

Интегрирани среди за развой



Автор: гл. ас. д-р инж. Любомир Богданов



Европейски съюз

ПРОЕКТ BG051PO001--4.3.04-0042

***„Организационна и технологична инфраструктура за учене през
целия живот и развитие на компетенции”***

Проектът се осъществява с финансовата подкрепа на
Оперативна програма „Развитие на човешките ресурси”,
съфинансирана от Европейския социален фонд на Европейския съюз

Инвестира във вашето бъдеще!



Европейски социален фонд

Съдържание

1. Разработка на системен софтуер
2. Етапи в създаването на изпълним код (build)
3. Откриване на грешки в кода (debug)
4. Проектиране на вградени системи

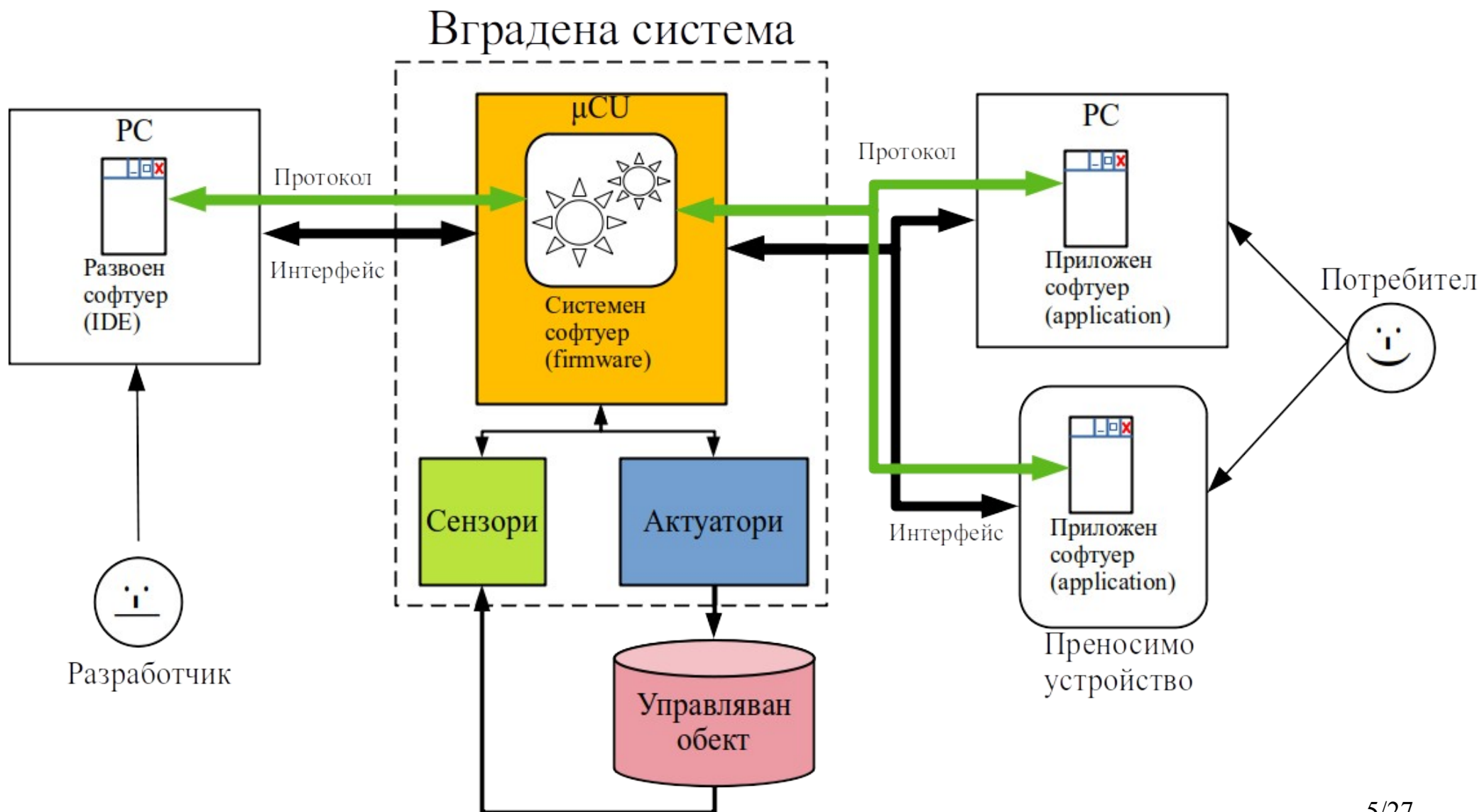
Разработка на системен софтуер

Системен софтуер (firmware) – софтуер, работещ на ниско ниво на абстракция, който модифицира регистри със специално предназначение и по този начин управлява хардуера.

Приложен софтуер (application) софтуер, работещ на високо ниво на абстракция, който задава команди и чете състоянието на вградената система посредством фърмуера и интерфейс, поддържащ даден протокол за обмен на данни.

Развоен софтуер (development software) – набор от програми, чрез които се създава изпълнимия код на системния софтуер. Използват се т.нар. развойни среди (виж по-следващия слайд).

Разработка на системен софтуер



Разработка на системен софтуер

Интегрирана развойна среда (Integrated Development Environment) – набор от програми от командния ред и програми с графичен интерфейс (Graphical User Interface) GUI), с помощта на които се създава изпълнимия код на системния софтуер (build), зарежда се в паметта на системата и позволява да се откриват и отстраняват грешки (debug).

IDE може да бъде разделена на 4 части:

- *GUI програми
- *Toolchain
- *Спомагателни програми
- *SDK

Разработка на системен софтуер

Развойна среда (IDE)

GUI tools

SDK (библиотеки,
