ХРИСТО КИЛИФАРЕВ

РЪКОВОДСТВО
за лабораторни упражнения по
МИКРОПРОЦЕСОРНА ТЕХНИКА

УНИВЕРСИТЕТСКО ИЗДАТЕЛСТВО
ГАБРОВО
2024

Христо Килифарев

РЪКОВОДСТВО

за лабораторни упражнения по

МИКРОПРОЦЕСОРНА ТЕХНИКА

УНИВЕРСИТЕТСКО ИЗДАТЕЛСТВО
"ВАСИЛ АПРИЛОВ"
ГАБРОВО, 2024

Рецензент: Проф. Николай Маджаров

Печат: Университетско издателство “Васил Априлов” – Габрово, 2024

ISBN: 978-954-683-706-6

Предговор

Ръководството за лабораторни упражнения по Микропроцесорна техника е предназначено за изучаване на 8–битовите микроконтролери от семейство AVR на фирма Atmel и експериментиране с тях. Примерите са тествани с микроконтролер ATmega8515.

За провеждане на експерименти „на физическо ниво“ е желателно да се разполага с апаратната платформа STK500, произведена от Atmel, която е система за развитие (на английски – kit).

STK500 се поддържа изцяло от съвместимата с нея безплатна и удобна програмна среда AVR Studio на Atmel. Последната постоянно се усъвършенства и улеснява значително технологията за създаване и симулиране на програми както на Асемблер, така и на C. Платформата STK500 разполага с вграден програматор за микроконтролерите от семейство AVR. Ако не се разполага със система STK500, програмите могат да се симулират сравнително удобно в средата на AVR Studio.

Материалът покрива въпроси от технологията за програмиране на Асемблер за микроконтролери AVR, задълбочава знанията по фундаментални въпроси от апаратната архитектура и програмното осигуряване на микроконтролери и микрокомпютри, като

- Бройни системи и кодове за представяне на числата,
- Специални регистри – флагов регистър и стеков указател,
- Стек, подпрограми и организация на адресното пространство,
- Програмни времезакъснения и работа с таймерен модул,
- Технология за обработка на апаратни прекъсвания,
- Работа с периферни устройства – бутони, светодиоди, 7-сегментни индикатори, LCD,
- Реализиране на серийна комуникация с компютър.

Разгледаните тематики в лабораторното ръководство предлагат изследователски елементи по алгоритмизиране и програмиране на реални задачи.

Всяка тема за лабораторно упражнение изисква не по-малко от три учебни часа, за да бъде развита в дълбочина.

Ръководството за лабораторни упражнения е предназначено за обучаващи се от специалности, изучаващи микропроцесорна и микроконтролерна техника, както и базирани на тях вградени системи на съвременно ниво. То може да бъде полезно също за студенти и специалисти, които искат да надградят знанията си по важни теми от указаната област.

Габрово, септември 2024 г. Авторът

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 1
Запознаване с развойната платка STK500, микроконтролер ATmega8515 и интегрираната програмна среда AVR Studio

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да се запознаят с апаратните характеристики и работата с развойна платка STK500, с основните функционални възможности и характеристики на микроконтролер ATmega8515, а също и с развойната среда AVR Studio за разработване на програми за микроконтролери от фамилията AVR.

КРАТКА ТЕОРИЯ

1. Развойна платка STK500

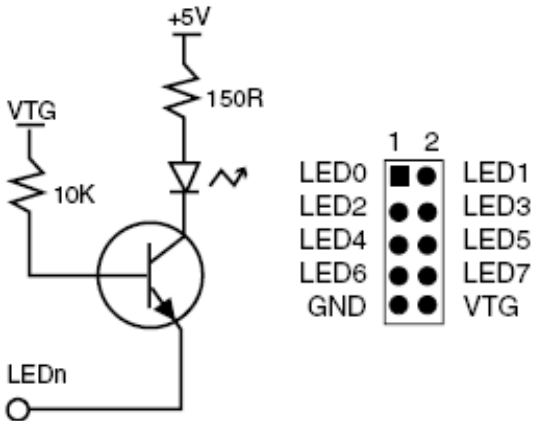
1.1. Основни характеристики

- Съвместимост с програмния пакет AVR Studio;
- RS-232 интерфейс за програмиране;
- Захранващо напрежение 10-15 V DC;
- Цокли за едночипови микроконтролери (ЕЧМК) с 8, 20, 28 и 40 извода;
- Паралелно и серийно програмиране на едночипов микроконтролер (ЕЧМК);
- Вътрешно-системно програмиране на ЕЧМК;
- 8 потребителски бутона с общо предназначение;
- 8 потребителски светодиода с общо предназначение;
- Всички входно-изходни портове на ЕЧМК са изведени върху 10-изводни рейки.

1.2. Апаратно описание

Развойната платка STK500 включва като периферия осем светодиода и осем бутона. Светодиодите и бутоните се свързват към рейки, които са разделени от останалата част на платката (фиг. 1.3). Те се свързват с ЕЧМК чрез 10-проводни лентови кабели към рейките за входно-изходните портове на ЕЧМК. Лентовите кабели трябва да се свързват директно, без да се усукват. Червеният проводник на всеки лентов кабел означава маркер за свързване към първи извод на рейките, който е означен с 1 на маската върху платката (1). Преди да се включи захранването трябва да се провери, дали той е свързан към първи извод на двете рейки.

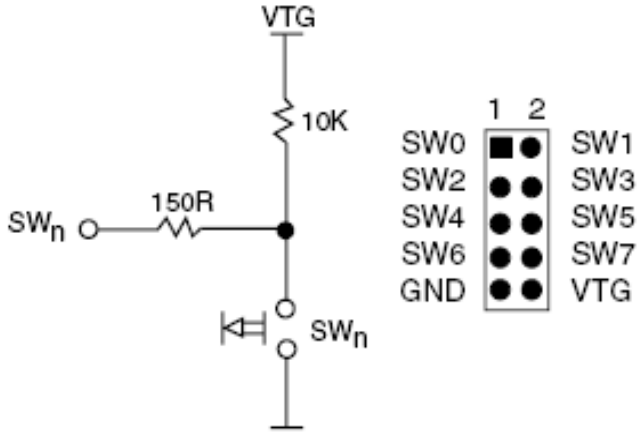
Схемата на свързване на светодиодите е показана на фиг. 1.1.



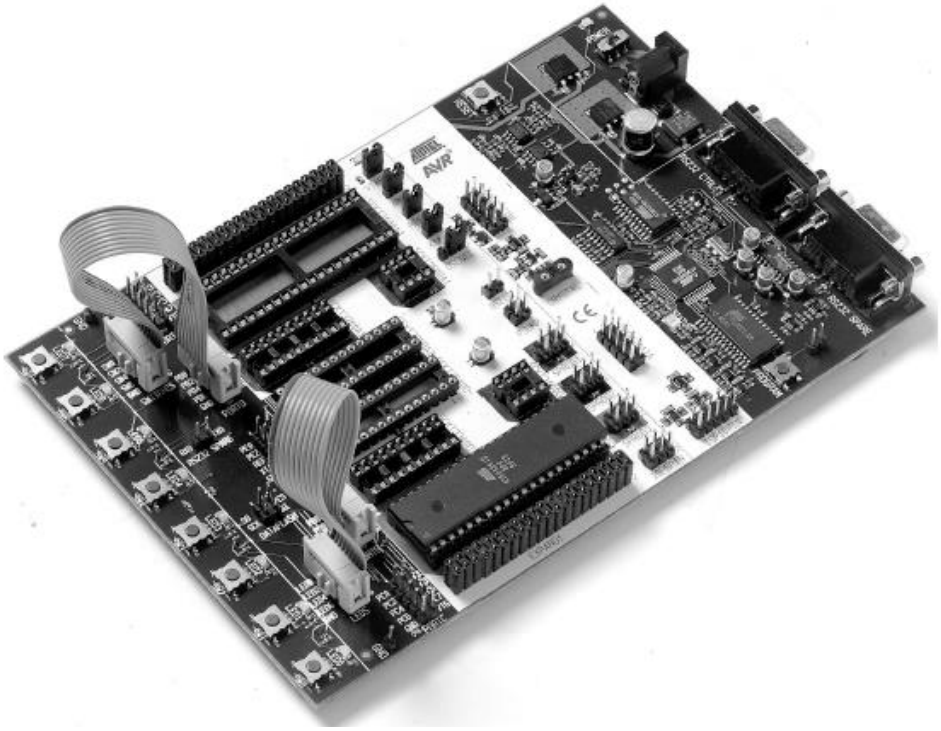
Фиг. 1.1. Схема на свързване на светодиодите и означение на съответстващата им рейка

Свързването на бутоните към съответната рейка е показано на фиг. 1.2. Натискането на бутон води до ниско ниво на съответния извод SWx (от рейката), а отпускането му подава захранващото напрежение (1,8 – 6,0 V) на същия извод. ЕЧМК позволява да се разрешат вътрешните изтеглящи резистори на всеки от изводите, като по този начин не е необходимо свързването на външни такива към бутоните. В развойната платка STK500 е добавен външен резистор със стойност 10kΩ, за да осигури логическа 1 на извода SWn, когато бутонът не е натиснат. Резисторът със стойност 150Ω ограничава тока към ЕЧМК.

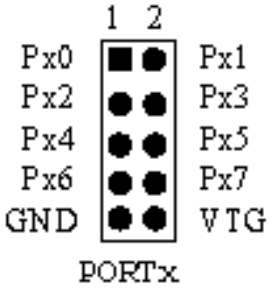
Всеки входно-изходен порт на ЕЧМК може да бъде свързан към светодиодите и бутоните чрез 10-проводни лентови кабели. Към рейките, освен сигналите за светодиодите, бутоните и тези от портовете, са изведени захранващо напрежение +5V (VTG) и маса (GND) (фиг. 1.1, 1.2, 1.4). Кабелите не трябва да се пресукват, за да не се промени номерацията на I/O портовете и свързваната външна периферия (светодиоди и бутони). За целта маркираният проводник от лентовия кабел трябва да се ориентира към извод с номер 1 от двата свързвани конектори.



Фиг. 1.2 Схема на свързване на бутоните и означение на съответстващата им рейка



Фиг. 1.3. Свързване на светодиодите и бутоните към рейките на входно-изходните портове



Фиг. 1.4. Схема на изводите на рейката за PORTx

2. Едночипов микроконтролер (ЕЧМК) ATmega8515

2.1. Изводи

Корпусът на микроконтролера е с 40 извода, които са разпределени по следния начин:

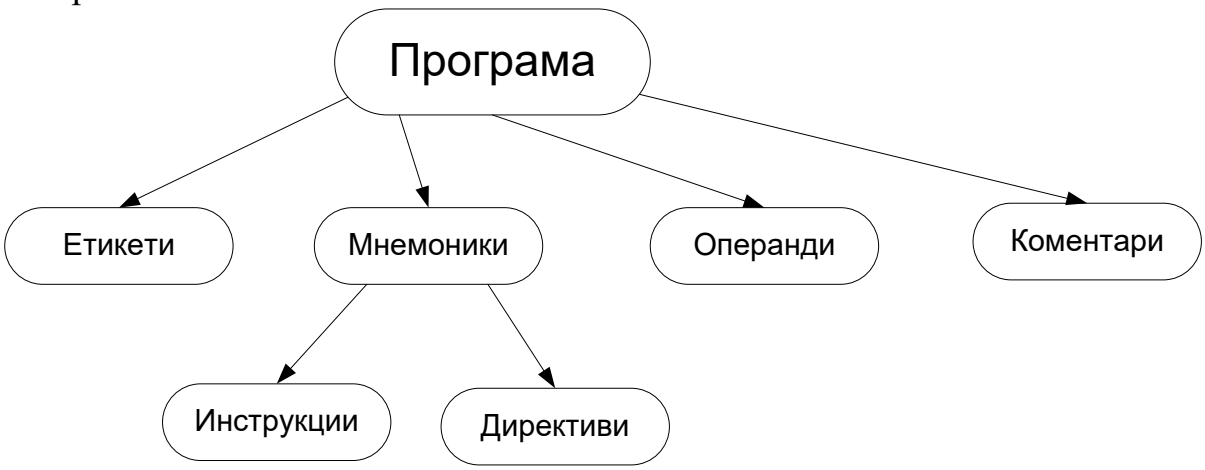
- 35 входно-изходни (I/O, В/И) извода за потребителски цели, разпределени между 5 В/И порта: 8-битовите A, B, C, D и 3-битовия E;
- RESET: вход за начално установяване (НУ) с активно ниско ниво;
- XTAL1 (In): вход за подаване на външен тактов сигнал и един от изводите за свързване на външен тактов генератор от тип кварцов резонатор;
- XTAL2 (Out): вторият от изводите за тактов генератор от тип кварцов резонатор;
- Vcc: захранващо напрежение (+5V);
- GND: маса на захранващото напрежение (-).

3. Език за програмиране Асемблер

3.1. Основни елементи

Всяка програма, съставена на език Асемблер (независимо от това за какъв микроконтролер е предназначена), съдържа следните основни елементи: етикети, мнемоники (инструкции и директиви), операнди и коментари (фиг. 1.5).

От структурната схема може да се види, че всички елементи имат точно определено място, като по този начин те образуват три основни колони в програмния код. Разделянето на колоните в програмния код може да стане чрез въвеждане на символ „Tab“ или с един или повече празни интервала.



Фиг. 1.5. Структура на програма, съставена на език Асемблер

Предназначението на всеки един от елементите е следното:

Етикетите винаги започват от първа колона и служат като идентификатор на реда или частта от програмата, към който (която) централния процесор прави преход по време на изпълнението и. Значението на етикетите се задава от програмиста, като обикновено има връзка с функциите, които изпълнява конкретната част от кода. При транслиране на програмата в машинен код на всеки етикет се присвоява адресът (от програмната памет), на който е написана инструкцията, към която той принадлежи. Етикетите не могат да съвпадат с имена на мнемоники (инструкции и директиви), а също в тях не е допустимо използването на специални символи и празни интервали, освен долно тире („_“). Разпознаването на етикетите в средата на AVR Studio става по символът за двоеточие „:“ веднага след последния символ на етикета.

Мнемониките могат да започват от втора колона или по-далеч спрямо етикетите, като те включват в себе си инструкции (характерни за съответния микропроцесор) и директиви. Инструкциите представляват команди, които се изпълняват от микропроцесора, а директивите са указания, които ще бъдат следвани от транслатора (или компилатора) при превода на програмата на машинен език. Особеното при директивите е, че

за разлика от инструкциите, те не зависят от типа и характера на конкретния микропроцесор. При работа с асемблер синтаксисът на директивите изисква те да започват със символа за точка („.“).

Операндите следват непосредствено след мнемониките, като по този начин образуват трета колона. В зависимост от инструкцията, операндът може да бъде адрес или съдържание на регистър, а също и непосредствена константа. Възможно е при някои инструкции и директиви да има повече от един операнд, тогава за разделител между тях се използва символът за запетая „,“, а може да се поставят около запетаята също и празни интервали или „Tab”.

Коментарите по синтаксис при асемблерен език започват със символа „точка и запетая” („;“) - всичко в дясно от знака на същия ред се приема за коментар. Те както при всеки език за програмиране са единствените елементи от програмата, за позицията на които няма ограничения, независимо от това, че в структурната схема са разположени в четвъртата колона т.е. могат да започват от всяка колона.

3.2. Често използвани инструкции в програмите на асемблер за AVR микроконтролери

В таблица 1.1 е представен синтаксиса на инструкцията от групата за трансфер на данни LDI, а таблиците 1.2 до 1.6 представят синтаксиса и действието на част от инструкциите за аритметични и логически операции.

