

Пловдивски Университет “Паисий Хилендарски”
Факултет по Математика и Информатика
Катедра Софтуерни технологии

Дипломна Работа на тема:
“Статична C библиотека за Arduino”

Дипломант:
Веселин Станчев
ФН: 1801321012
спец. СИ

Научен Ръководител:
гл. ас. д-р инж. Стоян Черешаров

Пловдив 2022 г.

Съдържание

Увод.....	4
I - Изследване на съществуващите C библиотеки и анализ на datasheet-овете на микроконтролерите за които е предназначена асемблерската част.....	9
Примери за C библиотеки:.....	9
Кратък анализ на datasheet-a на Atmega 328P.....	11
Кратък анализ на datasheet-a на RP2040.....	14
II - Анализ на целевите процесорни архитектури за които е предназначена библиотеката.....	18
Анализ на ARM архитектурата.....	18
Пример за аритметична операция на GNU Assembler:.....	18
Пример за разклоняване на потока на програма на GNU Assembler:.....	19
Пример за аритметична операция на GNU Assembler:.....	20
20.....	21
III - Дефиниране на изискванията към библиотеката.....	23
IV - Използвани софтуерни инструменти.....	25
Gcc- GNU C Compiler е компилатор който може да се използва за C, C++, Assembler. Използва се за компилация на source кода до обектен файл и до краен изпълним файл. Подават се допълнителни параметри за компилацията.....	26
Make- GNU Make е build система за автоматизация. Използва се за автоматизиране на компилацията за C, C++ и Assembler програми. В Makefile се пишат целите и файловете, нужни за тяхното изпълнение. След това се пише командата която се изпълнява при дадената цел.Изключително полезна е при големи проекти.....	26
GDB- GNU GDB е дебъгер за C и Assembler. Използва се след като програмата е компилирана със дебъг символи чрез параметъра -g. За да се проследи изпълнението на програмата е нужно да се използват breakpoints. Чрез командата run започва изпълнението на програмата в дебъгера. Чрез командата next дебъгера преминава към следващия ред на програмата.....	27
AVR-GCC е модифициран GCC компилатор, който компилира source файла до .hex файл, който да се качи на целевия микроконтролер.....	27
AVRDUDE е програмата, чрез която .hex файла се качи на целевия микроконтролер.....	27
BINUTILS съдържа свободния асемблер, който ще бъде използван за основната част на библиотеката, която е цел на настоящата дипломна работа.....	27
Vim е терминален текстов редактор който може да се използва за всички програмни езици. Поддържа възможност за редактиране на огромни файлове. Поддържа функциите Undo, Redo с пълната история на файла. Разполага с различни режими на работа:.....	28
ar представлява архиватор, чрез който се създават статичните библиотеки.....	28
V - Кодиране на библиотеката.....	30
VI - Постигнати резултати. Бъдещо Развитие.....	40
Заклучение.....	49
Използвани източници:.....	50

Увод

Със развитието на IoT все повече стават популярни едноплатковите development boards като Arduino Uno и Raspberry Pi Pico които могат да послужат за учебни/университетски проекти или домашна автоматизация. Съществуват няколко вида instruction sets:

- CISC -> Complex Instruction Set Computer
- RISC -> Redused Instruction Set Computer
- MISC -> Minimal Instruction Set Computer

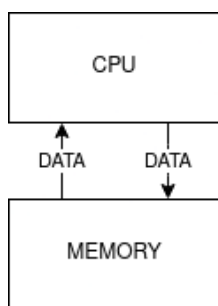
CISC се използва при x86_64 базираните настолни компютри и лаптопи.

RISC се използва при микроконтролерите Atmega 328p и RP2040.

Arduino Uno е базиран на Atmega 328p а Raspberry Pi Pico е базиран на RP2040.

Redused Instruction Set означава че по-сложните инструкции се свеждат до изпълнение на основните инструкции. Архитектурата на Atmega 328p и RP2040 е System on Chip (SoC).