примери, темплейти,
документация)

Toolchain

Спомагателни
програми

Разработка на системен софтуер

GUI програмите са това, което проектантът вижда на разойната среда. Те включват:

- ***GUI текстови редактор** – мястото, където се въвежда сорс кода в текстови вид;

- ***GUI дебъгер** – графична програма за управление на софтуерен дебъгер;

- ***GUI файлов експлорър** – гр. програма за представяне файловата йерархия на проекта/проектите;

- ***GUI команден ред** – гр. програма за показване на съобщенията на всички други програми, които работят от командния ред;

- ***GUI блоков редактор** – гр. програма за представяне и редактиране блоковата схема на система, реализирана върху FPGA;

- ***други.**

Разработка на системен софтуер

Toolchain - набор от програми за командния ред, чрез които се създава изпълнимия код на системния софтуер и позволява да се откриват и отстраняват грешки. Toolchain-ът съдържа:

- * **компилятор** на C/C++ (compiler)
- * **асемблер** (assembler)
- * **линкер** (linker)
- * **дебъгер** (debugger)
- * програми за **обработка на двоични файлове** (binary utilities)
- * програми за **статичен анализ** на кода (profiling tools)

Разработка на системен софтуер

Спомагателни програми - набор от програми за командния ред, спомагащи създаването на изпълнимия код:

- *команден ред (Command Line Interface, CLI)**
- *програми за автоматизация на създаването (build automation tools);**
- *програми за зареждане на системния софтуер в паметта на системата (flash utilities)**
- *програми за контрол на версията (version control)**
- *програми за автоматично документиране (documentation generator)**
- *други**

Разработка на системен софтуер

SDK (Software Development Kit) – спомагателен софтуер за фърмуера. Обикновено се дава от производителя на микроконтролера и включва следните компоненти:

