begin
  for I in RomType'range loop
   readline (RomFile, RomFileLine);
   hread (RomFileLine, ROM(I));
  end loop;
  return ROM;
 end function;
 signal ROM : RomType :=

 (X"00500113", X"00000000", X"00000000", X"00000000", X"00C00193",
  X"00000000", X"00000000", X"00000000", X"FF718393",
  X"00000000", X"00000000", X"00000000", X"0023E233",
  X"00000000", X"00000000", X"00000000", X"0041F2B3",
  X"00000000", X"00000000", X"00000000", X"004282B3",
  X"00000000", X"00000000", X"00000000", X"02728863",
  X"00000000", X"00000000", X"00000000", X"0041A233",
  X"00000000", X"00000000", X"00000000", X"00020463",

 X"00000000", X"00000000", X"00000000", X"00000293",
 X"00000000", X"00000000", X"00000000", X"0023A233",
 X"00000000", X"00000000", X"00000000", X"005203B3",
 X"00000000", X"00000000", X"00000000", X"402383B3",
 X"00000000", X"00000000", X"00000000", X"0471AA23",
 X"00000000", X"00000000", X"00000000", X"06002103",
 X"00000000", X"00000000", X"00000000", X"005104B3",
 X"00000000", X"00000000", X"00000000", X"008001EF",
 X"00000000", X"00000000", X"00000000", X"00100113",
 X"00000000", X"00000000", X"00000000", X"00910133",
 X"00000000", X"00000000", X"00000000", X"0221A023",
 X"00000000", X"00000000", X"00000000", X"00210063");
 begin
  process (addr)
  begin
   dout <= to_stdlogicvector(ROM(conv_integer(addr)));
        -- dout <= ROM(conv_integer(addr));
  end process;
  end syn;
```

Рис. 11. Пример 4: инициализация ПЗУ непосредственно из HDL-кода



Вы давно знаете нас как надежного и проверенного поставщика материалов для электронной промышленности. Сегодня мы перешли на следующий уровень и стали их производителем. Сделанные нами материалы уже применяются более чем в тысяче техпроцессов на российских производствах. Мы убеждены, что современные отечественные материалы не должны уступать ведущим мировым брендам по своим техническим и эксплуатационным характеристикам. И активно работаем над этим — в собственной лаборатории и на нашем производстве в России.

ГИДРОНОЛ — собственная линейка эффективных жидкостей для отмывки печатных узлов, очистки трафаретов и оборудования. Разработанные с учетом специфики российского производства высококачественные жидкости подходят для любого известного техпроцесса и технологии. Всегда в наличии независимо от условий и обстоятельств. С решениями Гидронол процесс отмывки полностью в ваших руках.

Сделано нами — сделано на совесть.

ostec
группа компаний

сделано нами

ОСТЕК-ИНТЕГРА
Технологические материалы для электронной промышленности
+7 495 788-44-44 | ostec-materials.ru

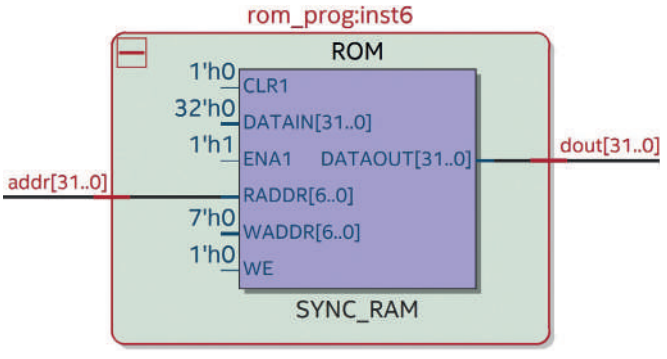


Рис. 12. RTL-представление из VHDL-кода ПЗУ (Пример 4)

В этом случае нужно будет заполнять дистанцию между реальными командами тремя пустыми значениями, например, X"00500113", X"00000000", X"00000000", X"00000000", X"00C00193", по причине того, что счетчик команд увеличивает свое содержимое на 4, поскольку команды в RISC-V являются 4-байтовыми. В этом случае потребуется 80 строк и использование 7-разрядной адресной шины RADDR[6..0] (рис. 12). Пример 5 (рис. 13) демонстрирует фрагмент сгенерированного mif-файла при инициализации ПЗУ непосредственно в HDL- коде.

Выявилась также проблема компиляции ОЗУ данных. VHDL-код из работ [2, 3] используется так же и для реализации ОЗУ MIPS-процессора [6]:

- mips.vhd
- From Section 7.6 of Digital Design & Computer Architecture
- Updated to VHDL 2008 26 July 2011 David_Harris@hmc.edu.

В VHDL-коде ОЗУ (пример 7.15 из работ [2, 3]) используется to_integer, оператор wait on и 8-разрядная адресная шина a(7 downto 2) (пример 6, рис. 14). Первые две причины приводят к ошибкам компиляции в САПР Quartus II, которые можно исправить, заменив to_integer на conv_integer с соответствующим пакетом IEEE.std_logic_unsigned.all, и использовать один оператор process на запись и считывание (пример 7, рис. 15).

Использование 8-разрядной адресной шины a(7 downto 2) ОЗУ данных приводит к неправильному функционированию процесса. Поэтому был

