A photograph of three apples on a white surface. In the foreground, a red apple is in sharp focus. Behind it, a green apple is slightly out of focus, and to the left, a yellow apple is also out of focus. The background is a plain white surface.

# Програмиране на С за вградени системи



гл. ас. д-р инж. Любомир Богданов

лаб. 1362, лаб. 1312

тел. +359 2 965 3362

[lbogdanov@tu-sofia.bg](mailto:lbogdanov@tu-sofia.bg)



# Съдържание

- 1** Въведение в микропроцесорните системи
- 2** Основни понятия в програмирането на  $\mu\text{C}$
- 3** Създаване на изпълним код
- 4** Сегменти на паметта
- 5** Видове библиотеки
- 6** Маски
- 7** Volatile променливи



# Въведение в микропроцесорните СИСТЕМИ

## Микроконтролер

**LDO**

**LCD**

**UART**

**Flash**

**μPU**

**USB**

**SRAM**

**I<sup>2</sup>S**

**Ethernet  
10/100**

## Персонален Компютър

**ATX  
Power**

**Video**

**UART**

**HDD**

**CPU**

**USB**

**DRAM**

**Sound**

**Ethernet  
10/  
100/  
1000**



# Въведение в микропроцесорните СИСТЕМИ

Микропроцесор –  $x(10 \div 100)$  Mhz

Памет за данни (SRAM) -  $x(10 \div 100)$  kB

Програмна памет (Flash) -  $x(10 \div 100)$  kB

Периферни модули:

GPIO

RTC

I<sup>2</sup>C

I<sup>2</sup>S

SPI

Timer

CAN

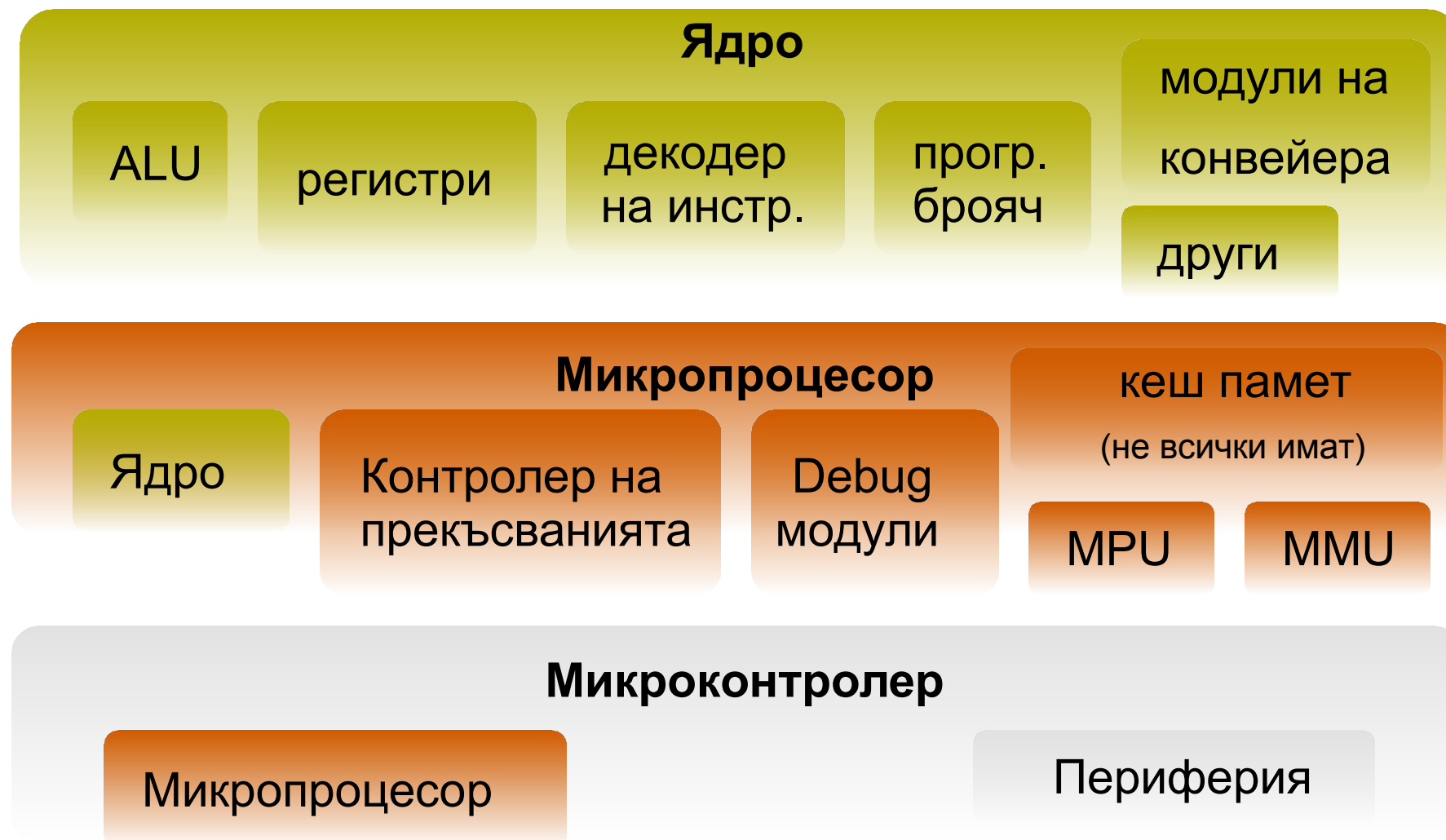
UART

USB

Ethernet



# Въведение в микропроцесорните системи





# Въведение в микропроцесорните системи

Видове микроконтролери:

## **General Purpose**

Разглеждани в курса Микропроцесорна схемотехника  
(MSP430/2, Tiva, PIC, STM32, AT32)

## **Application Specific**

Доближават се до персоналните компютри като  
параметри (оттам идва SoC) + специална периферия  
според областта на приложение (Sitara, i.MX, SPEAr).