Таблица 1.1. Инструкция LDI

Инструкция:	LDI Load Immediate
Синтаксис:	(i) LDI Rd, K
Операнди:	$16 \leq d \leq 31$, $0 \leq K \leq 255$
Операция:	(i)Rd <= K
Засегнати битове на STATUS:	Няма
Описание:	Зарежда 8-битова константа (K) непосредствено в регистър r16÷r31 (d)
Думи:	1
Цикли:	1

Таблица 1.2. Инструкция EOR

Инструкция:	EOR Exclusive OR
Синтаксис:	(i) EOR Rd, Rr
Операнди:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Операция:	Rd <= Rd xor Rr
Засегнати битове на STATUS:	I T H S V N Z C - - - <> 0 <> <> -

Описание:	Реализира логическа операция „Сума по модул две” между съдържанието на два регистъра (Rd и Rr) и разполага резултата в регистър Rd.
Думи:	1
Цикли:	1

Таблица 1.3. Инструкция OR

Инструкция:	OR Logical OR							
Синтаксис:	(i) OR Rd, Rr							
Операнди:	$0 \leq d \leq 31$, $0 \leq r \leq 31$							
Операция:	Rd <= Rd or Rr							
Засегнати битове на STATUS:	I	T	H	S	V	N	Z	C
	-	-	-	<>	0	<>	<>	-
Описание:	Реализира логическа операция „ИЛИ” между съдържанието на два регистъра (Rd и Rr) и разполага резултата в регистър Rd.							
Думи:	1							
Цикли:	1							

Таблица 1.4. Инструкция AND

Инструкция:	AND Logical AND							
Синтаксис:	(i) AND Rd, Rr							
Операнди:	$0 \leq d \leq 31$, $0 \leq r \leq 31$							
Операция:	Rd <= Rd and Rr							
Засегнати битове на STATUS:	I	T	H	S	V	N	Z	C
	-	-	-	<>	0	<>	<>	-
Описание:	Реализира логическа операция „И” между съдържанието на два регистъра (Rd и Rr) и разполага резултата в регистър Rd.							
Думи:	1							
Цикли:	1							

Таблица 1.5. Инструкция ADD

Инструкция:	ADD Add without Carry							
Синтаксис:	(i) ADD Rd, Rr							
Операнди:	$0 \leq d \leq 31$, $0 \leq r \leq 31$							
Операция:	Rd <= Rd and Rr							
Засегнати битове на STATUS:	I	T	H	S	V	N	Z	C
	-	-	<>	<>	<>	<>	<>	<>
Описание:	Реализира операция събиране между							

	съдържанието на два регистъра (Rd и Rr) и разполага резултата в регистър Rd без отчитане на пренос.
Думи:	1
Цикли:	1

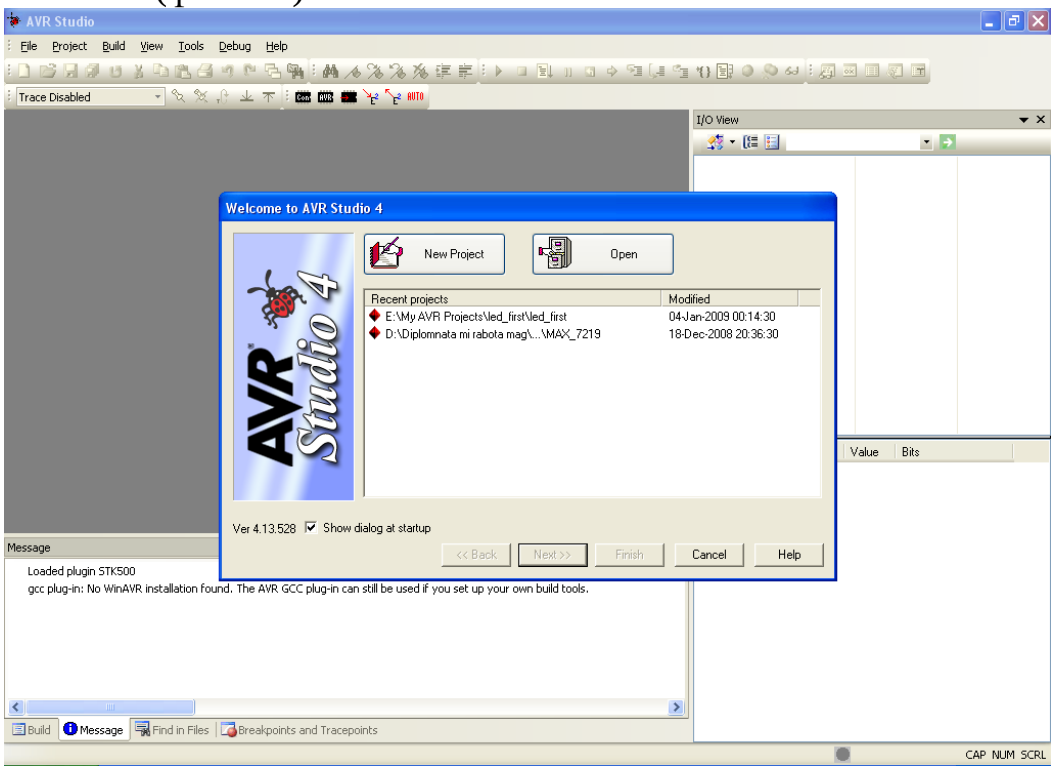
Таблица 1.6. Инструкция SUB

Инструкция:	SUB Subtract without Carry
Синтаксис:	(i) SUB Rd, Rr
Операнди:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Операция:	Rd <= Rd sub Rr
Засегнати битове на STATUS:	I T H S V N Z C - - <> <> <> <> <> <>
Описание:	Реализира операция изваждане между съдържанието на два регистъра (Rd и Rr) и разполага резултата в регистър Rd без отчитане на заем.
Думи:	1
Цикли:	1

4. Развойна среда AVR Studio 4

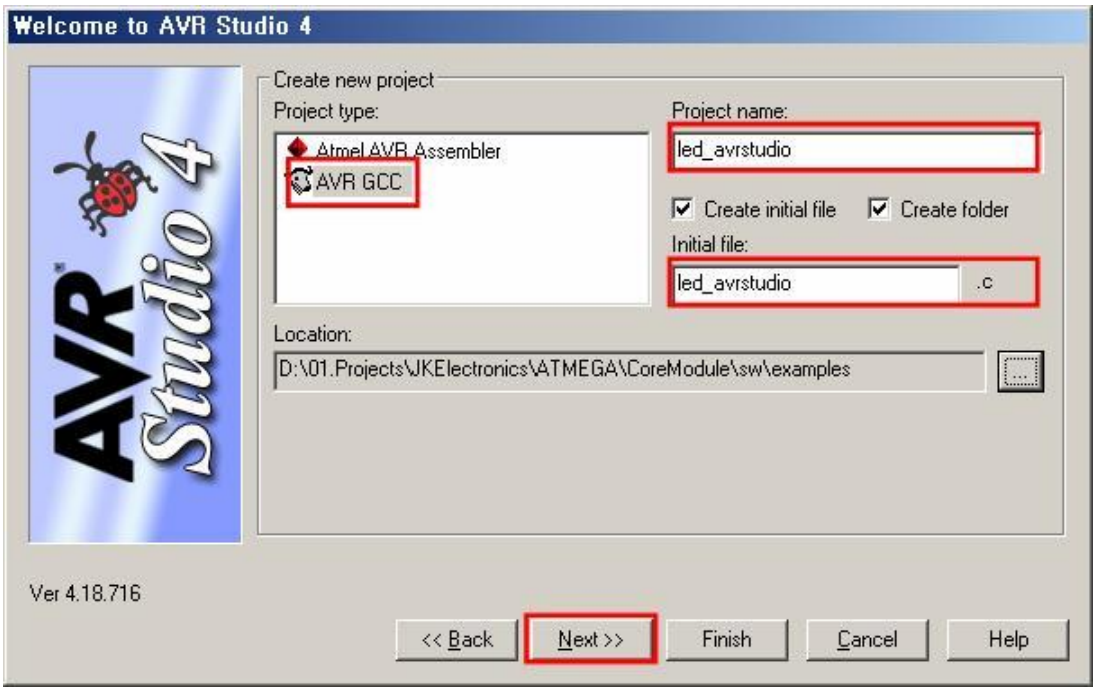
4.1. Създаване на проект с развойна среда AVR Studio 4

При стартиране на AVR Studio се отваря прозореца **Welcome to AVR Studio 4** (фиг. 1.6).



Фиг. 1.6. Прозорец Welcome to AVR Studio 4 на развойната среда

Същият прозорец може да се зареди и от основното меню Project/ Project Wizard. Натиска се бутон New Project за създаване на нов проект или може да се избере един от последно отваряните проекти.



Фиг. 1.7. Прозорец за избор на език и настройки на създавания проект

4.2. Настройване на проекта

При натискане на бутон **New Project** се появява подобен прозорец на показания на фиг. 1.7. В него може да се избере езикът за програмиране – асемблер или C, съответно при кликване върху **Atmel AVR Assembler** или **AVR GCC** (опцията AVR GCC се появява в списъка и работи, само ако е инсталиран правилно подходящ компилатор за C към AVR Studio 4). В същия прозорец се задават име на проекта (допустимо е използването на долно тире - без други специални символи и празни интервали!), посочва се също път, в който да се съхранят файловете на проекта (трябва да се работи само в папката, зададена от ръководителя на занятието!). Желателно е всички файлове на новосъздавания проект да са разположени в нова папка с името на проекта – това става автоматично, когато е избрана отметката **Create Folder**. По подразбиране имената на папката и

4.2.1. Настройка на проекта за работа с асемблер

- От полето Project Type се избира **Atmel AVR Assembler;**
- Задава се име на проекта от полето **Project Name**. Отметката **Create Initial File** трябва да е отбелязана. Името на файла не е задължително да съвпада с това на проекта, но е желателно. Създавания файл по подразбиране е с разширение „.asm”;

- В полето **Location** задаваме папка за проекта (зададена от ръководителя на занятието).
- Натиска се бутон **Next**.

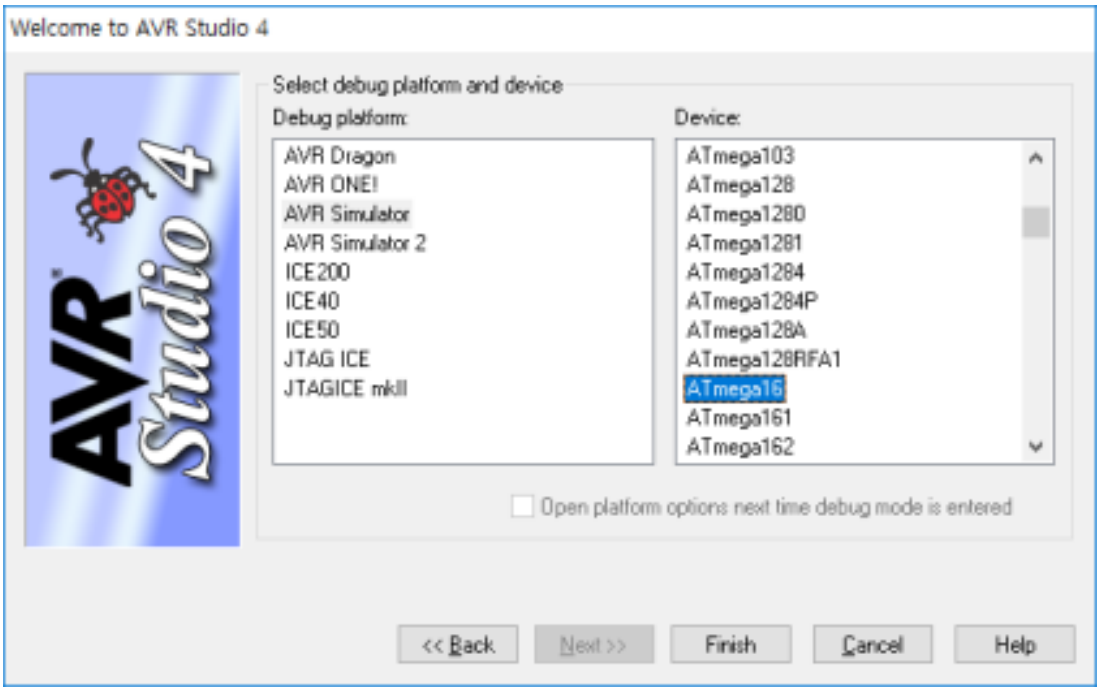
4.2.2. Настройка на проекта за работа с език C

- От полето Project Type се избира **AVR GCC;**
- Задава се име на проекта от полето **Project Name**. Отметката **Create Initial File** трябва да е отбелязана. Името на файла не е задължително да съвпада с това на проекта, но е желателно. Създавания файл по подразбиране е с разширение „.c”;
- В полето **Location** задаваме папка за проекта (зададена от ръководителя на занятието).
- Натиска се бутон **Next**.

4.3. Избор на платформа за дебъг и целеви микроконтролер

На фиг. 1.8 е показан примерен прозорец, в който от Debug platform се избира платформата за дебъг (постъпково проиграване и симулиране на програмите).

В дясната част на прозореца от Device се избира целевият микроконтролер, за който ще се компилира програмният код. Възможно е избраното целево устройство да бъде променено също и на по-късен етап от настройките на създадения проект.



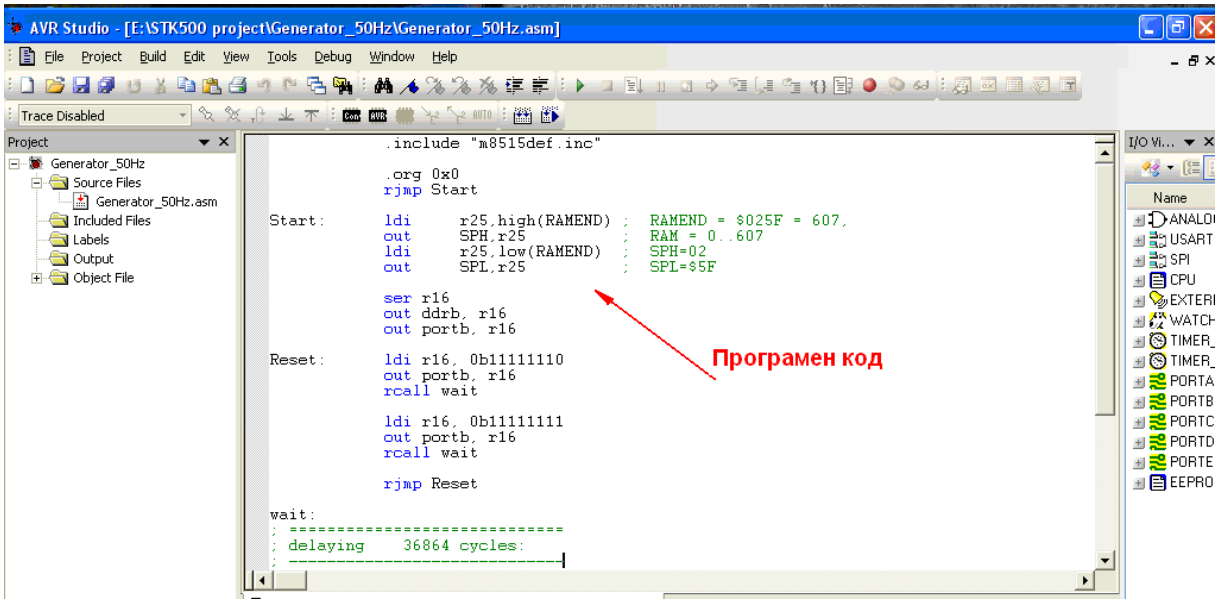
Фиг. 1.8. Прозорец за избор на платформа за дебъг и целеви микроконтролер

Настройването в този прозорец става по следната последователност:

- От полето **Debug Platform** се избира AVR Simulator;
- От полето **Device** се избира ATmega8515;
- Натиска се бутон **Finish**;

След натискане на бутон Finish се създава празен проект за избрания програмен език в началната стъпка. Отваря се в текстовия редактор на AVR Studio създаденият празен изходен файл, в който трябва да се въвежда програмен код на съответния език.

- В отворения от средата текстов редактор се набира потребителската програма (фиг. 1.9).



Фиг. 1.9. Разполагане на потребителската програма в текстовия редактор

4.4. Компилиране на потребителската програма

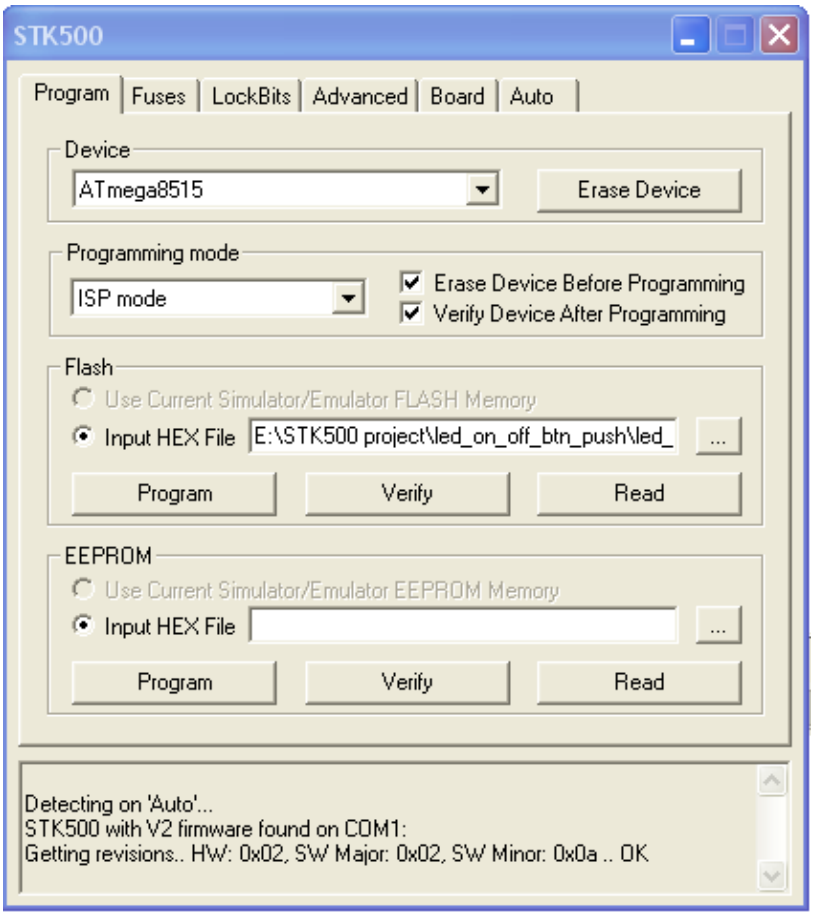
До различни опции за компилиране може да се достигне от менюто **Build**. Възможно е също бързо да се извърши чрез бутона  или с натискане на функционален бутон (F7);

4.5. Зареждане на потребителския изходен код .hex в микроконтролера

Изпълнява се следната последователност:

- От главното меню се избира Tools – Programm AVR – Connect;
- Отваря се диалогов прозорец STK500 (фиг. 1.10);
- В падащото поле Device се задава вида на микроконтролера (ако не е избран автоматично от настройките на проекта);
- В падащото поле Programming mode задавате режима на програмиране (ако не е селектиран той трябва да бъде ISP mode);

- В полето Flash задавате директорията, в която се намира изходният .hex файл (генерира се в папката на проекта при компилиране);
- Натискате бутон Program от полето Flash.



Фиг. 1.10. Диалогов прозорец за настройване на STK500 и програмиране на микроконтролери

4.6. Примерен код first1.asm

```
          .include "m8515def.inc"

          Ldi r16, 0b00001010
          Ldi r17, 0x05
          Add r17, r16
          Or r17, r16
          Eor r16, r17
          And r17, r16

Vechen_cikal:
          Nop
          Rjmp Vechen_cikal
```

4.7. Примерен код first2.asm

```
          .include "m8515def.inc"

          Ldi r16, 0b11001010
          Ldi r17, 0x45
          Add r17, r16
          Or r17, r16
          Eor r16, r17
          And r17, r16

Vechen_cikal:
          Nop
          Rjmp Vechen_cikal
```

4.8. Примерен код first3.asm

```
          .include "m8515def.inc"

          Ldi r16, 0b11001010
          Ldi r17, 0x45
          Sub r17, r16
          Eor r17, r16
          And r16, r17
          Or r17, r16

Vechen_cikal:
          Nop
          Rjmp Vechen_cikal
```

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се разучи развойна платка STK500 и документацията на микроконтролера ATmega8515.

2. Да се разучи средата за разработка AVR Studio 4.

3. Използвайки информацията за инструкциите от таблици 2.1 до 2.6 (или/и документацията на ATmega8515), определете теоретично, какъв ще е полученият краен резултат в регистрите r16 и r17 след изпълнение на примерните програми от точки 4.6, 4.7 и 4.8.

4. Да се създаде нов проект на асемблер.