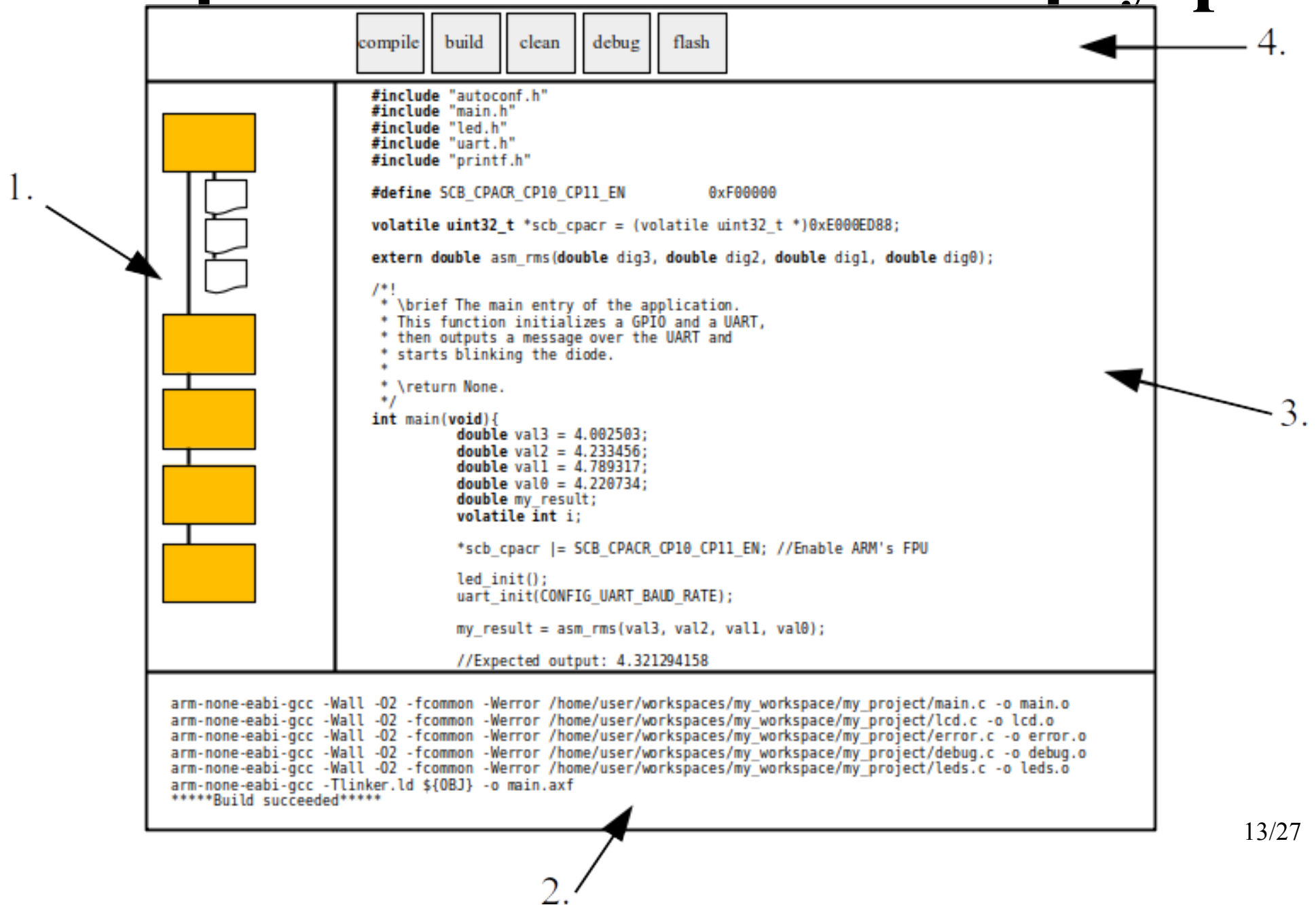
- ***библиотеки за μ PU** (съдържат вътрешни функции и дефиниции на регистри от микропроцесорната периферия);
- ***библиотеки за μ CU** (съдържат библиотеки за работа с периферните модули на контролера, Hardware Abstraction Layer, HAL);
- ***библиотеки за демо платки и еталонни дизайни** (такива библиотеки се наричат Board Support Package, BSP);
- ***външен софтуер** (наричат го third party: файлови системи, програми за статистика, CLI за микроконтролери, и т.н.)
- ***примерни проекти;**
- ***заготовки (темплейти)** за проекти;
- ***документация** (на библиотеките, на хардуера – технически спецификации, примерни приложения, и др.);
- ***други**

