

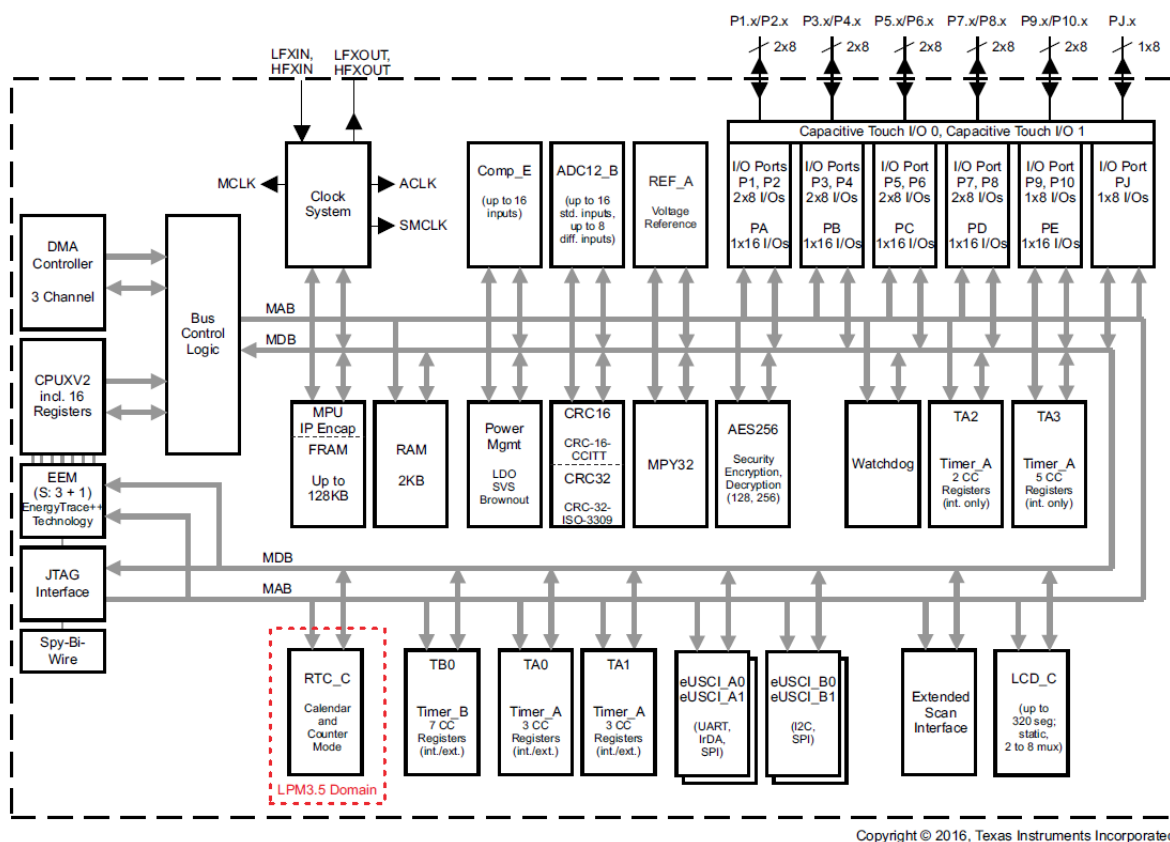
# Микропроцесорна системотехника

## Лабораторно упражнение №1

### “Запознаване с микроконтролер MSP430FR6989 и работа със средата Code Composer Studio”

**1.1. Запознаване с микроконтролера.** MSP430FR6989(IPZ) е 16-битов, 100-изведен RISC микроконтролер на фирмата Texas Instruments. Той работи с максимална тактова честота от 16 MHz и има 2 KB SRAM памет за данни. Програмната памет е 128 KB и е реализирана по технология Ferroelectric RAM (FRAM), докато по-старите контролери използват Flash памет. Предимствата на FRAM спрямо Flash са: 100 000 000 000 000 цикъла за запис (срещу 100 000 при Flash), около 250 пъти по-малка консумация, около 100 пъти по-кратки времена за достъп и около 10 пъти по-ниски нива на захранващото напрежение при запис (1.5 V). Големият брой цикли за запис позволява тя да бъде използвана като програмна и даннова памет едновременно.

На **фиг. 1.1** е показана блокова схема на микроконтролера.



Copyright © 2016, Texas Instruments Incorporated

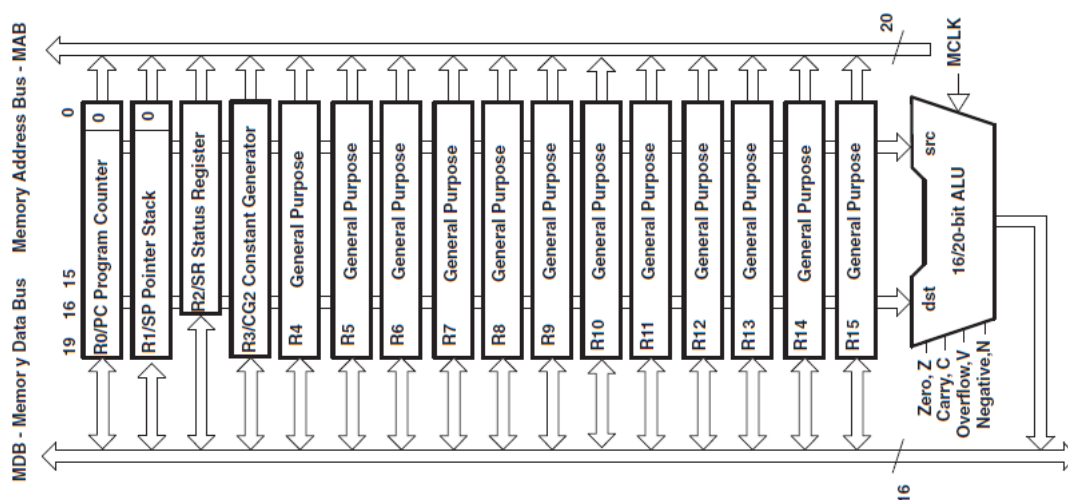
**Фиг. 1.1.** – Блокова схема на MSP430FR6989<sup>1</sup>

Отделните модули са:

- **CPU** – 16-мегагерцов микропроцесор с 20-битова магистрала за адреси и 16-битова магистрала за данни (**фиг. 1.2**). Използват се 27 инструкции (+ 24 емулирани инструкции). Разполага с:
  - 12 регистъра с общо преназначение (R4 ÷ R15).
  - програмен брояч (PC, разположен в R0).
  - указател на стека (SP, разположен в R1).
  - регистър на състоянието (SR, разположен в R2).