# Въведение в микропроцесорните системи

Начини за програмиране:

μCU, поставен директно в **програматор**

Програматор, свързан към разработваното устройство  
посредством **специален интерфейс (ICSP)**

**USB→JTAG** адаптер

**Стандартен интерфейс** (USB, UART, Ethernet и др.)

Директно във Flash  
посредством Bootloader

Във файловата система  
при използване на OS

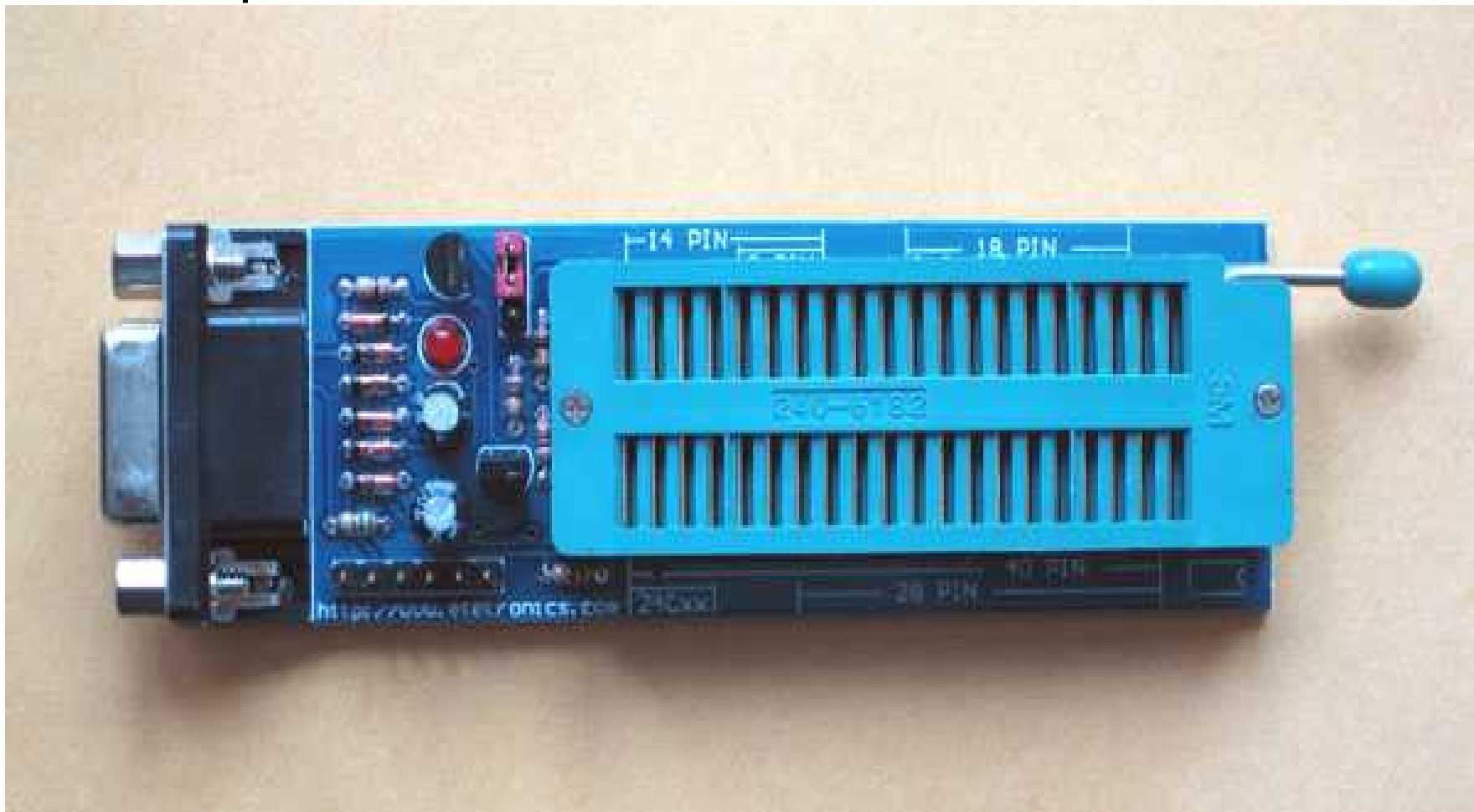
**ROM емулатор/програматор** (при използване на външна Flash)

Външна преносима **Flash памет** (SD Card)



