ТЕХНИЧЕСКИ УНИВЕРСИТЕТ-ГАБРОВО

Факултет „Електротехника и електроника”

маг. инж. Илиан Цвятков Върбов

МОДЕЛИРАНЕ И СИМУЛИРАНЕ НА КОМПОНЕНТИ НА КОМПЮТЪРНИТЕ СИСТЕМИ

А В Т О Р Е Ф Е Р А Т

на дисертация

за придобиване на образователна и научна степен „доктор”

Област на висше образование: 5. „Технически науки”

Професионално направление: 5.3. „Комуникационна и компютърна техника”

Докторска програма: „Автоматизация на инженерния труд и системи за автоматизирано проектиране”

Габрово, 2026 г.

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ-ГАБРОВО

Факултет „Електротехника и електроника”

маг. инж. Илиан Цвятков Върбов

МОДЕЛИРАНЕ И СИМУЛИРАНЕ НА КОМПОНЕНТИ НА КОМПЮТЪРНИТЕ СИСТЕМИ

А В Т О Р Е Ф Е Р А Т

на дисертация

за придобиване на образователна и научна степен „доктор”

Област на висше образование: 5. „Технически науки”

Професионално направление: 5.3. „Комуникационна и компютърна техника”

Докторска програма: „Автоматизация на инженерния труд и системи за автоматизирано проектиране”

Научен ръководител: доц. д-р инж. Валентина Стоянова Кукенска

Рецензенти: Проф. д-р инж. Станислав Денчев Симеонов
Проф. д-р инж. Мирослав Николов Гълъбов

Габрово, 2026 г.

Дисертационният труд е обсъден и насочен за официална защита на заседание на Разширен катедрен съвет на катедра „Компютърни системи и технологии” към факултет „Електротехника и електроника” на Технически университет – Габрово, проведен на 17.03.2026 г.

Дисертационният труд съдържа 197 страници. Научното съдържание е представено в увод, 4 глави и две приложения и включва 100 фигури и 42 таблици. Цитирани са 47 литературни източника. Номерацията на фигурите, таблиците и формулите в автореферата е в съответствие с тази в дисертацията.

Разработката и изследванията по дисертационния труд са извършени в катедра „Компютърни системи и технологии” към факултет „Електротехника и електроника” на Технически университет – Габрово.

Официалната защита на дисертационния труд ще се състои на 12.06.2026 г. от 13 ч. в зала 3312 на Технически университет – Габрово.

Благодарности:

Благодаря на цялата катедра по „Компютърни системи и технологии“ на Технически Университет - Габрово, за предоставените ресурси и техника, която подпомогна моето изследване.

Благодаря на моя научен ръководител, доц. д-р инж. Валентина Стоянова Кукенска за насърчителната подкрепа и редакторска помощ, които се оказаха решаващи за завършване на настоящия дисертационен труд.

Автор: Илиан Цвятков Върбов

Заглавие: МОДЕЛИРАНЕ И СИМУЛИРАНЕ НА КОМПОНЕНТИ НА КОМПЮТЪРНИТЕ СИСТЕМИ

А. ОБЩА ХАРАКТЕРИСТИКА НА ДИСЕРТАЦИОННИЯ ТРУД

АКТУАЛНОСТ НА ПРОБЛЕМА

Съвременният свят се характеризира с непрекъснато нарастване на изчислителните изисквания и зависимостта от компютърни системи. От мобилни устройства до високопроизводителни сървъри и вградени системи – всички те се основават на сложни хардуерни и софтуерни компоненти, които трябва да функционират бързо, надеждно и енергийно ефективно. За да се постигне това, са необходими задълбочени познания за архитектурите и технологиите, както и ефективни методи за моделиране, симулация и верификация на хардуерните компоненти и компютърни системи.

Развитието на компютърната техника през последните десетилетия преминава през етапи на нарастваща сложност, миниатюризация и интеграция. Традиционните методи за ръчно проектиране и тестване вече не са достатъчни при създаването на съвременни микропроцесори, памети и периферни устройства. Поради това, моделирането и симулирането на хардуерните блокове се превръща в задължителен етап от процеса на проектиране. Това позволява ранно откриване на грешки, оптимизация на архитектурните решения и съкращаване на времето за разработка.

Развитието на програмируемите логически устройства (FPGA), осигурява среда за бързо прототипиране и експериментиране с различни архитектурни решения. Възможността за описване на хардуер чрез специализирани езици, като VHDL, Verilog и TL-Verilog, улеснява реализацията на сложни цифрови системи и прави процеса на проектиране по-гъвкав. Същевременно симулационните среди като Vivado Design Suite, ModelSim, Verilator и Makerchip позволяват наблюдение и анализ на поведението на моделираните системи, което води до по-висока надеждност на крайния продукт.

Значимостта на изследванията в областта на моделирането и симулацията на компютърни системи се свързва с нарастващите изисквания за оптимизация на хардуерните решения и необходимостта от интегриране на нови архитектури. Създаването на универсални, мащабируеми и модулни модели на основни функционални блокове подпомага, както академичните изследвания, така и практическите разработки в индустрията. Особено важно е моделирането на микропроцесорни архитектури, които са в основата на повечето съвременни системи.

Настоящата дисертация е посветена на моделиране и симулиране на основни компоненти на компютърните системи. Изследването поставя акцент върху създаването на функционални и синтезируеми модели, които да бъдат използвани, както за обучение и анализ, така и за практическа реализация върху FPGA платформа.

ЦЕЛ И ЗАДАЧИ НА ДИСЕРТАЦИОННИЯ ТРУД

Основната цел на дисертацията е да се разработят и анализират модели на основни компоненти на компютърни системи с използване на съвременни езици за описание на хардуер и среди за симулация, да се създаде и имплементира модел на микропроцесор с AVR архитектура върху FPGA платформа.

За постигане на целта са поставени следните основни задачи:

1. Да се анализира състоянието на изследванията в областта на моделирането на компютърни системи и компоненти.
2. Да се разработят модели на цифрови схеми и основни компоненти на компютърните системи с използване на езици за описание на хардуер VHDL и Verilog.

3. Да се изследват разработените модели на цифрови схеми и основни компоненти на компютърните системи.
4. Да се разработят модели на микропроцесор с RISC архитектура като се използват езиците за хардуерно описание TL-Verilog, Verilog и VHDL.
5. Да се изследват предложените модели на микропроцесор с RISC архитектура.
6. Да се разработи модел на микропроцесор с AVR архитектура на регистрово ниво.
7. Да се имплементира модел на микропроцесор с AVR архитектура върху FPGA платформа.

МЕТОДИ НА ИЗСЛЕДВАНЕ

В дисертационния труд са използвани съвременни методи за функционално и симулационно моделиране, моделиране на регистрово ниво и компютърна симулация. Верификацията на разработените модели на компоненти на компютърни системи е реализирана посредством специализиран софтуер.

НАУЧНА НОВОСТ

В дисертационния труд са получени следните резултати с елементи на научна новост:

- ✓ Разработени са модели на микропроцесори с RISC архитектура чрез различни HDL езици, позволяващи изследване на тяхната структура и функциониране.
- ✓ Създаден е модел на микропроцесор с AVR архитектура на регистрово-трансферно ниво, осигуряващ възможност за детайлно изследване на работата на основните функционални блокове.
- ✓ Реализирана е FPGA имплементация на разработения модел, което позволява експериментална проверка на неговата работоспособност в реална хардуерна среда.
- ✓ Извършена е оценка на производителността и ресурсната ефективност на разработените модели при FPGA имплементация, която може да се използва при оптимизация на цифрови системи.

ПРИЛОЖИМОСТ

Резултатите от дисертационния труд могат да бъдат използвани за:

- бъдещи изследвания и разработки в областта на моделиране на компютърни архитектури и вградени системи;
- проектиране и реализация на цифрови системи върху FPGA платформи;
- разработване и изследване на модели на компютърни системи и микропроцесорни архитектури чрез езици за описание на хардуер;
- обучението на студенти по компютърни архитектури, цифрови системи, проектиране на цифрови схеми, устройства и системи с програмируема логика.

АПРОБАЦИЯ НА ДИСЕРТАЦИОННИЯ ТРУД

Дисертационната работа е докладвана и обсъждана на катедрен съвет и на разширен катедрен съвет на катедра „Компютърни системи и технологии” при Технически университет – Габрово.

Основните резултати от дисертационния труд са публикувани и докладвани в международни научни конференции в България.

- Международна научна конференция Унитех‘11 – Габрово, 18-19 Ноември 2011.
- Международна научна конференция Унитех‘12 – Габрово, 16-17 Ноември 2012.
- Международна научна конференция Унитех‘14 – Габрово, 21-22 Ноември 2014.
- Международна научна конференция Унитех’23 – Габрово, 20-21 Ноември 2020.

- Международна научна конференция International Conference Automatics and Informatics (ICAI) - Варна, 10 – 12 Октомври 2024.
- Международна научна конференция Унитех‘25 – Габрово,20-22 Ноември 2025.
- Научна конференция „Знание, наука, иновации, технологии“, Велико Търново, 27 Март 2026.

СТРУКТУРА И ОБЕМ НА ДИСЕРТАЦИОННИЯ ТРУД

Дисертационният труд съдържа увод, четири глави, заключение, списък на използваната литература и две приложения с общ обем от 197 страници. Включени са 100 фигури под формата на схеми, графики и диаграми, 42 таблици и 2 формули.

Номерацията на фигурите, таблиците, формулите и цитираната литература в автореферата съответства на тази в дисертационния труд и приложенията.

СЪДЪРЖАНИЕ НА ДИСЕРТАЦИОННИЯ ТРУД

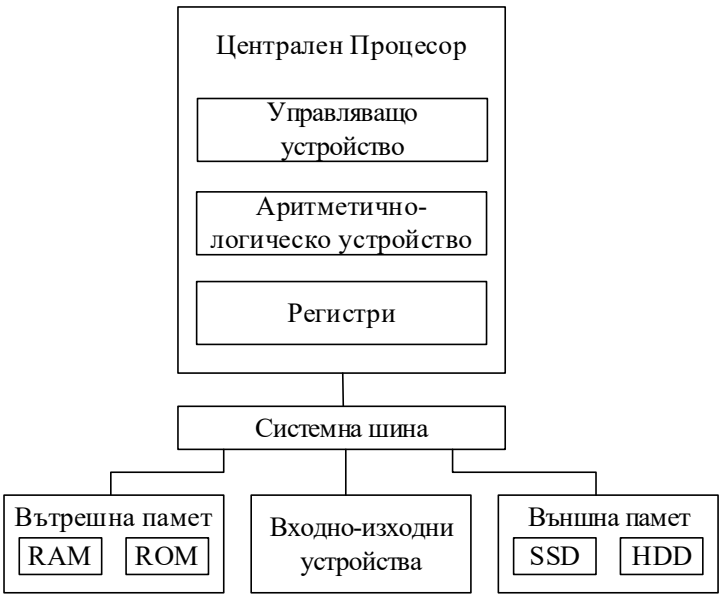
Глава I: Анализ на състоянието по темата на дисертацията

В тази глава е представен литературен обзор на съществуващи разработки в областта на моделирането и симулацията на процесорни ядра и компютърни системи. Анализирани са различни отворени архитектури на микропроцесори и техните реализации чрез езици за описание на хардуер. Основното внимание е насочено към RISC архитектури и процесорни ядра, използвани в академични и изследователски проекти.