<sup>1</sup> Datasheet на MSP430FR6989

- регистър за генериране на константи (CG, разположен в R3) – генерира 6 често използвани константи в зависимост от адресацията на изпълняваната инструкция. Тези константи могат да служат като „операнд по подразбиране” и така инструкция с един операнд може да се превърне в инструкция с два операнда (емулирана инструкция).



Фиг. 1.2. Блокова схема на микропроцесора, използван от микроконтролера MSP430FR6989<sup>2</sup>

- **JTAG** – интерфейс за програмиране на чипа и четене на вътрешните регистри.
- **EEM** – Embedded Emulation Module – хардуерен модул за реализация на “breakpoints”. Това са точки на прекъсване, в които изпълнението на програмата спира и чрез JTAG може да се прочете състоянието на даден регистър. Накратко – използва се за улеснение при дебъгването на програмата.
- **System Clock** – модул за генериране на тактови сигнали. Генерират се шест тактови сигнала, използвани от различни периферни модули на чипа:
  - MCLK (Main Clock) – тактов сигнал на микропроцесора
  - ACLK (Auxiliary Clock) – тактов сигнал за периферни модули
  - SMCLK (Sub-Main Clock) – тактов сигнал за периферни модули
  - VLOCLK (Very-LOW-Power Clock) – тактов сигнал за периферни модули
  - LFXTCLK (Low-Frequency Clock) – тактов сигнал за периферни модули
  - MODCLK (Low-power Clock) – тактов сигнал за периферни модули
- **FRAM** – програмна памет.
- **RAM** – даннова памет.
- **Boot ROM** – памет за съхранение на bootloader. Bootloader е специална програма, която помага за трансфер на потребителската програма (написана от програмиста) от персоналния компютър в програмната памет на микроконтролера.
- **Power Management & SVS (Supply Voltage Supervisor)** – модул за изработване на необходимите захранващи напрежения.
- **SYS Watchdog** – таймер, отброяващ интервал от време, при изтичането на което се подава сигнал за рестартиране на контролера. Програмистът трябва да заложи в програмата периодичното нулиране на този брояч. Така, ако някъде в програмата се получи „замръзване”, то броячът няма да се нулира и микроконтролерът ще се рестартира. В настоящите лабораторни упражнения този брояч е изключен.
- **REF** – еталонно напрежение, използвано от някое периферно устройство (ADC или компаратор в случая).

<sup>2</sup> User guide на MSP430FR6989

- **I/O Ports** – входно-изходни портове. Това са регистри, достъпни за потребителя и свързани с изходите на микроконтролера. В литературата се отбелязват още като GPIO – General Purpose Input Output pins. С тях може да се управляват външни устройства/схеми, приемащи CMOS нива.
- **DMA (Direct Memory Access controller)** – контролер за автоматично прехвърляне на данни (без намесата на микропроцесора) от един адрес на друг.
- **Comp\_E** – аналогов компаратор.
- **ADC12\_B** – 16-канално 12-битово АЦП.
- **UART, IrDA, SPI, I2C** – серийни синхронни (**SPI, I<sup>2</sup>C**) и асинхронни (**UART, IrDA**) интерфейси.
- **CRC16 (Cyclic Redundancy Check)** – модул за генериране на checksum.
- **RTC\_C** – чаовник за реално време.
- **Timer\_A, Timer\_B** – 16-битови таймери.
- **MPY32** – 32-битов хардуерен умножител.
- **AES256** - модул за хардуерно шифриране/дешифриране на информация.
- **LCD\_C** - модул за управление на сегментни LCD дисплеи.
- **ESI (Extended Scan Interface)** - модул за четене на оптични/магнитни ротационни енкодери и др.

## 1.2. Някои основни понятия свързани с микроконтролерите

**Фърмуер** (от англ. firmware) - софтуер, който се зарежда в постоянната памет на микроконтролера. Терминът се използва, за да не се бърка с приложния софтуер (от англ. application software), който се зарежда на персонален компютър. Микроконтролерните системи към днешна дата често се управляват от персонален компютър, което налага разграничаването между фърмуер и приложен софтуер. В настоящите упражнения се разглежда само разработката на фърмуер.

Фърмуерът може да бъде написан на асемблер (език, при който програмата се пише на ниво инструкции на процесора с някои улеснения), на език от високо ниво (като използваните при писане на приложен софтуер, при което се увеличава нивото на абстракция) или дори графично чрез блокови диаграми на алгоритмите.

**Хардуерен дебъгер** (от англ. hardware debugger) – устройство, с помощта на което развойната среда извършва следните действия (често обобщавани с изрази дебъгване):

- зарежда фърмуера в паметта на микроконтролера;
- стартира изпълнението на фърмуера;
- изпълнява програмата стъпка по стъпка, т.е. асемблерна инструкция след асемблерна инструкция, или ред след ред от C програма;
- чете от или записва във всички регистри на микроконтролера;
- следи за точки на прекъсване (места в програмата, на които микропроцесорът спира изпълнението на инструкции);
- следи за регистри, които са си променили съдържанието.

За да може дебъгерът да изпълнява всички тези функции, микроконтролерът трябва да съдържа хардуерни модули, които да изпълняват командите на дебъгера и да контролират микропроцесора. Такива са JTAG, SWD, Breakpoint, Watchpoint и др. модули. Повечето микроконтролери притежават поне JTAG или SWD.