5. Да се въведе примерната програма от точка 4.6.

6. Да се компилира и симулира нейното изпълнение чрез вградения в средата симулатор. По време на постъпковото изпълнение на програмата в симулатора, да се проследи при какви ситуации се променят състоянията на флаговете в статусния регистър (SREG) и да се направи сравнение с очакваното поведение според таблицата с инструкции за ATmega8515.

7. Запишете какво е съдържанието на регистрите r16 и r17 след достигането на безкрайно повтарящият се цикъл в края на симулираната програма.

8. Да се изпълнят задачи от 5 до 7 също и за останалите примерни програми 4.7 и 4.8 (Упътване: може да се редактират само различните инструкции и операнди в програмите).

9. Сравнете получените резултати от задачи 3 и 7 и направете изводи.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Какви са основните характеристики на микроконтролер ATmega8515?

2. Кои инструкции променят състоянието на статусния регистър?

3. Защо е нужно да се организира „вечен цикъл“ в края на примерните програми?

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 2
Преобразуване на числа в различни бройни системи, аритметични действия и побитови логически операции, представяне на двоични числа със знак

ЦЕЛ НА УПРАЖНЕНИЕТО

Да се изучат операциите събиране и изваждане в двоична и 16-чна бройна системи, преобразуването на числа от една бройна система в друга, представянето на двоични числа със знак, побитовите логически операции.

КРАТКА ТЕОРИЯ

Правилата за изпълнение на аритметичните действия с двоични числа са представени в таблица 2.1 за двоично събиране, изваждане и умножение.

Таблица 2.1. Правила за извършване на аритметични действия с двоични числа

Таблица за двоично събиране	Таблица за двоично изваждане	Таблица за двоично умножение
0 + 0 = 0	0 - 0 = 0	0 . 0 = 0
0 + 1 = 1	1 - 0 = 1	0 . 1 = 0
1 + 0 = 1	1 - 1 = 0	1 . 0 = 0
1 + 1 = 10	10 - 1 = 1	1 . 1 = 1

Двузначната сума в таблицата за двоично събиране означава, че при събирането на две двоични цифри равни на 1, в произволен разряд на двоичното число възниква пренос към съседния старши разряд.

При изваждане от двоичен разряд при нужда се взема „назаем" единица от следващия по-старши разряд. Тази взета единица е равна на две единици от дадения разряд. „Назаем" вземаме всеки път, когато цифрата в разряда на умалителя е по-голяма от съответната й в умаляемото.

Необходимостта от преобразуване на числата от една бройна система в друга възниква поради това, че ЦИМ работи в двоична система, програмите се записват в шестнадесетична или осмична система, а подготовката на данните при пресмятанията и извеждането на резултатите получени от машината след пресмятането, става в десетична система.

Обикновено привеждането на числата от една бройна система в друга се извършва от автоматичните устройства на машината. При програмирането на задачите обаче, а също така и при намеса на оператора в процеса на пресмятането може да възникне необходимост ръчно да бъде

приведено едно число или неголяма група от числа от една бройна система в друга.

Най-често при привеждане на числа, които представляват неправилни дроби от една бройна система в друга, цялата и дробната част се превеждат поотделно, като се спазват различаващи се едно от друго правила.

Правила за превеждане на цели числа: за превеждане на цяло число от бройна система с основа S в система с основа p е необходимо последователно да се дели числото на основата на новата система и получените частни отново да се делят на основата на системата, докато частното стане по-малко от p. Най-старшата цифра при записване на числото в системата ще бъде последното частно, а следващите я цифри се получават от остатъците, записани в последователност, обратна на тази при получаването им.

Правила за превеждане на дробни числа: за превеждане на правилни дроби от система S в бройна система с основа p е необходимо да се умножи изходната дроб, както и получаващите се след умножението дробни части по основата p, представена в система с основа S. Целите части на получаващите се произведения дават последователността от цифри при представянето на дробта в системата p.

Правила за превръщане на осмични и шестнадесетични числа в двоични и обратно. Правилата са извънредно прости, защото основата на осмичната и шестнадесетичната бройна система са степени на числото 2, а именно $8 = 2^3$ и $16 = 2^4$.

За превръщането на осмично число в двоично е необходимо само да заменим всяка цифра на осмичното число със съответстващото й триразрядно двоично число. По този начин за преминаване от шестнадесетична бройна система в двоична, всяка цифра от шетнадесетичното число ще трябва да се замени със съответстващото й четириразрядно двоично число. При това началните нули на най-старшата цифра могат да не се пишат.

За преминаване от двоична към осмична (или шестнадесетична) система постъпваме по следния начин: започвайки от запетаята на ляво и на дясно, разделяме двоичното число на групи по три (четири) разряда. При необходимост допълваме с нули крайната лява и крайната дясна група. След това всяка група заменяме със съответната й осмична (шестнадесетична) цифра.

В ЦИМ се използват две форми за представяне на числата: представяне на числата с фиксирана запетая и представяне на числата с плаваща запетая.

При представяне на числата с фиксирана запетая, позицията на запетаята се фиксира на определено място относно разрядите на числото и

запазва това място постоянно за всички числа, които се представят в дадената разрядна мрежа на машината.

В общия случай в машината с плаваща запетая числото се представя в неговият експоненциален вид (2.1):

$$X = m \cdot S^{p}, \qquad (2.1)$$

където m е мантиса на числото X;

S^{p} – характеристика на числото X;

p – порядък;

S – основа на характеристиката.

Обикновено числото S съвпада с основата или е цяла степен на основата на бройната система, която се използва за представяне на мантисата m в ЦИМ.

Порядъкът p може да бъде положително или отрицателно число, показващ положението на запетаята в числото X.

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се преведе десетичното число 145 в двоична система.

2. Да се преведе десетичната дроб 0,6875 в двоична система.

3. Да се съберат следните двойки 8-битови беззнакови числа и се определи значението (0 или 1) на флага за пренос C “навън” от старшия осми бит (b7):

А) b7……b0	Б) b7……b0	В) b7……b0	Г) b7……b0
1001 1011	0101 1101	0111 1110	1000 0000
C = 0010 0010	C = 0010 0010	C = 1011 0101	C = 1000 0001

4. Да се извадят двойките 8-битови беззнакови числа от задача 3 и се определи значението (0 или 1) на флага за заем (той е и флагът пренос C) от виртуалния девети бит, разположен вляво от осмия бит (b7).

5. Да се кодират в допълнителен код с 8 бита следните отрицателни числа, чийто модули (абсолютни стойности) са представени със седем бита:

А) b7 b6……b0	Б) b7 b6……b0	В) b7 b6……b0	Г) b7 b6……b0
100 1011	101 1101	111 1110	000 0001

6. Същите отрицателни числа да се кодират в допълнителен код с 16 бита.

7. Операцията изваждане S = (a - b) може да се замени с операцията събиране S = (a + (-b)), в която умалителят (b) е представен с допълнителния си код. Намерете S = (a - b) за следните двойки числа (a) и (b), като предварително преобразувате в допълнителен код умалителя (b). Забележете, че и двете числа във всяка двойка са представени чрез 7-битовите си модули (т.е. чрез положителни числа), докато 8-битовата им

разлика S = (a - b) може да бъде интерпретирана в някои случаи като представяща положително число, а в други – като представяща отрицателно такова.

А) b7 b6……b0	Б) b7 b6……b0	В) b7 b6……b0	Г) b7 b6……b0
a 001 1011	a 101 1101	a 111 1110	a 000 0000
b 010 0010	b 111 0111	b 011 0101	b 000 0001

8. Да се представят двоичните числа от задача 1 в 16-чен вид и да се извърши събирането им в 16-чна бройна система. Забележете, че резултатите, получени в 16-чен вид, когато се превърнат в двоичен вид, съвпадат със съответните им, получени при решаване на изходната задача 1.

9. Над двойките числа от задача 1 да се извърши побитова логическа операция AND (И).

10. Над двойките числа от задача 1 да се извърши побитова логическа операция OR (ИЛИ).

11. Над двойките числа от задача 1 да се извърши побитова логическа операция EOR (Изключващо ИЛИ).

12. Над всяко от двойките числа от задача 1 да се извърши побитова логическа операция COM (Побитово допълнение, One’s complement, допълнение „до единици”).

13. Над всяко от двойките числа от задача 1 да се извърши побитова логическа операция NEG (Побитово Two’s complement, допълнение „до нули”: (0000 0000) – (числото)).

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Какъв е обхвата на представяните стойности при 8-битовите беззнакови числа? А при 16-битовите?

2. Какъв е обхвата на представяните стойности при 8-битовите числа със знак? А при 16-битовите?

3. Заредете в програмната среда AVR Studio създаденият проект от Лабораторно упражнение №1 (или създайте нов, в който въведете една от примерните програми). Компилирайте програмата и стартирайте нейната симулация. Анализирайте настъпващите промени в стойностите на използваните регистри с общо предназначение r16 и r17 при изпълнението на аритметичните и логическите инструкции при зареждане на различни начални стойности в тях.

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 3
Организация на адресното пространство за данни, методи за адресиране, инструкции за трансфер и условно разклоняване на програми

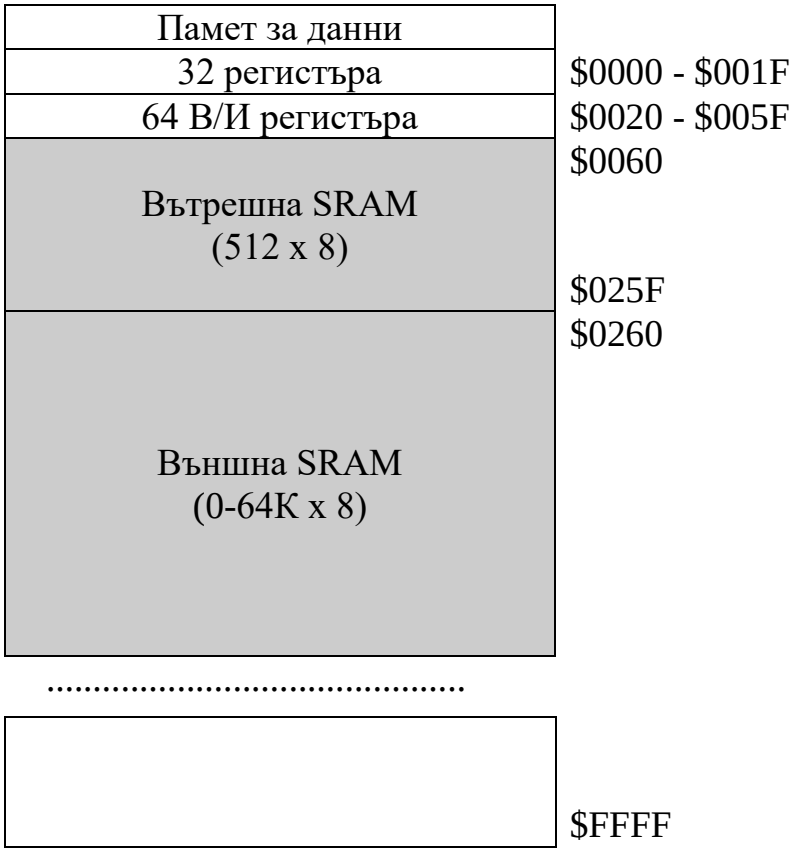
ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да разучат организацията на вградената в микроконтролер ATmega8515 SRAM памет за данни и различните регистри, методите за адресиране и групите от инструкции за трансфер на данни и условно разклоняване на програми.

КРАТКА ТЕОРИЯ

1. Организация на вградената памет за данни

На фиг. 3.1 е показана организацията на адресното пространство и SRAM (Static Random Access Memory, статична памет със случаен достъп) паметта при микроконтролер ATmega8515.



Фиг. 3.1. Организация на вградената памет за данни

Адресното пространство е непрекъснато и е с обем 608 клетки с 8-битова размерност като включва четири различни области, които са

последователно разположени. Първият адрес започва от 0. Започвайки от най-младшите адреси, областите са следните:

- 32 регистъра с общо предназначение – намират се в процесорното ядро на микроконтролера и се използват за временно съхраняване на стойности (константи, променливи, адреси) и междинни резултати при изпълнението на програми. Посочват се като операнди за голяма част от инструкциите в асемблер;
- 64 регистъра със специално предназначение – тази област е т. нар. „входно-изходна памет“ (I/O memory, В/И памет), тъй като повечето от тези регистри имат специализирани функции и се намират във вградените в чипа периферни модули (като портове, таймери, комуникационни интерфейси), които са извън процесорното ядро. Тези регистри се използват основно за конфигуриране, следене на статуса на периферните модули и цялостната работа на микроконтролера, както и за буферно съхраняване на входно-изходни данни;
- 512 клетки вътрешна памет за данни SRAM на микроконтролера;
- Следващите адреси до адрес $FFFF са заделени за достъп до външна SRAM памет (адресното пространство до 64KB), която може да се свърже към микроконтролера.

Тази област започва от адреса след вътрешната SRAM памет. Адресните пространства на регистрите с общо назначение, входно – изходното, разширеното (допълнителното) за вход и изход и вътрешната SRAM заемат младшите 608 байта в нормален режим, така, че когато се използва 64KB (65536 байта) външна памет, са достъпни само 64928 байта от външната памет.

Достъпът до външната SRAM отнема един допълнителен тактов цикъл за байт в сравнение с достъпа до вътрешната SRAM памет. Това означава, че инструкциите LD, ST, LDS, STS, LDD, STD, PUSH и POP отнемат един допълнителен такт. Ако Стекът е разположен във външната SRAM, прекъсванията, извикването на подпрограми и адресите за връщане отнемат три такта повече, защото съхраняването и извличането на съдържанието на двубайтовият Програмен брояч при достъпът до външната памет не се извършва с помощта на конвейрния /pipe-line/ достъп на вътрешната памет.

2. Методи за адресиране на вградената памет за данни SRAM

ATmega8515 поддържа шест режима на адресиране, които са както следва:

1. Директно (пряко);
2. Индиректно (непряко) с отместване;
3. Непряко;
4. Непряко с предварително намаляване (декремент);
5. Непряко с последващо увеличаване (инкремент);

6. Неявно.

Указателните регистри (наричани също „индексни“) за непряко адресиране са регистрите от R26 до R31.

Директното адресиране достъпва цялото пространство за данни, разположено в първите 256 адреса (8-битов адрес).

Режимът 'непряко адресиране с отместване' достъпва 63 адресни клетки от базовия адрес, зададен с регистри Y или Z.

При използване на режими за индиректно адресиране на регистри с автоматично предварително намаляване или последващо увеличаване, адресните регистри X, Y и Z се намаляват или увеличават (по стойност).

При неявното адресиране към инструкциите не се поставят операнди в програмния код. При тях действието не изисква операнди или те вече се намират в известни регистри.

Всичките 32 регистъра с общо предназначение, 64 В/И регистъра и 512-те байта от вътрешната памет за данни SRAM в ATmega8515 са достъпни, посредством тези режими за адресиране.

3. Индексни регистри X, Y и Z

Регистрите от R26 до R31 имат някои допълнителни функции при използването им за общи цели. Тези шест 8-битови регистри се използват като три 16-битови адресни указателя за непряко адресиране на адресното пространството за данни, когато се обединят по двойки. Организацията на тези регистри е представена фиг. 3.2.

Фиг. 3.2. Организация на индексни регистри X, Y и Z

4. Указател на стека

Стекът се използва главно за съхраняване на временни данни, за съхраняване на моментната стойност на променливи и за съхраняване на адреси на връщане след прекъсвания или извикване на подпрограми. Регистърът ‘указател на стека’ винаги показва началото (‘върха’) на стека.

Забележете, че стекът е реализиран като нарастващи местоположения от по-високи към по-ниски адреси на паметта. Това означава, че инструкцията PUSH намалява стойността на указателя на стека.

Указателят на стека сочи SRAM областта за данни на стека, където се разполага той при използване на подпрограми или прекъсвания. Това стеково пространство в SRAM паметта трябва да се дефинира в програмата преди да се изпълняват каквито и да е извиквания на подпрограми или да се разрешават прекъсвания. Указателят на стека трябва да се инициализира първоначално с адрес над $0060, за да е възможно той да бъде използван. Обикновено се разполага в последния адрес от вградената SRAM памет – адрес $025F. Указателят на стека се намалява с единица, когато данните се записват в Стека с инструкция 'PUSH' и се намалява с две единици, ако се записва в Стека адрес на връщане при извикване на подпрограма или използване на прекъсване. Указателят на стека се увеличава с единица, ако се четат данни с инструкция 'POP' и се увеличава с две единици, ако от стека се чете адрес на връщане от подпрограма с инструкция RET или връщане от прекъсване с инструкция RETI.

Указателят на стека на AVR е реализиран като два 8-битови регистъра във В/И адресно пространство фиг. 3.3.

бит	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
четене/ запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
начална стойност	0	0	0	0	0	0	0	0	

Фиг. 3.3. Указател на стека

5. Статусен регистър SREG

Статусният регистър SREG е наричан също „регистърът на състоянието", а също и „флагов регистър", защото битовете в него се наричат „флагове". Той съдържа информация за резултата от последната изпълнена аритметична, логическа, побитова или друга инструкция (вижте за справка таблицата с всички поддържани инструкции за ATmega8515 от неговата документация). Тази информация може да се използва за промяна на програмния ход с цел извършване на условни операции и разклонения в програмите. Обърнете внимание, че регистърът на състоянието се актуализира след всяка ALU операция. Това в много случаи ще премахне необходимостта от използване на специални инструкции за сравнение, което води до по-бърз и по-компактен код.

Регистърът на състоянието не се съхранява автоматично при влизане в програма за прекъсване и не се възстановява при връщане от

прекъсване. Това трябва да се обработва от потребителската програма (ако е нужно).

На фиг. 3.4 са представени битовете в SREG.

бит	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
четене/запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
начална стойност	0	0	0	0	0	0	0	0	

Фиг. 3.4. Битове (флагове) на статусния регистър SREG

5.1. Значение на флаговете на SREG

- Bit 7 – I: Global Interrupt Enable

Битът Global Interrupt Enable трябва да бъде в „1“, за да бъдат активирани прекъсванията. След това индивидуалното управление на разрешаването на прекъсването се извършва в отделни контролни регистри. Ако Global Interrupt Enable бита е в „0“, нито едно от прекъсванията не е разрешено, независимо от индивидуалните настройки за разрешаване на прекъсване. I-битът се нулира от хардуера след възникване на прекъсване и се установява в „1“ от инструкцията RETI, за да позволи последващи прекъсвания. I-битът може също да бъде зададен и изчистен от приложението с инструкциите SEI и CLI.