Разработка на системен софтуер

Всички развойни среди за μ CU имат общи черти и външният им изглед може да се обобщи, както е направено на следващия слайд. В **режим на въвеждане на код (edit)** има следните полета:

1. **Файлов експлорър** – показва директории и файлове на проекта/работното място (project/workspace);
2. **Команден ред** – показва кои команди се изпълняват в момента от средата;
3. **Текстов редактор** – съдържа сорс кода на фърмуера;
4. **Панел с бутони** – съдържа:
 - *compile – компилиране отворения в редактора файл;
 - *build – компилиране, асемблиране и линкване на всички сорс; файлове в проекта/работното място;
 - *clean – изтриване на всички обектови и двоични файлове;
 - *debug – стартиране на дебъг сесия;
 - *flash – програмиране на контролера без да се стартира дебъг сесия;

Разработка на системен софтуер



Разработка на системен софтуер

Всички развойни среди променят “лицето си” (изглежда, perspective), когато се влязат в **режим на дебъг сесия** (debug) при натискане на бутона debug. Отново може да се каже, че имат общи черти (виж следващия слайд):

5. панел с бутони за дебъгване;
- 6.панел, показващ съдържанието на регистрите на даден I/O модул;
- 7.панел, показващ съдържанието на регистрите на паметите (Flash, Ferro, SRAM);
- 8.панел, показващ съдържанието на регистрите на микропроцесорното ядро;
- 9.засветяване на ред от редактора, показващо докъде е стигнал μ PU в изпълнението на програмата;
- 10.точка на прекъсване, която ще пренуди μ PU да спре изпълнението на програмата и да върне контрола на хардуерния/софтуерния дебъгер ;

Разработка на системен софтуер

run	suspend	step over	step into	step out	run to cursor	asm. step over	restart	 Terminate session
-----	---------	-----------	-----------	----------	---------------	----------------	---------	-----------------------

```

#include "autoconf.h"
#include "main.h"
#include "led.h"
#include "uart.h"
#include "printf.h"

#define SCB_CPACR_CP10_CP11_EN

volatile uint32_t *scb_cpacr =

extern double asm_rms(double d

/*
 * \brief The main entry of th
 * This function initializes a
 * then outputs a message over
 * starts blinking the diode.
 *
 * \return None.
 */
int main(void){
    double val3 = 4.002503;
    double val2 = 4.233456;
    double val1 = 4.789317;
    double val0 = 4.220734;
    double my_result;
    volatile int i;

    *scb_cpacr |= SCB_CPACR_CP10_CP11_EN; //Enable ARM's FPU

    led_init();
    uart_init(CONFIG_UART_BAUD_RATE);

    my_result = asm_rms(val3, val2, val1, val0);

    //Expected output: 4.321294158
        
```

GPIOA

---+MODER

---+OTYPER

---+OSPEEDR

---+PUPDR

---+IDR

---+ODR

---+BSRR

0x800.0000: 00 00

0x800.000F: 3f fe

0x800.001E: ab 07

0x800.002D: ff ff

0x800.003C: 2a 3d

...

...

...

R0: 0x0000.0004

R1: 0xFF3F.240c

R2: 0xABCD.1234

R3: 0x1122.3344

R4: 0xCEFF.0311

R5: 0x5533.1202

R6: 0x1000.0004

R7: 0x0000.000a

R8: 0x0000.000a

R9: 0x8000.FFFF

R10: 0x23FF.16AC

```

****Starting debug session ~ ****
arm-none-eabi-gdb /home/user/workspaces/my_workspace/my_project/main.axf
target remote localhost:3333
monitor reset halt
load main.axf
program main.axf
b main
continue
        
```

Разработка на системен софтуер

Бутоните на дебъг панела са:

***run** - μ PU се пуска да изпълнява програмата безкрайно, докато тя не завърши, или докато не се натисне бутона **suspend**;

***suspend** – спира изпълнението на фърмуера от μ PU и се чакат команди от другите бутони;

***step over** – върви се ред по ред в програмата и ако се срещне функция, изпълнява се нейното тяло и контрола се предава обратно на дебъгера след края на функцията;