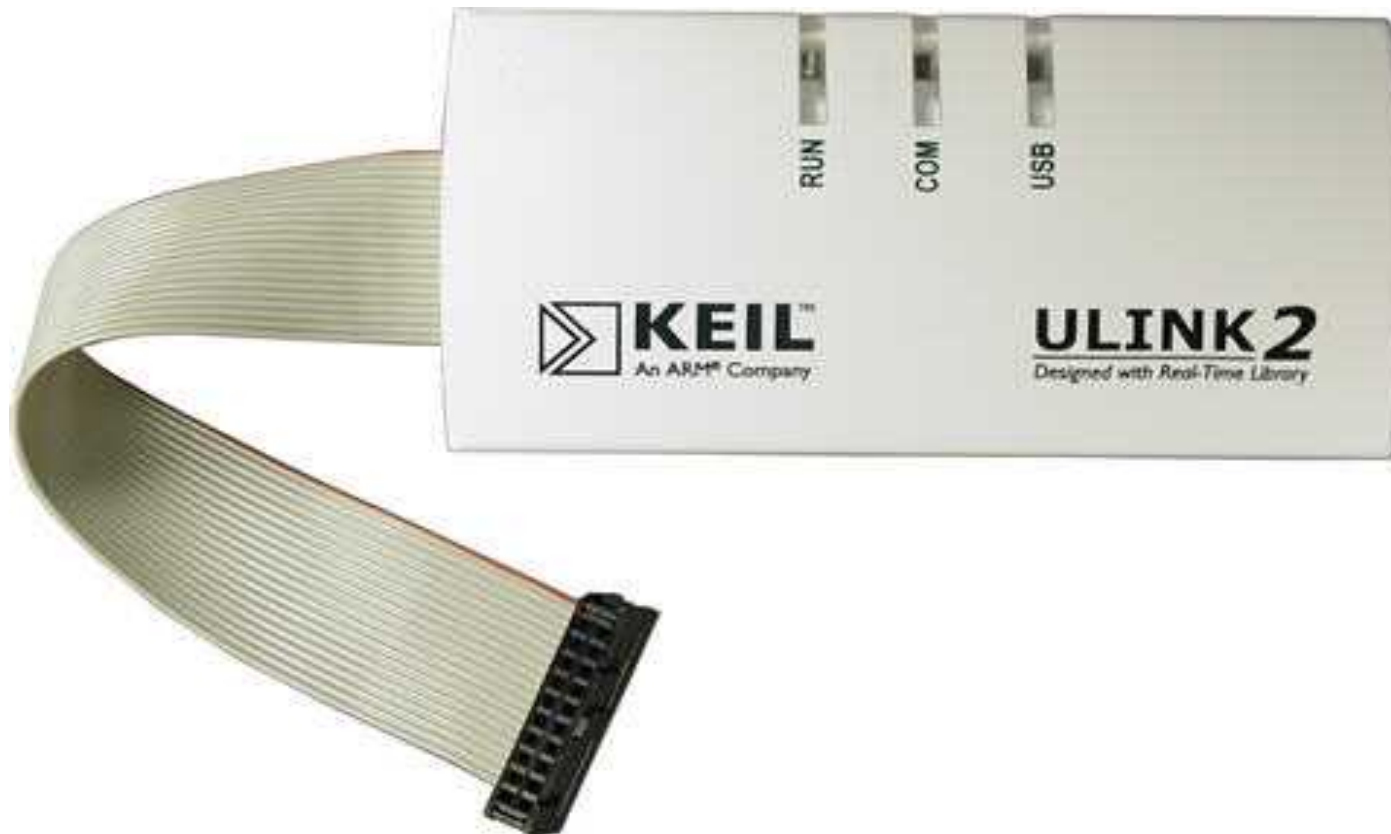
# Въведение в микропроцесорните системи

JDM програматор за програмиране на микроконтролери  
от Microchip



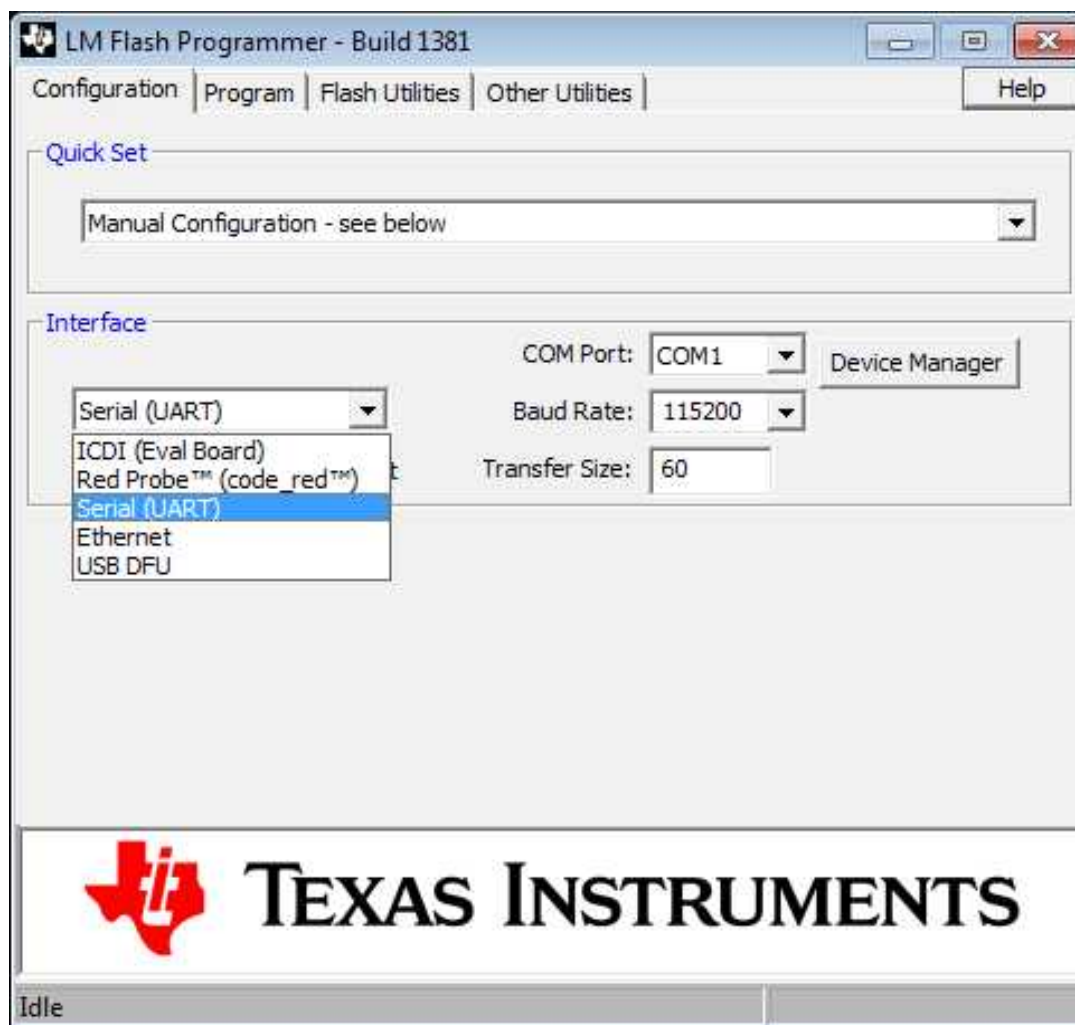
# Въведение в микропроцесорните системи

USB → JTAG адаптер за програмиране на микроконтролери от различни фирми



# Въведение в микропроцесорните СИСТЕМИ

Програмиране директно във Flash посредством Bootloader – изисква специална програма за комуникация. Показана е LM Flash.





# Въведение в микропроцесорните системи

Програмиране чрез директно копиране на програмата във файловата система при използване на OS. Използва се командата “secure copy” за Linux.

```
lbogdanov@laptop: ~  
File Edit View Search Terminal Help  
lbogdanov@laptop:~$ scp mydriver.a user@192.168.10.100:/drivers/destination
```



# Основни понятия в програмирането на $\mu C$

## Програми на асемблер

Състоят се от следните полета: директиви, етикети, инструкции, операнди и коментари.

На всеки един ред се въвежда по една инструкция с или без операнди.

Всяка инструкция съдържа следните битови полета:

- \* Код на операцията (=мнемоника);
- \* служебни полета (не се пишат, асемблерът сам ще ги определи).



# Основни понятия в програмирането на $\mu C$

## Програми на асемблер

След мнемониката се добавят **операндите**. Ако няма операнди се оставя празно поле:

*Пример:*

(Мнемоника) (Операнд 1) (Операнд 2)

add.w

R14,

R15

swpb

R14,

clra



# Основни понятия в програмирането на $\mu\text{C}$

## Програми на асемблер

MOV R5,R4

- Instruction code: 0x4504

| Op-code | S-reg   | Ad       | B/W     | As       | D-reg   |
|---------|---------|----------|---------|----------|---------|
| 0 1 0 0 | 0 1 0 1 | 0        | 0       | 0 0      | 0 1 0 0 |
| MOV     | R5      | Register | 16-Bits | Register | R4      |



# Основни понятия в програмирането на $\mu\text{C}$

При въвеждането на асемблерни програми в средите освен инструкцията се добавят етикети и коментари.

| Етикети | Мнемоника | Операнди | Коментари              |
|---------|-----------|----------|------------------------|
| L1      | xor.b     | #0xF0,   | &P3OUT<br>;Toggle pins |
|         | xor.w     | #0x0F,   | &PJOUT<br>;Toggle pins |
|         | call      | #Delay   | ;Call subroutine       |
|         | jmp       | L1       | ;Repeat program        |





# Основни понятия в програмирането на $\mu C$

## Програми на асемблер

- \* **Етикетите** се използват за реализация на цикли или подпрограми;
- \* **Коментарите** са за информация на програмиста и те не влизат в крайния двоичен код.



# Основни понятия в програмирането на $\mu\text{C}$

## Програми на асемблер

\* Обикновено в началото и края на програмата може да има специални думи, наречени **директиви**, които се използват от асемблера за конфигурация на проекта и/или подпомагане на програмиста:

```
ORG    0FFFFEh ;задай адрес на ресет хендлер  
#include "msp430.h" ;включи заглавен файл  
END ;край на асемблерния файл
```



# Основни понятия в програмирането на $\mu\text{C}$

Инструкциите на асемблер са различни за различните микропроцесори.