SoC означава, че най-важните компоненти за една компютърна система - процесора и паметта, според фон Ньоймановата архитектура са обединени в един чип.



Фиг. 1

Фиг. 1 показва разделението на процесора и паметта.

Начините за програмирането на Arduino са:

- чрез използването на C++ базирания диалект
- чрез използването на C
- чрез използването на Assembler

Езикът C е език от ниско ниво и затова гарантира бързина на изпълнение на програмата. Много по-добре е да се използва C отколкото официалния C++ диалект.

Ако се цели още по-голяма бързина, тогава се използва Assembler. Съществуват няколко вида архитектура на instruction set-a:

- x86_64 -> за настолни компютри и лаптопи.
- ARM -> за мобилни устройства, микроконтролери и едноплаткови компютри (Advanced RISC Machine).
- RISC-V -> open-source RISC базирана архитектура.

За различните архитектури на instruction set-a има различни асемблери.

Instruction Set Architecture	Assembly Language
X86_64	MASM, NASM
ARM	GNU Assembler
RISC-V	GNU Assembler; AVR Assembler

За X86_64 архитектурата на instruction set-a разполагаме със:

- ➔ Microsoft Assembler който може да се пише в директива `__asm{...}` в C/C++ source файл
- ➔ Netwide Assembler – свободен асемблер

За ARM архитектурата на instruction set-a разполагаме със:

- ➔ GNU Assembler - свободен асемблер за RISC базирани микроконтролери и процесори.
- ➔ AVR Assembler - свободен асемблер за широк спектър от Atmel базирани микроконтролери и процесори.

Примери за едноплаткови компютри:

- Raspberry Pi
- Olinuxino A20

Raspberry Pi е едноплатков компютър, базиран на ARM процесора Broadcom.

Той е RISC-базиран. Платката не е с отворен код но операционната система по подразбиране е Debian-базирана с отворен код. Като харддиск се използва SD карта.

Едноплатковите компютри помагат за по-лесния достъп на ученици и студенти до изучаването на компютърните науки. Те са с ниска крайна цена но достатъчно мощни за разработването на различни проекти.

Библиотеките съдържат предефинирани функции в езика.

Съществуват 2 типа библиотечни файлове – статични и динамични.

Цел на дипломната работа:

Да бъде създадена статична библиотека на C за Ардуино заедно със Асемблерска част която да бъде включена в библиотеката. Целта на библиотеката е да покаже взаимодействието между C и Assembler.

От тази цел произтичат следните задачи:

- *Изследване на съществуващите C библиотеки и анализ на datasheet-овете на микроконтролерите за които е предназначена асемблерската част*
- *Анализ на целевите процесорни архитектури за които е предназначена библиотеката.*
- *Дефиниране на изискванията към библиотеката*
- *Използвани софтуерни инструменти*
- *Кодиране на библиотеката*
- *Постигнати резултати. Бъдещо Развитие*

Глави:

Увод

- **Глава I - Изследване на съществуващите C библиотеки и анализ на datasheet-овете на микроконтролерите за които е предназначена асемблерската част**
- **Глава II - Анализ на целевите процесорни архитектури за които е предназначена библиотеката.**
- **Глава III - Дефиниране на изискванията към библиотеката**
- **Глава IV - Използвани софтуерни инструменти**
- **Глава V – Кодиране на библиотеката**
- **Глава VI – Постигнати резултати. Бъдещо Развитие**

Заклучение

I - Изследване на съществуващите C библиотеки и анализ на datasheet-овете на микроконтролерите за които е предназначена асемблерската част

Както стана ясно в увода, съществуват 2 типа библиотечни файлове (библиотеки) – статични и динамични. Статичната библиотека представлява архив с разширение .a , който се състои от обектни файлове с разширение .o . Динамичната библиотека от своя страна представлява файл с разширение .so (shared object) . Когато се работи под управление на GNU/Linux OS има основна директория, която съдържа динамичните библиотеки -> /usr/lib. По отношение на header файла той представлява файл с декларираните функции които ще бъдат налични в основната C програма или както е в този случай -> в библиотеката.