```
--begin_signature
--rom_prog
--end_signature
WIDTH=32;
DEPTH=128;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
     80:    00000000001000010000000001100011;
     …
     8:     11111111011100011000001110010011;
     7:     00000000000000000000000000000000;
     6:     00000000000000000000000000000000;
     5:     00000000000000000000000000000000;
     4:     00000000110000000000000110010011;
     3:     00000000000000000000000000000000;
     2:     00000000000000000000000000000000;
     1:     00000000000000000000000000000000;
     0:     00000000010100000000000100010011;
END;
```

Рис. 13. Пример 5: фрагмент сгенерированного mif-файла при инициализации ПЗУ непосредственно в HDL-коде

```
-- чтение или запись в память
 loop
if rising_edge(clk) then
       if (we = '1') then mem(to_integer(a(7 downto 2))) := wd;
       end if;
      end if;
      rd <= mem(to_integer(a(7 downto 2)));
      wait on clk, a;
     end loop;
```

Рис. 14. Пример 6: фрагмент VHDL-кода ОЗУ данных (пример 7.15, работы [1, 2])

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;
use IEEE.std_logic_unsigned.all;
entity dmem_heris is
 port(clk, we: in STD_LOGIC;
 a, wd: in STD_LOGIC_VECTOR(31 downto 0);
 rd: out STD_LOGIC_VECTOR(31 downto 0));
end;
architecture a of dmem_heris is
 type ramtype is array (63 downto 0) of
 STD_LOGIC_VECTOR(31 downto 0);
 signal mem: ramtype;
 begin
-- чтение или запись в память
process(clk,we,a)
 begin
if rising_edge(clk) then
 if (we = '1') then mem(conv_integer(a(31 downto 2))) <= wd;
 end if;
 end if;
 rd <= mem(conv_integer(a(31 downto 2)));
end process;
end a;
```

Рис. 15. Пример 7: фрагмент VHDL-кода ОЗУ данных с одним оператором process на запись и считывание

ХОЛДИНГ ПЛАНАР | 60 лет
1962—2022

ОптоЭлектронные Системы
Открытое акционерное общество

Посты оптического микроконтроля

- Микроскоп МИКРО 200

- Высокоразрешающий комплекс ГУФ

- Комплекс автоматизированный МА-300

- Комплекс аналитический «ФОТОН»

тел.: (+375 17) 377-90-64, e-mail: office@optes.by

ОАО «Оптоэлектронные системы»
220033, Республика Беларусь, г. Минск, Партизанский пр-т 2, корп. 2-15;
факс.: +375 17 272-71-21; тел: + 375 17 377-90-64; www.optes.by; email: office@opes.by