Примери за хардуерни дебъгери са Keil ULINK, Texas Instruments ICDI, LPC Link и др. Някои от тях са реализирани като отделни устройства, други – включени в демо платка и неделими от нея.

**Софтуерен дебъгер** (от англ. software debugger) – програма, която следи действията на програмиста в развойната среда (натискане на бутони) и в зависимост от тях изпраща команди на

хардуерния дебъгер. Той от своя страна изпраща команди на JTAG модула, който извършва желаното действие (например прочети даден регистър или спри микропроцесора).

Пример за софтуерен дебъгер е програмата с отворен код GDB, която има версии за различни фамилии микропроцесори от Intel до ARM.

### 1.3. Начини за създаване на фърмуер за микроконтролера

Типично, фърмуера за микроконтролерите се разработва с помощта на развойни среди (IDE – Integrated Development Environment), работещи на персонални компютри. IDE представляват пакет от програмни продукти, обединени в един потребителски интерфейс (прозорец). Повечето IDE включват текстови редактор, компилатор, линкер, асемблер, дебъгер и програма, генерираща изпълнимият код, често наричан фърмуер.

Основно, фърмуера може да бъде написан на асемблер или на език от високо ниво.

При асемблера, фърмуера се пише на ниво инструкции на процесора. Има някои улеснения, като например поставяне на етикети и др.

При писане на език от високо ниво, могат да се използват директно вече конструкции като if...else, while, for, switch...case, както и възможността за използване на променливи. Ако не се използват допълнителни библиотеки, записът и четенето в регистрите се извършва директно, т.е. кодът не е особено опростен спрямо кода на асемблер.

Ако се използват допълнителни библиотеки, нивото на абстракция се увеличава значително. Например, можете да установите в определено състояние даден пин, без да е необходимо да знаете, това чрез кой регистър се извършва.

По-долу е разгледан сегмент от фърмуер, който спира watchdog таймера, изключва високоимпедансното състояние на входно-изходните изводи, инициализира изводи P8.4, P8.5, P8.6 и P8.7 от порт 8 и ги установява в логическа нула. Представени са варианти на асемблер, на C без библиотеки и на C с използване на библиотеките MSP430Ware (start-up код не е показан).

От показаните примери се вижда ясно промяната в нивото на абстракция. На асемблер, трябва да се познават инструкциите на процесора, адресацията, както и регистрите. На C без използване на библиотеки, се абстрахираме от инструкциите на процесора и адресацията, но не и от регистрите. При C с използване на библиотеки, вече не се пише и името на регистъра, а се използва функция, която е по-разбираема за нас.

#### Сегмент от фърмуер на асемблер

```
StopWDT      mov.w    #WDTPW|WDTHOLD,&WDTCTL    ; Спиране на Watchdog таймера

              bic.w    #LOCKLPM5,&PM5CTL0        ; Изключи високоимпедансното състоя-
                                                ; ние на входно-изходните изводи

              bis.b     #11110000b, &P8DIR        ; Конфигуриране на изводите като
                                                ; изходи

              bic.b     #11110000b, &P8OUT        ; Инициализиране на тези изходи в
                                                ; логическа 0
```

#### Сегмент от фърмуер на C (директен регистров достъп)

```
WDTCTL = WDTPW | WDTHOLD;    // Спиране на Watchdog таймера

PM5CTL0 &= ~LOCKLPM5;        // Изключи високоимпедансното състоя-
                             // ние на входно-изходните изводи

P8DIR |= 0b11110000;         // Конфигуриране на изводите като
                             // изходи

P8OUT &= ~0b11110000;        // Инициализиране на тези изходи в
                             // логическа 0
```

```
WDT_A_hold(WDT_A_BASE);           // Спиране на Watchdog таймера

PMM_unlockLPM5();                  // Изключи високоимпедансното състоя-
                                   // ние на входно-изходните изводи

// Конфигуриране на изводите като изходи
GPIO_setAsOutputPin(GPIO_PORT_P8,
                    GPIO_PIN4 | GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7);

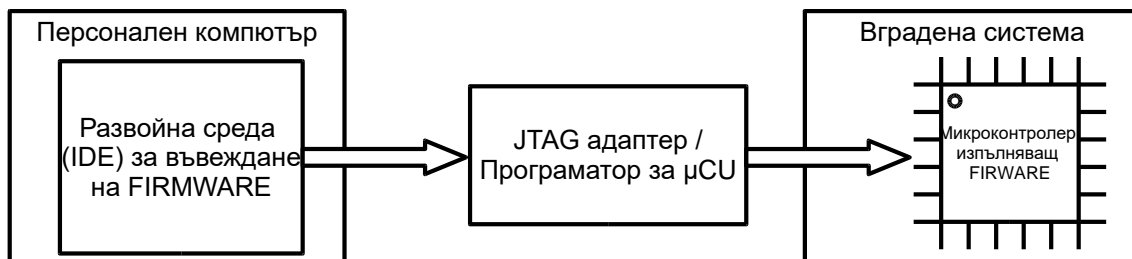
// Инициализиране на тези изходи в логическа 0
GPIO_setOutputLowOnPin(GPIO_PORT_P8,
                      GPIO_PIN4 | GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7);
```

### 1.3. Начини за запис на фърмуера в микроконтролера

Фърмуера за микроконтролерите се разработва с помощта на развойни среди (IDE – Integrated Development Environment), работещи на персонални компютри. IDE представляват пакет от програмни продукти, обединени в един потребителски интерфейс (прозорец). Повечето IDE включват текстови редактор, компилатор, линкер, асемблер, дебъгер и програма, генерираща изпълнимият код, често наричан фърмуер.