Асемблерните директиви са различни за различните развойни програми (toolchain).




# Основни понятия в програмирането на $\mu\text{C}$

Програмирането на C и C++ е независимо от използвания микроконтролер.

Все пак някои от библиотеките са зависими от използвания микроконтролер.

C и C++ позволяват без никакви (или с много леки) промени в кода една и съща програма да се изпълнява на два различни микроконтролера (Portability).



# Основни понятия в програмирането на $\mu\text{C}$

```
#include <msp430.h>
void main( void )
{
    unsigned int i;

    WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
    P3DIR |= 0x10; //P3DIR е микроконтролерно-зависим макрос
    P3OUT = 0x00;
    for( ; ; ) //for( ) е микроконтролерно-независима конструкция
    {
        P3OUT ^= 0x10;
        for(i = 0; i < 65530; i++) { }
    }
}
```



# Основни понятия в програмирането на $\mu\text{C}$

За да се разграничат понятията “приложни програми за персонален компютър” от “програми за микроконтролер”, то последните се наричат още фърмуер (**firmware**).

Когато се компилира програма, която директно ще се изпълнява от микроконтролера, тя се нарича **bare-metal firmware**.

Алтернативният вариант на bare-metal firmware-a е програма, която се изпълнява заедно с **embedded операционна система** (uClinux, FreeRTOS, SafeRTOS, TinyOS, Android, Windows CE и др.).



# Основни понятия в програмирането на $\mu C$

## Предимства на OS:

- (псевдо)паралелно изпълнение на две и повече програми
- независимост от хардуерната конфигурация (ако има съответните драйвери)
- лесно пренасяне на кода в други вградени системи (Portability).

## Недостатъци:

- използва по-голям хардуерен ресурс (ROM, RAM,  $\mu PU$  и др.)

# Основни понятия в програмирането на $\mu\text{C}$

**Програмен модел** – това е наборът от регистри на микропроцесора, които са достъпни (четене/запис) за програмиста.

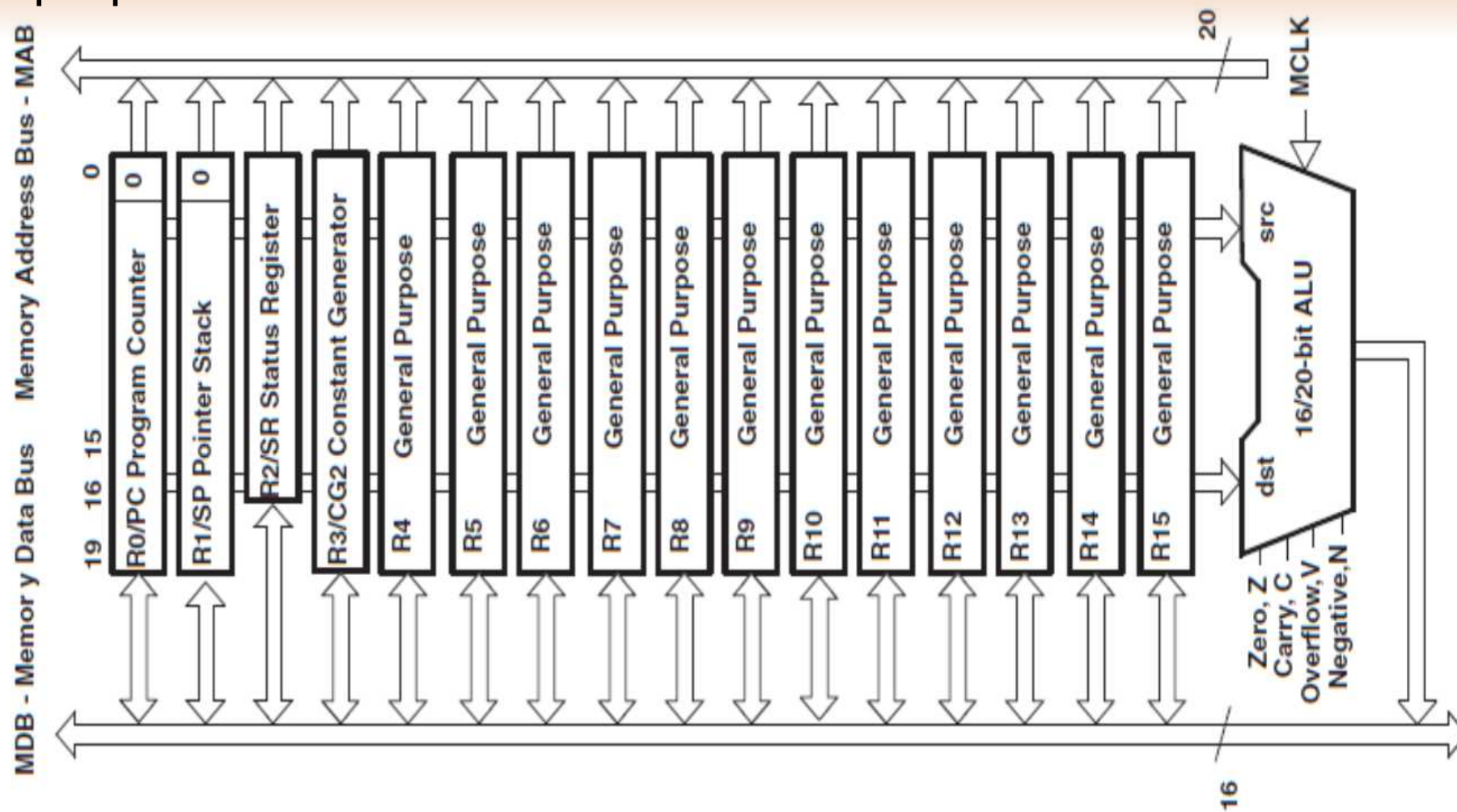


Figure 4-1. MSP430X CPU Block Diagram





# Основни понятия в програмирането на $\mu$ C

**Intrinsic functions** – елементарни функции, осигуряващи достъп до регистри на микропроцесора. Стандартният език C реализира програми, които могат да пишат/четат само в памет, външна за микропроцесора.

Ако се нуждаем от достъп до регистри вътре в самия микропроцесор, трябва да използваме intrinsic функции или inline асемблер (код на асемблер в C програма).