Когато се напише #include <mylib.h> -> header файла се търси в основната директория ->/usr/lib. Когато се напише #include "mylib.h" -> header файла се търси в конкретната директория, в която потребителят се намира в момента.

Примери за C библиотеки:

- stdlib.h
- stdio
- math.h

Нека разгледаме библиотеката libc и нейния header файл stdlib.h . Библиотеката libc съдържа основни функции на езика които могат да бъдат използвани в различни C програми. Header файлът stdlib.h съдържа различни функции като например:

- atoi
- atol
- malloc
- free

Функцията `atoi` получава като аргумент символ или символен низ- `string` и го преобразува в цяло число от тип `int`.

Функцията `atol` получава като аргумент символ или символен низ- `string` и го преобразува в число от тип `long`.

Чрез функцията `malloc` се запазват байтове в паметта. Например:
`malloc(sizeof(int)).`

Чрез функцията `free` се освобождават вече заети байтове в паметта. Например:

```
int a=5;
```

```
free(a);
```

Нека разгледаме библиотеката `stdio` и нейния header файл `stdio.h` . Библиотеката `stdio` съдържа основни функции на езика които могат да бъдат използвани за стандартни входно-изходни операции (I/O). Header файлът `stdio.h` съдържа различни функции като например:

- `fopen`
- `printf`

Чрез функцията `fopen` отваря stream от байтове в паметта. Получава като аргументи името на stream-а който трябва да отвори и различен режим за отваряне
Например:

```
fopen("example","r");
```

Нека разгледаме библиотеката `math` и нейния header файл `math.h` . Библиотеката `math.h` съдържа математически функции и дефинирани константи чрез препроцесорната директива `#define` . Header файлът `stdio.h` съдържа различни константи като например:

```
#define PI=3.14;
```

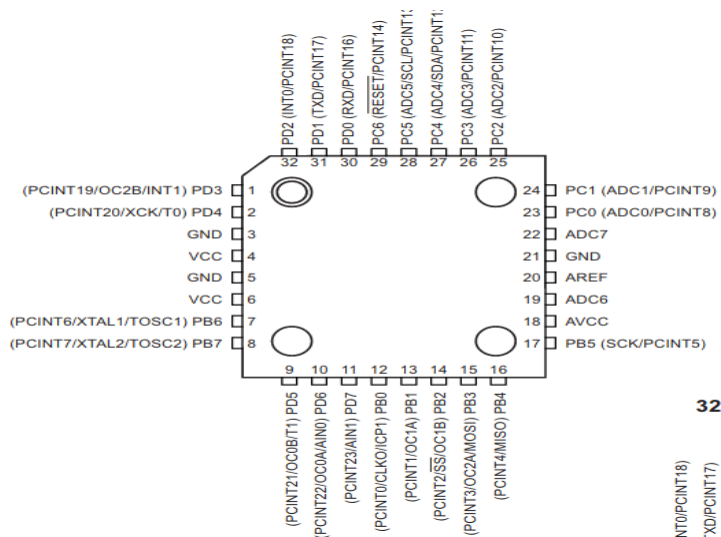
За разлика от разгледаните вече съществуващи библиотеки, статичната библиотека, която е цел на настоящата дипломна работа, ще съдържа асемблерска част, за да може действието ѝ да бъде най-бързо.

За да се запознаем със микроконтролерите за които е предназначена библиотеката -> Atmega 328P и RP2040 е необходимо да анализираме тяхната документация – datasheet-овете им.

Следва кратък анализ на datasheet-а на Atmega 328P, след това и на RP2040.

