25th International Scientific Conference UNITECH

INTERNATIONAL SCIENTIFIC CONFERENCE
20-22 November 2025
GABROVO

МОДЕЛИРАНЕ И ИМПЛЕМЕНТАЦИЯ НА МИКРОПРОЦЕСОР С AVR МИКРОКОНТРОЛЕРНА АРХИТЕКТУРА

Илиан Върбов*, Петър Минев, Матьо Динев, Валентина Кукенска

Технически Университет - Габрово, ул. "Х. Димитър" 4, гр. Габрово, България
Кореспондиращ автор: ivarbov@tugab.bg

MODELING AND IMPLEMENTATION OF AN AVR MICROCONTROLLER ARCHITECTURE

Ilian Varbov*, Petar Minev, Matyo Dinev, Valentina Kukenska

Technical University of Gabrovo, 4 H.Dimitar Str., Gabrovo, Bulgaria
**Corresponding author: ivarbov@tugab.bg*

Abstract
This paper presents the modeling and implementation of an AVR microcontroller architecture based on the extended avre core. The main structural components of the model are described, including the program counter, program and data memory, register file, arithmetic logic unit (ALU), control unit, and status register. The modeling process employs the RTL approach and the VHDL hardware description language. The model is simulated in the ISim environment, demonstrating arithmetic, logic, and conditional branch instructions. Synthesis and implementation were performed on a Spartan-3E (Basys 2) FPGA board, confirming the correct operation of the developed architecture. The proposed model is suitable for educational and research purposes.

Keywords: AVR, VHDL, model, microcontroller, FPGA

ВЪВЕДЕНИЕ

Архитектурата AVR представлява една от най-разпространените 8-битови микроконтролерни архитектури. Тя се характеризира с висока ефективност, ниска консумация на енергия и широко приложение във вградени системи. AVR процесорите се използват както в образователни, така и в индустриални среди, тъй като предлагат добър баланс между производителност и ресурсна оптимизация.
В доклада се разглежда моделирането и имплементацията на AVR микроконтролерна архитектура, базирана на разширеното ядро avre. За моделиране е използван RTL подход и езика за хардуерно описание VHDL. Извършена е симулация на модела, последвана от синтез и имплементация върху FPGA платка. По този начин се осигурява пълна проследимост – от архитектурната спецификация до реализацията в хардуер.

ИЗЛОЖЕНИЕ

1. Архитектура и принцип на действие на AVR микропроцесорите

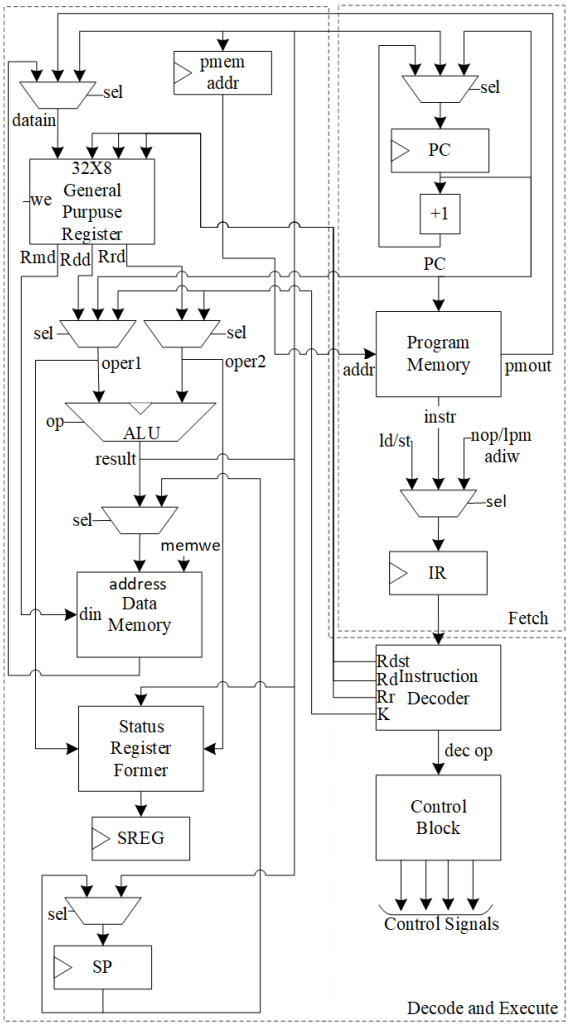
AVR (Advanced Virtual RISC) е 8-битова микроконтролерна архитектура, разработена от Atmel през 1996 година, която се отличава с ефективна харвардска архитектура, позволяваща паралелна обработка на инструкции и данни.