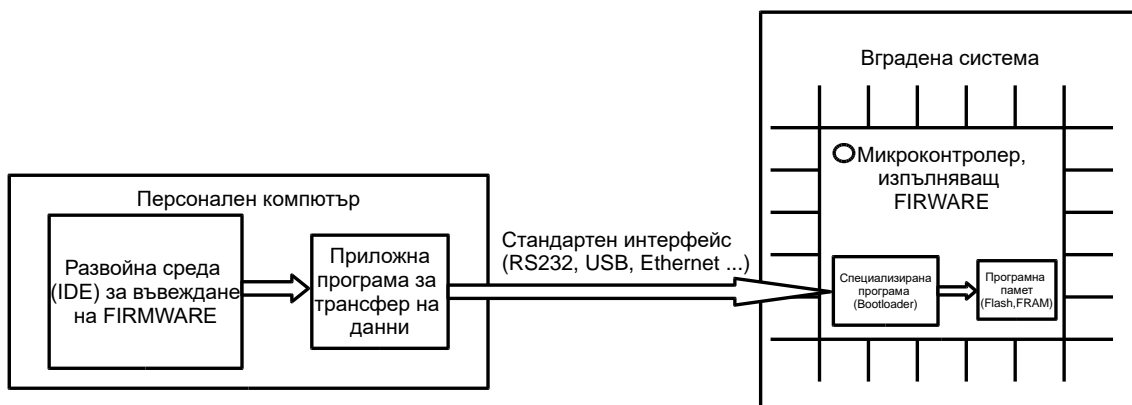
За да може фърмуерът да се изпълнява от микроконтролера, той трябва да бъде записан в неговата програмна памет. Записът в паметта може да стане по хардуерен или софтуерен път.

Хардуерните методи за зареждане на фърмуера в микроконтролера са посредством JTAG адаптер или програматор (фиг. 1.3).

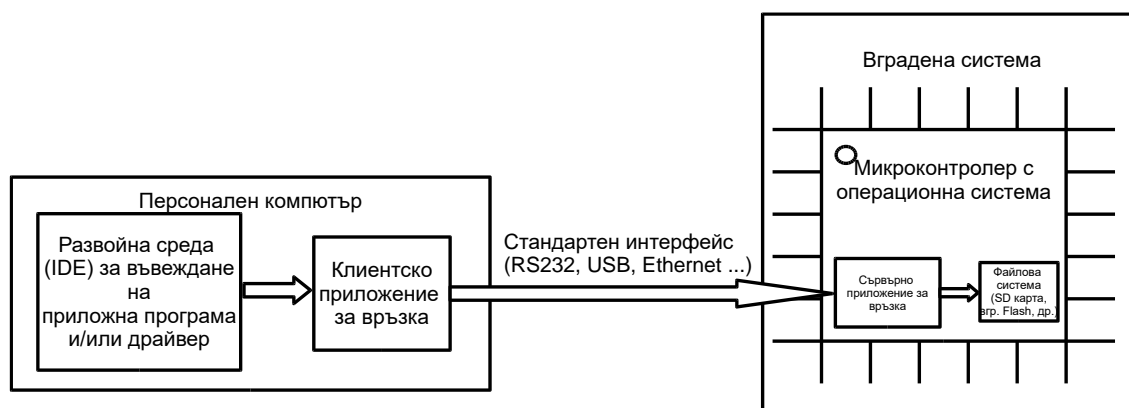


Фиг. 1.3. Програмиране на микроконтролер чрез JTAG адаптер или програматор

Често използван софтуерен метод за програмиране е чрез предварително зареден bootloader (Фиг. 1.4а). Bootloader-а е необходимо да бъде записан в паметта на микроконтролера чрез хардуерен метод. Обаче след като е записан веднъж, фърмуерът може да бъде зареждан чрез стандартен интерфейс. При микроконтролер с операционна система, фърмуерът може да се зареди и чрез копиране на файл във файловата система.



Фиг. 1.4. а) Програмиране чрез bootloader



Фиг. 1.4. б) Програмиране на микроконтролер с операционна система

В настоящото лабораторно упражнение ще се използва методът, показан на **фиг. 1.3**. Развойната платка е MSP-EXP430FR6989 (Experimenter's Board) на фирмата Texas Instruments и включва един микроконтролер MSP430FR6989, програматор/дебъггер за него, както и допълнителна периферия.

#### 1.4. Работа със средата Code Composer Studio

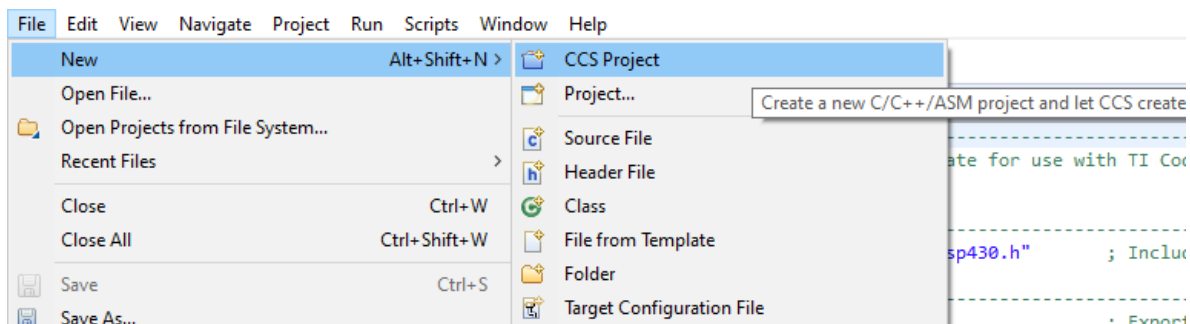
Средата за програмиране е Code Composer Studio на фирмата Texas Instruments. Тя се използва за писане на фърмуера, изчистване на синтактични грешки, запис на фърмуера в микроконтролера, дебъгване и други.

Програмата може да се стартира, като се щракне два пъти върху иконката ѝ на десктопа **Code Composer Studio X.X.X.** „X.X.X.“ е версията на програмата.