***step in** – върви се ред по ред в програмата и ако се срещне функция, влиза се в нея и започва да се върви ред по ред там;

Разработка на системен софтуер

***step out** – ако μ PU се намира в средата на много дълга функция, натискането на този бутон ще придвижи изпълнението до края на функцията и ще спре μ PU във функцията, която е едно ниво по-нагоре;

***run to cursor** – изпълнението на програмата се придвижва до позицията на курсора на мишката, след което се спира μ PU;

***assembly step over** – върви се ред по ред в програмата, но дебъгера спира μ PU след изпълнението на всяка инструкция (един ред на C може да е съставен от няколко реда на Асемблер), което прави стъпкуването по-фино;

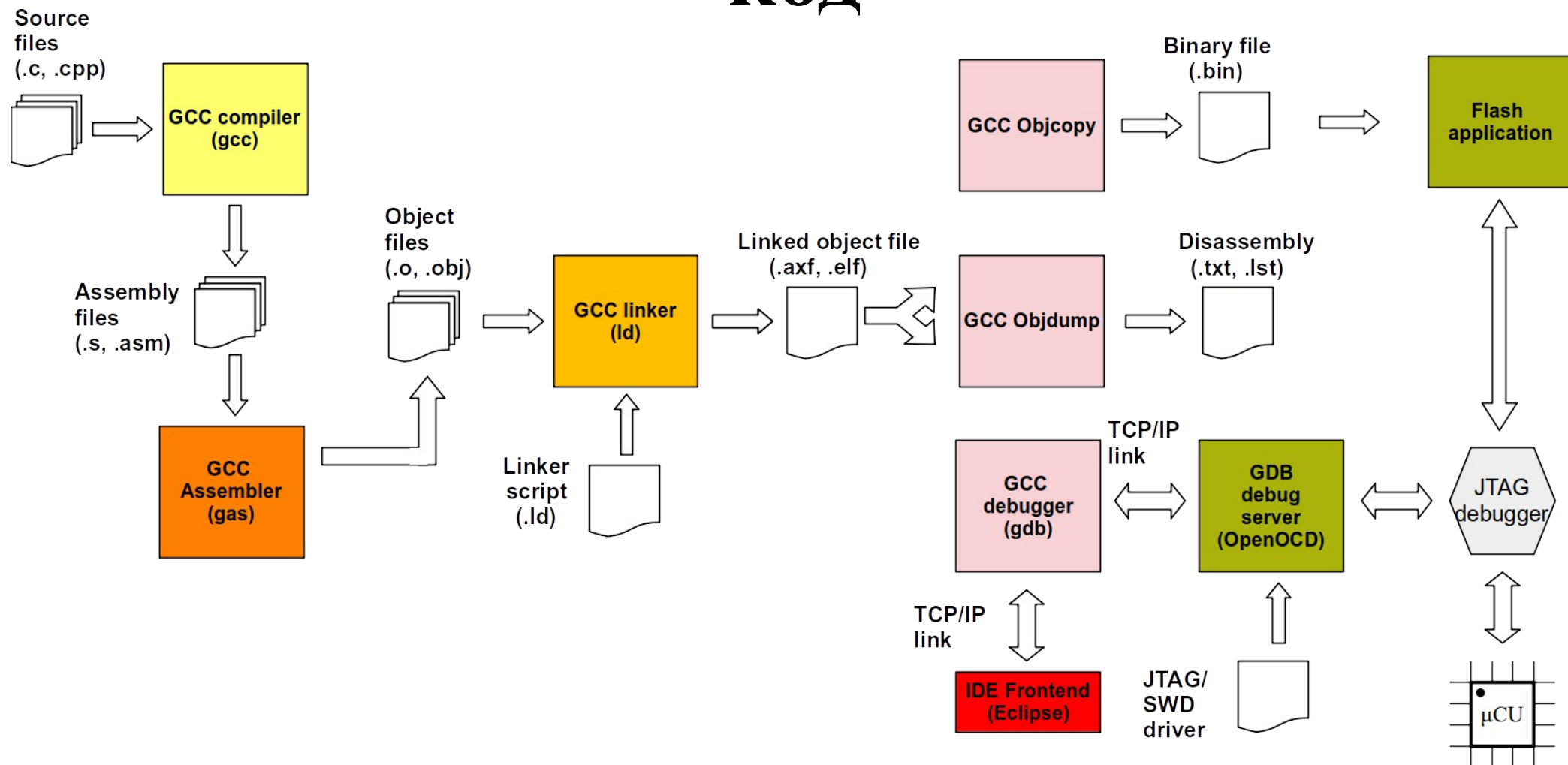
***restart** – връща се програмния брояч в началото на програмата.