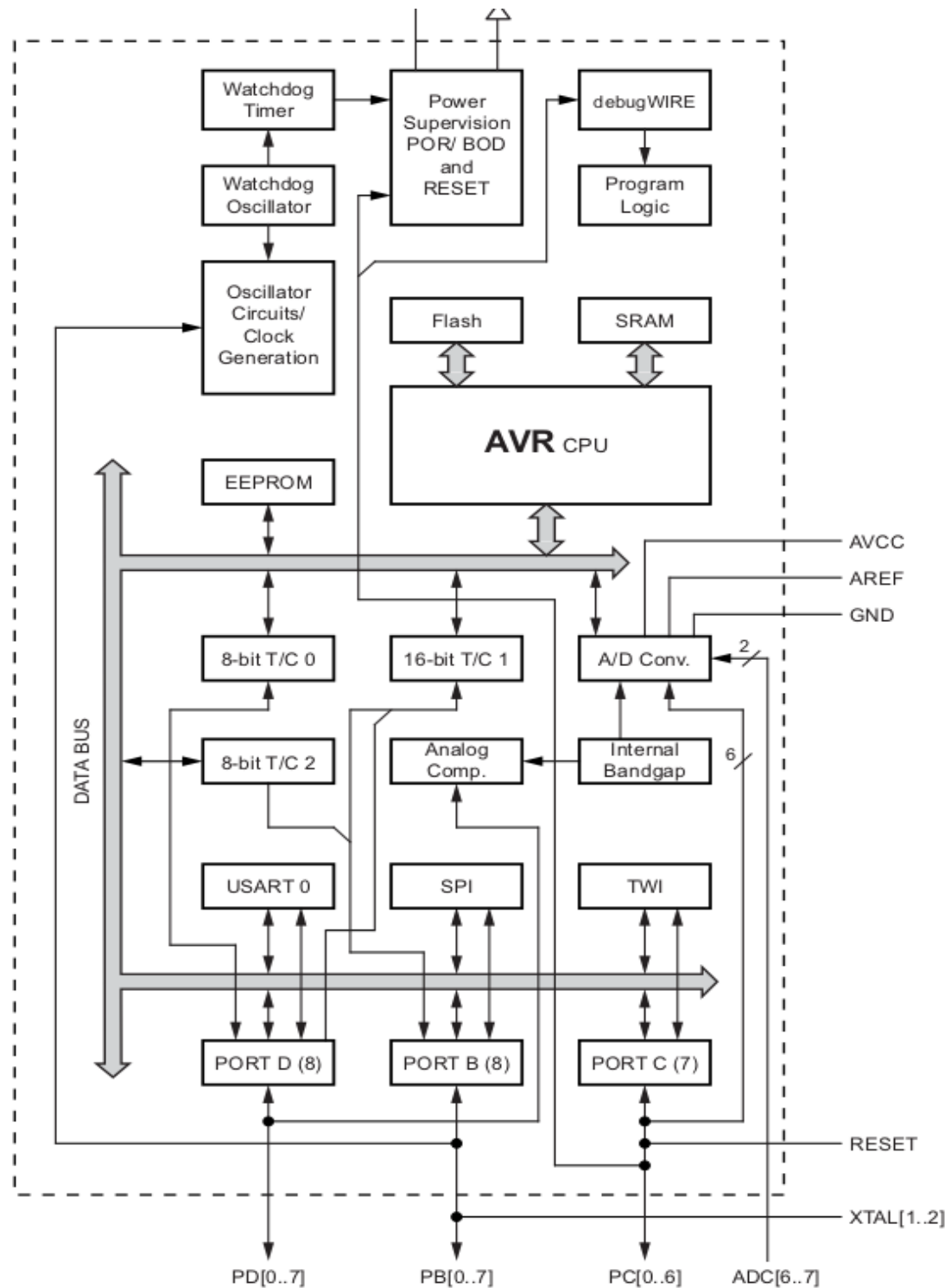
Кратък анализ на datasheet-а на Atmega 328P

Според datasheet-а Atmega 328P е 8-bit RISC базиран микроконтролер. Може да бъде използван GNU Assembler-а, който е съвместим с RISC-базирани устройства.



Фиг. 2

Фигура 2 показва достъпните пинове за използване на Atmega 328P.