- Bit 6 – T: Bit Copy Storage

Инструкциите за битово копиране BLD (Bit LoaD) и BST (Bit STore) използват T-бита като източник или дестинация за оперирания бит. Бит от регистърния файл може да бъде копиран в бит T чрез инструкцията BST, а стойността на бит T може да бъде копиран в избран бит в регистровия файл чрез инструкцията BLD.

- Bit 5 – H: Half Carry Flag

Флагът за полупренос H показва пренасяне между младшия и старшия полубайт при някои аритметични операции.

- Bit 4 – S: Sign Bit, S = N $\oplus$ V

S-битът винаги е свързан с изключващо или между флага N за отричателен резултат и флага за аритметично препълване V.

- Bit 3 – V: Two’s Complement Overflow Flag

Флагът за отразяване на ситуации на аритметично препълване.

- Bit 2 – N: Negative Flag

Флаг N показва отрицателен резултат при аритметична или логическа операция.

- Bit 1 – Z: Zero Flag

Флаг Z показва нулев резултат при аритметична или логическа операция.

- Bit 0 – C: Carry Flag

Флаг C показва пренасяне към 8-ми бит (или "заем") при аритметични или логически операции (например при събиране на числа, когато резултатът е по-голям от 255 или при изваждане, когато от по-малко число се изважда по-голямо).

6. Пример за условно разклонение в програма

```
          add r20, r22          ;Събира r22 към r20, резултатът – в r20
          brcc noCarry          ;Ако флаг C = 0 -> преход към noCarry
          ……                    ;други инструкции
          rjmp Next             ;безусловен преход към Next
noCarry:  nop                   ;Целева инструкция, ако преходът се извърши
          ……                    ;други инструкции
Next:     nop                   ;Целева инструкция на безусловния преход
```

7. Примерно реализиране и извикване на подпрограма

Подпрограмата представлява група инструкции, които са поставени на избран начален адрес в програмната памет, който най-често се именува с подходящ етикет. В края на подпрограмата е нужно да се постави инструкцията ret за връщане към мястото на извикване. Самото извикване на подпрограмата става чрез инструкцията rcall, за която трябва да се посочи като операнд адресът, където се намира тя, но представен като отместване (число със знак) от текущия адрес на инструкцията rcall. Полесно е да се постави „име_на_подпрограмата“, представляващ етикет на адреса, а компилаторът ще го изчисли като относителен адрес при компилирането.

7.1. Примерна програма за изваждане на две конкретни числа чрез подпрограма

```
          .include “m8515def.inc”
          .org  0
          rjmp  reset

;подпрограма за изваждане на стойностите в r16 и r17
          .org $20
Sub_ab:
          sub  r16,r17          ;след изваждане резултатът е в r16
          ret                   ;връщане към главната програма

;главна програма
reset:
;зареждане на стековия указател с начален адрес
          ldi  r16, $5f         ;може да се използва израза low(ramend)
          out  spl, r16
          ldi  r16, $02         ; може да се използва израза high(ramend)
          out  sph, r16

          ldi  r16, 0b10011011 ;първо число
          ldi  r17, 0b00100010 ;второ число

          rcall Sub_ab          ;извикване на подпрограмата

always:   rjmp always           ;loop завинаги
```

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Запознайте се подробно със списъка с инструкции за ATmega8515 в неговата документация.
2. Да се състави програмен код, с който да се запишат първите 10 байта от паметта с непосредствените константи 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, след което да се запише съдържанието на клетката с адрес $0065 в РОН R17 с използване на метода за пряка адресация.
3. Да се модифицира заданието от точка 1 с използване на останалите четири метода.
4. Да се състави програмен код за записване на посочените стойности в паметта, но това да се реализира чрез цикъл, който се повтаря необходимия брой пъти. В тялото на цикъла да се използват инструкции с подходящ метод за адресиране, така че да се намали броят използвани инструкции.
5. Да се симулира изпълнението на програмите с вградения в средата симулатор.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Колко и какви са наименованията и ролята на флаговете в SREG?
2. Съществуват ли инструкции в ATmega8515 за проверка на всички шест релации <, <=, =, !=, >=, > между числа със знак и между беззнакови числа?
3. Каква е ефикасността на инструкциите от тип „Skip IF …“ по отношение на еквивалентните им инструкции от тип „Branch IF …“, когато те се сравняват по дължина на кода си и по времената на изпълнението си (в брой периоди на тактовия генератор)?

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 4
Входно-изходна система на микроконтролер ATmega8515, програмни времезакъснения и управление на светодиодна индикация

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да изучат програмния модел на входно-изходната система и начина за програмна конфигурация с посока „изход” на паралелните входно-изходни изводи (Port A, Port B, Port C, Port D и Port E).Също така запознаване с начина за реализиране на програмни времезакъснения и съставяне на програми за управление на светодиодната индикация на STK500.

КРАТКА ТЕОРИЯ

1. Конфигуриране на входно-изходните портове на ATmega8515

1.2. Конфигуриране на входно-изходните портове

ЕЧМК ATmega8515 разполага с 5 входно-изходни порта, означени по следния начин:

- Port A (*PA7..PA0*)
- Port B (*PB7..PB0*)
- Port C (*PC7…PC0*)
- Port D (*PD7..PD0*)
- Port E (*PE2..PE0*)

1.3. Регистри асоциирани с входно-изходните портове

Общо три регистъра са асоциирани с входно-изходните портове:

- DDRx – за задаване на посока на данните (0 = вход; 1 = изход);
- PORTx – буферен регистър за цифровите сигнали към изходите;
- PINx – буферен регистър за прочетени цифрови сигнали на входовете.

Всеки портов извод се състои от три регистрови бита: DDxn, PORTxn и PINxn. DDxn битът в DDRx регистъра избира посоката на съответния извод с номер x (x = 7…0). Ако в DDxn е записана логическа единица, Pxn е конфигуриран като изходен извод. Ако в DDxn е записана логическа нула, Pxn е конфигуриран като входен извод. Ако в PORTxn е записана логическа единица, когато съответния извод е конфигуриран като вход, изтеглящият резистор е активиран. За да се изключи изтеглящият резистор в PORTxn трябва да бъде записана лог.”0” или съответния извод трябва да бъде конфигуриран като изход. Ако в PORTxn е записана логическа единица, когато съответния извод е конфигуриран като изход на съответния портов извод се извежда лог.”1”. Ако в PORTxn е записана

логическа нула, когато съответния извод е конфигуриран като изход на съответния портов извод се извежда лог.”0”.

2. Програмно генерирани времезакъснения

За изпълнението си всяка инструкция отнема един или повече такта на тактовия генератор. Ако е известна тактовата честота Fosc в [Hz], тогава продължителността на един такт Tosc в [s] представлява 1/ Fosc. Времето за изпълнение на даден програмен фрагмент ще бъде равно на сумата от времената за изпълнение на всяка от инструкциите. По този начин, чрез поставяне в програмата на инструкции с известна продължителност на изпълнение (например инструкция nop се изпълнява за 1 такт), може да се задават много точно кратки времеви интервали, в които се задържа изпълнението на следващата част от програмата.

Например ако Fosc = 1MHz, тогава продължителността на един такт ще бъде (4.1):

$$Tosc = 1/Fosc = 1.10^{-6}\ s = 1\ \mu s \qquad (4.1)$$

За постигането на закъснение от порядъка на 1 секунда ще бъдат нужни 1000000 такта или толкова поредни nop инструкции. По-ефективен начин е организирането на цикли, които повтарят многократно група инструкции с известна продължителност или друг вложен цикъл със зададен брой повторения. Когато се вложат няколко цикъла един в друг чрез промяна в броя повторения за всеки може да се постигне времезакъснение с голяма продължителност като в същото време програмният код остава сравнително кратък. Времезакъснението в този случай може да се пресметне в брой цикли като приблизително равно на произведението на броячите на всички вложени цикли, умножено по броя цикли за изпълнение на инструкциите в тялото на всеки вложен цикъл.

Пример за два вложени цикъла:

```
;цикъл за програмно генерирано закъснение
          Counter2 = Cy         ;инициализация Loop2 -> Tinit2
Loop2:
;тяло на цикъла Loop2

          Counter1 = Cx         ;инициализация Loop1 -> Tinit1
Loop1:
;тяло на цикъла Loop1                                    -> Tloop1
          Counter1 = Counter1 - 1
          ;проверка за край на Loop1
          if(Counter1 > 0) -> Loop1
;край на Loop1
                                                         -> Tloop2
          Counter2 = Counter2 - 1
          ;проверка за край на Loop2
          if(Counter2 > 0) -> Loop2
;край на Loop2
```

Ако се анализират двата вложени цикъла, разписани като псевдокод в примера по-горе, могат да се разграничат за всеки от циклите по две части – инициализация и тяло. Тялото на по-външния цикъл включва инициализацията и тялото на вътрешния цикъл. Това означава, че при пресмятането на времето за закъснение трябва да се добави и сумата от времето за инициализиране на всеки вложен цикъл, умножено по броя повторения на по-външния цикъл, в който е включено. Това може да се представи с обобщената формула за пресмятане на общото закъснение T на двата вложени цикъла от примера (4.2):

$$T = Tinit2 + Cy * (Tinit1 + Cx * Tloop1 + Tloop2) \qquad (4.2)$$

За да бъде точно изчислено времезакъснението е нужно да се отчете също и факта, че времето за изпълнение на инструкциите за проверка на условие, които се използват в края на циклите, е различно според това, дали се изпълнява проверяваното от тях условие или не.

2.1. Примерен код за програмно генерирано времезакъснение

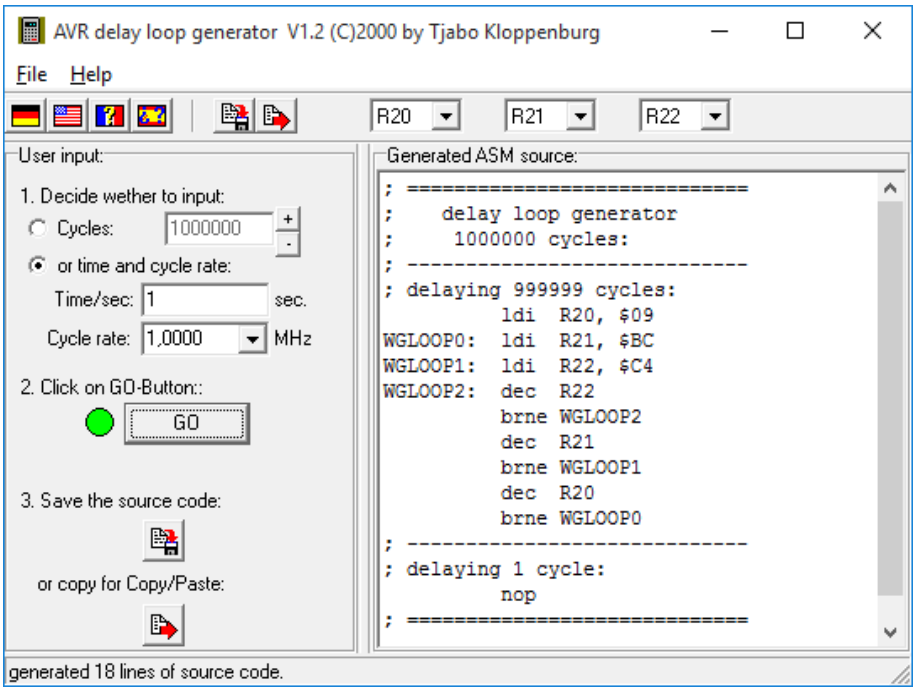
Следният програмен фрагмент ще задържи (забави) следващата част от програмата за точно 1000000 цикъла.

```
; ============================
;    delay loop generator
;     1000000 cycles:
; ----------------------------
; delaying 999999 cycles:
          ldi  R20, $09
WGLOOP0:  ldi  R21, $BC
WGLOOP1:  ldi  R22, $C4
WGLOOP2:  dec  R22
          brne WGLOOP2
          dec  R21
          brne WGLOOP1
          dec  R20
          brne WGLOOP0
; ----------------------------
; delaying 1 cycle:
          nop
; ============================
```

В началото се зареждат константни стойности в регистрите R20, R21 и R22, които се използват като броячи в три вложени цикъла. При всяко повторение на най-вътрешния вложен цикъл се декрементира стойността на R22 и когато тя стане равна на нула флаг Z ще стане „1“ и инструкцията brne (проверява за флаг Z=0, при вярно условие преминава към WGLOOP2) няма да даде верен резултат и ще премине към следващата инструкция, която декрементира R21 в тялото на по-външният цикъл. Ако R21 не е нула следващата инструкция brne завърта следващата итерация на този цикъл към етикета WGLOOP1, където R22 (броячът на

вътрешния цикъл) отново се зарежда с неговата начална стойност. По подобен начин се случват нещата и при нулирането на брояча на втория цикъл. Когато се нулира броячът на най-външния цикъл броят на изразходените тактове ще бъде 999999 и с добавянето на още една инструкция nop – те стават точно 1000000.

За удобство при създаването на такива вложени цикли за времезакъснения може да се използва например програмният продукт “AVR delay loop generator” (може да го намерите в папката с материалите за лабораторните упражнения). В неговия графичен интерфейс (фиг. 4.1) е важно да изберете трите регистъра с общо предназначение, които ще бъдат използвани в горната част на прозореца, а в лявата част е нужно да се зададе или броят цикли или времето в секунди, като също се избира една от възможните стойности за тактова честота на микроконтролера. При натискане на бутон „Go“ се генерира програмен код, който може да се копира и постави в средата на AVR Studio.



Фиг. 1.4. Графичен интерфейс на AVR delay loop generator

3. Управление на светодиодната индикация на STK500

Нужно е да се избере кой от портовете на микроконтролера ще бъде използван за управление на светодиодите на развойната платка STK500. Изведените изводи на платката за избрания порт се свързват към изводите на светодиодите с помощта на 10-проводен лентов кабел с конектори (вижте документацията на STK500 и Лабораторно упражнение №1). За светването на избран светодиод е нужно към него да се изведе сигнал с ниско ниво, за да се отпуши свързаният транзистор и да протече ток през светодиода към него. Нужно е предварително съответният извод на

микроконтролера да бъде конфигуриран да работи с функцията на цифров изход. Това става като се запише „1“ в съответния бит от DDRx регистъра за порта. След това, за да светне съответният светодиод е нужно да се запише в регистъра PORTx „0“ в съответстващия му бит.

3.1. Примерна програма led1

Примерната програма настройва изводите на PORTB като цифрови изходи, след което извежда на младшите 4 адреса сигнал лог. „0“ (съответства на ниво GND), а в старшите 4 адреса сигнал лог. „1“ (съответства на ниво +5 V на изводите).

```
          .include "m8515def.inc"

          .org 0
          rjmp reset            ;преход към началния адрес $20 на програмата

          .org $20
reset:    ser R16               ;установява всички битове на R16 в „1“
          out DDRB, R16         ;зарежда DDRB с числото в R16
                                ; и конфигурира изводите на PORTB
                                ; като изходи
          ldi R16, 0b11110000   ; зарежда в R16 константата 0b11110000
          out PORTB, R16        ;записва стойността на R16 в PORTB

loop:     rjmp loop             ;реализира безкраен цикъл към етикет loop
```

3.2. Примерна програма Sum_Led

Програма за събиране на две 8-битови конкретни стойности и извеждане на сумата им в двоичен вид върху осем светодиода, свързани към Port B.

```
          .include "m8515def.inc"
          .org  0
          rjmp  reset
          .org  $20

reset:    ldi  r16, 0b11111111
          out  ddrb, r16        ;port B (direction OUTPUT)трябва да бъде
                                ;свързан към LEDs
          out  portb, r16       ;LEDs са тъмни (0 за светване на LED)
          ldi  r16, 0b10011011  ;първо число
          ldi  r17, 0b00100010  ;второ число
          add  r16, r17         ;сумиране -> резултата е в r16
          com  r16              ;инвериране на 1 и 0 в r16 понеже
          out  portb, r16       ;LEDs светят с 0 и изгасват с 1
always:   rjmp always           ;loop завинаги
```

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Свържете светодиодите на развойната платка към PortB.

2. Да се състави програмен код, чрез който да се светнат всички светодиоди.

3. Променете заданието от точка 2, така че светодиодите Led0, Led1 и Led2 да светят, а останалите да са изгасени.

4. Да се реализира задачата от точка 3, но с използване на изводите от PortC.

5. Да се състави програмен код, така че на извод PC0 на PortC, да се изведе сигнал с честота 1Hz (Упътване: използвайте програма AVR Delay Loop Generator).

6. Да се реализира задачата от точка 5 с използване на подпрограма (Упътване: Използвайте теоретичните указания от Лабораторно упражнение №3).

7. Да се измери сигнала на задачата от точка 5 с помощта на цифров осцилоскоп.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Какъв е общият брой регистри, които са асоциирани с работата с всички входно-изходните портове?

2. Какво число е нужно да се запише в DDRA регистъра на PORTA, за да се конфигурират младшите 4 извода като входни, а старшите 4 извода – като изходни?

3. Модифицирайте програмния код на примерната програма в т.3.2 по такъв начин, че да се извежда съдържанието на статусния регистър SREG към светодиодите.

Упътване: Разположете на подходящо място в програмата следната инструкция:

```
In  r16, sreg
```

Проверете решението си с компилиране и изпълнение на така модифицираната програма.

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 5
Сканиране на бутони и механични контакти

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да разучат принципа за програмна конфигурация с посока „вход" на паралелните входно-изходни изводи на микроконтролера (Port A, Port B, Port C, Port D и Port E) и програмни методи за филтриране на трептенията на механичните контакти в бутони и превключватели и за надеждно отчитане на позицията на контактите.

КРАТКА ТЕОРИЯ

1. Конфигуриране на входно-изходните портове с функция за цифрови входове

Използвайте теоретичната част от Лабораторно упражнение №4 и документацията на микроконтролер ATmega8515.

За конфигуриране на функцията за цифров вход на даден извод от избран порт x, в DDRx регистърът на този порт трябва в съответния му бит да бъде записана нула.