Етапи в създаването на ИЗПЪЛНИМ КОД

Етапите в създаването на изпълним код са следните:

1. Въвежда се сорс код на C/C++ в текстови редактор.
2. Текстовият файл се подава на компилатор, който създава асемблерния еквивалент на C програмата с условни адреси.
3. Асемблерният еквивалент се подава на програмата асемблер, която създава обектов код с условни адреси.
4. Обектовият код с условни адреси се подава на линкера, който създава обектов код с абсолютни адреси и дебъгерна информация. Ако в проекта има други обектови файлове, те се линкват с настоящия.
5. Обектовият код с абсолютни адреси се подава на програма за обработка на двоични файлове и дебъгерната информация се премахва. Остава чист, изпълним, двоичен файл.
6. Двоичният изпълним файл се зарежда в паметта на системата посредством специализирана приложна програма.

Етапи в създаването на ИЗПЪЛНИМ КОД



Етапи в създаването на ИЗПЪЛНИМ КОД

Компилаторите, асемблерите и линкерите прилагат **оптимизации на кода**, за да се подобри някои от параметрите на програмата:

- *бързодействие

- *размер

Възможно е генерираният код да е грешен. Затова се използва програма, наречена дисасемблер.

Дисасемблер – програма, която преобразува двоичния код на фърмуера в код на Асемблер, и кода на Асемблер в код на С. Така може да станат видни оптимизациите на кода и да се поправи грешката чрез пренаписване оригиналния фърмуер, или чрез вмъкване на директиви за временно изключване на оптимизациите.

Етапи в създаването на ИЗПЪЛНИМ КОД

Едва ли има човек, който да напише сложна програма правилно от първия път. Създаването на изпълним код е итеративен процес, в който се използват **софтуерни и хардуерни дебъгери**, за да се отстранят грешките в кода.

На блоковата схема от по-предишния слайд IDE Frontend + GCC debugger + GDB debug server образуват **софтуерния дебъгер**, с чиято помощ се управлява μ PU, така че той да изпълнява команди при натискане на бутон от графичния интерфейс.

За да може софтуерния дебъгер да се свърже с μ PU, необходим е хардуерен интерфейс. Този интерфейс се осигурява от RS232/USB/Ethernet \leftrightarrow JTAG/SWD/SBW адаптер, който може да се нарече **хардуерен дебъгер**.

Етапи в създаването на ИЗПЪЛНИМ КОД

Пример: сорс код

```
int main(void){
    volatile int i;

    led_init();
    uart_init(CONFIG_UART_BAUD_RATE);

    printf("This is an example usage of printf and USART%d\n", 1);

    while (1){
        led_set();
        for(i = 0; i < LED_BLINK; i++){ }
        led_clear();
        for(i = 0; i < LED_BLINK; i++){ }
    }
}
```

Етапи в създаването на ИЗПЪЛНИМ

КОД

Пример: асемблерен еквивалент с
условни адреси.

main:

```
    push    {r7, lr}
    sub sp, sp, #8
    add r7, sp, #0
    bl led_init
    mov r0, #9600
    bl uart_init
    movs    r1, #1
    ldr r0, .L7
    bl printf_
```

.L6:

```
    bl led_set
    movs    r3, #0
    str r3, [r7, #4]
    b      .L2
```

.L3:

```
    ldr r3, [r7, #4]
    adds    r3, r3, #1
    str r3, [r7, #4]
```

.L2:

```
    ldr r3, [r7, #4]
    ldr r2, .L7+4
    cmp r3, r2
    ble .L3
    bl led_clear
    movs    r3, #0
    str r3, [r7, #4]
    b      .L4
```