optes.by

```
module aluRV(input  logic [31:0] a, b,
input  logic [2:0]  alucontrol,
output logic [31:0] result,
output logic     zero);
 logic [31:0] condinvb, sum;
 logic     v;        // overflow
 logic     isAddSub;      // true when is add or subtract operation
 assign condinvb = alucontrol[0] ? ~b : b;
 assign sum = a + condinvb + alucontrol[0];
 assign isAddSub = ~alucontrol[2] & ~alucontrol[1] |
           ~alucontrol[1] & alucontrol[0];
 always_comb
  case (alucontrol)
   3'b000:  result = sum;      // add
   3'b001:  result = sum;      // subtract
   3'b010:  result = a & b;    // and
   3'b011:  result = a | b;    // or
   3'b100:  result = a ^ b;    // xor
   3'b101:  result = sum[31] ^ v; // slt
   3'b110:  result = a << b[4:0]; // sll
   3'b111:  result = a >> b[4:0]; // srl
   default: result = 32'bx;
  endcase
 assign zero = (result == 32'b0);
 assign v = ~(alucontrol[0] ^ a[31] ^ b[31]) & (a[31] ^ sum[31]) & isAddSub;
endmodule
```

Рис. 16. Пример 8: код АЛУ с поддержкой команды SLT

использован оригинальный SystemVerilog-код ОЗУ с 32-разрядной адресной шиной. Как вариант можно использовать доработанный VHDL-код (см. пример 7, рис. 15).

В работах [2, 3] разработка VHDL-кода АЛУ вынесено в самостоятельную задачу. Однако, код АЛУ можно взять из [6] от MIPS-процессора (пример 8, рис. 16). VHDL-код в точности соответствует схеме на рис. 5.18, б из работы [2]. АЛУ поддерживает команду SLT (set if less than – «установить, если меньше, чем», когда A < B, Result = 1), а также логические сдвиги влево (LSL) или вправо (LSL).

В работе [2] приведена программа тестбенча (обьект testbench, пример 7.12). Он проверяет наличие записи в память данных и сообщает об успехе, если правильное значение (25) записано по адресу 100.

Но мы не будем его использовать, а подтвердим правильность разработки функциональным моделированием с использованием симулятора ModelSim Altera Starter Edition и векторного редактора. Функциональное моделирование показывает, что тест проходит успешно (рис. 17). Значение 25 (0×19) записывается в ОЗУ по адресу 100 (0×64). Проект работает на максимальной частоте 58 МГц (табл. 1).

Проверить правильность выполнения команд можно с помощью бесплатной онлайн-программы «кодер/декодер команд RISC-V» [7] (рис. 18). На рис. 18 показан перевод команды FF718393 (третья команда с адресом 8 на рис. 3), представленной в шестнадцатеричном формате, в ассемблерный код addi x7, x3, –9. В результате выполнения этой команды в регистр с адресом x7 будет занесено число 3: x7 = (12–9) = 3 (см. рис. 17).

Команда FF718393 представляется в двоичном коде с выделением полей команд процессора RV32I. На рис. 18 видно, что это команда относится к типу I. Если сравнить

Таблица 1. Используемые ресурсы ПЛИС Cyclone V 5CGXFC7C7F23CB

Проект	Логические ресурсы			Максимальная тактовая частота синхросигнала, МГц (модель Slow 1100 мВ, 85 °С)
	Адаптивные логические модули (ALM)	Адаптивные таблицы перекодировок (ALUT) для реализации комбинационных функций	Выделенные триггеры логических элементов	
Асинхронное ПЗУ и синхронное ОЗУ на SystemVerilog	2347	1773	3008	58