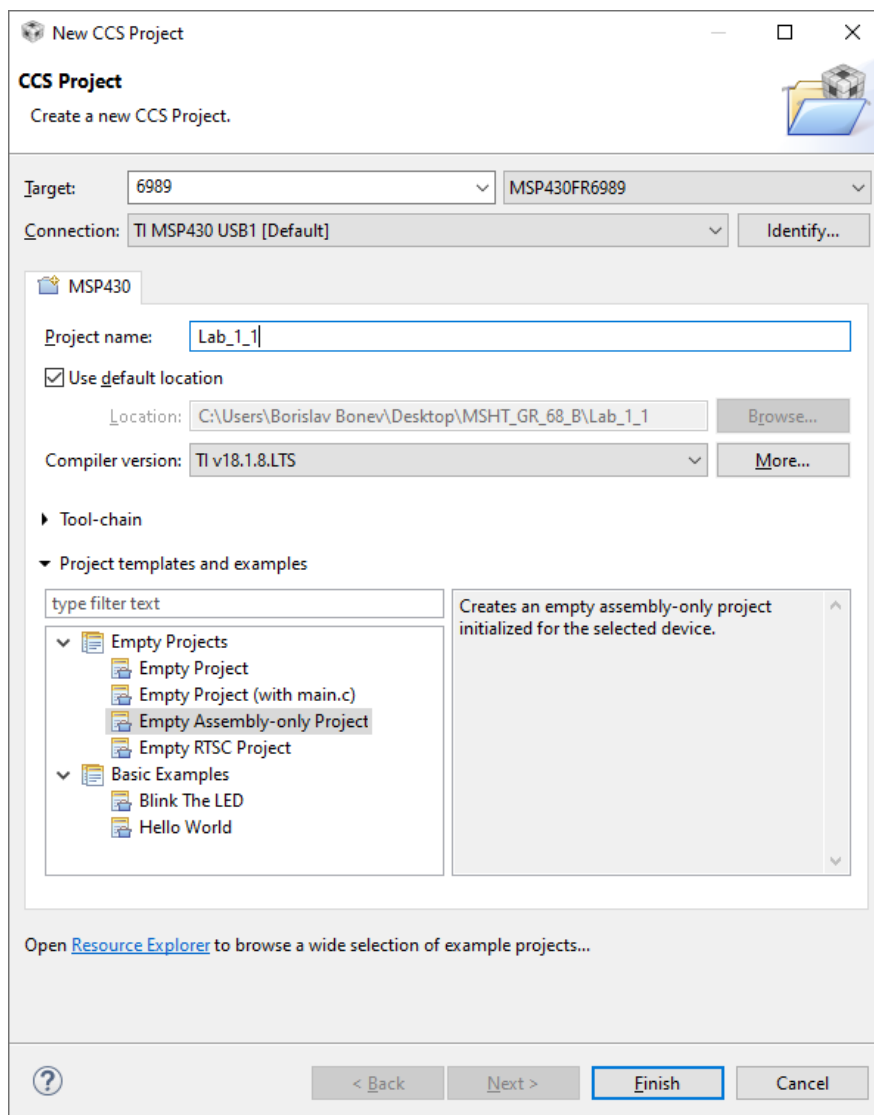
##### Програмиране на асемблер

Трябва да се премине през следните стъпки:

- Стартира се програмата, като се щракне два пъти върху иконката ѝ на десктопа **Code Composer Studio X.X.X.**
- Ако се появи прозорец **Select a directory as workspace**, изберете директория за съхраняване на проектите на Вашата подгрупа:  
**Desktop/MSHT\_GR\_68\_B** или **Desktop/MSHT\_GR\_69\_A**  
Ако не се появи този прозорец, проверете дали като workspace е избрана правилната папка. Ако не е правилната папка, изберете **File → Switch Workspace** и изберете коректната папка.  
**Тази стъпка е важна**, тъй като по този начин ще имате лесен достъп до **Вашите проекти**, които ще Ви бъдат от полза при **курсовото проектиране** по дисциплината.
- Изберете **File → New → CCS Project** (фиг. 1.5). Отваря се прозореца **New CCS Project**.
- От падащото меню **Target** се избира модела на микроконтролера (**MSP430FR6989**).
- В Connection изберете това, което е по подразбиране (**TI MSP430 USB1 [Default]**). Можете да кликнете върху бутона **Identify**, за да се уверите, че няма проблеми с комуникацията между компютъра и развойната платка.
- В **Project name** въведете името на проекта. Например „**Lab\_1\_1**“.
- В **Project Templates and examples** се избира **Empty Assembly-only Project**.
- Щраква се върху бутон **Finish** (фиг. 1.6).