.L5:

```
    ldr r3, [r7, #4]
    adds    r3, r3, #1
    str r3, [r7, #4]
```

.L4:

```
    ldr r3, [r7, #4]
    ldr r2, .L7+4
    cmp r3, r2
    ble .L5
    b      .L6
```

.L8:

```
    .align 2
```

.L7:

```
    .word   .LC0
    .word   399999
```

Етапи в създаването на ИЗПЪЛНИМ КОД

Пример: асемблерен обектов

еквивалент с **условни адреси**.

00000000 <main>:

```
0: b580      push    {r7, lr}
2: b082      sub     sp, #8
4: af00      add     r7, sp, #0
6: f7ff fffe  bl      0
a: f44f 5016  mov.w   r0, #9600
e: f7ff fffe  bl      0
12: 2101      movs    r1, #1
14: 480d      ldr     r0, [pc, #52]
16: f7ff fffe  bl      0
1a: f7ff fffe  bl      0
1e: 2300      movs    r3, #0
20: 607b      str     r3, [r7, #4]
22: e002      b.n     2a
24: 687b      ldr     r3, [r7, #4]
26: 3301      adds    r3, #1
28: 607b      str     r3, [r7, #4]
2a: 687b      ldr     r3, [r7, #4]
```

```
2c: 4a08      ldr     r2, [pc, #32]
2e: 4293      cmp     r3, r2
30: ddf8      ble.n   24
32: f7ff fffe  bl      0
36: 2300      movs    r3, #0
38: 607b      str     r3, [r7, #4]
3a: e002      b.n     42
3c: 687b      ldr     r3, [r7, #4]
3e: 3301      adds    r3, #1
40: 607b      str     r3, [r7, #4]
42: 687b      ldr     r3, [r7, #4]
44: 4a02      ldr     r2, [pc, #8]
46: 4293      cmp     r3, r2
48: ddf8      ble.n   3c
4a: e7e6      b.n     1a
4c: 00000000  andeq   ...
50: 00061a7f  andeq   ...
```


Етапи в създаването на ИЗПЪЛНИМ КОД

Пример: асемблерен обектов еквивалент с **абсолютни адреси** (след линкване).

```
080001f8 <main>:
80001f8: b580      push    {r7, lr}
80001fa: b082      sub     sp, #8
80001fc: af00      add     r7, sp, #0
80001fe: f000 f825 bl     800024c
8000202: f44f 5016 mov.w   r0, #9600
8000206: f000 f86f bl     80002e8
800020a: 2101      movs    r1, #1
800020c: 480d      ldr     r0, [pc, #52]
800020e: f001 fdeb bl     8001de8
8000212: f000 f849 bl     80002a8
8000216: 2300      movs    r3, #0
8000218: 607b      str     r3, [r7, #4]
800021a: e002      b.n     8000222
800021c: 687b      ldr     r3, [r7, #4]
800021e: 3301      adds    r3, #1
8000220: 607b      str     r3, [r7, #4]
8000222: 687b      ldr     r3, [r7, #4]
8000224: 4a08      ldr     r2, [pc, #32]
8000226: 4293      cmp     r3, r2
8000228: ddf8      ble.n   800021c
800022a: f000 f84d bl     80002c8
800022e: 2300      movs    r3, #0
8000230: 607b      str     r3, [r7, #4]
8000232: e002      b.n     800023a
8000234: 687b      ldr     r3, [r7, #4]
8000236: 3301      adds    r3, #1
8000238: 607b      str     r3, [r7, #4]
800023a: 687b      ldr     r3, [r7, #4]
800023c: 4a02      ldr     r2, [pc, #8]
800023e: 4293      cmp     r3, r2
8000240: ddf8      ble.n   8000234
8000242: e7e6      b.n     8000212
8000244: 08002b6c stmdeaq ...
8000248: 00061a7f andeq   ...
```

Откриване на грешки в кода

TO DO

Литература

[1]