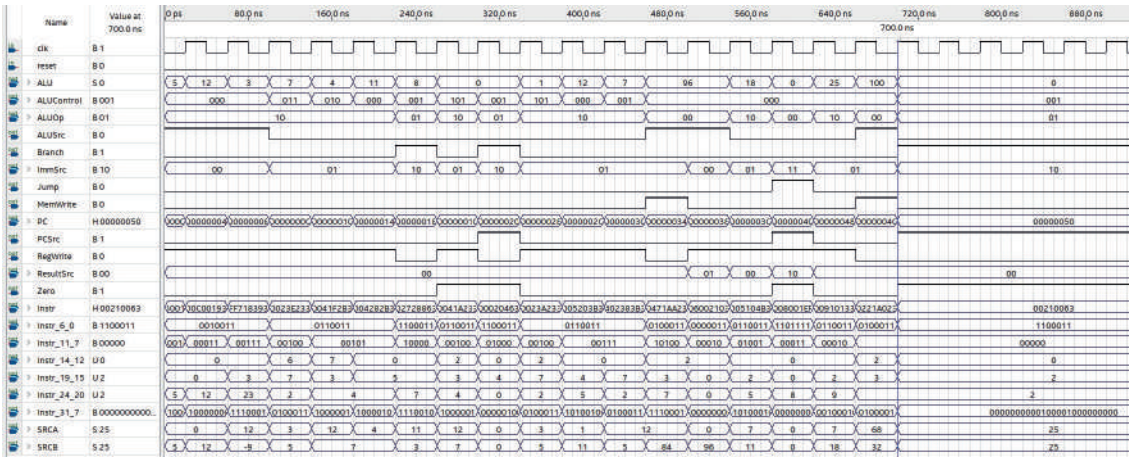


Рис. 17. Временная диаграмма успешного прохождения теста, представленного на рис. 3

рис. 17 и 18, то можно увидеть, что поля команды, формируемые проектом процессора RISC-V в САПР Quartus II, правильно заполнены значениями.

ЗАКЛЮЧЕНИЕ

Несмотря на незначительные проблемы с адаптацией HDL-кода однотактного процессорного ядра RISC-V, в данной работе удалось успешно реализовать его в базисе ПЛИС Cyclone V с применением САПР Altera Quartus II. При этом максимальная рабочая частота данной реализации составила 58 МГц.

Потребовалась доработка VHDL-кода ПЗУ команд и ОЗУ данных, а также замена разрядности шин данных в некоторых функциональных блоках с 8 на 32 (регистр с сигналом сброса flopr, пример 7.8 [2]; шинный мультиплексор 2:1, пример 7.10 [2]; шинный мультиплексор 3:1, пример 7.11 [2]).

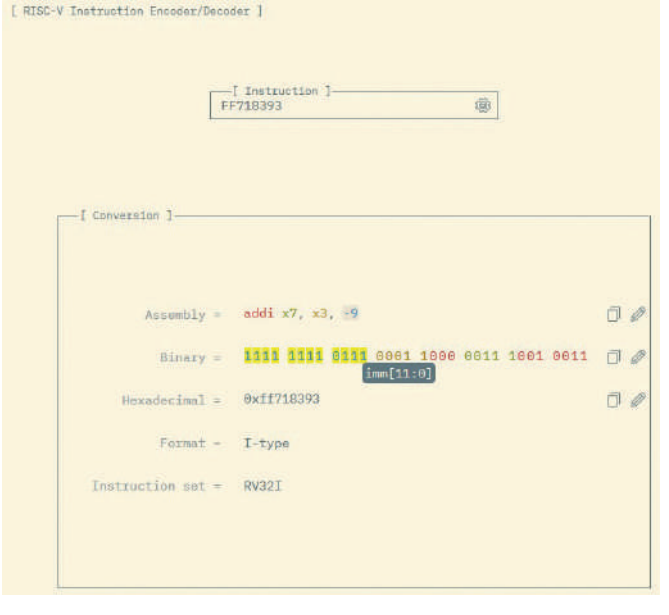


Рис. 18. Окно онлайн-программы «Кодер/декодер команд RISC-V», позволяющей переводить машинный код в ассемблерный с выделением полей команд в двоичном коде

ЛИТЕРАТУРА

1. **Строгонов А.В., Бордюжа О.Л., Строгонов А.И.** Эффективный подход в разработке управляющих автоматов микропроцессорных ядер // ЭЛЕКТРОНИКА: Наука, Технология, Бизнес. 2024. № 1. С. 78–86.
2. **Харрис С.Л., Харрис Д.** Цифровая схемотехника и архитектура компьютера RISC-V / Пер. с англ. В.С. Яценкова, А.Ю. Романова; под ред. А.Ю. Романова. М.: ДМК Пресс, 2021. 810 с.
3. **Harris S.L., Harris D.** Digital Design and Computer Architecture RISC-V Edition. 2022. ISBN: 978-0-12-820064-3.
4. Intel Quartus Prime Pro Edition User Guide. Design Recommendations. UG-20131. ID: 683082. Version: 2021.10.04.
5. XST User Guide. 10.1. ROMs Using Block RAM Resources HDL Coding Techniques // www.xilinx.com.
6. https://pastebin.com/ew0SACWy.
7. https://luplab.gitlab.io/rvcodecjs/#q=00C00193&abi=false&isa=AUTO.