За да се гарантира прочитането на валиден сигнал от даден вход, когато той не е свързан към захранваща линия или цифров изход, добра практика е конфигурирането и активирането на съответния му изтеглящ резистор. По този начин в ситуации, когато не е свързан ще бъде прочитана лог. „1".

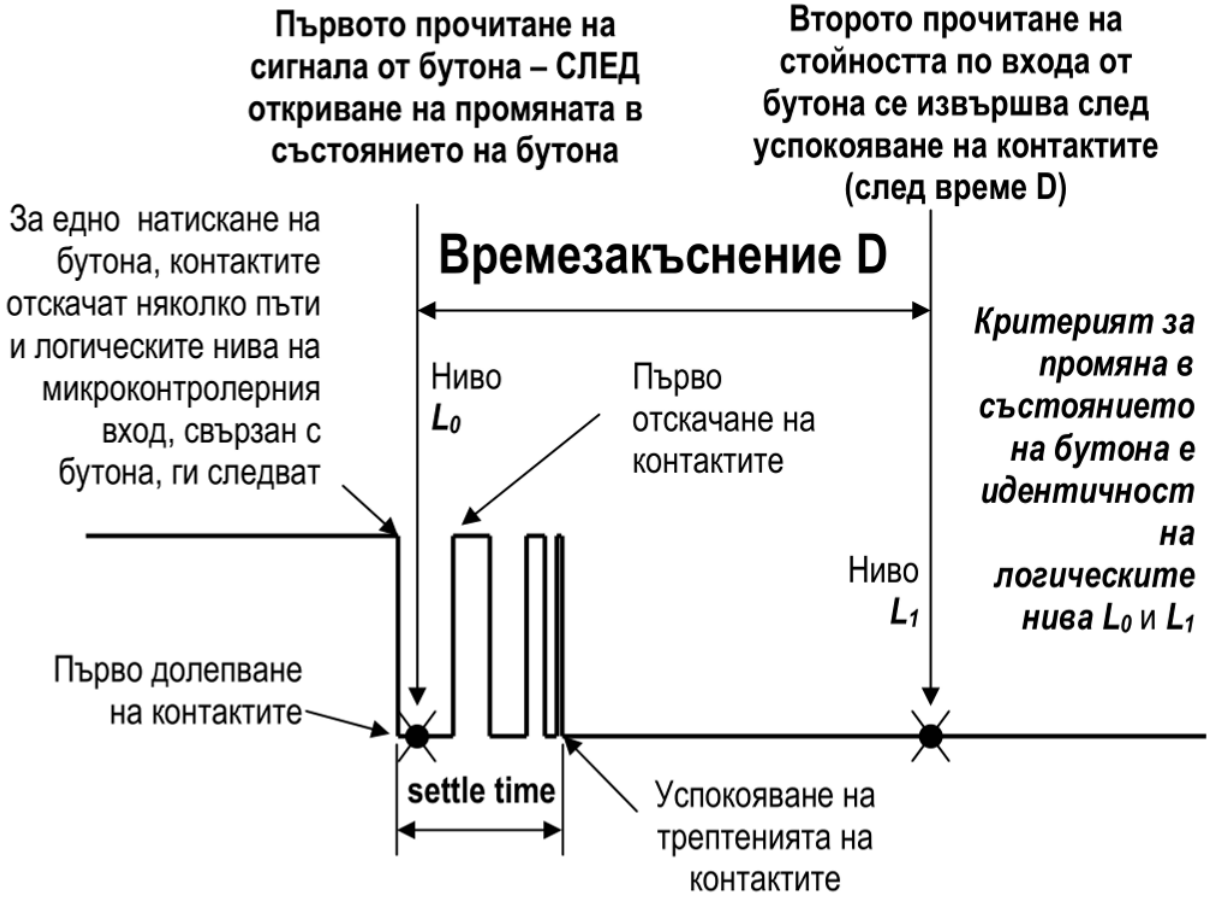
Ако в PORTxn е записана логическа единица, когато съответния извод е конфигуриран като вход изтеглящия резистор е активиран. За да се изключи изтеглящия резистор в PORTxn трябва да бъде записана лог."0" или съответния извод трябва да бъде конфигуриран като изход.

2. Явлението трептене на механичните контакти

Трептене на контактите на различни механични контактни системи (бутони, превключватели, контакти на релета и др.) се наблюдава при всяко от крайните състояния. При бутоните трептене има както след натискане, така и след отпускане. Изходният сигнал се установява в едно стабилно ниво след изтичането на определено време след събитието на преместване на контактния механизъм, което е познато като време за успокояване на контактите (settle time) – фиг. 5.1.

Обикновено времето за успокояване на контактите е от порядъка на **няколко милисекунди** за леки контактни системи (например бутони, монтирани върху печатна платка или такива с гумени контактни площадки) до **няколко десетки ms** за по-груби контактни системи като тези в крайни изключватели (машини с ЦПУ, асансьори), силови релета и

контактори в индустриални системи и др. Времето за успокояване на контактите след натискане на бутона **се различава** от това след неговото отпускане.



Фиг. 5.1. Времедиаграма на отчетения входен сигнал от бутон при неговото натискане

2.1. Филтриране на трептенията

Трептенето на контактите, ако не се филтрира по апаратен или по програмен начин, може да доведе до погрешни алгоритмични резултати. Ако една програма отброява колко пъти е натиснат даден бутон, тя е възможно при всяко натискане да отброява по няколко събития за натискане.

2.2. Избор на времезакъснението D

Времезакъснението D за прочитане на логическата стойност на входа от бутона, след като е била открита първата промяна в положението на контактите, зависи от механиката на бутонните контакти.

Филтрирането на трептенията по програмен път се осъществява чрез подходящ избор на времезакъснението D, след което се предполага с голяма вероятност, че прочетеното логическо ниво на входа от

микроконтролера, свързан с контакта, е именно установилото се значение за контактната система.

Величината на D се избира да бъде по-голяма от settle time със запас, за да се покрие неговото увеличаване, настъпващо със стареене на контактната система.

2.3. Алгоритъм за филтриране на трептенията на контактите

Алгоритъмът намира отговор на въпроса „Има ли наистина настоятелно натискане на бутона?“ За да бъде сигурно, че откриването на първата промяна в нивото на контактната система не е било вследствие на лъжливо трепване по случайни причини, вибрации и др. подобни. Алгоритъмът за филтриране на трептенията за откриване на едно настоятелно натискане на бутона може да се изгради по следния начин.

След като е било прочетено нивото (L0) на сигнала непосредствено след откритата промяна в положението на контактите, алгоритъмът продължава с изчакване на време D. След изтичане на това време се прочита отново нивото (L1) на сигнала. Нивата (L0) и (L1) се сравняват. Ако те са **идентични** (и двете са нула или и двете – единица) и **съвпадат** с приетото ниво за натиснат бутон, тогава се приема че е налице **настоятелно** натискане на бутона. Ако нивата са различни, откритата в началото промяна на положението на контактите се игнорира като случайно трепване на контактите.

Надеждността на тази констатация зависи пряко от избора на стъпката за диференциране D.

3. Програма за филтриране на трептенията на бутони в STK500

Приема се следното:

1. Бутоните ще се сканират през PortD, т.е. трябва да са свързани предварително с лентов кабел контактните рейки между PORTD и SWITCHES;

2. Светодиодите на STK500 ще се управляват през PORTB, т.е. трябва да се свържат с лентов кабел контактните рейки между PORTB и LEDS;

3. При натискане на бутон SW0 ще се инкрементира брояч, който ще се визуализира на светодиодите.

```
;*******************************************
; Програма за брояч, инкрементиращ при натискане на бутон SW0
; Двойна проверка за филтриране на трептенията (debouncing)
; Бутоните са свързани към PORTD ; Светодиодите са свързани към PORTB
; ATmega8515, тактова честота 4MHz
;*******************************************
          .include "m8515def.inc"   ; Включва дефинициите за ATmega8515
          .equ DEBOUNCE_DELAY = 200 ; Задаваме време за филтриране на
                                    ; трептенията
```

```
          .org 0x0000               ; Адрес на началото на програмата
          rjmp INIT                 ; Пренасочване към етикета INIT

;*******************************************
; Инициализация на портове и регистри
;*******************************************
INIT:
;зареждане на стековия указател с начален адрес
          ldi  r16, $5f         ;може да се използва израза low(ramend)
          out  spl, r16
          ldi  r16, $02         ; може да се използва израза high(ramend)
          out  sph, r16

          ldi r16, 0x00         ; Изчистваме регистър r16 (брояч)
          out DDRD, r16         ; PORTD като вход (бутони)
          ldi r16, 0xFF         ; Всички пинове на PORTB като изход
          out DDRB, r16         ; PORTB като изход (светодиоди)
          out PORTB, r16        ; Изчистваме светодиодите
          ldi r17, 0xFF         ; Рег. r17 пази старото състояние на бутона

MAIN_LOOP:
          in r18, PIND          ; Четем състоянието на PORTD (бутоните)
          sbrc r18, 0           ; Проверяваме дали SW0 (бит 0 на PORTD)
                                ; е натиснат
          rjmp MAIN_LOOP        ; Ако бутонът не е натиснат, връщаме се
                                ; в главния цикъл
          rcall DEBOUNCE        ; Извикваме дебоунс подпрограмата
          in r18, PIND          ; Прочитаме отново състоянието на PORTD
                                ; след дебоунс
          sbrc r18, 0           ; Проверяваме дали бутонът все още
                                ; е натиснат (SW0)
          rjmp MAIN_LOOP        ; Ако бутонът вече не е натиснат, връщаме се
                                ; в главния цикъл
          cp r18, r17           ; Сравняваме текущото състояние със старото
          breq MAIN_LOOP

; Ако бутонът е задържан (няма промяна), пропускаме инкрементацията
          inc r16               ; Инкрементираме брояча
          out PORTB, r16        ; Показваме стойността на брояча
                                ; върху светодиодите
          mov r17, r18          ; Запомняме новото състояние на бутона
          rjmp MAIN_LOOP        ; Връщане в главния цикъл

;*******************************************
; Подпрограма за филтриране на трептенията на бутоните (debouncing)
;*******************************************
DEBOUNCE:
          ldi r18, DEBOUNCE_DELAY   ; Зареждаме времето за дебоунс
DEBOUNCE_LOOP:
          dec r18               ; Намаляваме брояча
          brne DEBOUNCE_LOOP    ; Изчакваме докато не приключи
                                ; дебоунс периода
          ret                   ; Връщане от подпрограмата
```

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Свържете бутоните на развойната платка към PortD.

2. Да се състави програмен код със следното функционално действие:

- Натискането на бутон SW0 да светва светодиод Led4, а при отпускане да изгасва.

3. Променете програмата от точка 2, така че натискането на бутон SW0 да светва или изгасва светодиода Led4, а отпускането му да не променя състоянието на светодиода.

4. Да се състави програма реализираща двоичен брояч, който да се увеличава с натискане на бутон SW0, намалява с натискане на бутон SW1 и нулира с натискане на бутон SW2.

5. Свържете периферната платка с двупозиционни превключватели към PortD. Съставете програмен код със следното функционално действие:

- Да се преобразуват двоичните значения, задавани от превключвателите в шестнадесетичен код, който да се извежда върху светодиодната индикация.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Какъв е общият брой регистри, които са асоциирани с работата с един входно-изходен порт?

2. Какво е нужно да се направи, за да се конфигурират изтеглящите резистори на PORTB, така че да са включени само за младшите 4 извода. Също така младшите 4 извода да са конфигурирани за цифрови входове, а старшите – за цифрови изходи с изведено активно ниво лог. „0“?

3. Възможно ли е да се „спести“ повторното прочитане на входния сигнал от бутон в алгоритъма за филтриране на трептения, а да се разчита само на по-дълго времезакъснение? Тествайте идеята практически.

4. Помислете и предложете алтернативни алгоритми за програмно филтриране на трептенията на механични контактни системи. Опитайте се да ги реализирате практически и да тествате работоспособността им.

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 6
Запознаване с архитектурата, функционалните възможности и режими на работа на Таймер/броячен модул 0 в ATmega8515

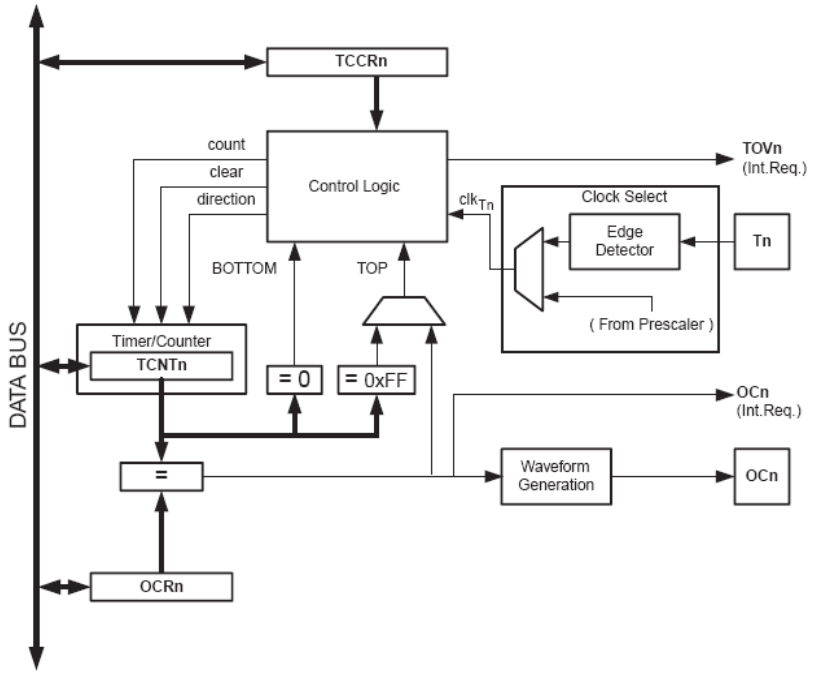
ЦЕЛ НА УПРАЖНЕНИЕТО

Целта е студентите да се запознаят с архитектурата, програмния модел и функционалните възможности на вградената в микроконтролера таймер/броячна система 0 и особености при конфигуриране.

КРАТКА ТЕОРИЯ

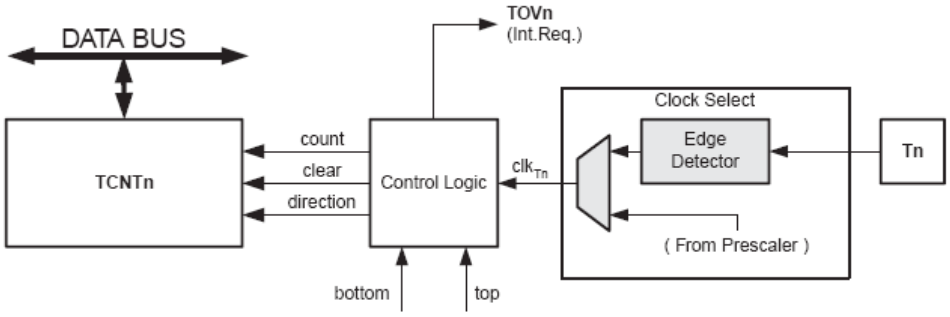
Таймерите са периферни устройства, с помощта на които се измерват различни времена. Това са едни от най-често използваните модули от един микроконтролер, тъй като в почти всяко приложение, по една или друга причина се налага измерване на предварително зададено време. Най-общо, всеки таймер представлява регистър, стойността на който се променя под въздействието на тактов сигнал. Колкото повече битове съдържа регистърът на даден таймер, толкова с по-голяма прецизност той ще измерва отделните времеинтервали. Освен това тези устройства могат да се използват за организиране на броячи. В този случай, ако се свърже сензор за движение към броячния вход може да се отчете броя на преминалите през обхвата на сензора обекти (например хора, различни предмети в даден производствен процес и. тн.).

1. Структура на таймер/броячния блок в микроконтролер ATmega8515



Фиг. 6.1. Блокова схема на таймер/брояч0

1.1. Броячен блок (Counter)



Фиг. 6.2. Схема на броячния блок (Counter)

Означения и предназначение на сигналите:

count- Инкрементира / декрементира с 1 регистър TCNT0.

direction- Избира между функциите инкремент / декремент;

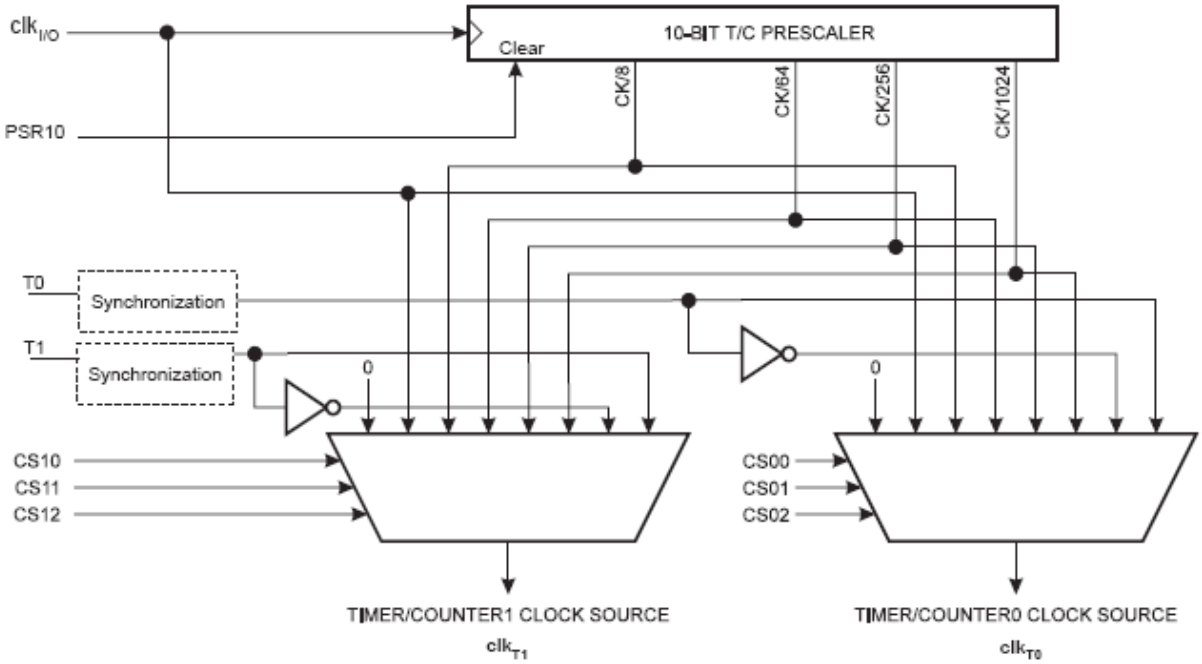
clear- Нулира регистър TCNT0;

clkTn- Тактов сигнала, обозначаван като clkT0;

top- Сигнализира, че TCNT0 е достигнал максималната стойност за броене;

bottom- Сигнализира, че TCNT0 е достигнал максималната стойност за броене (нула).