Разгледани са различни процесорни реализации като Rocket Chip Generator, VexRiscv, Ibex Core, CV32E40P, OpenRISC OR1200, LEON3, NEORV32, AVR Core, Plasma CPU и OpenSPARC T1. За всяка архитектура са анализирани архитектурните особености, използваните HDL технологии, подходите за моделиране и възможностите за симулация и верификация.

Направеният анализ показва, че голяма част от съществуващите реализации са ориентирани към индустриални приложения и използват сложни методи за проектиране. Това ги прави по-трудни за използване в учебна среда. В същото време те предоставят ценна информация за различни подходи при разработването на процесорни архитектури.

Описани са основните компоненти на компютърните системи (централен процесор, памет, входно-изходни устройства и системна шина) представени на фигура 1.1.



Фигура 1.1 Типична структура на компютърна система

Представени са основните характеристики, приложения и предимства на езиците за описание на хардуер (Hardware Description Languages – HDL). Направен и сравнителен анализ между VHDL, Verilog и TL-Verilog показан на таблица 1.1.

Таблица 1.1. Сравнение между VHDL, Verilog и TL-Verilog

Характеристика	**VHDL**	**Verilog**	**TL-Verilog**
Синтаксис	Строг, близък до Ada/Pascal	По-интуитивен, близък до C	Разширен, абстрактен
Основни предимства	Четимост, формализъм, надеждност	Лесен за усвояване, широко използван	Автоматизация, намаляване на код
Основни недостатъци	Обемен код, по-сложен синтаксис	По-слаба строгост	Ограничена поддръжка и разпространение
Ниво на абстракция	RTL, структурно, поведенческо	RTL, поведенческо	Transaction-Level, RTL

Описани са основните подходи за създаване на модели на компютърни системи, както и ролята на симулацията при анализа на тяхното поведение. Разгледани са различните нива на моделиране, включително поведенческо моделиране и моделиране на регистрово-трансферно ниво (RTL). Направено е сравнение на програмни среди, използвани при моделиране и симулация на цифрови системи (таблица 1.2).

Таблица 1.2. Сравнителен анализ на симулационни среди

Среда	**Поддържани езици**	**Основни предимства**	**Основни недостатъци**	**Типични приложения**
ISim (Xilinx)	VHDL, Verilog	Интеграция с ISE WebPACK, безплатна	Ограничена функционалност, по-бавна симулация	Учебни цели, малки проекти
Vivado Design Suite (Xilinx/AMD)	VHDL, Verilog, SystemVerilog	Съвременен интерфейс, бърза симулация, интеграция със синтез	Високи изисквания към ресурси, лицензиране	FPGA разработка, професионални проекти
ModelSim (Intel/Siemens EDA)	VHDL, Verilog, SystemVerilog	Стандарт в индустрията, мощни отладъчни инструменти	Платена версия с ограничения в безплатната	ASIC/FPGA индустриални проекти
Verilator	SystemVerilog (subset), Verilog	Отворен код, много висока скорост, C++/SystemC интеграция	Липсва пълна поддръжка на VHDL, по-сложна настройка	Научни изследвания, симулация на RISC-V
Makerchip	Verilog, TL-Verilog	Онлайн среда, визуализация в реално време, лесен достъп	Ограничени възможности за сложни проекти, зависимост от интернет	Образование, изследване на конвейерни CPU

Изводи към първа глава

В резултат на извършения анализ са направени следните изводи:

- Моделирането на компоненти на компютърни системи е ключов етап от процеса на проектиране, който позволява предварителна проверка на архитектурни решения, оптимизация на функционални блокове, намаляване на времето и разходите за разработка.
- Микропроцесорите с RISC архитектури са подходящ обект на изследване и моделиране поради своята относителна простота и широката приложимост в практиката и в образователния процес.
- Езиците за описание на хардуер VHDL и Verilog са основни средства за създаване на модели на компютърни системи и компоненти на различни нива на абстракция (системно, функционално, логическо и схемно).
- За моделиране на конвейерни микропроцесорни архитектури е удачно използването на TL-Verilog.
- За моделиране, изследване и имплементация на модели на компоненти на компютърни системи са приложими програмните среди Xilinx Project Navigator, ISim и Makerchip.

Глава II: Моделиране на компоненти на компютърни системи

В тази глава се представени модели на цифрови схеми, памет с произволен достъп и аритметично-логическо устройство. За построяването им са използвани езиците за хардуерно описание VHDL и Verilog. Представените модели са изследвани с Xilinx Project Navigator.

Моделиране на аритметично-логическо устройство

Аритметично-логическо устройство (АЛУ) е основен функционален блок на процесора. За построяване на неговия модел са използвани входно – изходните сигнали представени в таблица 2.1.

Таблица 2.1 Входно-изходни сигнали на моделите

Сигнал	Тип	Посока	Функция
A	std_logic_vector (7 downto 0)	вход	операнд
B	std_logic_vector (7 downto 0)	вход	операнд
Opcode	std_logic_vector (4 downto 0)	вход	избор на операция
Result	std_logic_vector (7 downto 0)	изход	резултат
c_out	std_logic	изход	изходен пренос

Създадени са два модела на АЛУ. Единият модел е базиран на VHDL (фиг.2.2), а другият - на Verilog (фиг. 2.3).

```
architecture Behavioral of ALU_VHDL is
signal Result_tmp : std_logic_vector (7
downto 0);
signal tmp: std_logic_vector (8 downto 0);
begin
    process(A,B,Opcode)
    begin
        case(Opcode) is
            when "00000" => -- ADD
                Result_tmp <= A + B;
            when "00001" => -- SUB
                Result_tmp <= A - B;
            when "00010" => -- MUL
                Result_tmp <=
std_logic_vector(to_unsigned((to_integer(
unsigned(A)) *
to_integer(unsigned(B))),8));
            when "00011" => -- DIV
                Result_tmp <=
std_logic_vector(to_unsigned(to_integer(u
nsigned(A)) / to_integer(unsigned(B)),8));
            when "00100" => -- SLL
```

```
                Result_tmp <=
std_logic_vector(unsigned(A) sll N);
            when "00101" => -- SRL
            Result_tmp <=
std_logic_vector(unsigned(A) srl N);
            when "00110" => -- ROL
                Result_tmp <=
std_logic_vector(unsigned(A) rol N);
            when "00111" => -- ROR
                Result_tmp <=
std_logic_vector(unsigned(A) ror N);
            when "10000" => -- AND
                Result_tmp <= A and B;
            when "10001" => -- OR
                Result_tmp <= A or B;
            when "10010" => -- XOR
                Result_tmp <= A xor B;
            when "10011" => -- NOR
                Result_tmp <= A nor B;
            when "10100" => -- NAND
                Result_tmp <= A nand B;
            when "10101" => -- XNOR
                Result_tmp <= A xnor B;
            when "10110" => -- NOTA
                Result_tmp <= not A;
            when "10111" => -- NOTB
                Result_tmp <= not B;
            when "11000" => -- LT
                if(A<B) then
                    Result_tmp <= x"01";
                else
                    Result_tmp <= x"00";
                end if;
            when "11001" => -- EQ
                if(A=B) then
                    Result_tmp <= x"01";
                else
                    Result_tmp <= x"00";
                end if;
            when "11010" => -- GT
                if(A>B) then
                    Result_tmp <= x"01";
                else
                    Result_tmp <= x"00";
                end if;
            when others => Result_tmp <=
x"00";
        end case;
    end process;
    Result <= Result_tmp;
    tmp <= ('0' & A) + ('0' & B);
    c_out <= tmp(8); -- Carryout
end Behavioral;
```

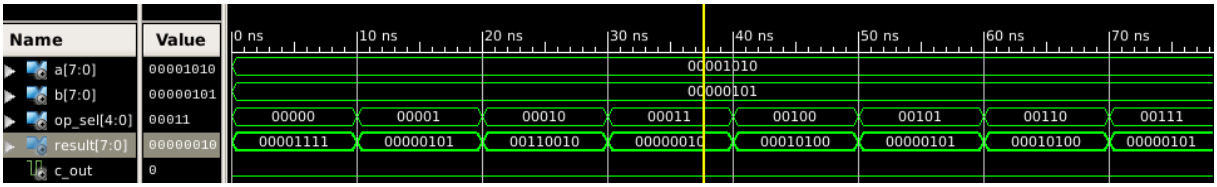
Фигура 2.2 VHDL модел на аритметично-логическо устройство

```
assign Result = Result_tmp;
    assign tmp = {1'b0,A} + {1'b0,B};
    assign c_out = tmp[8]; // Carryout
    always @(*)
    begin
        case(Opcode)
            5'b00000: // ADD
                Result_tmp = A + B;
            5'b00001: // SUB
                Result_tmp = A - B;
            5'b00010: // MUL
                Result_tmp = A * B;
            5'b00011: // DIV
                Result_tmp = A / B;
            5'b00100: // SLL
                Result_tmp = A << 1;
            5'b00101: // SRL
                Result_tmp = A >> 1;
            5'b00110: // ROL
                Result_tmp = {A[6:0],
A[7]};
            5'b00111: // ROR
                Result_tmp = {A[0],
A[7:1]};
            5'b10000: //  AND
                Result_tmp = A & B;
            5'b10001: //  OR
                Result_tmp = A | B;
            5'b10010: //  XOR
                Result_tmp = A ^ B;
            5'b10011: //  NOR
                Result_tmp = ~(A | B);
            5'b10100: //  NAND
                Result_tmp = ~(A & B);
            5'b10101: //  XNOR
                Result_tmp = ~(A ^ B);
            5'b10110: //  NOTA
                Result_tmp = ~(A ^ B);
```

```
            5'b10111: //  NOTB
                Result_tmp = ~(A ^ B);
            5'b11000: // GT
                Result_tmp = (A >
B)?8'd1:8'd0;
            5'b11001: // EQ
                Result_tmp = (A ==
B)?8'd1:8'd0;
            5'b11010: // LT
                Result_tmp = (A <
B)?8'd1:8'd0;
                default: Result_tmp = 5'b00000;
        endcase
    end
endmodule
```

Фигура 2.3 Verilog модел на аритметично-логическо устройство.

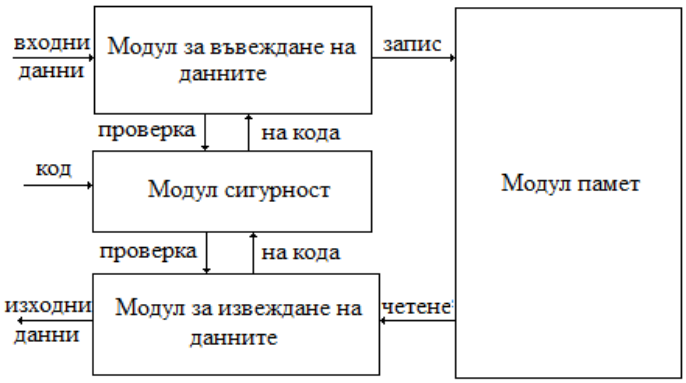
Част от получените резултати от симулацията на първия модел на АЛУ са представени на фиг. 2.4.