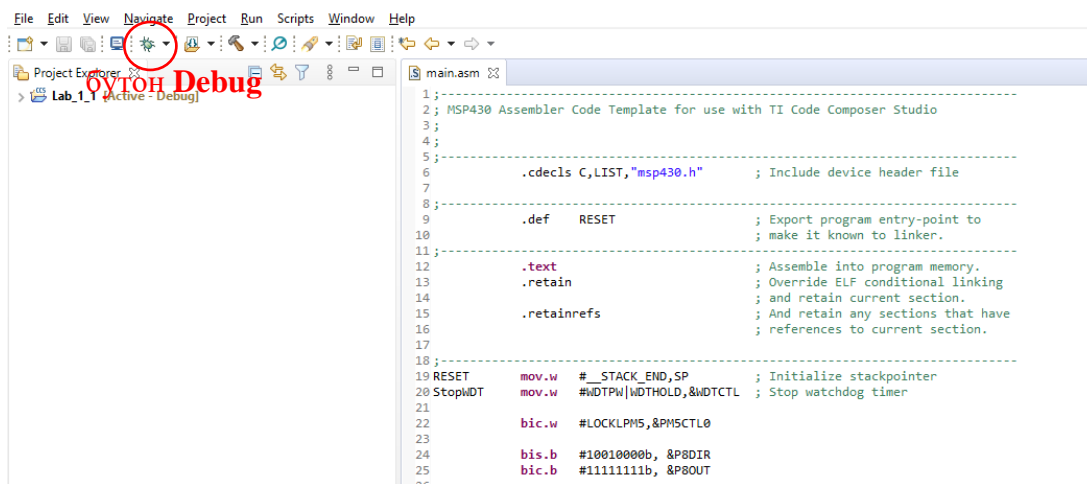


Фиг. 1.5. Отваряне на помощника за създаване на нов проект



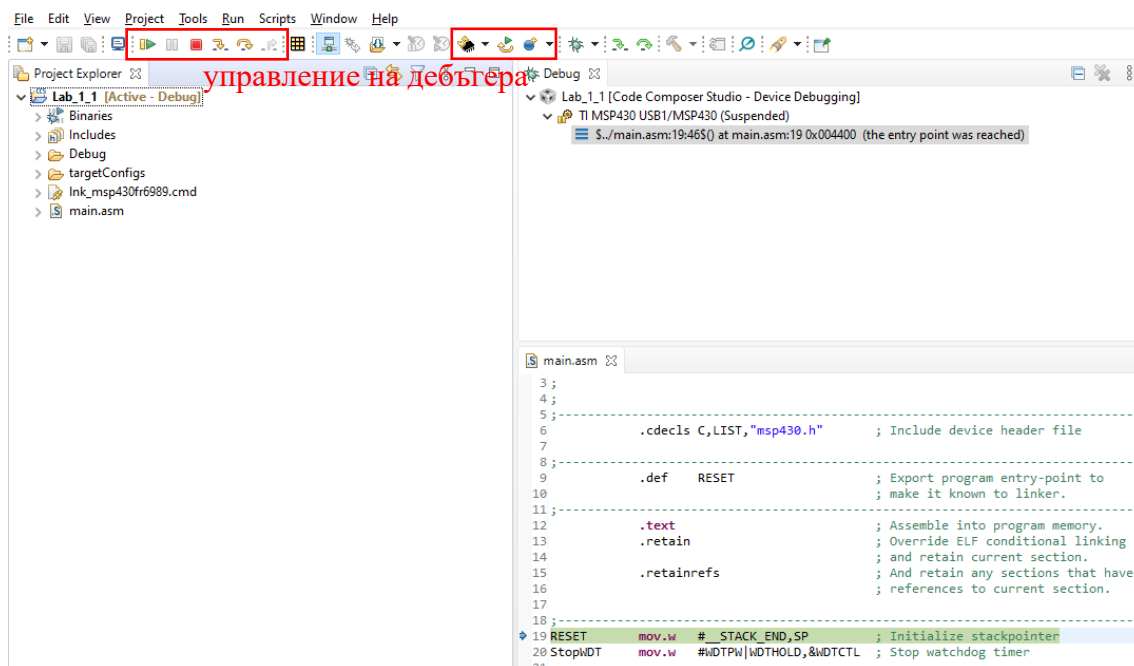
Фиг. 1.6. Създаване на нов проект на асемблер

- Така проектът вече е създаден. Създаден е и файл `main.asm`, в който се въвежда фърмуера. От етикета **RESET** нататък се пишат новите инструкции. Редът с етикета **RESET** е мястото, от което започва изпълнението при включване на захранването или при рестартиране (reset) на микроконтролера.
- След това се натиска бутона **Debug** (фиг. 1.7). Преди натискането на този бутон макетът трябва да е включен с USB кабел към компютъра.




Фиг. 1.7. Местоположение на бутона Debug


- Ако няма грешки в програмата, пречеци на компилирането, както и ако няма проблеми с комуникацията между развойната платка и компютъра, фърмуера ще бъде зареден в микроконтролера и ще се появи в развойната среда лента с бутони, управляващи дебъгването (фиг. 1.8). В началото изпълнението на програмата е спряно на първата инструкция (при етикета **RESET**).




Фиг. 1.8. Местоположение на бутоните за управление на дебъгера


Описание на бутоните (важи и при дебъгване на C/C++ програми):

Бутонът  служи за изпълнение на програмата от настоящото положение до края на програмата. Ако има поставени точки на прекъсване (breakpoints), то изпълнението ще спре при тях.


Бутонът  служи за спиране изпълнението на програмата без приключване на сесията на дебъгване.


Бутонът  служи за приключване на сесията на дебъгване.

Бутонът  служи за изпълняване на следваща инструкция (извикване на C/C++ функция) с влизане в подпрограми (в самата C/C++ функция).

Бутонът  служи за изпълняване на следваща инструкция (извикване на C/C++ функция) без да се влиза в подпрограми (в самата C/C++ функция).

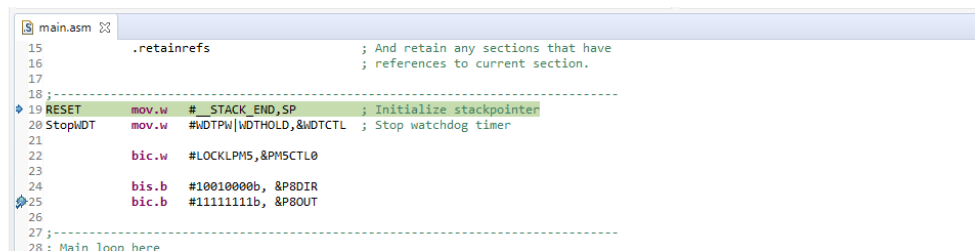


Бутонът  служи за излизане от C/C++ функцията, в която се намираме.

Бутонът  служи за reset на микроконтролера. Изпълнението на програмата започва от етикета **RESET**.

#### Поставяне на точка на прекъсване:

Щраква се с десен бутон на мишката на реда, на който искаме изпълнението на програмата да спре. Избира се **Breakpoint (Code Composer Studio) → Breakpoint**. Срещу избрания ред ще се появи синя точка (фиг. 1.9). В конкретния случай, изпълнението ще спре на 25-ти ред.