1.2. Блок „предделители" за Таймер0 и Таймер1



Фиг. 6.3. Схема на предделителния блок за двата таймер/брояча

Източникът на тактовия сигнал за броячния регистър (TCNT0) се управлява от трите бита CS02, CS01 и CS00 на управляващия регистър

(TCCR0). Тяхната комбинация по реда на старшинството им <CS02:CS01:CS00> избира източникът на тактов сигнал за регистъра TCNT0 както следва:

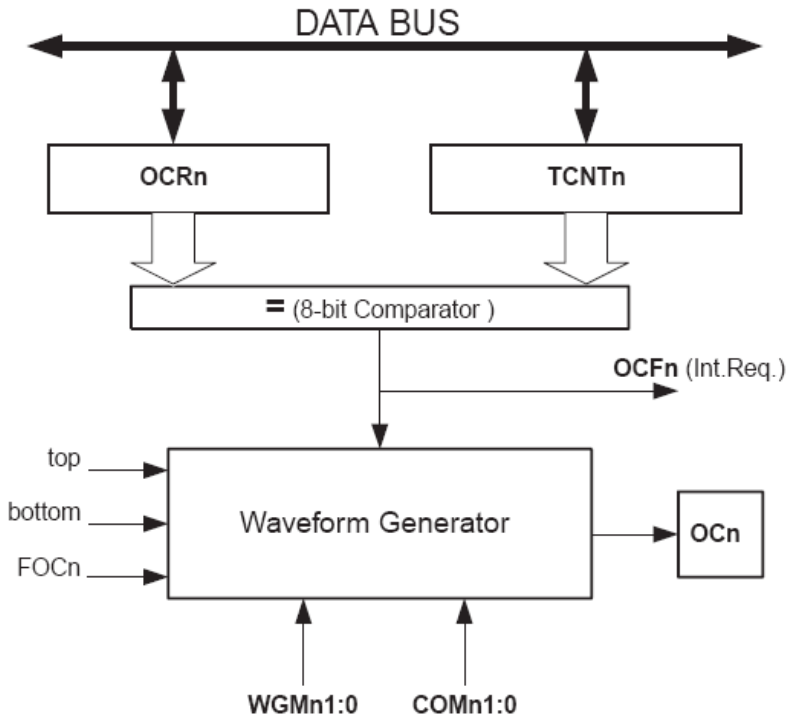
• към никакъв тактов сигнал (стоп на таймер/брояча) (за комбинация <000>),

• към вътрешния ТГ clkIO (коефициент на делене = 1) (за комбинация <001>),

• към коефициенти на делене 1/8, 1/64, 1/256, 1/1024 (за комбинации <010>, <011>, <100>, <101>).

• последните две комбинации могат да се използват за режим брояч за броене по падащите (за <110>) или по нарастващи фронтове (за <111>) на тактиращия сигнал от входния извод T0 на микроконтролера.

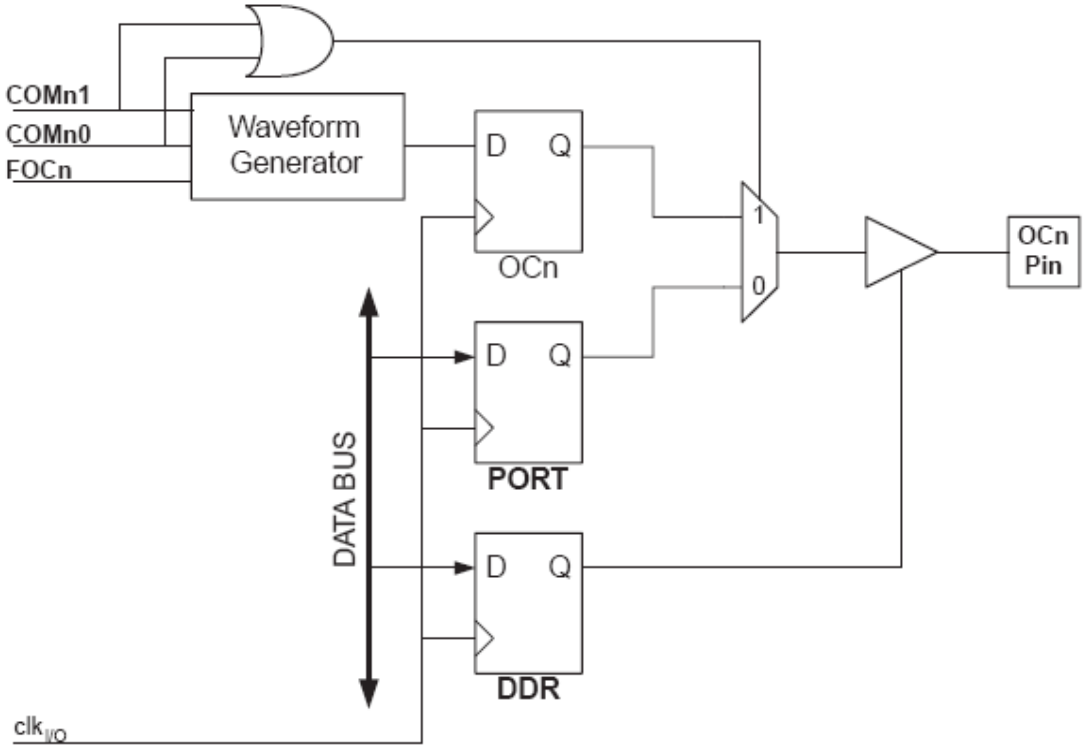
1.3. Блок „Изход от Сравняване” (Output Compare)



Фиг. 6.4. Схема на блок „Изход от Сравняване” (Output Compare)

Непрекъснатото следене на текущото съдържание на броячния регистър е функционалност, която намира приложение в широк кръг практически важни проблеми. **Функцията за следене** на текущото съдържание на броячния регистър се поддържа от вградена в чипа апаратна компараторна логика, чиято цел е да **сравнява непрекъснато** текущото съдържание на **броячния регистър** TCNT0 и това на отделен 8-битов регистър, наречен **компараторен** или **прагов** регистър OCR0 (от Output-Compare Match Register 0).

1.4. Блок „Изход при Съвпадение” (Output Compare Match)



Фиг. 6.5. Схема на блок „Изход при Съвпадение” (Output Compare Match)

Основната В/И функция на извода PB0 от PortB може да се подтисне от функцията на изхода от сравняването (OC0), осъществявана от генератора на формата (Waveform Generator), когато поне един от двата бита COM01:0 е програмиран в 1.

Важно: битът с мнемоника DDR от регистъра за посока на данните на PortB (Data Direction Register – DDRB), управляващ посоката на извод PB0, в конкретния случай OC0, трябва да бъде записан в 1, за да може да се разреши изходният драйвер с 3 състояния, за да се изгради изходящ тракт за сигнал OC0 до съответния извод OC0 на микроконтролера (извод с номер 1 на корпуса).

2. Режими на работа на Timer/Counter0

- NM: Нормален режим (Normal Mode);
- CTC: Нулиране на таймера при съвпадение след сравняването (Clear Timer on Compare Match);
- FPWM: Бърза ШИМ (Fast PWM);
- PCPWM: ШИМ с корекция на фазата (Phase Correct PWM); разликата с режим за бърза ШИМ е тази, че сигналът на таймера е триъгълен, а не трионообразен, както е при бърза ШИМ. По този начин таймер-броячния регистър, вместо да се нулира след преминаване през максималното си съдържание 255, както е при бърза ШИМ, преминава в

режим на изваждане и започва да се декрементира на всеки тактов импулс (254, 253, ..., 0), след което отново преминава в режим събиране (1, 2, 3, ...). Изходът OC0 не се променя при достигане на максималната си стойност 255, а преминава в противоположното си състояние при всяко изравняване на съдържанието на броячния регистър с това на компараторния регистър, независимо от посоката на натрупване на брояча (в плюс или в минус).

3. Конфигуриране на таймер/броячна система 0

Пет регистъра, асоциирани с конфигурацията на Таймер0:

TCCR0: Timer/Counter Control Register – Управляващ регистър;

TCNT0: Timer/Counter Register – Броячен регистър;

OCR0: Output Compare Register – Регистър за сравнение (компараторен);

TIMSK: Timer/Counter Interrupt Mask Register – Регистър с маските за прекъсване;

TIFR: Timer/Counter Interrupt Flag Register – Регистър с флаговете за прекъсване.

Програмиране на управляващия регистър на таймер/брояч0 – **TCCR0** – Timer/Counter Control Register (фиг. 6.6).

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Фиг. 6.6. Управляващ регистър TCCR0

Таблица 6.1. Избор на режим на работа

Режим	WGM01 (CTC0)	WGM00 (PWM0)	Режим на операцията	TOP	Обновяване на OCR0 при	TOV0
0	0	0	Нормален	0xFF	Непосредствено	MAX
1	0	1	ШИМ с корекция на фазата	0xFF	TOP	BOTTOM
2	1	0	Нулиране на таймера при съвпадение след сравняването (CTC)	OCR0	Непосредствено	MAX
3	1	1	Бърза ШИМ	0xFF	TOP	MAX

В бит 7 с наименование **FOC0** (Force Output Compare) трябва да се вписва винаги **0**, когато се избира режим за **някакъв вид ШИМ**. За

режими, които **не са ШИМ**, вписването на **1** в него генерира **строб** върху изход **OC0**. При четене на регистър TCCR0, този бит се чете винаги като 0.

Режимът на работа на таймер/броячната система се избира с битове 6 (**WGM01**) и 3 (**WGM0**) от управляващия регистър **TCCR0** на таймера както следва (таблица1):

Двата бита b5 и b4 с наименования **COM01** и **COM00** от регистър **TCCR0** имат функции, зависещи от режима на работа, избиран с битове **6** (**WGM01**) и **3** (**WGM00**).

За **нормален режим без ШИМ**, значенията на битове са както следва (таблица 6.2):

Таблица 6.2. Значения на COM01 и COM00 за **нормален режим без ШИМ**

COM01	COM00	Описание
0	0	Нормална входно-изходна операция за извод PB0. Функция OC0 е забранена.
0	1	Преобръщане на OC0 при съвпадение след сравняване.
1	0	Нулиране на OC0 при съвпадение след сравняване.
1	1	Установяване в единица на OC0 при съвпадение след сравняване.

За режим **бърза ШИМ**, значенията на битове са следните (таблица 6.3):

Таблица 6.3. Значения на COM01 и COM00 за режим **бърза ШИМ**

COM01	COM00	Описание
0	0	Нормална входно-изходна операция за извод PB0. Функция OC0 е забранена.
0	1	Резервирана
1	0	Нулиране на OC0 при съвпадение след сравняване, установяване в единица при TOP.
1	1	Установяване в единица на OC0 при съвпадение след сравняване, нулиране при TOP.

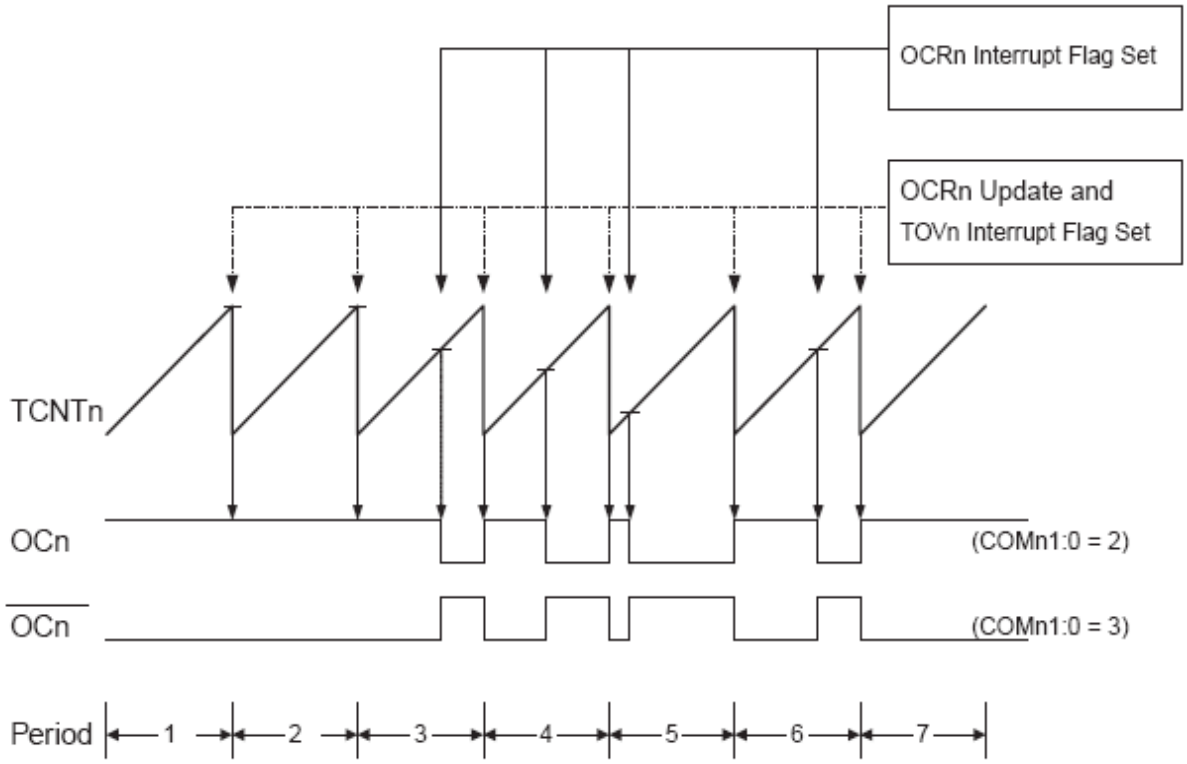
Забележка към таблица 6.3: Настъпва специален случай, когато стойността на OCR0 е равна на TOP и COM01 е установен в единица. В този случай, съвпадението след сравняване се игнорира, но установяването на OC0 в единица или нулирането се извършва при TOP.

Честотата на генериране на импулсите за режим бърза ШИМ се определя от формулата (6.1):

$$f_{OCnPWM}=f_{clk\_io} / N.256, \qquad (6.1)$$

където N е коефициентът на деление на честотата в предделителя (1, 8, 64, 256 или 1024).

На фиг. 6.7 е представена времедиаграмата на режим бърза ШИМ.



Фиг. 6.7. Времедиаграма на режим бърза ШИМ

За режим **ШИМ с корекция на фазата**, значенията на битове са следните (таблица 6.4):

Таблица 6.4. Значения на COM01 и COM00 за режим **ШИМ с корекция на фазата**

COM01	COM00	Описание
0	0	Нормална входно-изходна операция за извод PB0. Функция OC0 е забранена.
0	1	Резервирана
1	0	Нулиране на OC0 при съвпадение след сравняване когато брояча брои **нагоре**. Установяване в единица при съвпадение след сравняване когато брояча брои **надолу**.
1	1	Установяване в единица на OC0 при съвпадение след сравняване когато брояча брои **нагоре.** нулиране при съвпадение след сравняване когато брояча брои **надолу**.

Забележка към таблица 6.4: Настъпва специален случай, когато стойността на OCR0 е равна на TOP и COM01 е установен в единица. В

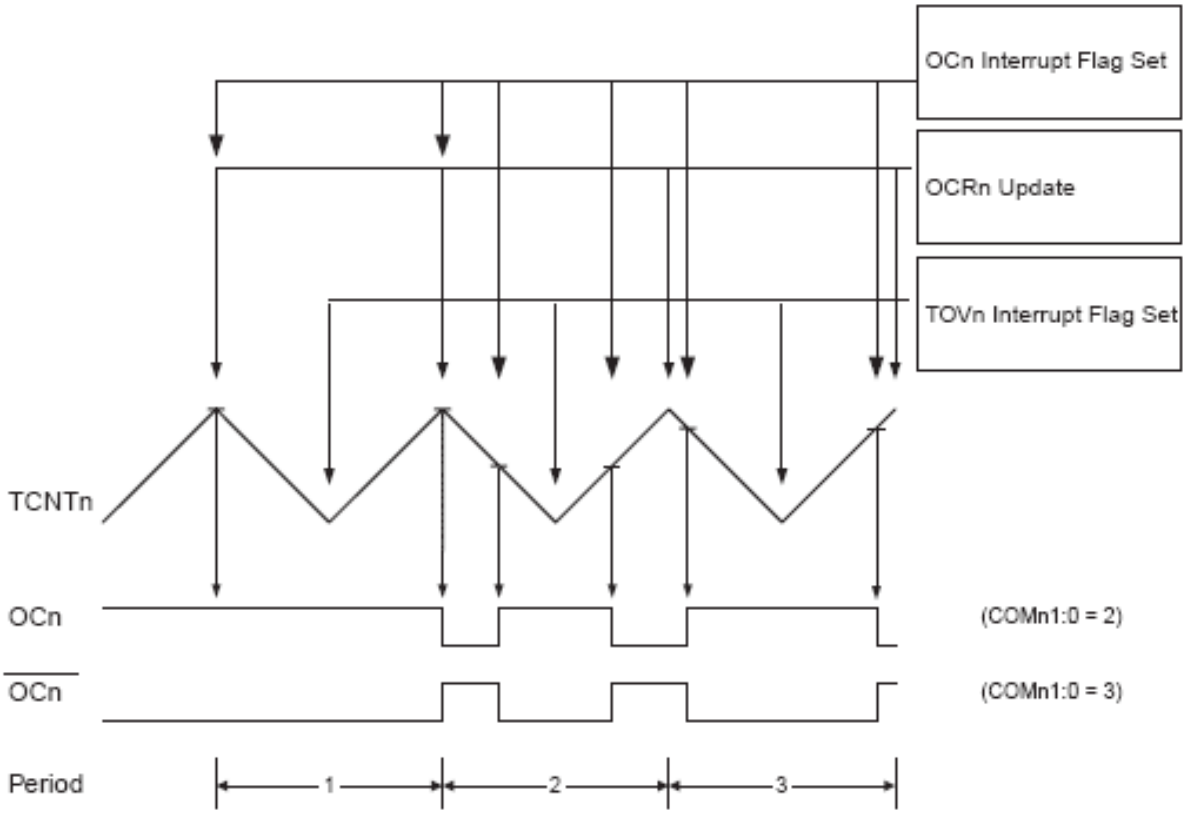
този случай, съвпадението след сравняване се игнорира, но установяването на OC0 в единица или нулирането се извършва при TOP.