Фигура 2.4 Симулация на VHDL модела на АЛУ

Моделиране на памет с произволен достъп

Един от най-важните компоненти за изграждане на една компютърна система е оперативната памет (ОП). В блоковата схема на RAM паметта могат да се обособят четири основни модула – за въвеждане на данни, за сигурност, за извеждане и модул памет (фиг.2.5).



Фигура 2.5 Блокова схема на RAM памет

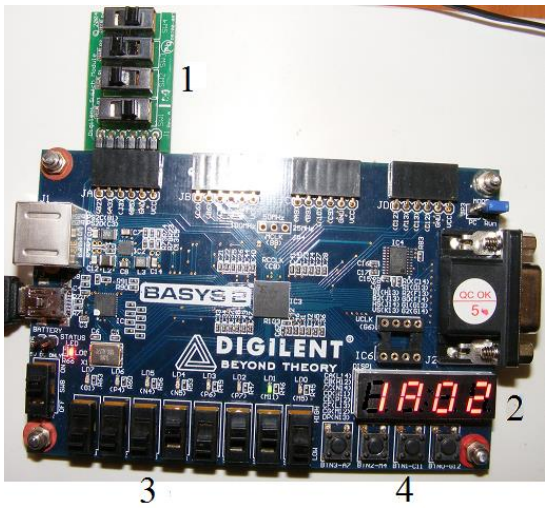
Разработен е модел на памет с произволен достъп с езика за хардуерно описание VHDL. На фигура 2.6 е представена част от него.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use
IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ram is
port (
        address : in integer range 0 to 15;
        ce : in std_logic;
        re : in std_logic;
        clr : in std_logic;
    datain : in std_logic_vector(7 downto 0);
     enck : in std_logic_vector(3 downto 0);
     encr : in std_logic_vector(3 downto 0);
```

```
    x : out std_logic_vector(3 downto 0);
    l: out std_logic_vector(3 downto 0);
    y : out std_logic_vector(3 downto 0)
        );
end ram;
architecture Behavioral of ram is
signal dataout : std_logic_vector(7 downto
0);
signal we : std_logic;
begin
memory : process (address, ce, we, re, clr,
enck, encr, datain) is
type ram_array is array (integer range 0 to
15) of std_logic_vector (7 downto 0);
variable mem : ram_array;
begin
we <= '1';
if clr = '1' then
dataout <= (others => 'Z');
elsif (ce = '0') then
    if (re = '1' and enck = encr) then
        mem (address) := datain;
    elsif (we = '1' and enck = encr) then
        l <= "1010";
        dataout <= mem(address);
    else
    dataout <= not mem(address);
    l <="1111";
    end if;
end if;
end process memory;
x <= dataout(7 downto 4);
y <= dataout(3 downto 0);
end Behavioral;
```

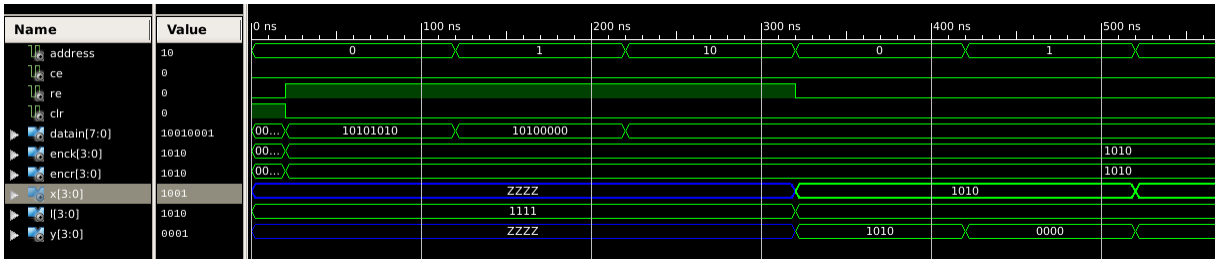
Фигура 2.6 VHDL модел на модула памет

Тестването на предложения модел на памет с произволен достъп е реализирано с развойна платка Basys 2 (фиг. 2.7).



Фигура 2.7 Визуализация на адрес „1“ с записана в него информация „02“.

Моделът е изследван с програмната средата ISim. Част от резултатите от симулацията са представени на фиг. 2.9.

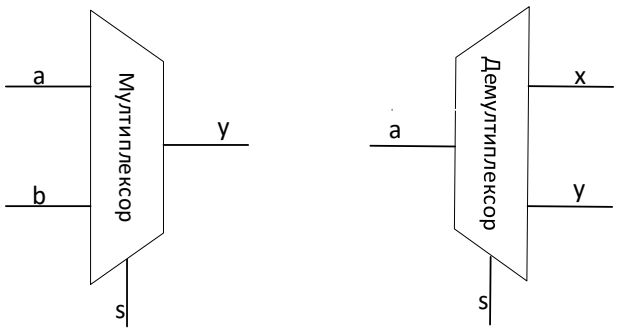


Фигура 2.9 Резултати от симулацията на VHDL модела на RAM

Създадени са по два модела на цифрови схеми (мултиплексор, демултиплрксор, шифратор, дешифратор, полусуматор, пълен суматор, брояч, регистър и компаратор) на VHDL и Verilog. Изследвани са чрез средата ISim на Xilinx. Моделите са синтезирани и имплементирани в FPGA.

Моделиране на мултиплексор

Блокови схеми на мултиплексор и дешифратор са показани на фигура 2.12.



Фигура 2.12 Схема на мултиплексор и демултиплексор

Разработените модели на мултиплексор са представени на фиг. 2.16.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MUX8b_VHDL is
    Port ( a,b : in  STD_LOGIC_VECTOR
(7 downto 0);
           s : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR (7
downto 0));
end MUX8b_VHDL;

architecture Behavioral of MUX8b_VHDL
is
begin
    process(s,a,b) begin
        if (s = '0') then
            y <= a;
        else
            y <= b;
        end if;
    end process;
end Behavioral;
```

a) VHDL

```
module MUX8b_Verilog(
    input [7:0] a,b,
    input s,
    output reg [7:0] y );

    always @ (s,a,b)
        if (s == 0)
            y = a;
        else
            y = b;
endmodule
```

b) Verilog

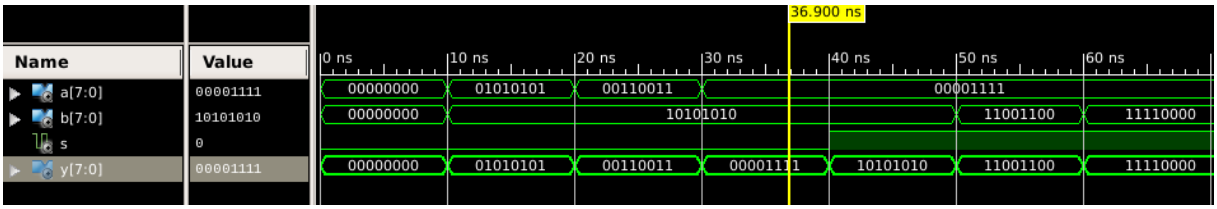
Фигура 2. 16 Модели на 8 битов мултиплексор 2x1

В таблица 2.6 са представени използваните ресурси от програмируемата логическа интегрална схема.

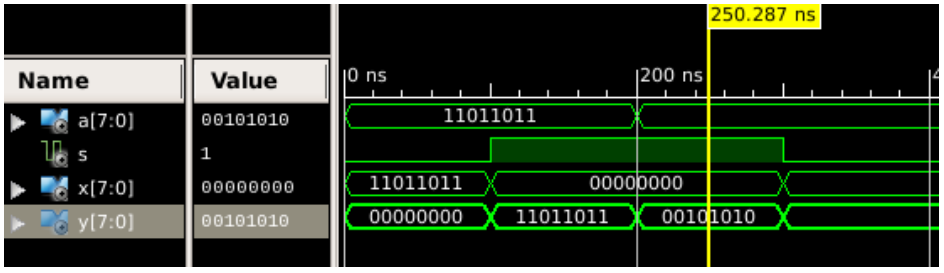
Таблица 2.6 Използвани ресурси от FPGA след синтеза

Използвани ресурси от FPGA	VHDL и Verilog
Брой на слайсовете	4
Брой на 4-входни LUTs:	8
Брой на свързани IOBs:	25

Част от резултатите от симулациятиите на моделите на 8-битовия мултиплексор са представени на фигура 2.17 и 2.20.



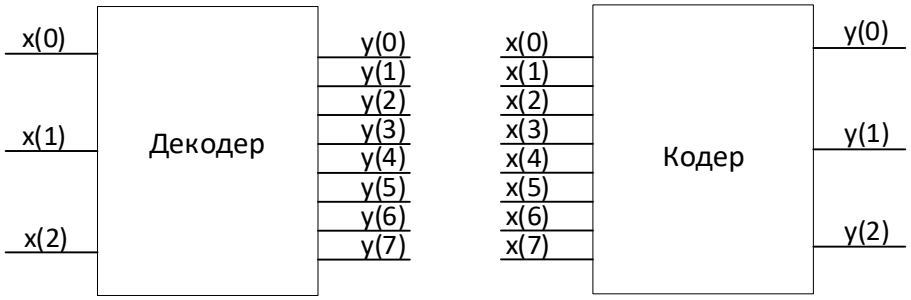
Фигура 2.17 Симулация на модел на 8 битов мултиплексор



Фигура 2.20 Симулация на Демултиплексор

Моделиране на дешифратор

Блоковите схеми на шифратор и дешифратор са показани на фиг.2.21, а на фиг.2.22 разработените модели.