Print ISSN: 1313-230X, Online ISSN: 3033-1404

Благодарение на своята компактност, ниска консумация на енергия и висока скорост на изпълнение, AVR микроконтролерите се използват за вградени системи, управление на устройства и приложения в инду-стрията, автоматизацията и образовател-ни приложения. Тези микроконтролери използват модифицирана харвардска архитектура, при която програмната памет и паметта за данни са физически разделени. Това позволява едновременното извличане на инструкции и четене/запис на данни. Повечето AVR устройства използват двуфазен конвейер (Fetch-Execute), което позволява изпълнението на по-голямата част от инструкциите за един тактов цикъл. Това минимизира време-закъсненията и максимизира пропускателната способност. Архитектурата е с ниска консумация на енергия, специално разработена, за да се даде възможност за ефективно компилиране на програми на езика C. Централно място в архитектурата заема регистровият файл, състоящ се от 32 обобщени 8-битови регистъра (R0–R31). Регистрите R26–R31, които образуват три 16-битови индиректни адресни регистъра (X, Y, Z), се използват за до-стъп до паметта.

Архитектурата на системата инструкции (ISA) на AVR се характеризира с компактни, предимно 16-битови инструкции. Тя включва набор от аритметични, логически инструкции, инструкции за прехвърляне на данни, манипулация на битове и управление на реда на изпълнение в програмата. Към класическия набор инструкции през годините са добавяни нови. Така са се оформили следните разновидности: AVR; AVRe; AVRe+; AVRxm; AVRxt. Освен тях съществува и набор с по-малко инструкции от класическия, наречен AVR tiny или AVRrc.

Създаденият модел на AVR архитектура съдържа следните основни блокове (фиг.1.):



***Фиг. 1** Блокова диаграма на създаденият AVR модел*

Програмен брояч (Program Counter, PC) е регистър, който съдържа адреса на текущата инструкция. На всеки такт, стойността му се обновява според избрания сигнал sel, който определя дали:

- да се изпълни нормален инкремент (PC + 1);
- да се извърши преход в зависимост от зададено условие (branch). Адресът на който трябва да се осъществи прехода се изчислява от ALU;
- да се извърши безусловен преход (jump). Адресът на който трябва да се осъществи прехода се изчислява от ALU;

Програмната памет (Program Memory) съдържа програмния код. Той връща инструкцията, съхранена на адреса, подаден от PC. Изходът се подава

към регистъра за инструкции (Instruction Register, IR) за съхранение и декодиране.

Регистърът за инструкции е регистър, в който се съхранява текущата инструкция за декодиране. При всяка промяна на тактовия сигнал текущото състояние на регистъра се обновява със стойността на следващото състояние.

Декодерът на инструкции (Instruction Decoder) извлича полетата Rdst, Rd, Rr, K и операционния код (opcode) от инструкцията. Rd и Rr се подават като адреси към регистровият файл, а K е непосредствената стойност. Операционният код се подава към блока за контрол.

Блокът за контрол (Control Block) генерира контролни сигнали на база opcode. Той управлява (всички мултиплексори на схемата) операциите в ALU, запис в регистри, достъп до памет, и др.

Регистровият файл (General Purpose Register, GPR) съдържа 32 8-битови регистъра. Входни данни може да му се подават от паметите или от аритметично логическото устройство, в зависимост от това каква инструкция се изпълнява. Изходите Rdd и Rrd изпращат данните прочетени от съответния регистър към аритметично логическото устройство.

Аритметично-логическото устройство (Arithmetic Logic Unit, ALU) извършва всички аритметични и логически операции. Изчислява адреса в програмната памет при условни и безусловни преходи и адреса в паметта за данни при операции четене и запис. Входните операнди се подават от регистровия файл или от самата инструкция. Може да му се подаде текущата стойност на програмният брояч при инструкциите за преход.

Паметта за данни (Data Memory) е оперативна памет, достъпна чрез инструкциите за запис и четене. На адресния вход в зависимост от инструкцията може да постъпи адрес изчислен от аритметично логическото устройство или адрес съхраняван в стековия указател. При операции за запис, данните се подават директно от регистровия файл. При операции за четене, данните прочетени от паметта се подават на регистровия файл.