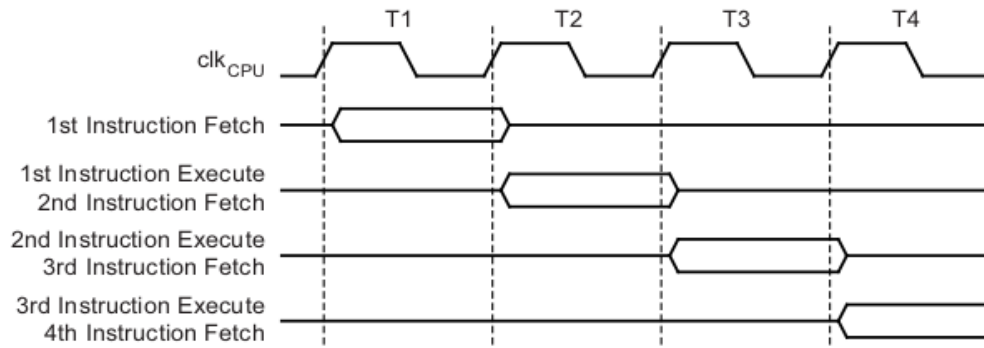
Фиг. 3

Фигура 3 показва процесора, паметите и адресната шина на Atmega 328P

Всеки един процесор изпълнява следните задачи върху процесорна инструкция:

- прихващане на инструкцията
- декодиране на инструкцията
- изпълнение на инструкцията

Figure 6-4. The Parallel Instruction Fetches and Instruction Executions



Фигура 6.4 от datasheet-а показва как на всеки 1 clock-cycle на clock сигнала последователно се прихващат, декодират и изпълняват инструкциите.

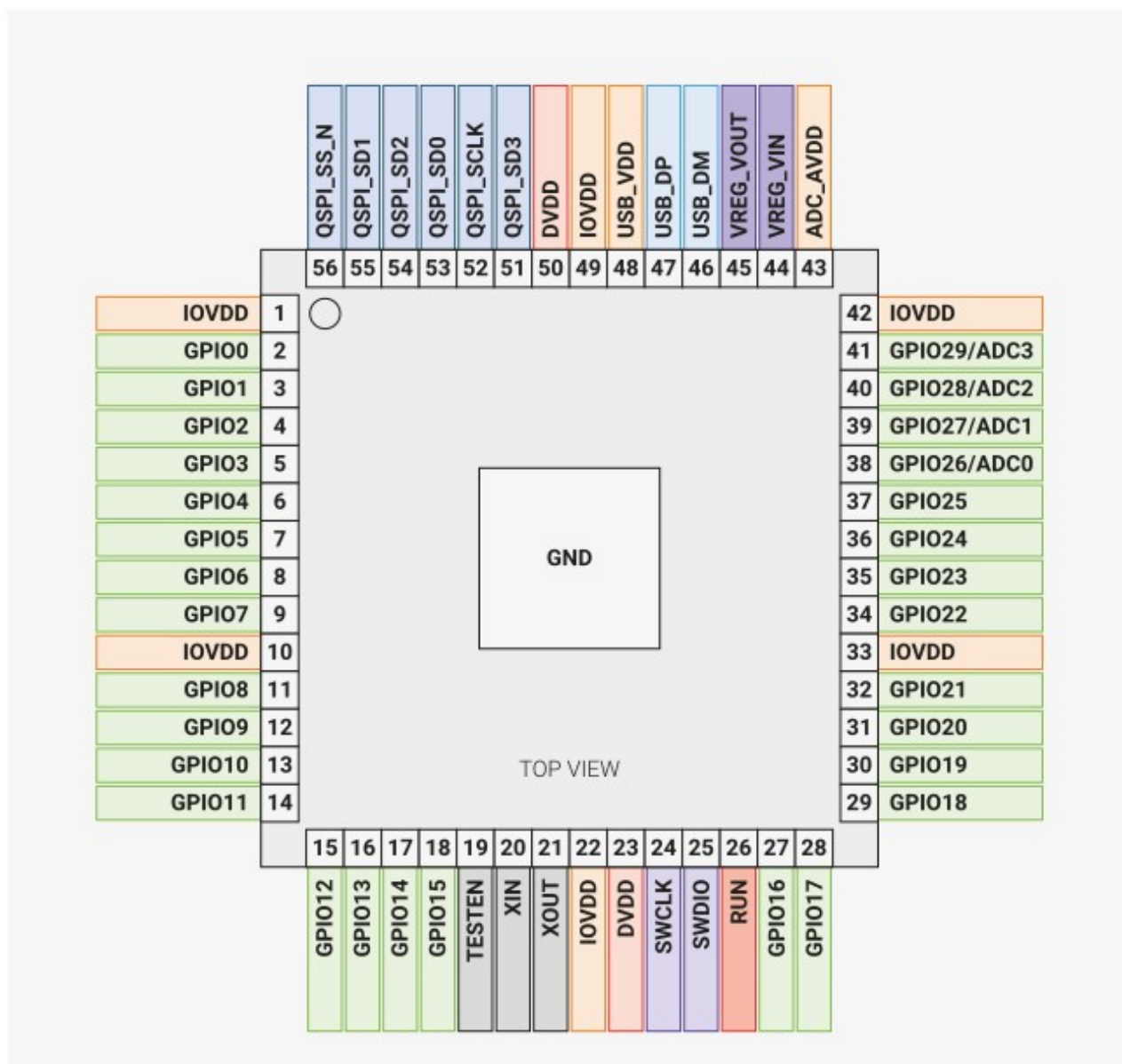
За да можем да програмираме на Assembler е нужно да знаем какви регистри на процесора на микроконтролера са достъпни за използване.