Фигура 2.21 Схема на дешифратор и шифратор

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DEC3x8_VHDL is
    Port ( x : in  STD_LOGIC_VECTOR (2
downto 0);
           y : out  STD_LOGIC_VECTOR (7
downto 0));
end DEC3x8_VHDL;

architecture Behavioral of DEC3x8_VHDL
is
begin
    process (x) begin
        case x is
            when "000" => y <= "00000001";
```

```
        when "001" => y <= "00000010";
        when "010" => y <= "00000100";
        when "011" => y <= "00001000";
        when "100" => y <= "00010000";
        when "101" => y <= "00100000";
        when "110" => y <= "01000000";
        when "111" => y <= "10000000";
        when others => y <= "00000000";
    end case;
  end process;
end Behavioral;
```

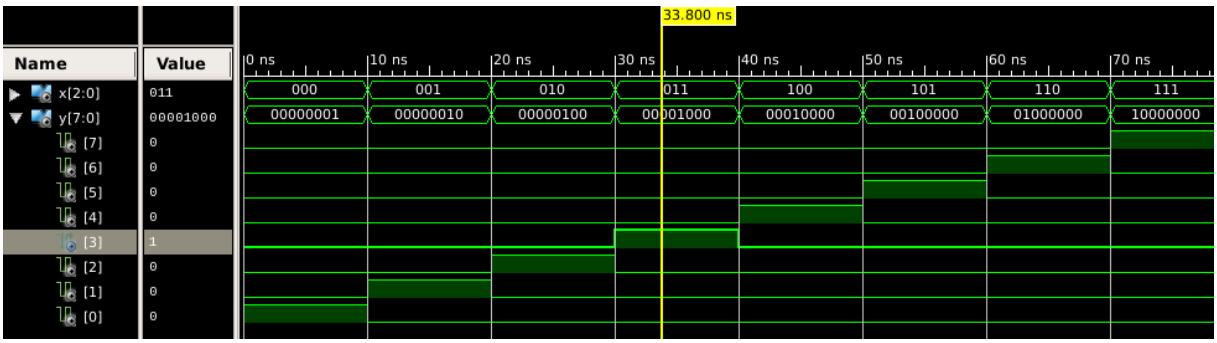
a) VHDL

```
module DEC3x8_Verilog(
   input [2:0] x,
   output reg [7:0] y
   );
initial
   y = 8'b0;
always @ (x)
  case(x)
   3'b000 : y = 8'b00000001;
   3'b001 : y = 8'b00000010;
   3'b010 : y = 8'b00000100;
   3'b011 : y = 8'b00001000;
   3'b100 : y = 8'b00010000;
   3'b101 : y = 8'b00100000;
   3'b110 : y = 8'b01000000;
   3'b111 : y = 8'b10000000;
    endcase
endmodule
```

b) Verilog

Фигура 2.22 Модели на дешифратор 3 към 8

На фиг. 2.23 са представени част от резултатите от симулацията на моделите на дешифратора.



Фигура 2.23 Симулация на дешифратор

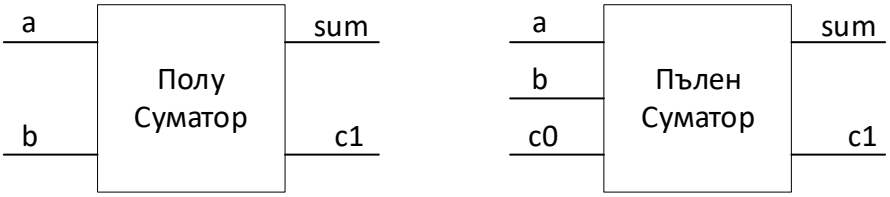
Моделите от фигура 2.22 са използвани за синтез и имплементация в FPGA. В таблица 2.10 са представени използваните ресурси от интегралната схема.

Таблица 2.10 Използвани ресурси от моделите на дешифратора

Използвани ресурси от FPGA	VHDL и Verilog
Брой на слайсовете	4
Брой на 4-входни LUTs:	8
Брой на свързани IOBs:	11

Пълен суматор

Блоковите схеми на полусуматор и пълен суматор са показани на фигура.2.28. Разработените са модели на пълен суматор представени на фиг.2.31.



Фигура 2.28 Схема на полу суматор и пълен суматор

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use
IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TEMP_VHDL is
    Port ( a, b : in std_logic_vector (1
downto 0);
           c_in: in std_logic;
           sum : out std_logic_vector (1
downto 0);
           c_out: out std_logic);
end TEMP_VHDL;

architecture Behavioral of TEMP_VHDL
is
signal tmp : std_logic_vector(2 downto 0);
begin
    tmp <= ('0' & a) + ('0' & b) + c_in;
    sum <= tmp(1 downto 0);
    c_out <= tmp(2);
end Behavioral;
```

a) VHDL

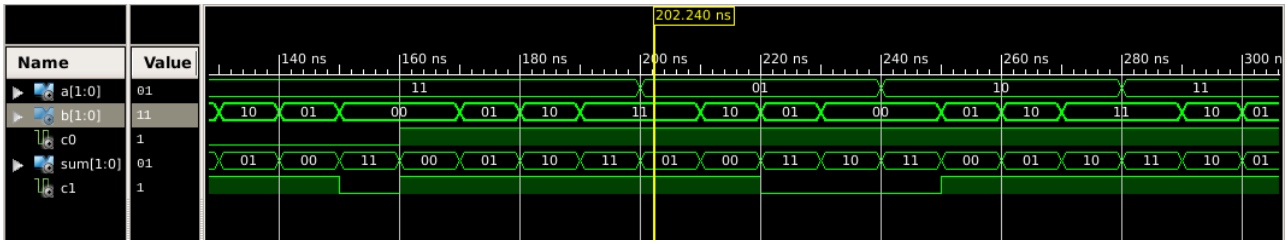
```
module FSUM_Verilog(
    input [1:0] a,b,
    input c_in,
    output reg[1:0] sum,
    output reg c_out );

    always @ (a, b, c_in)
        {c_out, sum} <= a + b + c_in;
endmodule
```

b) Verilog

Фигура 2.31 Модели на пълен суматор

На фиг. 2.32 са представени част от резултатите от симулацията на моделите на пълен суматор.



Фигура 2.32 Симулация на пълен суматор

Моделите от фигура 2.31 са използвани за синтез и имплементация в FPGA. В таблица 2.16 са представени използваните ресурси.

Таблица 2.16 Използвани ресурси от FPGA след синтеза

Използвани ресурси от FPGA	VHDL и Verilog
Брой на слайсовете	3
Брой на 4-входни LUTs:	5
Брой на свързани IOBs:	8

Моделиране на регистър

Разработените модели на регистър са представени на фигура 2.37.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity REG_VHDL is
    Port ( clk, res, sh_l, sh_r : in
STD_LOGIC;
           input : in  STD_LOGIC_VECTOR (7
downto 0);
           out_p : out STD_LOGIC_VECTOR
(7 downto 0);
           out_r : out STD_LOGIC_VECTOR (7
downto 0);
           serial : out STD_LOGIC);
end REG_VHDL;

architecture Behavioral of REG_VHDL is
signal reg: std_logic_vector(7 downto 0);
begin
    process(clk, res, sh_l,sh_r) begin
        if res = '1' then
            reg <= "00000000";
            serial <= '0';
        elsif (sh_l = '0' and sh_r = '0') then
            reg <= input;
        elsif(rising_edge(clk)) then
            if (sh_l = '1' and sh_r = '0') then
                reg <= reg(6 downto 0) & '0';
                serial <= reg(7);
            elsif (sh_l = '0' and sh_r = '1') then
                reg <= '0' & reg(7 downto 1);
                serial <= reg(0);
            else
                reg <= input;
            end if;
        end if;
    end process;
    out_p <= reg;
    out_r <= reg(7) & not reg(6 downto 0);
end Behavioral;
```

a) VHDL

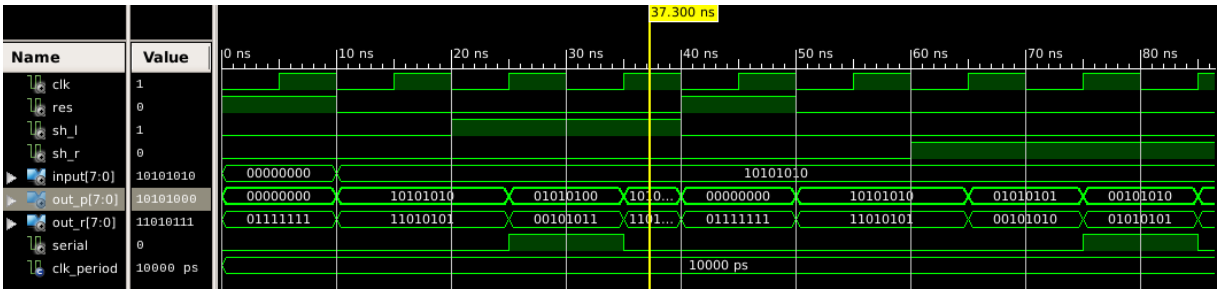
```
module REG_Verilog(
    input clk, res,
    input sh_l, sh_r,
    input [7:0] inputt,
    output reg [7:0] out_r,
    output reg [7:0] out_p,
    output reg serial );