Блокът за промяна на стойността на статус регистъра (Status Register Former) формира статус флаговете (carry, zero, negative, overflow и др.).

Статус регистърът SREG (Status Register) е 8-битов регистър, съхраняващ статус флагове. Използва се при условни преходи и прекъсвания.

Стековият указател SP (Stack Pointer) е регистър, съхраняващ адреса за начало на стека в оперативната памет. Стековата памет се използва при подпрограми и прекъсвания. Актуализира се при инструкции PUSH и POP.

В модела има няколко мултиплексора. Тяхната функция е да подават правилните сигнали на съответния блок. Работа им се управлява от блока за контрол.

2. Видове архитектури в AVR семейството

В зависимост от функционалността, възможностите и поддържаните инструкции, архитектурите на AVR микроконтролерите могат да бъдат класифицирани, както следва:

Класическа AVR архитектура (avr)

- Представлява оригиналната версия на ядрото.
- Използва 16-битови инструкции.
- Поддържа ограничен набор от инструкции, без разширени такива като MOVW или LPM Rn, Z+.
- Характеризира се с ограничена памет (до 8 KB Flash, до 128 байта SRAM).
- Използвана основно в най-ранните модели (напр. AT90S1200, ATtiny11).

Разширено ядро (avre)

- Поддържа допълнителни инструкции: MOVW, ADIW, SBIW, LPM R, Z+.
- Разширен достъп до паметта – до 128 KB Flash и до 4 KB SRAM.
- Включва пълен набор от 32 работни регистъра.

- Широко използвана в класически ATmega микроконтролери (напр. ATmega8, ATmega16, ATmega32).

Разширено ядро + (avre+)
- Разширява възможностите на avre с допълнителни инструкции и хардуерни подобрения.
- Оптимизиран достъп до Flash памет.
- Подобрено декодиране и изпълнение на инструкции.
- Използвана в устройства като ATmega328P (включително в платформата Arduino Uno), ATmega1280 и други.

XMEGA архитектура (avrxm)
- Напълно нова архитектура, насочена към по-комплексни и високопроизводителни приложения.
- Поддръжка на периферни модули като DMA, Event System, усъвършенствано управление на захранване и др.
- Значително разширен обем на паметта – до 16 MB Flash, до 128 KB SRAM.
- Подходяща за индустриални и професионални приложения (напр. ATxmega128A1, ATxmega32A4).

Tiny (XT) Core (avrxt)
- Представлява новото поколение на tinyAVR (0/1/2/4 сериите).
- Подобрен instruction decoder с цел по-висока енергийна ефективност.
- Включва компактни инструкции, оптимизирани за ниска консумация.
- Модерни функции: многофункционални пинове, TWI/I2C, USART, RTC и др.

Архитектурата avre (Enhanced AVR Core) е разширение на класическото AVR ядро (avr) и представлява баланс между хардуерна простота и софтуерна мощност. Тя поддържа богат набор от инструкции, които покриват широк спектър от функционалности – аритметика, логика, работа с битове, пренос на данни и контрол на потока. Наборът от инструкции на AVR е оптимизиран за ефективност при вградени приложения. Повечето инструкции се изпълняват за един такт, което повишава производителността при ниска честота. Инструкциите са 16 или 32-битови, поддържащи различни режими на адресиране.

Avre архитектурата въвежда разширения, които я отличават от базовото ядро:
- 32 универсални 8-битови регистъра (R0–R31).
- Поддръжка на инструкции с 16-битови константи (ADIW, SBIW).
- Подобрени инструкции за четене от Flash: LPM Rn, Z, LPM Rn, Z+.
- Поддръжка на инструкции с двойни регистри, като MOVW (за пренос на 16-битови данни).
- Инструкции за I/O и работа със стека.

Въвеждат се инструкции, оптимизирани за по-голяма гъвкавост, по-бърза работа и по-добро управление на ресурси. Тя е стандарт за повечето ATmega микроконтролери, използвани както в индустрията, така и в образованието (включително Arduino-платформи). Поради баланса между сложност и производителност, avre остава предпочитан избор в широк кръг от вградени (embedded) приложения.

Изпълнението на една инструкция преминава през няколко етапа:
- Извличане – инструкцията се извлича от програмната памет чрез програмния брояч.
- Декодиране – управляващата логика разпознава операцията и определя участващите регистри.
- Изпълнение – аритметично логическото устройство извършва операцията, резултатите се записват в регистър или паметта за данни.