Честотата на генериране на импулсите за режим ШИМ с корекция на фазата се определя от формулата (6.2):

$$f_{OCnPCPWM}=f_{clk\_io} / N.510, \quad (6.2)$$

където N е коефициентът на деление на честотата в предделителя (1, 8, 64, 256 или 1024).

На фиг. 6.8 е представена времедиаграмата на режим ШИМ с корекция на фазата.



Фиг. 6.8. Времедиаграма на режим ШИМ с корекция на фазата

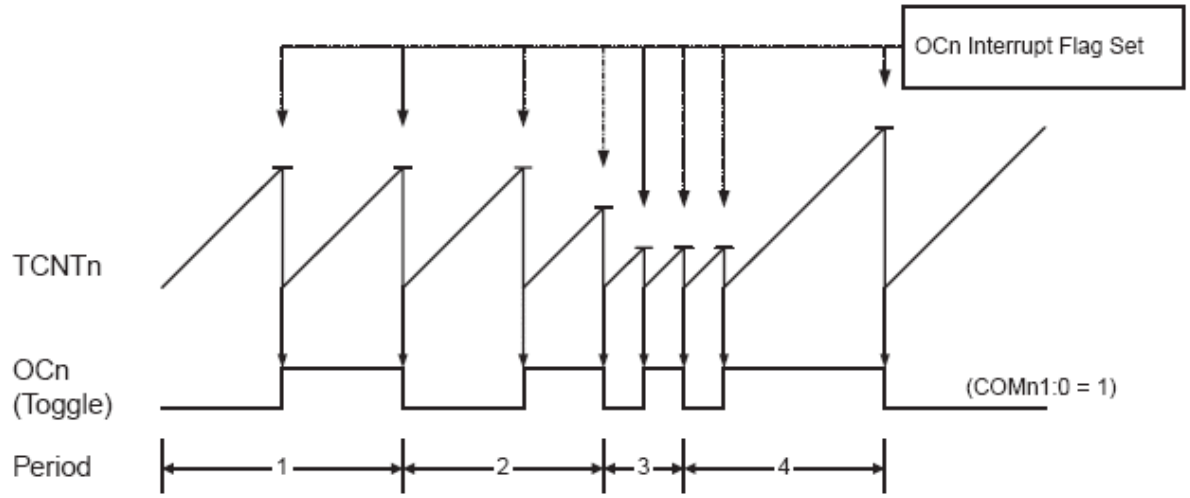
Режимът **нулиране на таймера при съвпадение след сравняването** (WGM01:0 = 2), компараторния регистър (OCR0) се използва за манипулиране на резолюцията на брояча. При този режим брояча се нулира със стойност нула, когато стойността на броячния регистър (TCNT0) съвпадне с тази на компараторния (OCR0). Стойността записана в компараторния регистър (OCR0) определя **top** стойността на брояча, оттам и неговата резолюция. Този режим позволява по-добър контрол на честотата на генериране на импулсите. Той също така опростява работата при преброяване на външни събития.

Честотата на генериране на импулсите за режим нулиране на таймера при съвпадение след сравняването се определя от формулата (6.3):

$$f_{OCn}=f_{clk\_io} / 2.N.(1+OCRn), \quad (6.3)$$

където N е коефициентът на деление на честотата в предделителя (1, 8, 64, 256 или 1024).

На фиг. 6.9 е представена времедиаграмата на режим нулиране на таймера при съвпадение след сравняването.



Фиг. 6.9. Времедиаграма на режим нулиране на таймера при съвпадение след сравняването

Изборът на тактов сигнал (външен или вътрешен) и на коефициента на делене се определя чрез програмиране на битове 2, 1 и 0 (CS02, CS01, CS00) (таблица 6.5):

Таблица 6.5. Избор на тактов сигнал (външен или вътрешен) и на коефициента на делене

CS02	CS01	CS00	Тактов сигнал за Таймер / Брояч 0
0	0	0	Стоп на таймера и на брояча (Липса тактов сигнал)
0	0	1	$clk_{I/O}$ /1 (Без делене на тактовата честота)
0	1	0	$clk_{I/O}$ /8 (От предделителя)
0	1	1	$clk_{I/O}$ /64 (От предделителя)
1	0	0	$clk_{I/O}$ /256 (От предделителя)
1	0	1	$clk_{I/O}$ /1024 (От предделителя)
1	1	0	Режим БРОЯЧ: Външен тактов сигнал от входен извод T0;
1	1	1	Режим БРОЯЧ: Външен тактов сигнал от входен извод T0;

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се състави програмен код, чрез който таймер-брояч0 да се конфигурира по следния начин:
   - „Нормален” режим;
   - Коефициент на деление 8;
2. Да се състави програмен код, така че функционалността на таймер-броячната система да се конфигурира както следва:
   - Режим “нулиране на таймера при съвпадение след сравняването”
   - Установяване в единица на извод OC0 при съвпадение след сравняване”
   - Коефициент на деление 64;
   - Прагова стойност в регистър OCR0=76.
3. Да се състави програмен код, чрез който таймер-брояч0 да се конфигурира по следния начин:
   - Режим “ШИМ с корекция на фазата”;
   - Нулиране на извод OC0 при съвпадение след сравняване;
   - Коефициент на деление 256;
   - Прагова стойност в регистър OCR0=56.
4. Да се състави програмен код, така че функционалността на таймер-броячната система да се конфигурира както следва:
   - Режим “Бърза ШИМ”;
   - Установяване в единица на извод OC0 при съвпадение след сравняване;
   - Коефициент на деление 1024;
   - Прагова стойност в регистър OCR0=92.
5. Да се пресметне продължителността на получения времеви интервал при програмната конфигурация от точка 1, за задачата от точка 2 да се пресметне честотата на сигнала изведен на извод OC0, а за задачите от точки 3 и 4 да се пресметне честотата и коефициента на запълване на сигнала изведен на извод OC0. Да се измерят и сравнят получените теоретични параметри на сигнала за всяка задача с помощта на цифров осцилоскоп.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Колко и какви са възможните режими на работа на таймер/броячен модул 0?
2. Коя комбинация на управляващите битове SC02, CS02 и CS00 предизвиква изключване на таймерната система?
3. Какво означава понятието широчинно-импулсна модулация (ШИМ), на английски Pulse Width Modulation (PWM)?

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 7
Система за прекъсвания в микроконтролер ATmega8515 и обработка на прекъсвания от таймер/брояч 0

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта е студентите да се запознаят със системата за прекъсвания на микроконтролера и да овладеят метода за разрешаване, прехващане и обработка на прекъсвания от таймер/брояч 0.

КРАТКА ТЕОРИЯ

1. Прекъсвания

Прекъсването е временно преустановяване на изпълнението на текущата програма (инструкция) по някаква причина, най-често по сигнал от настъпило апаратно събитие, с автоматично съхраняване на контекста (адреса за връщане, флаговия (статусния) регистър и др.) и преход към предварително подготвен манипулатор (процедура) – по един за всяка причина (заявка) за прекъсване.

Последната инструкция в манипулатора е обикновено инструкцията за връщане от прекъсване – RETI за ATmega8515, с която управлението се предава към прекъснатата програма в точката на нейното прекъсване (оригиналният контекст от точката на прекъсване се възстановява от инструкцията RETI).

Започване на изпълнението на манипулатора за прекъсване се нарича "възприемане на (заявката за) прекъсване".

Изпълнението на програмата-манипулатор се вмъква между две инструкции от основната програма и удължава времето за изпълнението й.

Моментът на възприемане на заявката за прекъсване е асинхронен по отношение на събитията в хода на изпълнение на основната програма.

1.1. Верига от събития в един процесор при възприемане на прекъсвания

Апаратният автоматизъм за възприемане от процесора на заявка за прекъсване включва следните действия:

1. Завършва се изпълнението на текущата инструкция;
2. Следва съхраняване на процесорния статус в момента на прекъсване:

- Микроконтролер ATmega8515 вписва автоматично в стека съдържанието на прграмния брояч (PC); последният съдържа адреса на инструкцията, следваща текущата (тя може и да НЕ Е съседната!); този

единствен запис съставлява в ядро AVR ATmega8515 процесорния статус в момента на прекъсване;

- Микроконтролер ATmega8515 не вписва в стека статусния регистър SREG!

Други процесори обикновено правят това. Съхраняването на SREG е задача на програмиста, който кодира манипулатора за обработка на прекъсването. Ето един примерен подход:

```
          MANIP_Int:            ; име на манипулатора
          in r23, SREG          ; съхраняване на статусния регистър
; тяло на манипулатора, което не разрушава регистъра r23 – копие на SREG!!!
          out  SREG, r23        ; възстановяване на статусния регистър
          reti                  ; последната инструкция във всеки
                                ; манипулатор
```

3. Флагът за разрешаване на прекъсванията I в статусния регистър SREG се нулира апаратно след възприемане на прекъсване; последствието е важно – при влизане в манипулатора прекъсванията са маскирани и процесорът не може да бъде прекъсван;

По-късно, по време на изпълнението си, последната в манипулатора инструкция за връщане от прекъсване RETI (RETurn from Interrupt) установява в 1 бит I автоматично, като по този начин разрешава възприемане на последващи заявки за прекъсвания след изпълнение на манипулатора.

4. По номера на прекъсването k, съответстващ на причината за заявка, процесорът определя адреса на съответния манипулатор за обработка, като използва данните от ред k в таблица IVT.

- Програмният брояч се зарежда с АДРЕСА, вписан във вход k от таблица IVT, а в него ОС е вписала инструкция rjmp за преход към първата инструкция от съответния манипулатор за прекъсване;

5. Изброените действия се изпълняват автоматично на апаратно ниво.

Макар и бързи, тези действия изискват определено време, известно като "Латентност за превключване от фоновата програма към манипулатор за обслужване на прекъсване".

1.2. Латентност за превключване на контекста

Латентността за превключване от фоновата програма към 1-та инструкция от манипулатора за обработка на прекъсване k съставлява в микроконтролер ATmega8515 повече от 7 периода на тактовия генератор и включва:

1. Времето до завършване на текущата инструкция (то е случайна величина),

2. 4 тактови периода за съхраняване в стека:

- на "адреса за връщане", т. е. адреса на инструкцията след, на която ще се предаде управлението, след като се изпълни и бъде завършен манипулаторът за обработка на прекъсването;
- за зареждане на програмния брояч с адреса на вход k във векторната таблица IVT;

3. 3 тактови периода за изпълнение на инструкция rjmp, която се разполага на адрес k в таблица IVT.

2. Номера, причини и адреси за прекъсване в микроконтролер ATmega8515

Векторите за рестартиране и прекъсване заемат в ATmega8515 първите 17 начални адреси в програмната памет Flash, т.е. в адресното пространство на програмата. Това са 17-те адреса от $0 до $10 включително.

Те са представени в таблица 7.1, известна като IVT (Interrupt Vector Table). За всеки вектор на прекъсване е определен свой уникален адрес на началото на манипулатора (програмата) за неговата обработка. Това е адресът на първата инструкция, с която манипулаторът започва.

Тъй като в таблица IVT са резервирани само по един адрес за манипулатор, а манипулаторите в общия случай заемат повече от един адрес, то горната таблица се попълва с инструкции за преход от типа rjmp към "истинското" начало на съответния манипулатор.

Съдържанието на таблица IVT се изгражда от ОС или от системния програмист през етапа на инициализация на микрокомпютъра, когато прекъсванията не са още разрешени.

Всяка възможна причина (събитие, функционален блок) за прекъсване може да бъде подтисната по програмен начин така, че да не генерира прекъсване или, обратното – да го генерира, чрез записване на 0, съответно на 1, в определен обособен бит от специален регистър за разрешаване на прекъсванията от нейна страна.

Таблица 7.1. Векторна таблица на прекъсванията в микроконтролер ATmega8515

№ на вектор	Адрес в паметта	Източник/събитие	Условие за възникване на прекъсването
0	$0000	RESET (рестартиране, или НУ – начално установяване)	От външен извод или ***НУ*** при включване на захранването, откриване на ***brown-out reset*** (пропадане на захранването), ***watchdog reset*** (НУ от стражевия таймер)
1	$001	INT0	Външна заявка по обособен извод за прекъсване 0

2	$002	INT1	Външна заявка по друг извод за прекъсване 1
3	$003	TIMER**1** CAPT	Прехващане на събитие от
4	$004	TIMER**1** COMPA	Съвпадение при сравнение с Таймер/Брояч 1 по канала му A
5	$005	TIMER**1** COMPB	Съвпадение при сравнение с Таймер/Брояч 1 по канала му B
6	$006	TIMER**1** OVF	Препълване на Таймер/Брояч 1
7	$007	TIMER0 OVF	Препълване на Таймер/Брояч 0
8	$008	SPI, STC	Завършване на обмена на данни по серийния интерфейс SPI
9	$009	USART, RXC (Universal Synchronous-Asynchronous Receiver	Приемането в УСАПП завършено (УСАПП - Универсален Синхронен и Асинхронен Приемник/Предавател)
10	$00A	USART	Регистъра за данни в УСАПП е празен
11	$00B	USART, TXC	Предаването през УСАПП е завършено
12	$00C	ANA_COMP	Аналогов компаратор
13	$00D	INT2	Външна заявка за прекъсване 2
14	$00E	TIMER0 COMP	Съвпадение при сравнение с
15	$00F	EE_RDY	EEPROM е в готовност
16	$010	SPM_RDY	Готовност за запис в програмната памет чрез инструкция SPM

Ако заявките за прекъсване от някакъв източник са забранени, то е без значение съдържанието на думата по адреса на неговия вектор в таблица IVT. Но ако заявките от даден източник k за прекъсване за разрешени, то е задължително думата по адреса на неговия вектор k в таблица IVT да съдържа инструкция rjmp към началото на съответния манипулатор k.

За да могат заявките от различни източници на прекъсване да се възприемат от процесора и съответстващите им манипулатори да се обработват, са необходими разрешения на две нива:

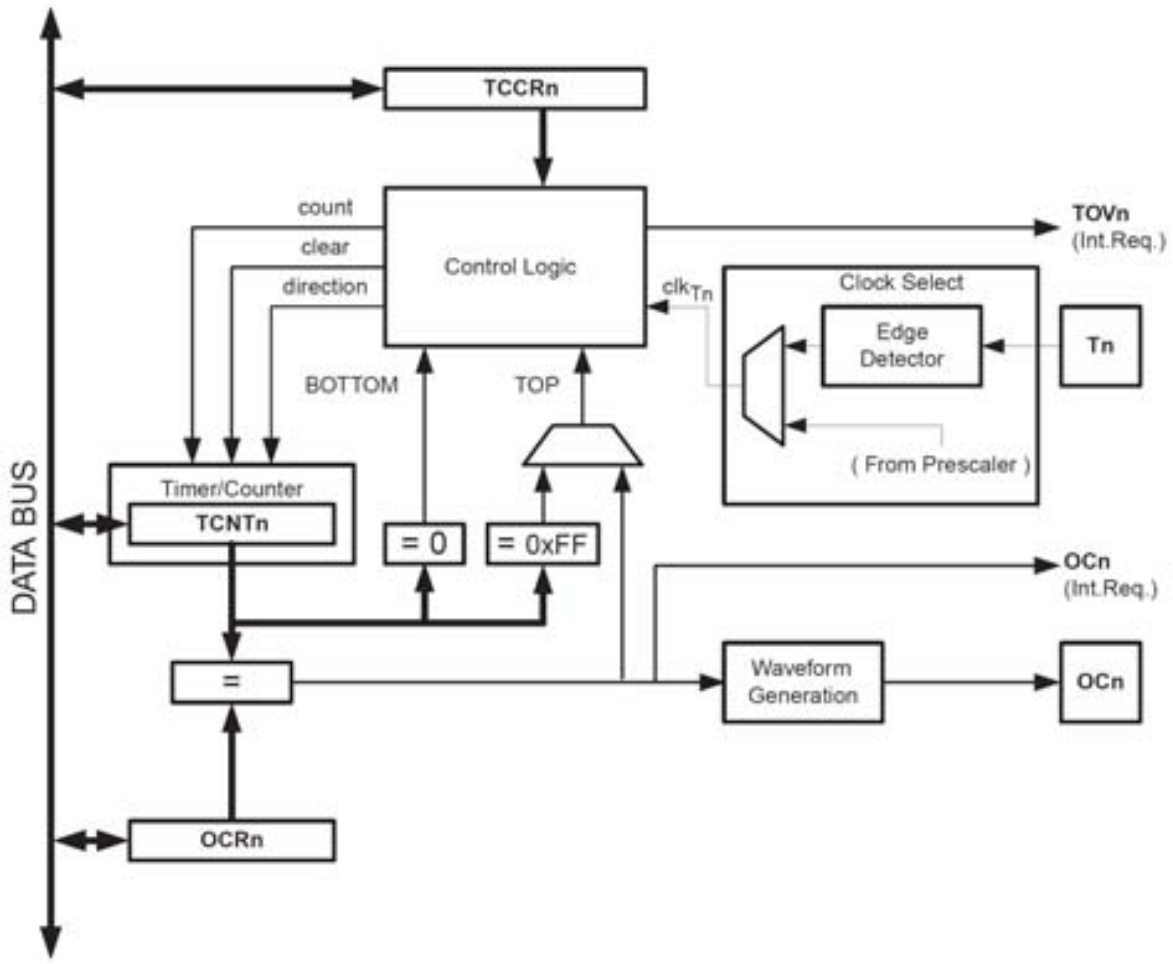
На глобално ниво (ниво на процесорното ядро): чрез установяване в 1 на бит I от статусния регистър SREG, с което се разрешават заявките за прекъсване на всички източници, чиито заявки за прекъсване са разрешени и на локалните си нива:

- В ATmega8515 инструкция **SEI** (от SEt I) установява в 1 бит I в статусния регистър SREG, с което разрешава прекъсванията на глобално ниво;
- Инструкция **CLI** (от CLear I) нулира бит I, с което забранява всички прекъсвания, в т. ч. и онези, които са разрешени на локалните си нива.