    always @(posedge clk, posedge res)
        if (res == 1) begin
            out_r = 8'b00000000;
            out_p = 8'b01111111;
            serial = 1'b0; end
        else if (sh_l == 0 & sh_r == 0) begin
            out_r = inputt;
            out_p = {out_r[7], ~out_r[6:0]};
end
        else if (clk) begin
            if (sh_l == 1 & sh_r == 0) begin
                serial = out_r[7];
                out_r = {out_r[6:0], 1'b0};
                out_p = {out_r[7],
~out_r[6:0]}; end
            else if (sh_l == 0 & sh_r == 1)
begin
                serial = out_r[0];
                out_r = {1'b0, out_r[7:1]};
                out_p = {out_r[7],
~out_r[6:0]}; end
            else begin
                out_r = inputt;
                out_p = {out_r[7],
~out_r[6:0]};
            end
        end
endmodule
```

b) Verilog

Фигура 2.37 Модели на регистър

На фиг. 2.38 са представени част резултатите от симулацията.



Фигура 2.38 Симулация на регистър

Моделите от фигура 2.37 са използвани за синтез и имплементация, а в таблица 2.23 са представени използваните ресурси. От таблицата се вижда, че за VHDL модела на регистъра са използвани повече LUTs, а Verilog модела - повече тригери.

Таблица 2.23 Ресурси използване от моделите

Използвани ресурси от FPGA	VHDL	Verilog
Брой на слайсовете	18	15
Брой на слайсове с тригери	9	16
Брой на 4-входни LUTs:	39	30
Брой на свързани IOBs:	29	29
Брои на GCLKs:	1	1

Изводи към втора глава

- Разработените модели на цифрови схеми и компоненти на компютърните системи са приложими за изграждане на модели на компютърните системи.
- Получените резултати от симулацията на разработените модели показват, че използваният език за хардуерно описание не е определящ.
- Необходимото количество използвани хардуерни ресурси при логически синтез на компонентите не зависи от избрания език за описание.

Глава III: Моделиране на микропроцесор с RISC архитектура

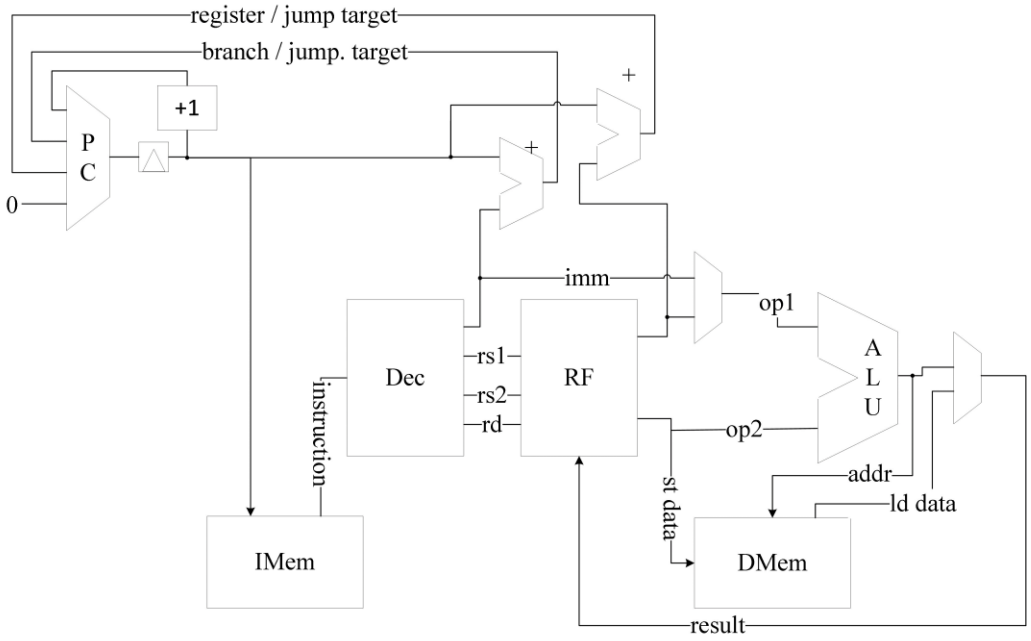
В тази глава са разгледани основните типове микропроцесорни архитектури. Направено е сравнение между RISC и CISC подходите по отношение на сложност на инструкциите, брой операции и ефективност на изпълнението.

Основни характеристики на RISC архитектурите са:

- използване на опростен набор от инструкции;
- инструкции с фиксирана дължина;
- разделяне на операциите за достъп до памет и изчисления;
- използване на регистров файл за обработка на данни;

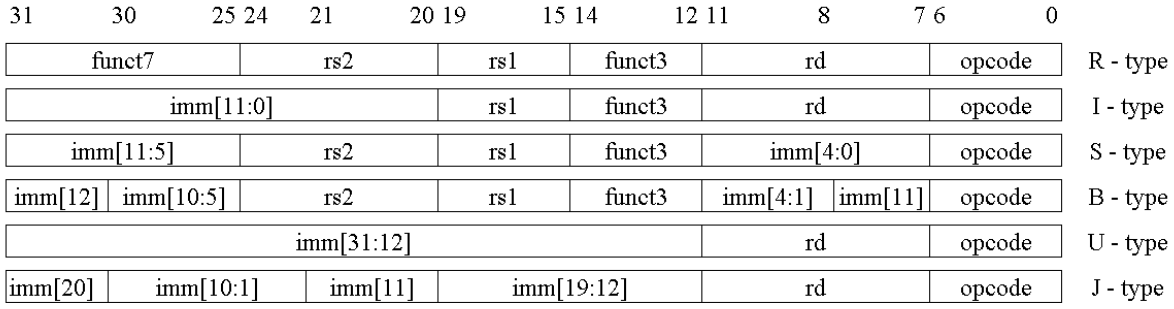
Моделиране на микропроцесор с RISC-V архитектура

На фигура 3.1 е представена структурна схема на микропроцесор с RISC-V архитектура.



Фигура 3.1 Структурна схема на микропроцесор с RISC-V архитектура

Микропроцесорите с RISC-V архитектура дефинират различни типове инструкции. Всяка инструкция се характеризира с определен формат. На фигура 3.4 е показан формата на различните типове инструкции.



Фигура 3.4 Формати на инструкциите за всеки тип

Разработеният модел на микропроцесор с RISC-V архитектура е описан чрез езика за хардуерно описание TL-Verilog. Моделът включва всички основни функционални блокове и реализира изпълнението на инструкции съгласно тази архитектурата.

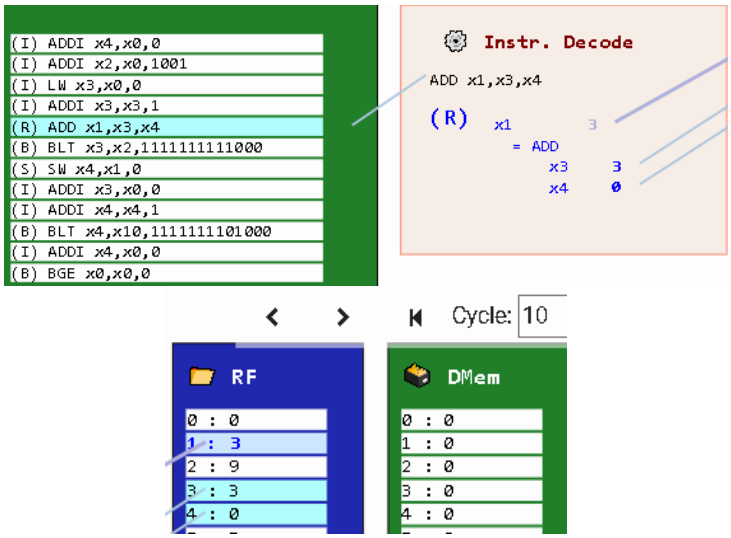
Част от модела на аритметично-логическото устройство е показана на фигура 3.9.

```
$result[31:0] =    $is_addi  ? $src1_value + $imm :
$is_add   ? $src1_value + $src2_value :
$is_andi  ? $src1_value & $imm :
$is_ori   ? $src1_value | $imm :
$is_xori  ? $src1_value ^ $imm :
$is_slli  ? $src1_value << $imm[5:0] :
$is_srli  ? $src1_value >> $imm[5:0] :
$is_and   ? $src1_value & $src2_value :
$is_or    ? $src1_value | $src2_value :
$is_xor   ? $src1_value ^ $src2_value :
```

```
$is_sub   ? $src1_value - $src2_value :
$is_sll   ? $src1_value << $src2_value[4:0] :
$is_srl   ? $src1_value >> $src2_value[4:0] :
$is_sltu  ? $sltu_rslt :
$is_sltiu ? $sltiu_rslt :
$is_lui   ? {$imm[31:12],12'b0} :
is_auipc ? $pc + $imm :
$is_jal   ? $pc + 32'd4 :
$is_jalr  ? $pc + 32'd4 :
$is_slt   ? (($src1_value[31] == $src2_value[31]) ? $sltu_rslt : {31'b0, $src1_value[31]}) :
$is_slti  ? (($src1_value[31] == $imm[31]) ? $sltiu_rslt : {31'b0, $src1_value[31]}) :
$is_sra   ? $sra_rslt[31:0] :
$is_srai  ? $srai_rslt[31:0] :
32'b0;
```

Фигура 3.9 Част от модела на аритметично логическото устройство

Валидирането на модела на микропроцесор с RISC-V микроархитектура е реализирано в средата Makerchip. На фигура 3.14 са показани част от резултатите от симулацията на модела.



Фигура 3.14 Резултата от симулацията на модела в процесорен цикъл 10

Разработена тестовата програма за валидиране на модела (фигура 3.13).

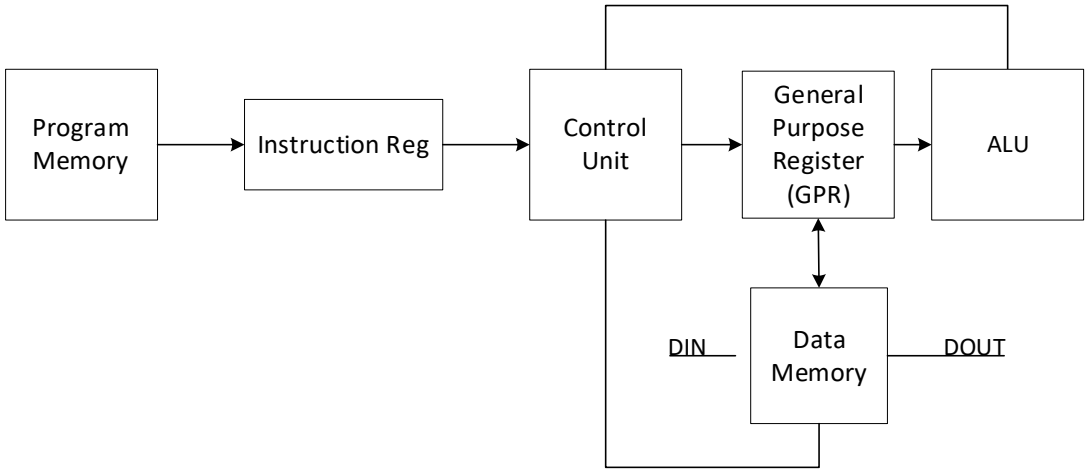
```
m4_asm(ADDI, x4, x0, 0)
    m4_asm(ADDI, x2, x0, 1001)
    m4_asm(LW, x3, x0, 0)
    m4_asm(ADDI, x3, x3, 1)
    m4_asm(ADD, x1, x3, x4)
    m4_asm(BLT, x3, x2, 1111111111000)
    m4_asm(SW, x4, x1, 0 )
    m4_asm(ADDI, x3, x0, 0)
    m4_asm(ADDI, x4, x4, 1)
    m4_asm(BLT, x4, x10, 1111111101000)
    m4_asm(ADDI, x4, x0, 0)
    m4_asm(BGE, x0, x0, 0)
```