Figure 6-2. AVR CPU General Purpose Working Registers

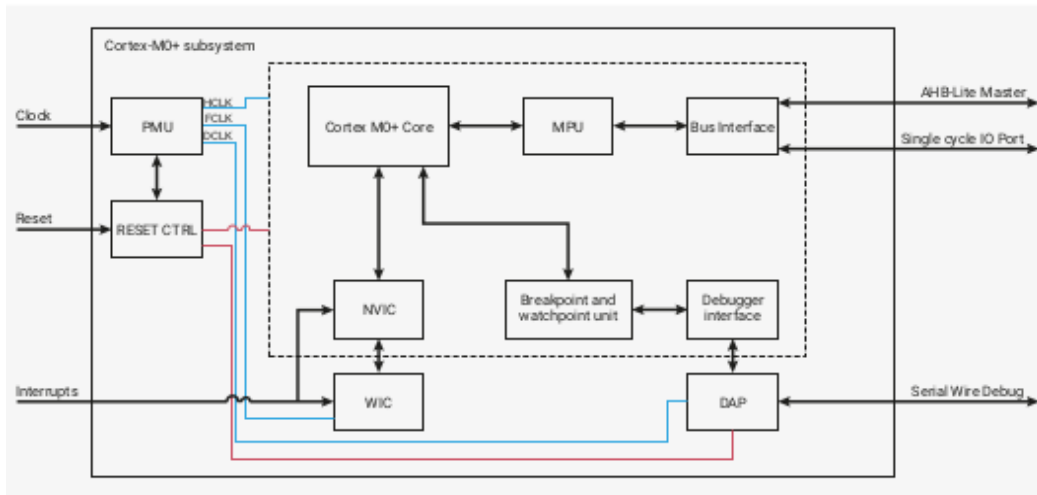
	7 0		Addr.	
General Purpose Working Registers		R0	0x00	
		R1	0x01	
		R2	0x02	
		...		
		R13	0x0D	
		R14	0x0E	
		R15	0x0F	
		R16	0x10	
		R17	0x11	
		...		
		R26	0x1A	X-register Low Byte
		R27	0x1B	X-register High Byte
		R28	0x1C	Y-register Low Byte
		R29	0x1D	Y-register High Byte
		R30	0x1E	Z-register Low Byte
		R31	0x1F	Z-register High Byte

Фигура 6.2 показва достъпните регистри.