*Пример:*

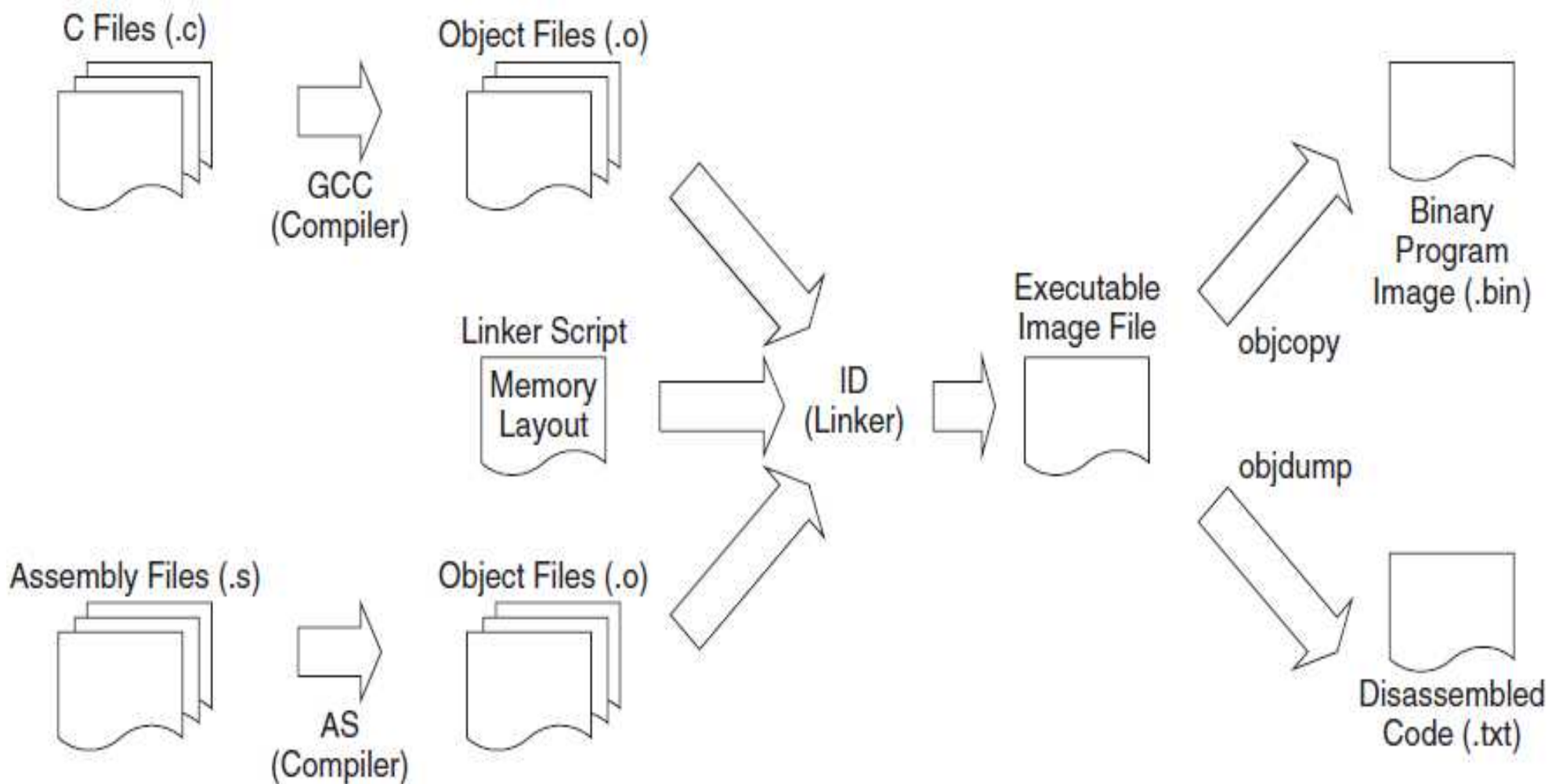
```
unsigned short __set_SP_register(unsigned short src);
```

*Пример:*

```
__asm("      mov      r1, #0x2000\n");
```

# Създаване на изпълним код

## Етапи в създаването на изпълнимия код



# Създаване на изпълним код

Демонстрация на пример във вградена система.





## Създаване на изпълним код

Анализ на използваните ресурси – написали сме програмата си, как да я оценим по качество?

За колко време се изпълнява тя?

Каква мощност се консумира от системата при нейното изпълнение?

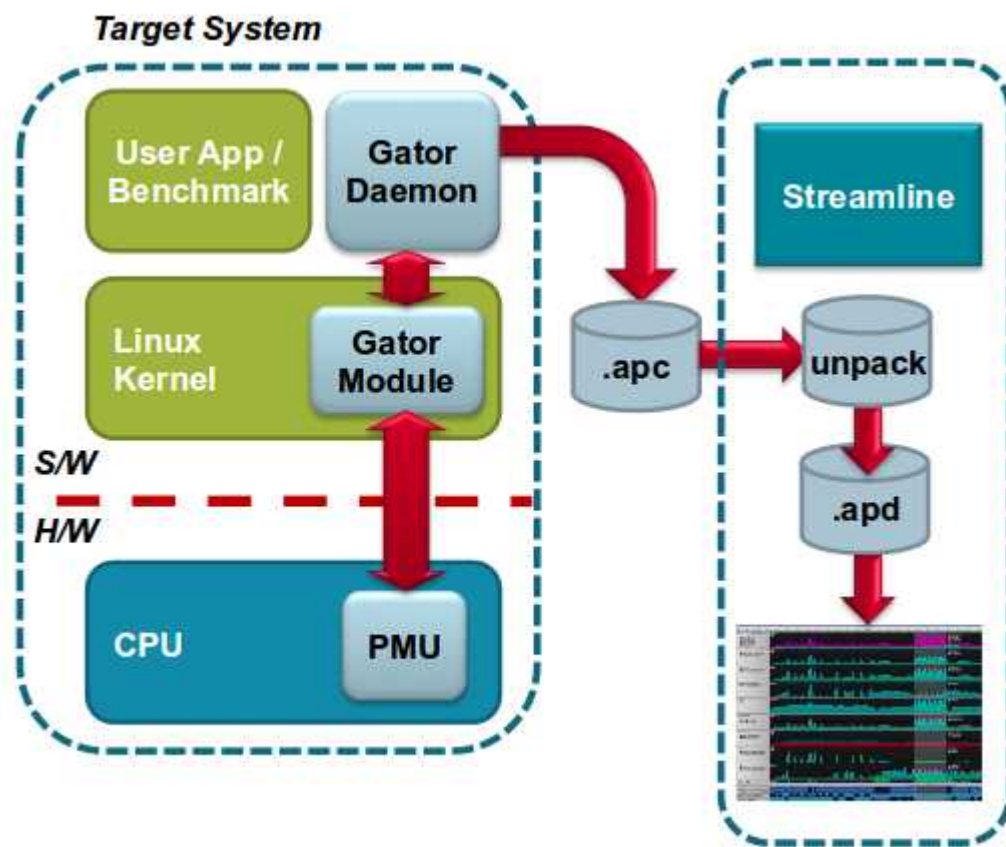
Колко четения/записи от SD карта са настъпили (ако има)?

Колко четения/записи от кеш паметта са настъпили (ако има)?