```
m4_asm_end()
m4_define(['M4_MAX_CYC'], 315)
```

Фигура 3.13 Тестова програма

Модели на микропроцесори с RISC архитектура

На фигура 3.17 е показана блокова схема на структурата на микропроцесорите с RISC архитектура.



Фигура 3.17 Блокова схема на микропроцесора

Разработени са два модела описано моделирането на всеки блок. В таблица 3.4 е представен форматът на инструкциите те са три основни типа R, I и J.

Таблица 3.4 Полета на инструкциите в процесора

R-Type	Field	Opcode	rd	rs1	imm?	Rs2	Unused	Total
	Bit	5	5	5	1	5	11	32
I-Type	Field	Opcode	rd	rs1	imm?	Immediate		Total
	Bit	5	5	5	1	16		32
J-Type	Field	Opcode	Unused			Address		Total
	Bit	5	11			16		32

Инструкциите за аритметични и логически операции са представени в таблици 3.5 и 3.6.

Таблица 3.5 Инструкции за аритметични операции

MOVS RDST;	RDST <= SGPR (при операция умножение)
MOV RDST, RSRC;	RDST <= RSRC
MOVI RDST, IMM;	RDST <= immediate data
ADD RDST, RSRC1, RSRC2;	RDST <= RSRC1 + RSRC2
ADDI RDST, RSRC1, IMM;	RDST <= RSRC1 + immediate
SUB RDST, RSRC1, RSRC2;	RDST <= RSRC1 – RSRC2
SUBI RDST, RSRC1, IMM;	RDST <= RSRC1 – immediate
MUL RDST, RSRC1, RSRC2;	RDST <= RSRC1 * RSRC2 [15 : 0] SGPR <= MUL_OUT [31 : 16]
MULI RDST, RSRC1, IMM;	RDST <= RSRC1 * immediate [15 : 0] SGPR <= MUL_OUT [31 : 16]

Таблица 3.6 Инструкции за логически операции

OR RDST, RSRC1, RSRC2;	RDST <= RSRC1 | RSRC2
ORI RDST, RSRC1, IMM;	RDST <= RSRC1 | immediate
AND RDST, RSRC1, RSRC2;	RDST <= RSRC1 & RSRC2
ANDI RDST, RSRC1, IMM;	RDST <= RSRC1 & immediate
XOR RDST, RSRC1, RSRC2;	RDST <= RSRC1 ^ RSRC2
XORI RDST, RSRC1, IMM;	RDST <= RSRC1 ^ immediate
XNOR RDST, RSRC1, RSRC2;	RDST <= RSRC1 ~^ RSRC2
XNORI RDST, RSRC1, IMM;	RDST <= RSRC1 ~^ immediate
NAND RDST, RSRC1, RSRC2;	RDST <= ~ (RSRC1 & RSRC2
NANDI RDST, RSRC1, IMM;	RDST <= ~ (RSRC1 & immediate)
NOR RDST, RSRC1, RSRC2;	RDST <= ~ (RSRC1 | RSRC2
NORI RDST, RSRC1, IMM;	RDST <= ~ (RSRC1 | immediate)
NOT RDST, RSRC1, RSRC2;	RDST <= ~ RSRC1
NOTI RDST, RSRC1, IMM;	RDST <= ~ immediate

В създадените модели на микропроцесорите са използвани следните условни флагове: Carry Flag, Zero Flag, Sign Flag и Overflow Flag. Те определят дали прехода ще се осъществи или не. Инструкциите за преход и разклонение са представени в таблица 3.9.

Таблица 3.9 Инструкции за преход

JUMP ADDR;	PC <= ADDR
JC ADDR; (if carry)	= 0 → PC <= PC + 1, = 1 → PC <= ADDR
JNC ADDR; (if no carry)	= 1 → PC <= PC + 1, = 0 → PC <= ADDR
JS ADDR; (if sign)	= 0 → PC <= PC + 1, = 1 → PC <= ADDR
JNS ADDR; (if no sign)	= 1 → PC <= PC + 1, = 0 → PC <= ADDR
JZ ADDR; (if zero)	= 0 → PC <= PC + 1, = 1 → PC <= ADDR
JNZ ADDR; (if no zero)	= 1 → PC <= PC + 1, = 0 → PC <= ADDR
JO ADDR; (if overflow)	= 0 → PC <= PC + 1, = 1 → PC <= ADDR
JNO ADDR; (if no overflow)	= 1 → PC <= PC + 1, = 0 → PC <= ADDR
HALT;	Stop <= 1 (може да се премахне само след рестарт)

Блокът за контрол е създаден на базата на краен автомат с шест състояния: idle, fetch_inst, dec_exec_inst, next_inst, sense_halt и delay_next_inst;

За единия модел на микропроцесор с RISC архитектура е използван езика за хардуерно описание Verilog. Част от него (на блока му за контрол) е показан на фигура 3.25.

```
always @(posedge clk) begin
    if (sys_rst)
        state <= idle;
    else
        state <= next_state; end
always @(*) begin
    case(state)
idle: begin
        IR = 32'h0;
        PC = 0;
        dout = 16'h1;
        next_state = fetch_inst; end
fetch_inst: begin
        IR = inst_mem[PC];
```

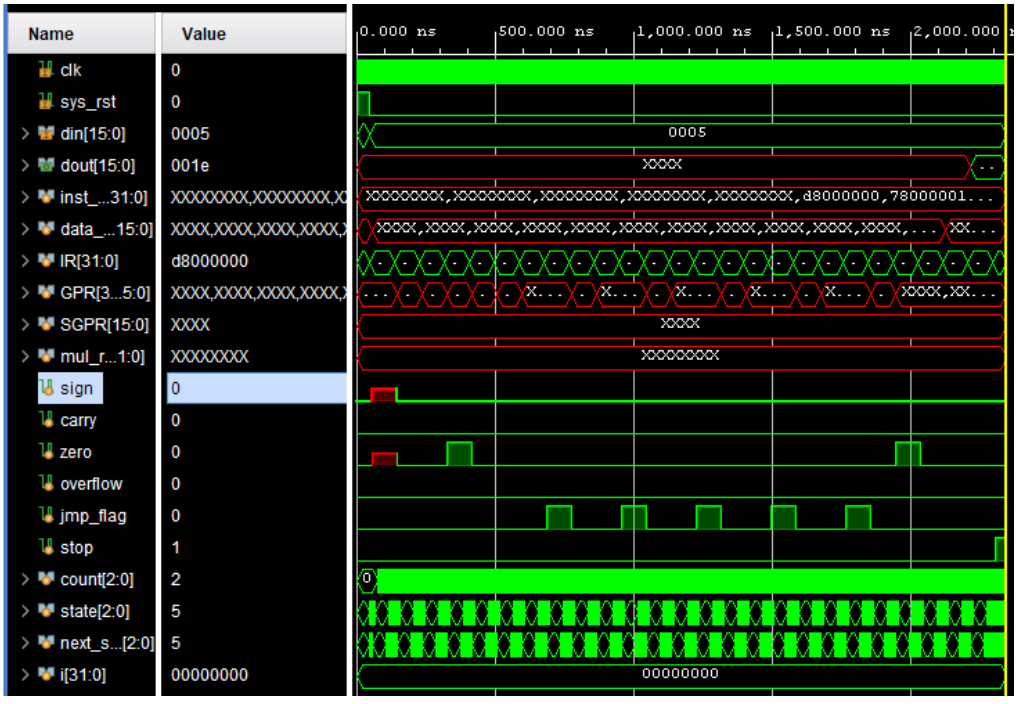
```
            next_state = dec_exec_inst; end
        dec_exec_inst: begin
            decode_inst();
            decode_condflags();
            next_state = delay_next_inst;
end
        delay_next_inst: begin
            if (count < 4)
                next_state = delay_next_inst;
            else
                next_state = next_inst; end
        next_inst: begin
            next_state = sense_halt;
            if (jmp_flag == 1'b1)
                PC = `isrc;
            else
                PC = PC + 1; end
        sense_halt: begin
            if (stop == 1'b0)
                next_state = fetch_inst;
            else if (sys_rst == 1'b1)
                next_state = idle;
            else
                next_state = sense_halt; end
        default: next_state = idle;
endcase end
```

Фигура 3.25 Verilog модел на блока за контрол

За валидиране на изградения с Verilog модел на микропроцесор с RISC архитектура е използвана средата за разработка на Vivado. За нуждите на валидирането е съставена тестова програма на асемблер. В нея са включени основни инструкции от различни типове.

```
0   STORE   0;
1   LOADR   R0, 0;
2   MOVI R1, 6;
3   MOVI R2, 0;
4   MOV  R3, R1;
5   ADD   R2, R2, R0;
6   SUBI  R3, R3, 1;
7   JNZ    @5;
8   STORER  1, R2;
9   LOAD     1;
10  HALT
```

Част от получените резултати от тестването на модела микропроцесора са представени на фиг. 3.28.



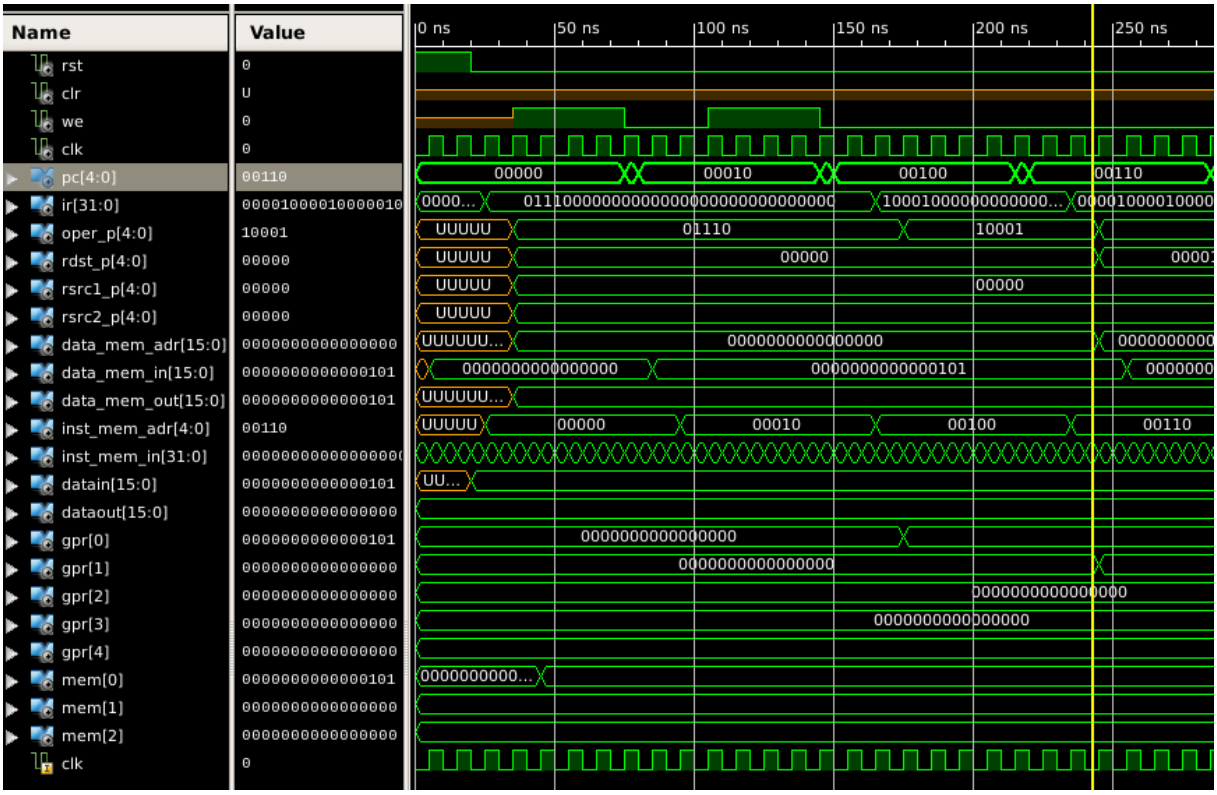
Фиг.3.28. Резултати от тестове на микропроцесорен модел

За другия разработен модел на микропроцесор с RISC архитектура е използван на езика за хардуерно описание VHDL. Част от модела на аритметично-логическото му устройство е представен на фигура 3.29.