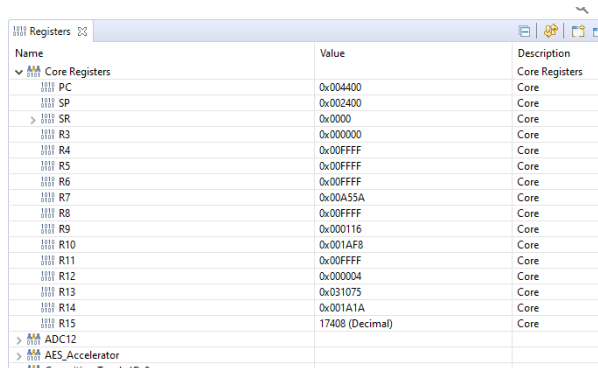


Фиг. 1.9. Поставяне на точка на прекъсване

#### Наблюдение на регистрите на микроконтролера или на променливи:

Докато изпълнението на програмата е спряно може да се наблюдават регистрите на микроконтролера благодарение на JTAG адаптера. В дебъг режим разнообразната информация може да се визуализира от менюто:

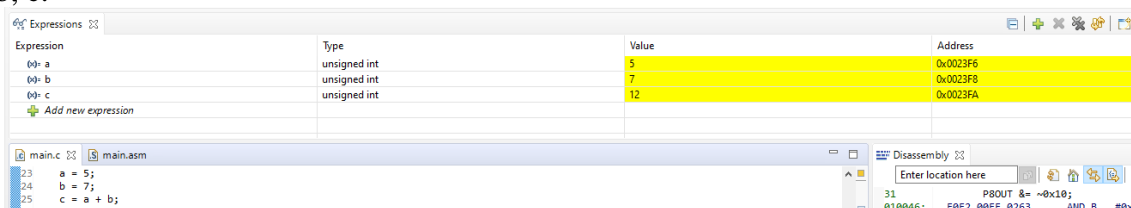
- **View → Registers** – за наблюдение на всички регистри в микроконтролера. От падащото меню **Register** се избира периферен модул или микропроцесор. На **фиг. 1.10** са дадени моментните стойности на регистрите на микропроцесора от **фиг. 1.2**.



Name	Value	Description
Core Registers		Core Registers
PC	0x004400	Core
SP	0x002400	Core
SR	0x0000	Core
R3	0x000000	Core
R4	0x00FFFF	Core
R5	0x00FFFF	Core
R6	0x00FFFF	Core
R7	0x00A55A	Core
R8	0x00FFFF	Core
R9	0x000116	Core
R10	0x001AF8	Core
R11	0x00FFFF	Core
R12	0x000004	Core
R13	0x031075	Core
R14	0x001A1A	Core
R15	17408 (Decimal)	Core
ADC12		
AES_Accelerator		

Фиг. 1.10. Визуализация на стойностите, записани в регистрите на микроконтролера. В конкретния случай, стойностите на регистрите на процесора на микроконтролера

- **View → Expressions** - в отворения се прозорец, в **Add new expression** се записва име на променливата или регистрите, които искаме да инспектираме. На **фиг. 1.11** е показана програма на C и наблюдаване на 3 променливи от целочислен тип a, b, c.