Обновяване на флагове – ако инструкцията влияе на състоянието, се обновяват флаговете в статус регистъра.

3. Симулация на модела

Моделът е реализиран в средата на Xilinx – Project Navigator 14.7. Като софтуер за симулиране се използва вградения в средата ISim.

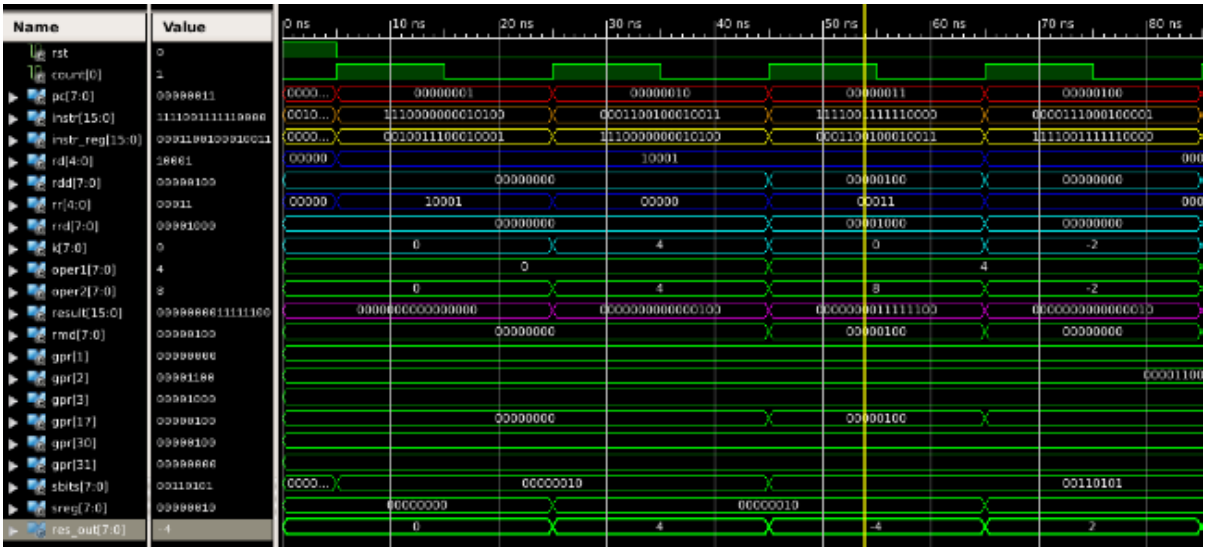
Тестовата програма е записана в програмната памет под формата на инструкции и е показана на фигура 2.

Инструкция EOR нулира регистъра R17 (работи като инструкция CLR). Следващата инструкция LDI зарежда регистър 17 с непосредствена стойност 4. Инструкцията SUB изважда от стойността на R17 стойността на R3.

```
begin                                   -- inst|  regs
    imem(0) <= "0010011100010001";  -- EOR  R17, R17
    imem(1) <= "1110000000010100";  -- LDI  R17, 4
    imem(2) <= "0001100100010011";  -- SUB  R17, R3
    imem(3) <= "1111001111110000";  -- BRCS -2
    imem(4) <= "0000111000100001";  -- ADD  R2, R17
    imem(5) <= "1111001111100001";  -- BREQ -4
    imem(6) <= "0101111000011100";  -- SUBI R17 -20
    imem(7) <= "1111011111100000";  -- BRCC -4
    imem(8) <= "0001111000010001";  -- ADC  R1, R17
    imem(9) <= "0010011111111111";  -- CLR  R31
    imem(10)<= "1110000011100100";  -- LDI  R30, 4
    imem(11)<= "1001001000011111";  -- PUSH R1
    imem(12)<= "1001011000110001";  -- ADIW Z, 1
    imem(13)<= "1001010000011000";  -- BSET ZERO
```

***Фиг. 2** Тестова програма*

В регистър R3 предварително е заредена стойност 8. Инструкцията за преход BRCS проверява флага за пренос (Carry). Ако той е вдигнат, правим преход на адрес 2 (в паметта за инструкции). Инструкциите за преход се изпълняват в два такта. Инструкцията ADD събира стойностите на двата регистъра R2 и R17 и записва резултата в R17. R2 има предварително зададена стойност 12. Инструкция за преход BREQ проверява флага Zero ако, той е 1-ца се прави преход на адрес 2 (в паметта за инструкции). Инструкция SUBI изважда от стойността на R17 непосредствената стойност -20 и резултата се записва в R17. ADC събира стойностите на R1, R17 и Carry, а резултата се записва в R1. Следващата инструкция изпълнява CLR на регистър 31. LDI зарежда стойност 4 в регистър 30. Инструкция PUSH записва съдържанието на регистър (R1) в стека. Инструкция ADIW добавя непосредствена стойност към дума. Буквата Z показва, че това са регистри 31 и 30. Резултата се записва отново в тези регистри. Инструкция BSET вдига условен флаг (zero) в единица.