Други

# Създаване на изпълним код

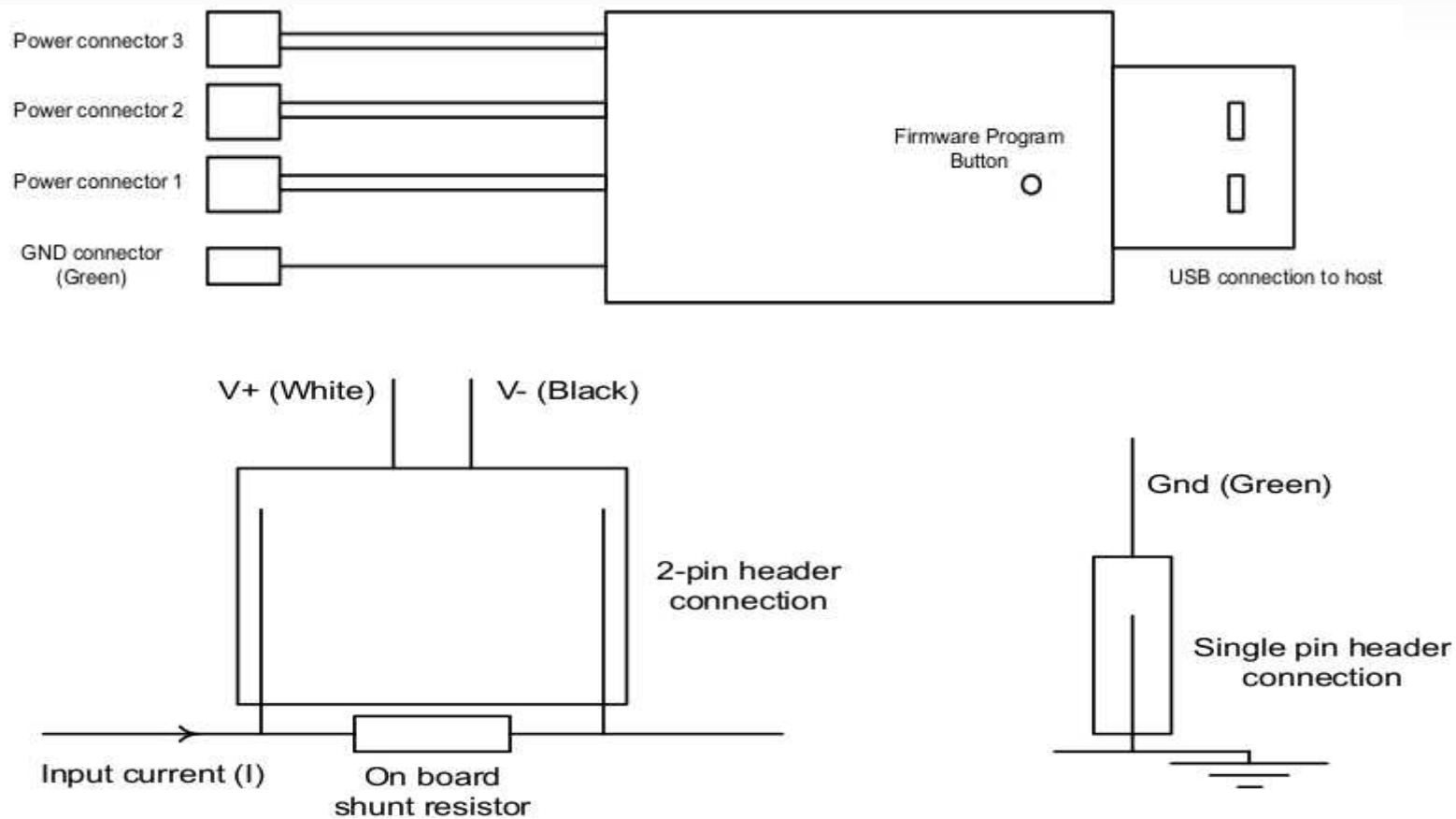
Използват се допълнителни (third party) програми. Техният брой е огромен. Тук се разглежда ARM Streamline Performance Analyzer, част от средата DS-5 Development Studio на ARM [2]. *Валиден за ARM микропроцесори.*



- Relies on “gator” kernel module and daemon
- Reads out counters and process information and dumps to file

# Създаване на изпълним код

С помощта на ARM Energy Probe може да се измери и консумираната енергия от програмата [3].



# Създаване на изпълним код





# Създаване на изпълним код

C/C++ - /home/lbogdanov/Desktop/Capture\_04.apc - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access C/C++

Project Explorer

- 10x20\_LED\_master\_i...
- 10x20\_LED\_matrix\_te...
- 11192014\_msp430g2...
- 11192014\_msp430g2...
- AM335X\_StarterWare...
- blinky\_msp430g2553...
- blinky\_msp430g2553...
- i2c\_master
- i2c\_slave
- msp430g2553\_4asovi...
- MSP430G2553\_uartst...
- seconddrv\_2

Outline

An outline is not available.

Streamline Data

192.168.7.2:3456

Filter

- Capture\_03
- Capture\_04

Capture\_03 Capture\_04

Timeline Functions

50ms

CPU Activity

- User
- System

Branch

- Mispredicted

Clock

1.1 GHz

[for\_loops.bin #2...]

Heat Map

Filter map processes

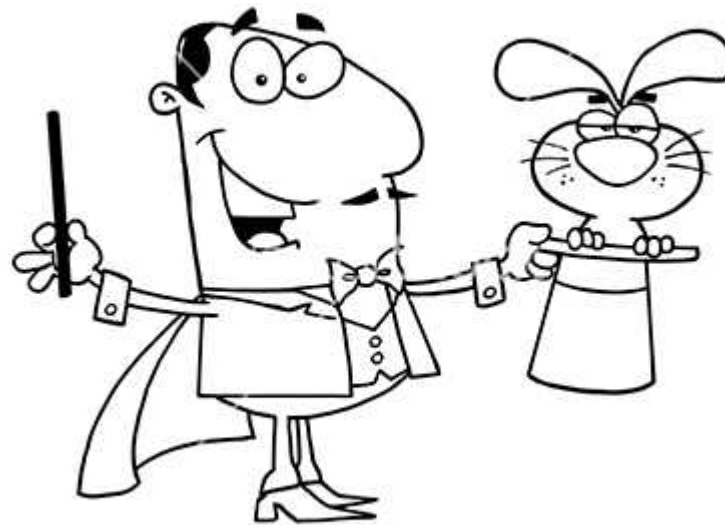
Problems Tasks Console Properties Call Graph

Click here to hide all windows and show the desktop.



# Създаване на изпълним код

Демонстрация на пример с ARM DS-5  
Streamline.









## Създаване на изпълним код

За да се автоматизира процеса на компилация се използват **Makefile** (под Linux) и **Batch** (под Windows) файлове. В тях са въведени командите, които се изпълняват от командния ред.

В IDE средите данните за компилацията се въвеждат в менюта. При компилирането на проекта тези данни се предават като параметри на съответните програми (компилатор, асемблер, линкер и т.н.)