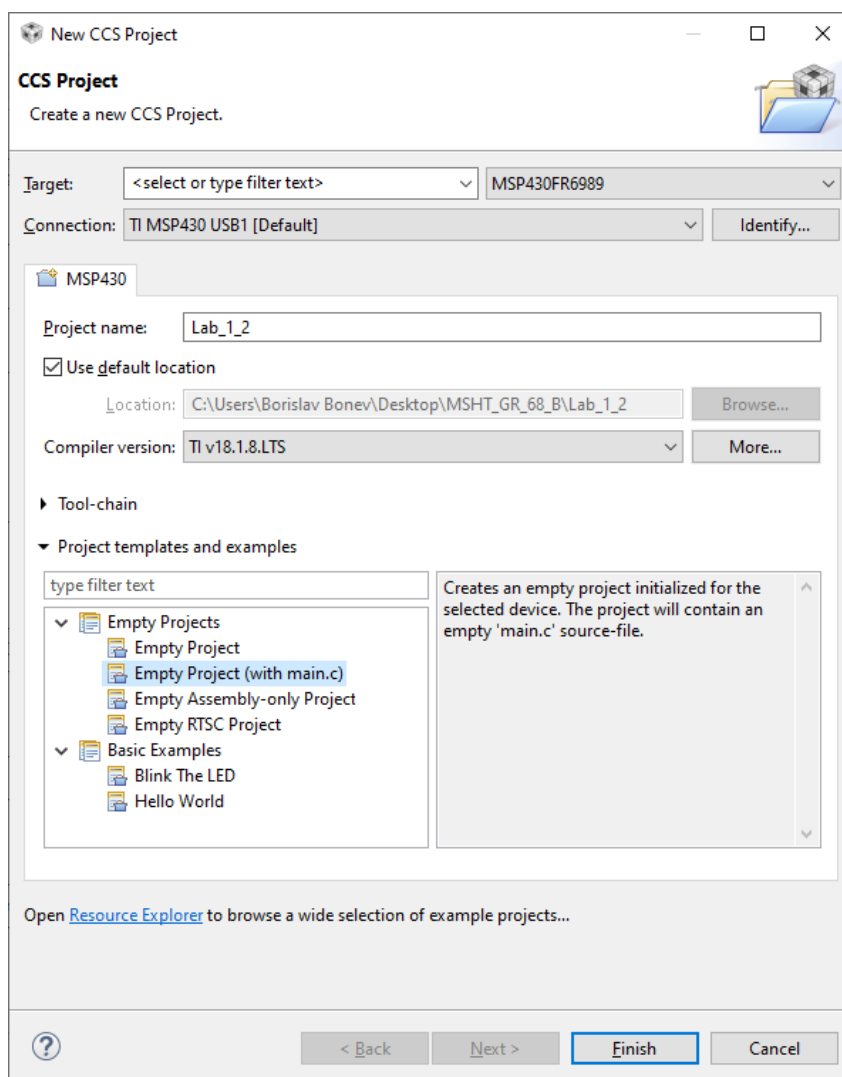


Фиг. 1.11. Наблюдаване на конкретни променливи

## Програмиране на C

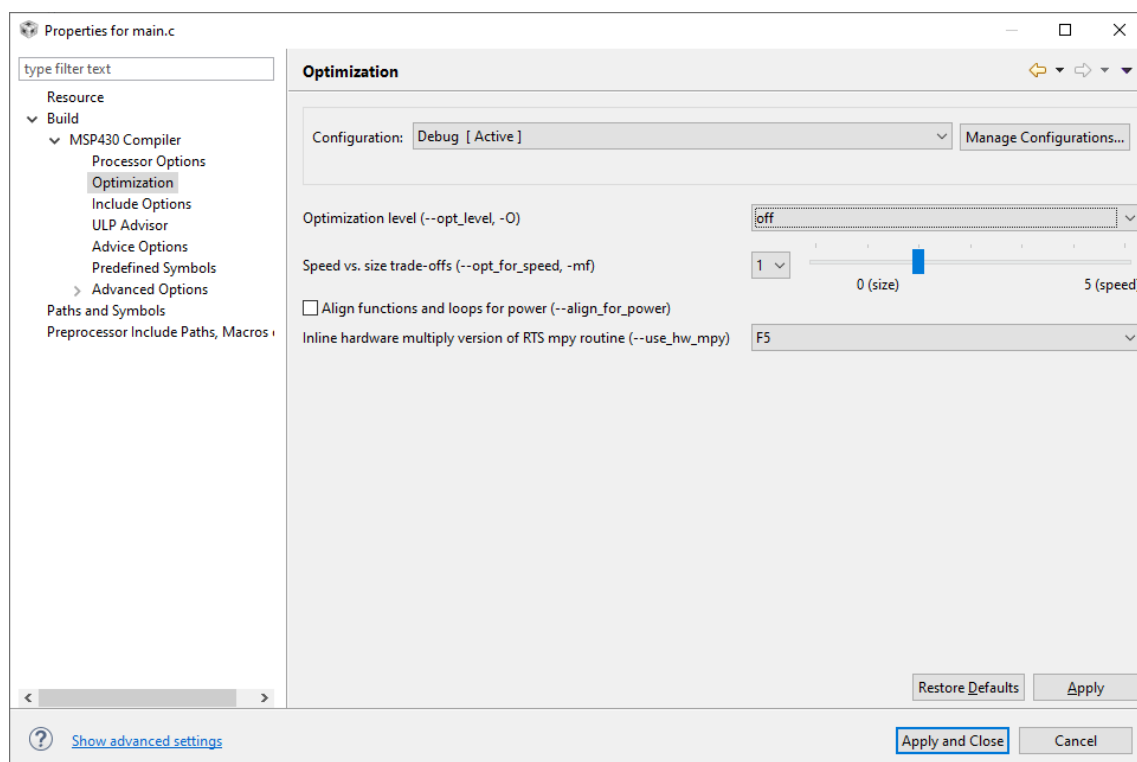
Трябва да се премине през следните стъпки:

- Стартира се програмата, като се щракне два пъти върху иконката ѝ на десктопа **Code Composer Studio X.X.X**.
- Ако се появи прозорец **Select a directory as workspace**, изберете директория за съхраняване на проектите на Вашата подгрупа:  
**Desktop/MSHT\_GR\_68\_B** или **Desktop/MSHT\_GR\_69\_A**  
Ако не се появи този прозорец, проверете дали като workspace е избрана правилната папка. Ако не е правилната папка, изберете **File → Switch Workspace** и изберете коректната папка.  
**Тази стъпка е важна**, тъй като по този начин ще имате **лесен достъп до Вашите проекти**, които ще Ви бъдат от полза при **курсовото проектиране** по дисциплината.
- Избира се **File → New → CCS Project**. Отваря се прозореца **New CCS Project**.
- От падащото меню **Target** се избира модела на микроконтролера (**MSP430FR6989**).
- В Connection изберете това, което е по подразбиране (**TI MSP430 USB1 [Default]**). Можете да кликнете върху бутона **Identify**, за да се уверите, че няма проблеми с комуникацията между компютъра и развойната платка.
- В **Project name** въведете името на проекта. Например „**Lab\_1\_2**“.
- В **Project Templates and examples** се избира **Empty Project (with main.c)**.
- Щраква се върху бутон **Finish** (фиг. 1.12).



Фиг. 1.12. Създаване на нов проект на C

- Оптимизациите при компилиране се изключват за лабораторните упражнения с цел по-ясно показване работата на микроконтролера. Това се прави от File → Properties. Като се отвори прозорец с настройките, се избира таб Build → MSP430 Compiler → Optimization. От падащото меню Optimization level се избира off. Натиска се Apply and Close (фиг. 1.13).



**Фиг. 1.13.** Изключване на оптимизациите при компилиране

- От дървото на проекта се избира файлът main.c и програмата на C се въвежда след реда WDTCTL = WDTPW + WDTNOLD;
- Всички стъпки като компилирането, програмирането и дебъгването, описани за асемблер, също са приложими тук.

## ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се създаде нов проект с име **Lab\_1\_1** в папка **/Desktop/MSHT\_GR\_XX\_N**, да се копира програмата на асемблер, която последователно включва само един от светодиодите LED 0 ÷ 7 (ефект „бягаща точка”). Реализирано е софтуерно закъснение между всяко преместване на позицията, за да може да се наблюдава ефекта.

2. Да се създаде нов проект с име **Lab\_1\_2** в папка **/Desktop/MSHT\_GR\_XX\_N**, да се копира програмата на C, която извършва същите действия като програмата в предходната точка. Да се изпълни програмата стъпка по стъпка. Да се наблюдават регистрите на микропроцесора и изходните портове (P8 и P9). Да се постави точка на прекъсване на избрано място. Да се наблюдава изменението на променливата **i**: чрез **View → Expressions**, в **Add new expression** се записва името на променливата.

3. Да се покаже съответствието между инструкциите на асемблерната програма (от задача 1) и операциите на C програмата (от задача 2).

4. Коя програма е с по-добро бързодействие? Опишете, как преценявате бързодействие ѝ.

5. Коя програма използва по-малък обем от програмната памет и от RAM паметта на микроконтролера? Каква е причината според Вас?