```
when movsgpr => --Special MOV
    GPR(rdstt) <= SGPR;
when mov => --MOV & MOVI
    if (imm_mode = '1') then
        GPR(rdstt) <= isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t);
    end if;
when add =>--ADD & ADI
    if (imm_mode = '1') then
        GPR(rdstt) <= std_logic_vector(
unsigned (GPR(rsrc1t)) + unsigned(isrc));
    else
        GPR(rdstt) <= std_logic_vector
(unsigned (GPR(rsrc1t)) + unsigned(
GPR(rsrc2t)));
    end if;
when sub => --SUB @ SUBI
    if (imm_mode = '1') then
        GPR(rdstt) <= std_logic_vector(
unsigned (GPR(rsrc1t)) - unsigned(isrc));
    else
        GPR(rdstt) <= std_logic_vector(
unsigned (GPR(rsrc1t)) - unsigned(
GPR(rsrc2t)));
    end if;
when mul => --MUL & MULI
    if (imm_mode = '1') then
        mul_res<=std_logic_vector(unsigned
(GPR(rsrc1t)) * unsigned(isrc));
    else
        mul_res<=std_logic_vector(unsigned
(GPR(rsrc1t)) * unsigned(GPR(rsrc2t)));
        GPR(rdstt) <= mul_res(15 downto 0);
        SGPR      <= mul_res(31 downto 16);
    end if;
when rror => --OR & ORI
    if (imm_mode = '1') then
        GPR(rdstt) <= GPR(rsrc1t) or isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t) or
GPR(rsrc2t);
    end if;
when rand => --AND & ANDI
    if (imm_mode = '1') then
        GPR(rdstt) <= GPR(rsrc1t) and isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t) and
GPR(rsrc2t);
    end if;
when rxor=> --XOR & XORI
    if (imm_mode = '1') then
        GPR(rdstt) <= GPR(rsrc1t) xor isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t) xor
GPR(rsrc2t);
    end if;
when rxnor => --XNOR & XNORI
    if (imm_mode = '1') then
        GPR(rdstt) <= GPR(rsrc1t) xnor isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t) xnor
GPR(rsrc2t);
    end if;
when rnand => --NAND & NANDI
    if (imm_mode = '1') then
        GPR(rdstt) <= GPR(rsrc1t) nand isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t) nand
GPR(rsrc2t);
    end if;
when rnor => --NOR & NORI
    if (imm_mode = '1') then
        GPR(rdstt) <= GPR(rsrc1t) nor isrc;
    else
        GPR(rdstt) <= GPR(rsrc1t) nor
GPR(rsrc2t);
    end if;
when rnot => --NOT & NOTI
    if (imm_mode = '1') then
        GPR(rdstt) <= not isrc;
    else
        GPR(rdstt) <= not GPR(rsrc1t);
    end if;
```

Фигура 3.29 VHDL модел на аритметично-логическо устройство

За валидиране на изградения VHDL модел на микропроцесор с RISC архитектура е използвана средата за разработка на Isim на Xilinx. На фигура 3.36 са показани част от резултатите от симулацията.



Фигура 3.36 Резултат от симулацията на първите 4 инструкции

Изводи към трета глава

- Разработените модели са приложими за изследване на микропроцесори с RISC архитектура.
- Предложените модели могат да се използват за изграждане на модели на компютърни системи от различен клас.
- Резултатите от симулацията показват функционалната работоспособност на разработените модели на микропроцесорни архитектури.

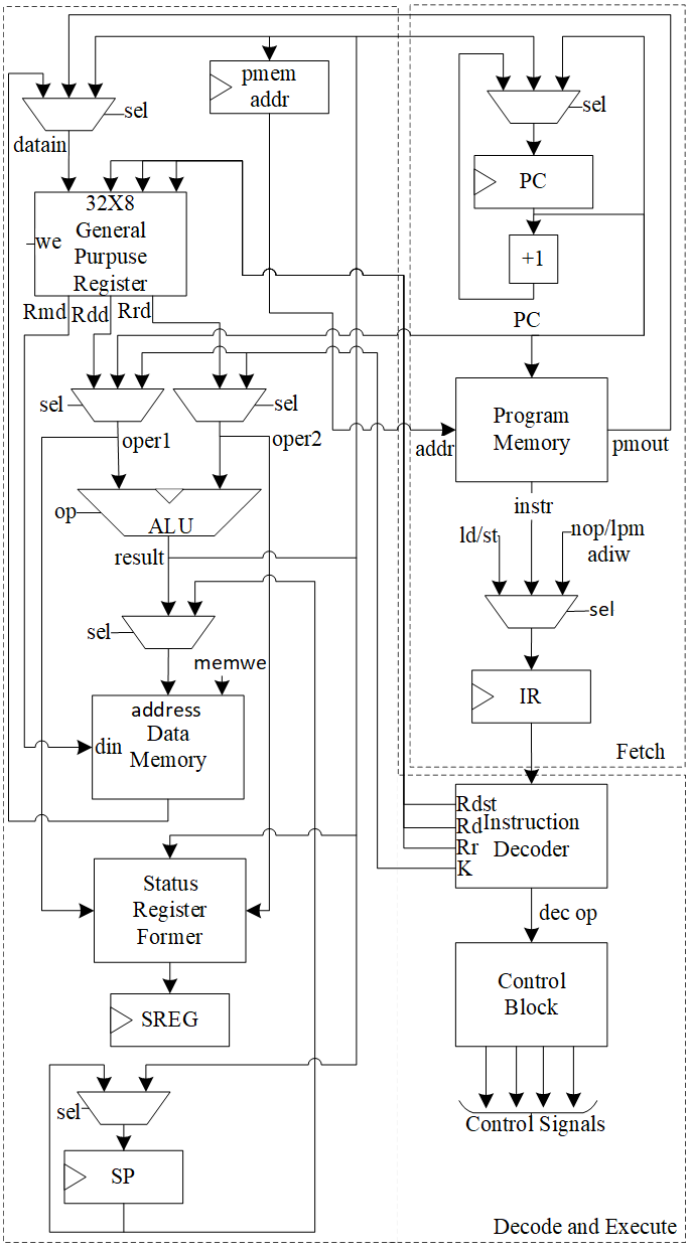
Глава IV: Моделиране и имплементация на микропроцесор с AVR архитектура

В тази глава е разгледана архитектурата на AVR микропроцесорите и принципа на тяхното функциониране.

Блокова диаграма на създаденият модел на AVR микропроцесор е представена на фигура 4.2.

Архитектурата включва следните основни функционални блокове: Програмен брояч (Program Counter), Програмна памет (Program Memory), Регистър за инструкции (Instruction Register, IR)**,** Декодер на инструкции (Instruction Decoder), Блок за контрол (Control Block)**,** Регистров файл (General Purpose Register, GPR), Аритметично-логическо устройство (Arithmetic Logic Unit, ALU)**,** Памет за данни (Data Memory), Блок за промяна на стойността на регистъра на състоянието (Status Register Former), Регистър на състоянието (Status Register, SREG) и Стеков указател (Stack Pointer, SP)**;**

За всеки от блокове в нея е създаден VHDL модел.



Фигура 4.2. Блокова диаграма на създаденият модел на AVR микропроцесор

Част от модела на аритметично-логическото устройство е представен на фигура 4.9.

```
case(op) is
   when "000000" => --NOP
       result(7 downto 0) <= "00000000";
   when "000001" => -- ADD
       result(7 downto 0) <= oper1 + oper2;
   when "000010" => --ADC
       result(7 downto 0) <= oper1 + oper2 + carry;
   when "000100" => --EOR
       result(7 downto 0) <= oper1 xor oper2;
   when "000011" => --SUBI
       result(7 downto 0) <= oper1 - oper2;
```

```
when "000101" => --ASR
    result(7 downto 0) <= oper1(7) & '0' & oper1(6 downto 1);
when "000110" => --ADIW
    result<= (oper1 & oper1m) + (oper2ext & oper2);
```

Фигура 4.9. Част от VHDL моделът на АЛУ

Част от модела на декодера на инструкции (инср. ADD) е представен фиг. 4.6.

```
elsif instr_reg(15 downto 10) = "000011" then
decoded_op <= OP_ADD;
RDest <= instr_reg(8 downto 4);
Rd <= instr_reg(8 downto 4);
Rr <= instr_reg(9) & instr_reg(3 downto 0);
```

Фигура 4.6. Декодиране на инструкцията ADD

Моделът на паметта за данни е представен фигура 4.10.