# Създаване на изпълним код

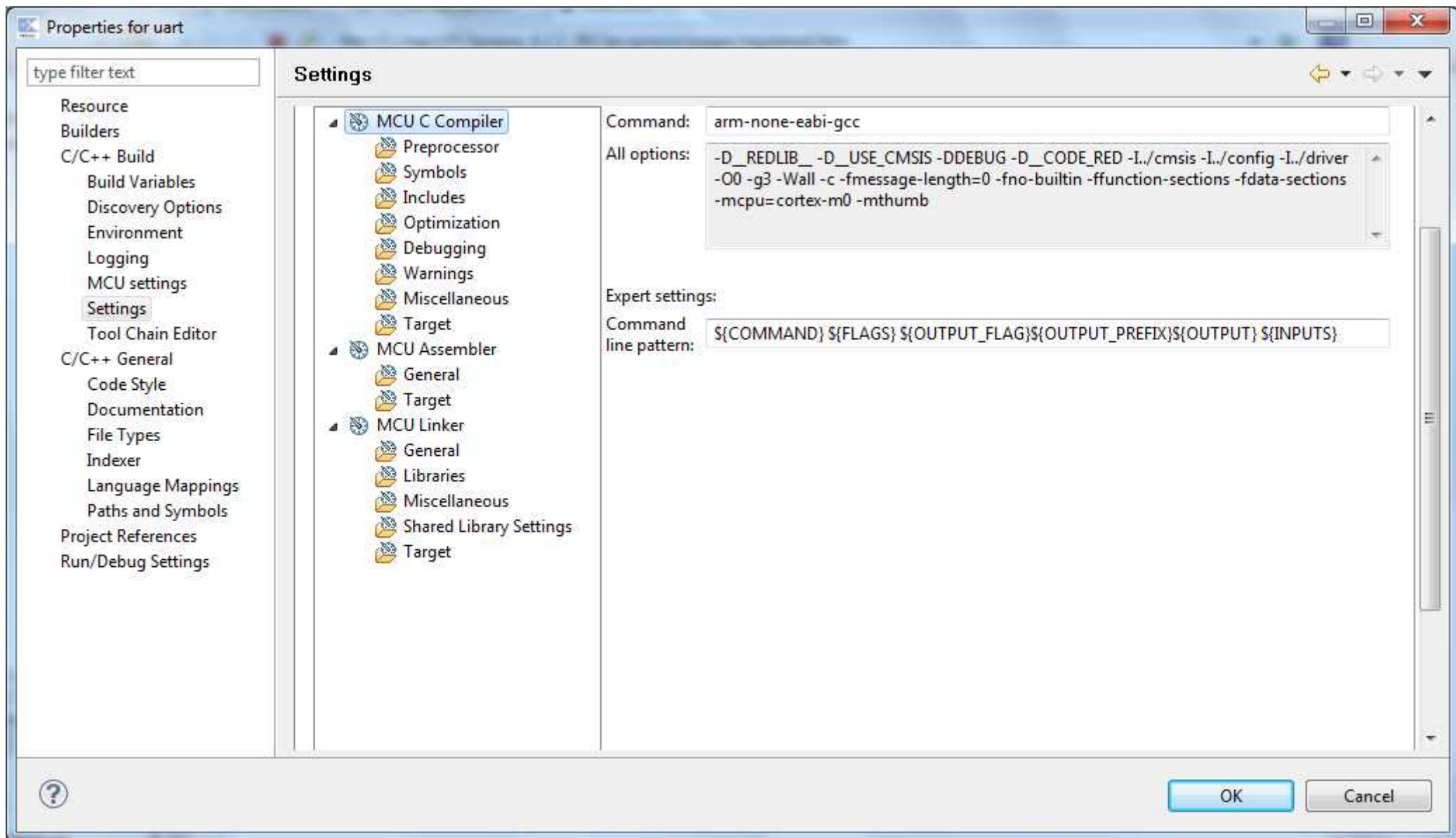
|   |                       |         |       |
|---|-----------------------|---------|-------|
|  blinky.c      | 17.3.2014 г. 13:43 ч. | C File  | 3 KB  |
|  blinky.ld     | 5.6.2012 г. 16:41 ч.  | LD File | 2 KB  |
|  Makefile      | 17.3.2014 г. 14:25 ч. | File    | 1 KB  |
|  startup_gcc.c | 5.6.2012 г. 16:41 ч.  | C File  | 10 KB |

```
1 IPATH=../../../../
2
3
4 all: blinky.bin blinky.lst
5
6 blinky.o: blinky.c
7     arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb -Wall -O2 -std=c99 -c -I${IPATH} blinky.c -o blinky.o
8
9 startup_gcc.o: startup_gcc.c
10    arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb -Wall -O2 -std=c99 -c -I${IPATH} startup_gcc.c -o startup_gcc.o
11
12 blinky.axf: blinky.o startup_gcc.o
13    arm-none-eabi-ld -Tblinky.ld blinky.o startup_gcc.o -o blinky.axf
14
15 blinky.bin: blinky.axf
16    arm-none-eabi-objcopy -O binary blinky.axf blinky.bin
17
18 #Дисасемблерен файл, може и без него.
19 blinky.lst: blinky.axf
20    arm-none-eabi-objdump -S -D blinky.axf > blinky.lst
21
22 flash:
23    LMFlash.exe -q ek-lm3s3748 -v -r blinky.bin
24
25 clean:
26    rm blinky.o startup_gcc.o blinky.axf blinky.bin blinky.lst
```

[ИЗХОД]: [ВХОД]  
[ТАБ] [ПРАВИЛО]

[ИЗХОД]: [ВХОД]  
[ТАБ] [ПРАВИЛО]

# Създаване на изпълним код





## Сегменти на паметта

**bss** - съдържа глобални и статични неинициализирани променливи (в C → глобални, static, extern променливи). Тези променливи могат да бъдат четени и презаписвани.

**data** – съдържа глобални и статични променливи, инициализирани от програмиста (в C → глобални, static и extern променливи). Тези променливи могат да бъдат четени и презаписвани.

**text** (или **code**) - съдържа const променливи, и променливи инициализирани от програмиста, които по време на изпълнение на програмата (runtime) отиват в data сегмента. Инструкциите на програмата се помещават в този сегмент.



# Сегменти на паметта

**Stack** – в програмирането под стек се подразбира място в паметта на микроконтролера, където се съхранява LIFO буфер, който отговаря за съхраняването на данни при извикванията на отделните функции (return addr, статус регистър, аргументи на функции и локални променливи, които не са static). Запис и четене от буфера става със специални инструкции (push, pop).

**Heap** – в програмирането под хийп се подразбира място в паметта на микроконтролера, където се съхраняват динамично заделените променливи (с malloc( ), new и т.н.)

**Hardware Stack** – в електрониката под хардуерен стек (dedicated hardware stack) се подразбира модул от микропроцесора, съставен от няколко регистъра, в които се записват адресите на последните инструкции от програмата (return address) преди да се извика дадена подпрограма. В някои MCU автоматично се записва и SR. Използват се специални за целта инструкции (push, pop).