***Фиг. 3** Резултати от симулацията*

На графиката от фиг. 3 е представено изпълнението на първите 4 инструкции от програмата.

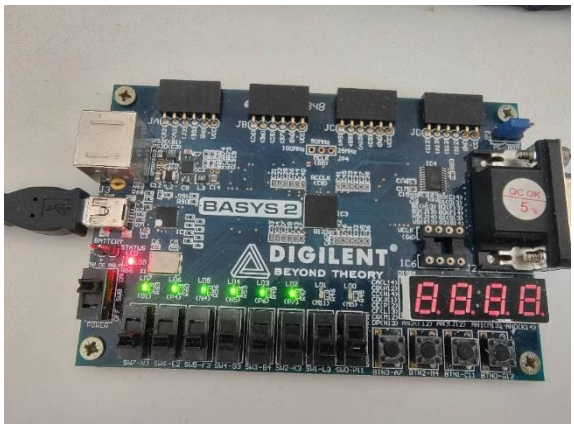
4. Синтез и имплементация

За имплементация е използвана развойната платка на Digilent – Basys 2

(фиг.4). Тя е оборудвана с FPGA чип на Xilnx - Spartan-3E (XC3S250E).

За да можем да дебъгваме и отчетем резултат на платката в модела са направени няколко промени. Добавен е делител на тактовия сигнал, който понижава честотата от 50 MHz, до 0.4Hz. За да се отчете, че имплементацията е успешна и всички инструкции се изпълняват правилно на изхода res_out се присвоява сигналът rfdatain. В повечето случай на него пристига резултата от АЛУ. При инструкции за четене от паметите на него се присвояват прочетените данни. Този сигнал е удобен за проверка на голяма част от инструкциите.

На фигура 4 се вижда резултата при изпълнение на инструкция SUB в двоичен вид – 11111100.



Фиг. 4 *Развойната платка в изпълнение на създаденият модел*

Пълният VHDL код на създаденият модел може да бъде разгледан на следният адрес:
https://github.com/tER-ROR6239/avr/blob/main/avr_core_10_04_25_11_15.vhd

ЗАКЛЮЧЕНИЕ

В рамките на настоящия доклад е моделиран и реализиран микропроцесор, базиран на AVR архитектурата и по-конкретно на разширеното ядро *avre*. Чрез прилагане на VHDL и RTL подход е създаден модел, включващ всички основни блокове на архитектурата. Моделът е успешно симулиран и имплементиран върху FPGA платка Spartan-3E. Резултатите от симулацията и хардуерната реализация доказват коректността на набора от инструкции и надеждността на архитектурната организация. Разработеният модел е приложим за образователни и изследователски цели.

Източник на финансиране:
Представената разработка е реализирана с финансовата подкрепа на проект: НИП2025-15 "Приложение на съвременни IT технологии за представяне, опазване и съхраняване на културно-историческото наследство – етап II".

ЛИТЕРАТУРА

[1] Bogen A. E., Wollan V. AVR enhanced RISC microcontrollers, Atmel Technical Document, 1999.
[2] Myklebust G. The AVR microcontroller and C compiler co-design, ATMEL Development Center, Trondheim, Norway, 1996.
[3] Atmel Corporation. (2015). 8-bit Atmel Microcontroller with 2/4/8K Bytes In-System Programmable Flash (ATtiny25/45/85 Datasheet).
[4] Atmel Corporation. (2021). AVR Instruction Set Manual.
[5] Microchip Technology Inc., ATmega 808/809/1608/1609 Data Sheet, Document No. GUID-010260B4-17F3-4FE8-8E04-90F3705BF991, [Online]. Available: https://onlinedocs.microchip.com/oxy/GUID-010260B4-17F3-4FE8-8E04-90F3705BF991-en-US-15/index.html. [Accessed: April 16, 2025].