```
process (clk, memre, memwe, rst, dmemaddr)    begin
    addr <= dmemaddr(4 downto 0);
    if (rst = '1') then
        dmem <= (OTHERS => (OTHERS => '0'));
        ready <= '0';
    elsif (memwe = '0') then
        dmemout <= dmem(to_integer(unsigned(addr)));
    elsif (rising_edge(clk)) then
        dmem(to_integer(unsigned(addr))) <= dmemin;
        ready <= '1';
    end if;
end process;
```

Фигура 4.10. VHDL модел на паметта за данни

Направена е класификация и сравнение между AVR архитектурите (таблица 4.1).

Таблица 4.1. Сравнение между AVR архитектурите

Архитектура	Характеристики	Flash памет	SRAM	Типични микроконтролери
AVR	Класическо ядро, базов instruction set	до 8 KB	до 128 B	AT90S1200, ATtiny11
AVRe	Разширен instruction set	до 128 KB	до 4 KB	ATmega16, ATmega32
AVRe+	Подобрено ядро, оптимизиран достъп до памет	до 256 KB	до 8 KB	ATmega328P, ATmega1280
AVRxm	XMEGA с разширени периферии	до 16 MB	до 128 KB	ATxmega128A1
AVRxt	Компактно, модерно ядро, подходящо за low-power	до 32 KB	до 8 KB	ATtiny1616, ATtiny817

Архитектурата AVRe (Enhanced AVR Core) е разширение на класическото AVR ядро. Нейните предимства са показани в таблица 4.2.

Таблица 4.2. Сравнение на AVRe с AVR

Характеристика	AVR (класически)	AVRe (разширен)
Инструкции	Базов набор	+ MOVW, LPM R,Z+, ADIW, SBIW
Flash памет	до 8 KB	до 128 KB
SRAM	до 128 B	до 4 KB
Регистри	32	32
Примерен MCU	AT90S1200	ATmega16, ATmega32

Видовете инструкции използвани при моделиране на AVRe микропроцесора са показани в таблици 4.3 ÷ 4.6.

Таблица 4.3. Аритметични и логически инструкции

Инструкция	Описание
ADD, ADC	Събиране на 8-битови стойности
SUB, SBC, SBCI	Изваждане с/без пренос
MUL, MULS, FMUL	Умножение (резултат в R0:R1)
INC, DEC, NEG	Инкрементиране, декрементиране, отрицание
AND, OR, EOR, COM	Побитови логически операции

Таблица 4.4. Инструкции за сравнение и управление на потока

Инструкция	Описание
CP, CPC, CPI	Сравнение на регистри с регистри/константи
RJMP, IJMP, JMP	Преходи (къси и дълги)
RCALL, CALL, RET	Подпрограми
BRNE, BREQ, BRLT и др.	Условни преходи (на база флагове от регистъра SREG)

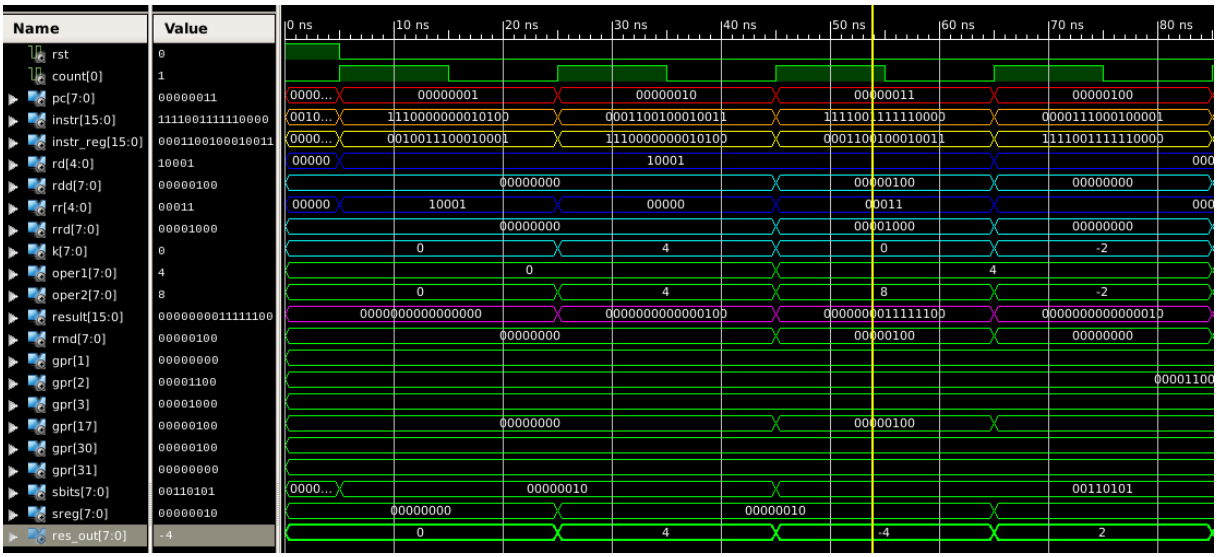
Таблица 4.5. Инструкции за пренос на данни

Инструкция	Описание
MOV, MOVW	Прехвърляне между регистри (вкл. двойни регистри)
LD, LDS, LDD	Зареждане от SRAM чрез индексиране с X, Y, Z
ST, STS, STD	Запис в SRAM чрез регистри
LPM, ELPM	Четене от Flash чрез Z или RAMPZ:Z
IN, OUT	Пренос от/до I/O регистри
PUSH, POP	Стекови операции

Таблица 4.6. Инструкции за работа на ниво бит

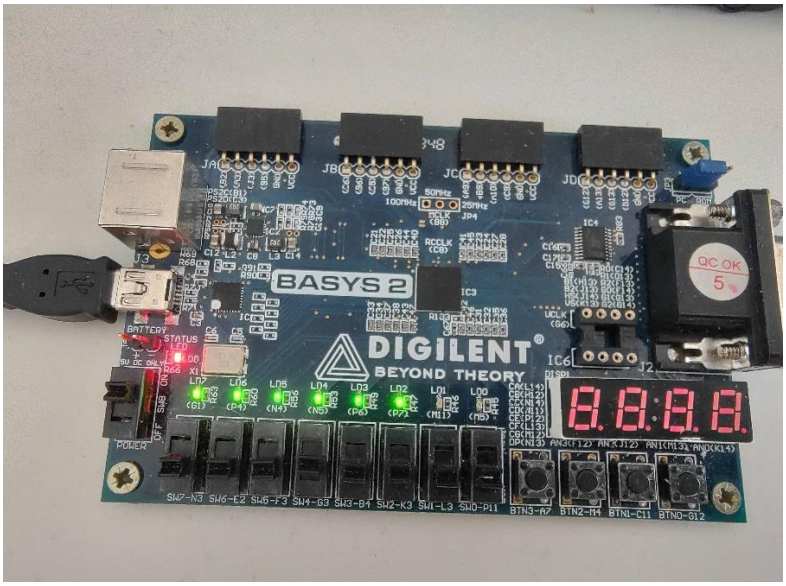
Инструкция	Описание
SBI, CBI	Задаване/нулиране на бит в I/O регистър
SBIC, SBIS	Условен пропуск на инструкция в зависимост от бит
BST, BLD	Трансфер на бит между регистър и T-бит
SBR, CBR	Задаване/нулиране на битове в регистър (с маска)

Разработеният модел на AVR микропроцесор е реализиран в средата на Xilinx – Project Navigator 14.7. На фигура 4.21 са показани част от резултатите от симулацията.



Фигура 4.21. Резултати от симулацията

Моделът е синтезиран и имплементиран в препрограмируема матрица. За целта е използвана развойната платка Basys2 (фигура 4.23).



Фигура 4.23. Развойната платка в изпълнение на създаденият модел

Направено е сравнение между съществуващите разработки и предложените модели на микропроцесори (таблица 5.1). Основните предимства на разработените модели са:

- по-ясна връзка между архитектурната концепция и поведенческия модел;
- възможност за експериментиране с нетипични синхронни и асинхронни зависимости;
- по-ниско ниво на сложност на модела;
- възможност за сравнителен анализ;
- приложими са за изграждане на модел на компютърни архитектури.

Таблица 5.1 Сравнение между съществуващи разработки и предложените модели

Критерий	Съществуващи разработки	Модели на RISC архитектура (глава 3)	Модел на AVR архитектура (глава 4)
Основна цел	Индустриално приложение или готов SoC	Обучение и архитектурен анализ	Обучение и хардуерна реализация
Архитектура	RISC-V, SPARC, MIPS, AVR	RISC архитектура	AVR микроконтролерна
HDL език	VHDL / Verilog / SystemVerilog	VHDL, Verilog, TL-Verilog	VHDL
Ниво на моделиране	Строго RTL или генеративно RTL	Архитектурно / поведенческо	Строго RTL
Спазване на RTL принципите	Да	Не	Да
Цел на симулацията	Верификация и оптимизация	Обяснение и изследване	Функционална коректност
FPGA имплементация	Да	Не	Да
Проблеми при синтез	Не	Да (синхронизация, инструкции)	Не
Гъвкавост за експерименти	Ограничена	Висока	Средна
Подходящо за студенти	Ограничено	Много подходящо	Много подходящо
Научен фокус	Производителност и надеждност	Обучение и изследване	RTL проектиране и FPGA

Изводи към четвърта глава

- Разработеният модел на микропроцесор с AVR архитектура е подходящ за хардуерна реализация и отговаря на предварително зададените функционални изисквания.
- Разработеният модел на микропроцесор с AVR архитектура на регистрово ниво гарантира коректна синхронизация при физическата му реализация.
- Моделът на микропроцесор с AVR архитектура може да се използва при проектиране на вградени системи с ограничени ресурси.
- Получените резултати от симулацията на предложените модели потвърждават тяхната функционална работоспособност и приложимост.
- Направената оценка на производителността и ресурсната ефективност на предложените модели може да се използва за оптимизация на бъдещи проекти.

ПРИНОСИ НА ДИСЕРТАЦИОННИЯ ТРУД

В резултат на разработката могат да се обобщят следните приноси:

1. Създадени са модели на основни цифрови компоненти на компютърните системи – аритметично-логически устройства, памет с произволен достъп, мултиплексори, демултиплексори, шифратори, дешифратори, броячи, регистри и компаратори – реализирани чрез езиците за хардуерно описание VHDL и Verilog.
2. Разработени са модели на микропроцесори с RISC архитектура на базата на езиците за хардуерно описание TL-Verilog, Verilog и VHDL. Чрез тези модели е изследвана организацията и взаимодействието между основните блокове – аритметично-логическо устройство, регистров файл, памет и контролен блок.
3. Разработен е модел на микропроцесор с AVR архитектура на регистрово ниво.
4. Имплементиран е модел на микропроцесор с AVR архитектура върху FPGA платформа.
5. Направена е оценка на производителността и ресурсната ефективност на HDL модели при имплементация върху FPGA, която може да се използва за оптимизация на бъдещи проекти.
6. Направен е сравнителен анализ на съществуващите разработки и предложените модели на микропроцесори с RISC архитектури.

СПИСЪК НА ПУБЛИКАЦИИТЕ ПО ДИСЕРТАЦИОННИЯ ТРУД

1. Върбов И., В. Кукенска, П. Минев, Моделиране на последователен входно-изходен интерфейс, Международна научна конференция “УНИТЕХ’11”, Габрово, 18 – 19 Ноември, 2011, том I, стр. 388-392, ISSN 1313-230X.
2. Върбов И., Моделиране и симулиране на микропроцесор с RISC архитектура, Знание, наука, иновации, технологии, Велико Търново, 27 Март, 2026, под печат, ISSN 2815-3480.
3. Върбов И., В. Кукенска, П. Минев, Моделиране и изследване на памет с произволен достъп, Международна научна конференция “УНИТЕХ’12”, Габрово, 16 – 17 Ноември, 2012, том I, стр. 414-418, ISSN 1313-230X.
4. Върбов И., В. Кукенска, П. Минев, VHDL модел на FLASH памет, Международна научна конференция “УНИТЕХ’14”, Габрово, 21 – 22 Ноември, 2014, том II, стр. 298-301, ISSN 1313-230X.
5. Varbov I., V. Kukenska, P. Minev, M. Dinev, Application of TL-Verilog for Modelling Components of Microprocessor Architectures, Proceedings of International Scientific Conference Unitech’2023, Vol. 1, Gabrovo, 2023, pp. I 273-279. ISSN 1313-230X.
6. Varbov, P. Minev, M. Dinev and V. Kukenska, "Modeling a microprocessor with RISC architecture," 2024 International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 2024, pp. 399-402, doi: 10.1109/ ICAI63388.2024.10851508, IEEE.
7. И. Върбов, П. Минев, М. Динев, В. Кукенска, Моделиране и имплементация на микропроцесор с AVR микроконтролерна архитектура, Международна научна конференция “УНИТЕХ’25”, Габрово, 20 – 22 Ноември, 2025.

MODELING AND SIMULATION OF COMPONENTS OF COMPUTER SYSTEMS

Ilian Tsvyatkov Varbov

ABSTRACT

The first chapter presents an analysis of the current state of research in the field of modeling computer architectures. Existing developments of processor cores such as Rocket Chip Generator, VexRiscv, Ibex Core, CV32E40P, and others are reviewed. An overview of the main components of computer systems, as well as the hardware description languages and simulation environments used, is provided.

The second chapter develops models of basic digital circuits and functional blocks used in the design of computer systems. Models of an arithmetic logic unit, random access memory, multiplexers, adders, registers, and counters are presented, implemented using HDL languages and analyzed through simulation.

The third chapter is devoted to the modeling of a microprocessor with a RISC architecture. The developed RISC microprocessor models are implemented using TL-Verilog, Verilog, and VHDL. The models are analyzed and simulated using the Makerchip, Vivado, and ISim environments.

The fourth chapter presents a register-transfer level model of a microprocessor based on AVR architecture. The structure of the microprocessor and the interaction between its main functional blocks are analyzed. The developed model is implemented on a Digilent Basys-2 FPGA platform with a Spartan-3E device.

Keywords
hardware description languages, VHDL, Verilog, TL-Verilog, RISC architecture, AVR microprocessor, FPGA, microprocessor, architectures, computer architecture modeling, digital system simulation.