Кратък анализ на datasheet-а на RP2040



Фигура 4 показва достъпните пинове за използване на RP2040



RP2040 е базиран на процесор Cortex M0+. Той е 32 битов RISC-базиран процесор. Поддържа защита на паметта.

II - Анализ на целевите процесорни архитектури за които е предназначена библиотеката.

Както стана ясно от I -ва глава, библиотеката е предназначена за микроконтролерите Atmega 328P и RP2040. RP2040 е ARM RISC -базиран. Atmega 328P е Atmel-RISC базиран. С развитието на едноплатковите компютри RISC-базирани устройства ще увеличават своя дял.

Анализ на ARM архитектурата

ARM архитектурата разполага с общо 32 регистъра. Има регистри с общо и специално предназначение.

Примери за регистри със общо предназначение са:

- r0
- r1
- r2

Примери за регистри със специално предназначение са:

- r13 -> Stack pointer
- r14 -> Link Register
- r15 -> Program Counter

r14 съдържа адреса на следващата инструкция след branch инструкцията.

r15 може да се използва като указател към следващата процесорна инструкция която трябва да бъде изпълнена.

ARM-базираният процесор поддържа следните видове инструкции:

- аритметично-логически операции
- операции за разклоняване на потока – branch
- multiply операции

Пример за аритметична операция на GNU Assembler:

```
.text
.global main
.func main
main:
mov r0, #7
mov r1, #4
sub r2,r0,r1
bx lr
```

Пример за разклоняване на потока на програма на GNU Assembler:

```
.text
.global main
.func main
main:
mov r0, #7
mov r1, #4
cmp r0,r1
beq eql
bne noteql
eql:
mov r0,#1
b end
noteql:
mov r0,#0
b end
end:
bx lr
```

Пример за аритметична операция на GNU Assembler:

```
.text
.global main
.func main
main:
mov r0, #5
mov r1, #6
mul r2, r0, r1
bx lr
```

Пример за побитова операция на GNU Assembler:

```
.text
.global main
.func main
main:
mov r0, #1
mov r1, #0
mov r2, r0 & r1
bx lr
```


III - Дефиниране на изискванията към библиотеката

В резултат от анализа на ARM архитектурата направен във втора глава, и от развитието на едноплатковите компютри в момента, може да се каже, че статичната библиотека трябва да бъде налична за RISC-базирани устройства. Статичната библиотека трябва да поддържа микроконтролерите:

- Atmega 328P
- RP2040

За основната асемблерска част на библиотеката е необходимо да се използва асемблер който е съвместим с ARM RISC процесорната архитектура. Такъв асемблер е GNU Assembler-a. За него съм споменал още в Увода.

Асемблерът е част от GNU проекта, свободен за използване.

Нужно е използването на свободен асемблер заради независимостта при разработване. Използването на свободен софтуер и свободния асемблер улеснява работата в екип при разрастване на проекта.

Библиотеката трябва да има следните функционалности:

-

IV - Използвани софтуерни инструменти



За реализацията на проекта е избран сетът от инструменти GNU Tools.

Причината за избора е, че всички инструменти са или с отворен лиценз като редактора Vim или са част от GNU Project.

Те са свободни мултиплатформени софтуери със терминален (не графичен) интерфейс, което позволява по-бърза работа. Налични са за повечето GNU/Linux дистрибуции както и за mac OS и MS Windows.

GCC (GNU Compiler Collection) е колекция от компилатори за Fortran, Ada, C,C++.

Използваните инструменти са:

- gcc
- make
- gdb
- avr-gcc
- avrdude
- binutils
- Vim
- ar

Gcc- GNU C Compiler е компилатор който може да се използва за C, C++, Assembler. Използва се за компилация на source кода до обектен файл и до краен изпълним файл. Подават се допълнителни параметри за компилацията.