# Сегменти на паметта

Памет на

Микроконтролера

Най-голям адрес

**Stack**



**Heap**

**.bss**

**.data**

**.text**

Най-малък адрес

.bin файлът  
се записва  
тук





# Сегменти на паметта

```
MEMORY
{
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x00040000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00010000
}

SECTIONS
{
    .text :
    {
        _text = .;
        KEEP(*(.isr_vector))
        *(.text*)
        *(.rodata*)
        _etext = .;
    } > FLASH

    .data : AT(ADDR(.text) + SIZEOF(.text))
    {
        _data = .;
        *(vtable)
        *(.data*)
        _edata = .;
    } > SRAM

    .bss :
    {
        _bss = .;
        *(.bss*)
        *(COMMON)
        _ebss = .;
    } > SRAM
}
```





# Видове библиотеки

## Статични

.a под Linux

.lib под Windows

## Динамични

.so под Linux

.dll под Windows



## Видове библиотеки

Използване в проект:

Включват се съответните хедърни файлове.

Включва се пътя до библиотеката (в Makefile или в меню на развойната среда → по-късно автоматично се предава на линкера).

# Маски

Числа, които чрез логически оператори (&, |, ^ и др.) се използват за промяна на индивидуални битове от даден регистър без да се засягат останалите битове.



*Пример:*

В регистър P1OUT има записана стойност 0x7A. Искаме да установим битове 1 и 3 в логическа 0, а бит 7 – в логическа единица.



# Маски

P1OUT

|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>0</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>0</b> | <b>1</b> | <b>0</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

0x7A



# Маски

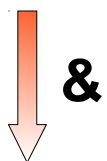
|       |   |   |   |   |   |   |   |   |      |
|-------|---|---|---|---|---|---|---|---|------|
| P1OUT | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0x7A |
|       | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |
| MASK  | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0xF5 |
|       | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |



# Маски

|       |   |   |   |   |   |   |   |   |      |
|-------|---|---|---|---|---|---|---|---|------|
| P1OUT | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0x7A |
|       | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |

|      |   |   |   |   |   |   |   |   |      |
|------|---|---|---|---|---|---|---|---|------|
| MASK | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0xF5 |
|      | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |





# Маски

**P1OUT**

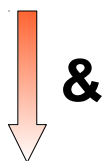
|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>0</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>0</b> | <b>1</b> | <b>0</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

**0x7A**

**MASK**

|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>1</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>0</b> | <b>1</b> | <b>0</b> | <b>1</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

**0xF5**



**P1OUT**

|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>0</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

**0x70**



# Маски

P1OUT

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

0x70





# Маски

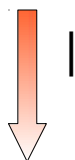
|       |   |   |   |   |   |   |   |   |      |
|-------|---|---|---|---|---|---|---|---|------|
| P1OUT | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0x70 |
|       | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |
| MASK  | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x80 |
|       | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |



# Маски

|       |   |   |   |   |   |   |   |   |      |
|-------|---|---|---|---|---|---|---|---|------|
| P1OUT | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0x70 |
|       | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |

|      |   |   |   |   |   |   |   |   |      |
|------|---|---|---|---|---|---|---|---|------|
| MASK | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x80 |
|      | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |





# Маски

**P1OUT**

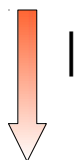
|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>0</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

**0x70**

**MASK**

|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>1</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

**0x80**



**P1OUT**

|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>1</b> | <b>1</b> | <b>1</b> | <b>1</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |
| 7        | 6        | 5        | 4        | 3        | 2        | 1        | 0        |

**0xF0**

# Маски



//На този ред P1OUT е 0x7A

**P1OUT &= 0xF5;** (аналогично на P1OUT = P1OUT & 0xF5;)

**P1OUT |= 0x80;** (аналогично на P1OUT = P1OUT | 0x80;)

//На този ред P1OUT е 0xF0



## Volatile променливи

Използват се, за да укажем изрично на компилатора ДА НЕ оптимизира дадена променлива. Оптимизацията на кода понякога е нежелана.

Такива примери са:

Когато се дефинира указател към адрес

Когато се използва обща променлива (shared variable) в многонишкова програма

Когато променливата се използва за празни операции

# Volatile променливи

```
void main(void)
{
    unsigned long var;

    initLED();

    while(1)
    {
        turnLEDon();
        for(var = 0; var < 200000; var++){ } //Времеzakъснение

        turnLEDoft();
        for(var = 0; var < 200000; var++){ } //Времеzakъснение
    }
}
```

?????

```
00000130 <main>:
130:    b508                push    {r3, lr}
132:    f7ff ffd0           bl      f0 <initLED>
136:    f7ff ffeb           bl      110 <turnLEDon>
13a:    f7ff fff1           bl      120 <turnLEDoft>
13e:    e7fa                b.n     136 <main+0x6>
```

# Volatile променливи

```
void main(void)
{
    volatile unsigned long var;

    initLED();

    while(1)
    {
        turnLEDOn();
        for(var = 0; var < 200000; var++){ } //Времеzakъснение

        turnLEDOff();
        for(var = 0; var < 200000; var++){ } //Времеzakъснение
    }
}
```

```
00000130 <main>:
130: b507          push    {r0, r1, r2, lr}
132: f7ff ffd8     bl      f0 <initLED>
136: f7ff ffeb     bl      110 <turnLEDOn>
13a: 2300          movs    r3, #0
13c: e001          b.n     142 <main+0x12>
13e: 9b01          ldr     r3, [sp, #4]
140: 3301          adds    r3, #1
142: 9301          str     r3, [sp, #4]
144: 9a01          ldr     r2, [sp, #4]
146: 4b07          ldr     r3, [pc, #28] ;
148: 429a          cmp     r2, r3
14a: d9f8          bls.n   13e <main+0xe>
14c: f7ff ffe8     bl      120 <turnLEDOff>
150: 2300          movs    r3, #0
152: e001          b.n     158 <main+0x28>
154: 9b01          ldr     r3, [sp, #4]
156: 3301          adds    r3, #1
158: 9301          str     r3, [sp, #4]
15a: 9a01          ldr     r2, [sp, #4]
15c: 4b01          ldr     r3, [pc, #4] ;
15e: 429a          cmp     r2, r3
160: d9f8          bls.n   154 <main+0x24>
162: e7e8          b.n     136 <main+0x6>
164: 00030d3f     andeq   r0, r3, pc, lsr sp
```

The header features a decorative background. On the left, there is a close-up of a red apple and a green apple. To the right of the apples, the background consists of several horizontal stripes in shades of yellow, orange, and red. The word 'Литература' is written in white text across the top of these stripes.

# Литература

1. Yiu, Joseph, The Definitive Guide to the ARM Cortex-M3, Newnes, USA, 2007
2. Dam Sunwoo, Visualizing gem5 via ARM DS-5 Streamline, workshop, 2012
3. ARM DS-5 Streamline User Guide, version 5.23, 2015





**Thank you !**