Пример:

```
gcc -c -o example.o example.c //Компилиране до обектен файл
```

```
gcc example.o -o example //Компилиране до краен изпълним файл
```

```
./example //Изпълнение на изпълнимия файл
```

Make- GNU Make е build система за автоматизация. Използва се за автоматизиране на компилацията за C, C++ и Assembler програми. В Makefile се пишат целите и файловете, нужни за тяхното изпълнение. След това се пише командата която се изпълнява при дадената цел. Изключително полезна е при големи проекти.

Пример:

```
example.o: example.c
```

```
    gcc -c -o example.o example.c //Компилиране до обектен файл без дебъг символи
```

```
example: example.o
```

```
    gcc example.o -o example //Компилиране до краен изпълним файл
```

```
run: example
```

```
    ./example //Изпълнение на изпълнимия файл
```

GDB- GNU GDB е дебъгер за C и Assembler. Използва се след като програмата е компилирана със дебъг символи чрез параметъра -g. За да се проследи изпълнението на програмата е нужно да се използват breakpoints. Чрез командата run започва изпълнението на програмата в дебъгера. Чрез командата next дебъгера преминава към следващия ред на програмата.

Пример:

```
gcc -c -g -o example.o example.c
```

```
gcc example.o -o example
```

След като е компилиран крайния файл се дебъгва чрез:

```
gdb ./example
```

```
breakpoint main
```

```
run
```

```
next
```

AVR-GCC е модифициран GCC компилатор, който компилира source файла до .hex файл, който да се качи на целевия микроконтролер.

AVRDUDE е програмата, чрез която .hex файла се качи на целевия микроконтролер.

BINUTILS съдържа свободния асемблер, който ще бъде използван за основната част на библиотеката, която е цел на настоящата дипломна работа.

Vim е терминален текстов редактор който може да се използва за всички програмни езици. Поддържа възможност за редактиране на огромни файлове. Поддържа функциите Undo, Redo с пълната история на файла. Разполага с различни режими на работа:

- Normal -> за навигация в текста
- Visual -> за операции върху текста (Cut,Copy,Paste)
- Insert -> за въвеждане
- Visual Block -> за операции върху множество редове
- Replace -> за замяна в текста

ar представлява архиватор, чрез който се създават статичните библиотеки

V - Кодирание на библиотеката

VI - Постигнати резултати. Бъдещо Развитие

В темата на настоящата дипломна работа централно място заемат С и Асемблер и връзката между тях. Във Въведението бяха поставени 6 цели. Всяка глава представлява изпълнение на поставена преди това цел.

В първа глава бяха разгледани микроконтролерите за които е предназначена библиотеката и сравнение между тях, с цел по-доброто разбиране на Асемблера за тях. Във втора глава беше разгледана ARM архитектурата. В трета глава бяха поставени изискванията към библиотеката. В четвърта глава бяха разгледани софтуерните инструменти, използвани за реализацията на проекта. Свободните софтуерни инструменти гарантират по-голяма независимост за разработването на проекта и позволява бъдещо взаимодействие в екип. В пета глава беше разгледано кодирането на статична библиотека, включваща в себе си С част и основна асемблерска част.

Асемблерската част е представена в отделен файл. Той е компилиран до обектен файл и обектния файл заедно със С обектния файл са архивирани в статична библиотека чрез ar. При компилация на програма в която се използва библиотеката е необходимо да се посочи, че библиотеката която се използва не се намира в стандартната директория за библиотеките `/usr/lib` а в конкретната директория в която се прави компилацията.

Проектът е изцяло open-source заради по-доброто бъдещо развитие. С развитието на микроконтролерите за обучение като RP2040 паралелно се развиват и асемблерните езици на които могат да бъдат програмирани. Развиват се нови асемблери които могат да покриват множество устройства.

Например AVR Assembler, с който се покриват множество AVR- базирани устройства.

За свободните асемблери

Заключение

Използвани източници:

1. [Arduino Atmega 328P Datasheet](#)
2. [RP2040 Datasheet](#)
- 3.