На локално ниво (на ниво функционален блок): чрез програмиране на техни обособени битове за разрешаване/забрана на прекъсванията, разположени в управляващи регистри, специализирани за разрешаване на прекъсванията.

3. Причини за прекъсвания от 8-битовата таймер/броячна система

Причините, източниците, събитията за генериране на прекъсвания от 8-битовата таймер-броячна система са две: съвпадение при сравняване и препълване на таймер/брояч0 (фиг. 7.1 и таблица 7.2).



Фиг. 7.1. Блокова схема на таймер/брояч0

Сигналът за заявките за прекъсване (съкратен с ***Int.Req*** на фигура 1.) е видим в 8-битовия флагов регистър за прекъсванията от таймер/брояч0 (Timer Interrupt Flag Register (TIFR)). Всички прекъсвания се маскират индивидуално чрез съответни битове от 8-битовия управляващ регистър на маските (Timer Interrupt Mask Register (TIMSK).

Таблица 7.2. Вектори на прекъсване от Таймер/брояч 0

№ на вектор	Адрес в паметта	Източник/събитие	Условие за възникване на прекъсването
7	$007	TIMER0 OVF	При препълване на Таймер/Брояч 0
14	$00E	TIMER0 COMP	Съвпадение при сравнение на Таймер/Брояч 0

3.1. Масков регистър на прекъсванията от таймер/броячни системи 0 и 1 TIMSK

Битовете за разрешаване на всяко от тези две прекъсвания на **локално ниво** в тази подсистема са групирани в регистър с маски за прекъсване – **TIMSK** (Timer/Counter Interrupt Mask Register). Тези битове са:

Бит 0 в TIMSK, с наименование **OCIE0** (Output Compare Inetrrupt Enable 0), разрешава (**когато в него се впише 1**) прекъсванията при съвпадение след сравняване в таймер/брояч 0; всяка такава заявка за прекъсване кара процесорът да предаде управлението към вектор k =7 от таблица IVT, където програмистът е трябвало да впише съответна инструкция **rjmp**;

Бит 1 в TIMSK, с наименование **TOIE0** (Timer Overflow Inetrrupt Enable 0) разрешава (**когато в него се впише 1**) прекъсванията при препълване на брояча (преминаване през стойност 255); всяка такава заявка за прекъсване кара процесорът да предаде управлението към вектор k =14 от таблица IVT, където програмистът е трябвало да впише съответна инструкция **rjmp**.

3.2. Реакция на uCU в случай на съвпадение при сравняване

Реакцията на uCU в случай на съвпадение при сравняване е следната:

- Флаг OCF0 (бит 0) от флаговия регистър на прекъсванията TIFR се вдига в логическа 1;
- Ако в същото време ПРЕКЪСВАНИЯТА по тази причина (по това събитие) са разрешени, което е трябвало да бъде направено чрез:
  - ❖ вписване на логическа 1 в бит OCIE0 от масковия регистър TIMSK за маскиране на прекъсванията от таймерите и в същото време

❖ бит I в процесорния статусен регистър е в логическа 1,

То събитието “Compare Match” подава заявка за прекъсване към процесора и той прави преход към съответния манипулатор за неговата обработка. Като резултат флаг OCF0 се нулира автоматически.

- В случай, че прекъсването е било маскирано, флаг OCF0 може да се нулира програмно по необикновен начин – чрез запис на 1 (!!!) в него.

3.3. Реакция на uCU в случай на препълване на таймер/брояч0

Реакцията на uCU в случай на препълване на таймер/брояч0 е следната:

- Флаг TOV0 (бит 1) от флаговия регистър TIFR се вдига в логическа 1;
- Ако в същото време ПРЕКЪСВАНИЯТА по тази причина (по това събитие) са разрешени, което е трябвало да бъде направено чрез:

❖ вписване на логическа 1 в бит TOIE0 от масковия регистър TIMSK за маскиране на прекъсванията от таймерите и в същото време

❖ бит I в процесорния статусен регистър е в логическа 1,

То събитието “Timer Overflow” подава заявка за прекъсване към процесора и той прави преход към съответния манипулатор за неговата обработка. Като резултат флаг TOV0 се нулира автоматически.

- В случай, че прекъсването е било маскирано, флаг TOV0 може да се нулира програмно по необикновен начин – чрез запис на 1 (!!!) в него.

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се състави програмен код реализиращ генератор на правоъгълни импулси с период T≈1s и коефициент на запълване 50% с използване на прекъсване от таймер/брояч 0 по препълване при тактова честота на процесорното ядро и таймер-брояча F_{cpu}=3.6786MHz. Изведете сигнала на извод PB0 микроконтролера. Пресметнете теоретично периода на сигнала;
2. С използване на цифров осцилоскоп да се измерят параметрите на сигнала за задачата от точка 1;
3. Съставете програмен код реализиращ двоичен брояч, с извеждане на моментната натрупана стойност върху светодиодната индикация с използване на прекъсване от таймер/брояч 0 по препълване;
4. Да се състави програмен код, реализиращ ефекти върху светодиодната индикация с възможност за избор с натискане на бутон SW0;
5. Реализирайте задачата за управление на извод OC0 с прекъсване по препълване от таймер 0 и съвпадение след сравняване.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Колко и какви са източниците на прекъсвания в ATmega8515?

2. Ако има разрешени прекъсвания от таймер0 при препълване, при записване на комбинация на управляващите битове SC02, CS02 и CS00 в управляващия регистър на таймер0, ще се изпълни ли поне един път подпрограмата (манипулатора) за обработка на прекъсвания от таймер0?

3. Възможно ли е при ATmega8515 да се обработват едновременно (чрез вложени прекъсвания) две различни събития, за които има активирани прекъсвания и обработващи подпрограми? Какво е нужно да се направи, за да е възможно да се обработи ново прекъсване по време на обработката на вече възникнало прекъсване?

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 8
Управление на седемсегментен дисплей

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да разучат методите за управление на седем сегментен дисплей чрез микроконтролер и съответното програмно осигуряване.

КРАТКА ТЕОРИЯ

1. Методи за управление на седем сегментен дисплей чрез едночипов микроконтролер

Въпреки, че управлението на седем сегментните дисплеи е по-лесно в сравнение с това на LCD индикаторите, свързването (и управлението) им към (чрез) едночипов микроконтролер създава някои проблеми.

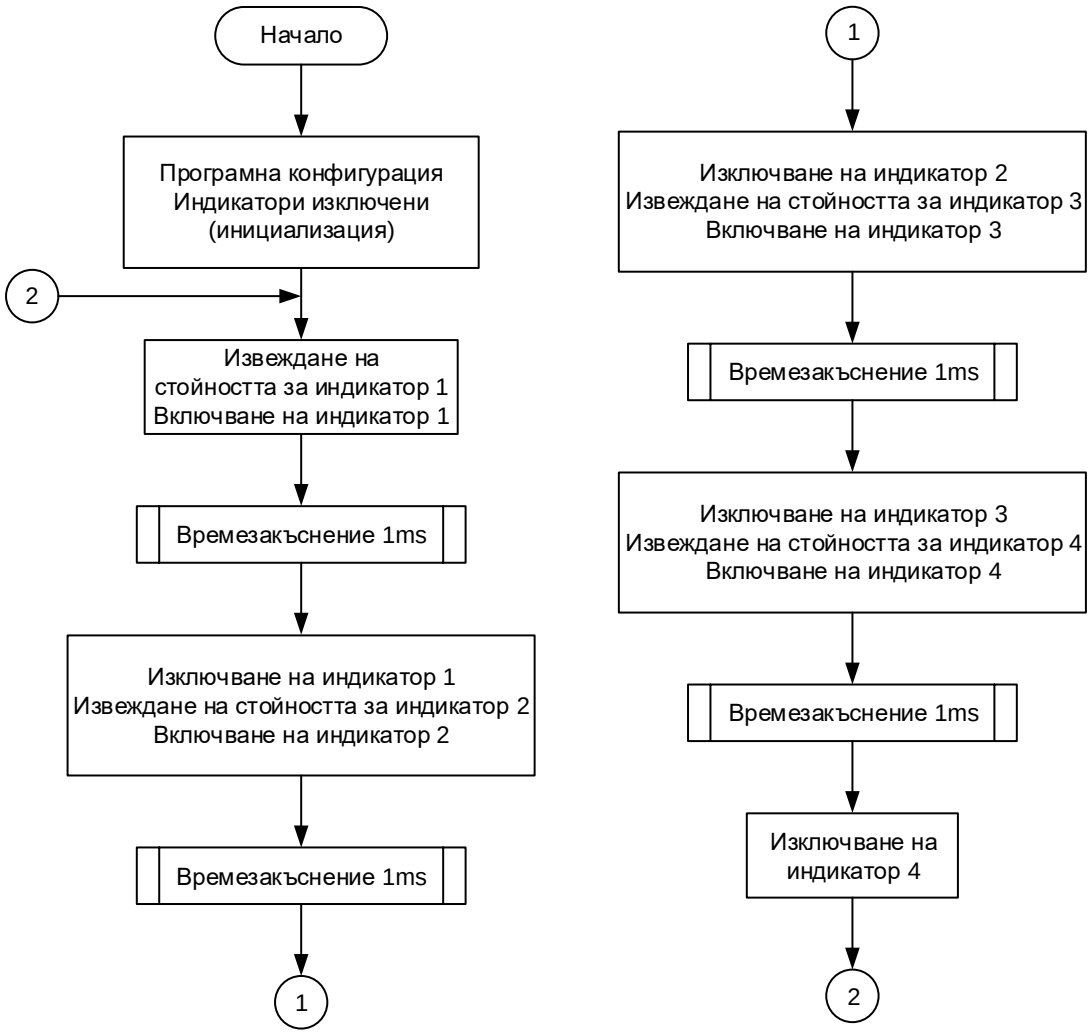
1.1. Статичен метод

Този метод се харакреризира с това, че всеки сегмент на индикаторите се включва към отделен извод на микроконтролера, а общите аноди (или катоди) се свързват директно към положителния (отрицателния) полюс на захранването (статично управление). От това следва, че за захранването на седемте сегмента (заедно с десетичната точка) на един индикатор са необходими осем свободни входно-изходни извода от микроконтролера. По този начин броят на индикаторите води до значителен разход на входно-изходни ресурси. Това означава, че броят на индикаторите може да се увеличи по два начина. Единият е с използване на микроконтролер с толкова на брой входно-изходни изводи, колкото са индикаторите плюс всички необходими за покриване на функционалните нужди на конкретната задача. В случая, управлението на индикаторите се състои в зареждане на необходимите двоични константи на всеки от изводите в зависимост от числото, което трябва да се изобрази. Така съдържанието на изводите ще остава непроменено, докато не се наложи промяна в показанието на някой от индикаторите. Този метод поради необходимостта от сравнително голям брой изводи, не е особено ефективен и сравнително рядко се използва.

1.2. Динамичен метод с използване на програмно времезакъснение

За да се увеличи броя на индикаторите без разход на входно-изходни ресурси е необходимо използването на другия начин за свързване на седем сегментен дисплей към микроконтролер известен още като динамично управление. Този метод се основава на инертността на човешкото зрение и е сходен с принципа за получаване на неподвижно изображение в телевизионните приемници. Отнесен към управлението на

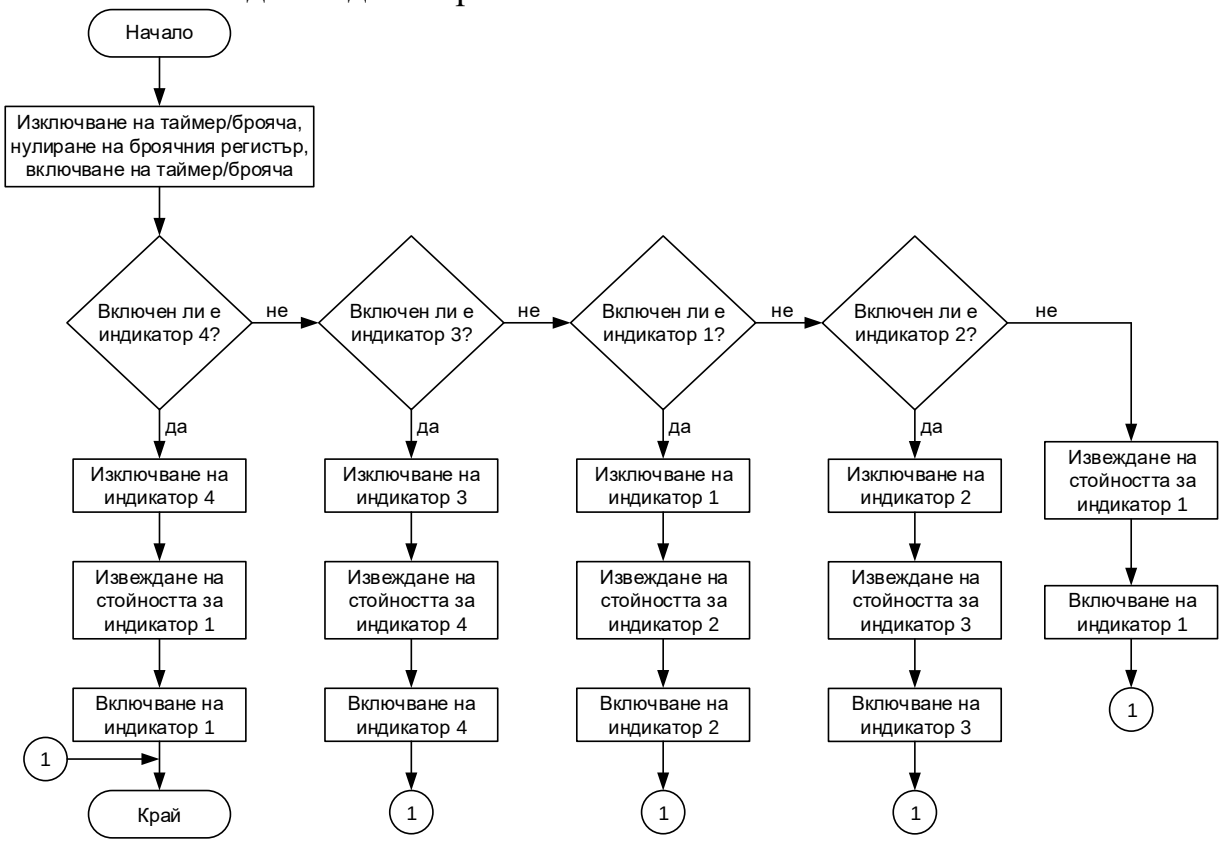
светодиодните индикатори, той се състои в свързването на сегментите на всеки от индикаторите към обща шина и последователно обхождане (опресняване) на всеки от тях. Времето, за което даден индикатор е изгасен, трябва да е по-малко от времето за задържане на възприятието. В общия случай, добре е това време да е по-малко или равно на 20ms. По този начин, ако се управляват четири индикатора, в определен момент от времето ще свети само един от тях, но резултатното изображение за наблюдателя ще бъде неподвижно четирицифрено число. Този метод е удобен от гледна точка на това, че използва по-малко входно-изходни ресурси, но за сметка на процесорни ресурси (процесорно време) необходими за превключване на всеки индикатор. На фиг. 8.1 е представен алгоритъма за динамично управление на четири индикатора. Превключването на всеки индикатор се извършва чрез програмно времезакъснение от 1ms. По този начин процесорното време не се използва ефективно заради обработката на вложените цикли необходими за реализиране на времезакъснението.



Фиг. 8.1 Алгоритъм за управление на седем сегментен дисплей чрез програмно времезакъснение

1.3. Динамичен метод с използване на таймер/брояч

За да се постигне по-голямо бързодействие е необходимо превключването да не се извършва с програмно времезакъснение, а чрез другия известен начин, който се основава на използването на един от вградените в даден микроконтролер таймер-броячи (фиг. 8.2). Обхождането на четирите индикатора се извършва чрез обслужване на прекъсване от таймер-брояча и „край“ в алгоритъма представлява връщане към главната програма. Предимството на този начин е, че процесора се разтоварва от тежката задача, свързана с измерване на времето от 1ms, тъй като с това е ангажиран таймер-брояча. Недостатъка е, че се използват ресурсите на таймер-брояча. Освен това се изразходва и процесорно време необходимо, както за обслужване на прекъсването, така и за цикъла, чрез който се обхождат индикаторите.

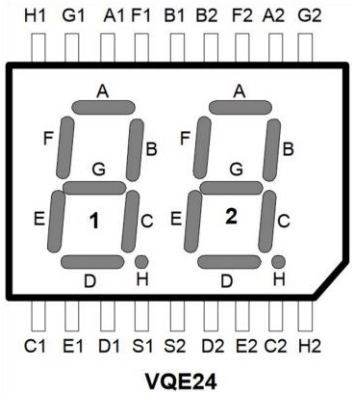


Фиг. 8.2. Алгоритъм за управление на седем сегментен дисплей чрез апаратно времезакъснение с обслужване на прекъсване от таймер-брояч

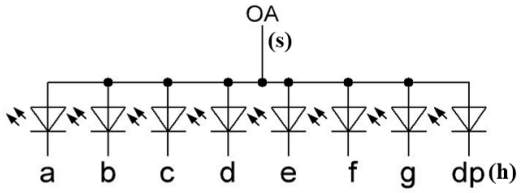
2. Схема на свързване на модул 7-сегментни индикатори към ATmega8515

На фиг. 8.3в е представена схемата и външния вид (фиг. 8.3г) на разработен модул с 4 броя 7-сегментни индикатори с общ анод (базирани на VQE24), която е предназначена за динамичен метод за извеждане. Схемата на свързване на сегментите (светодиодите, фиг. 8.3b) е Общ Анод (ОА), където S1 е ОА на първия индикатор (1), а S2 е ОА на втория индикатор (2). Максималният ток през сегмент (светодиод) е 20mA.

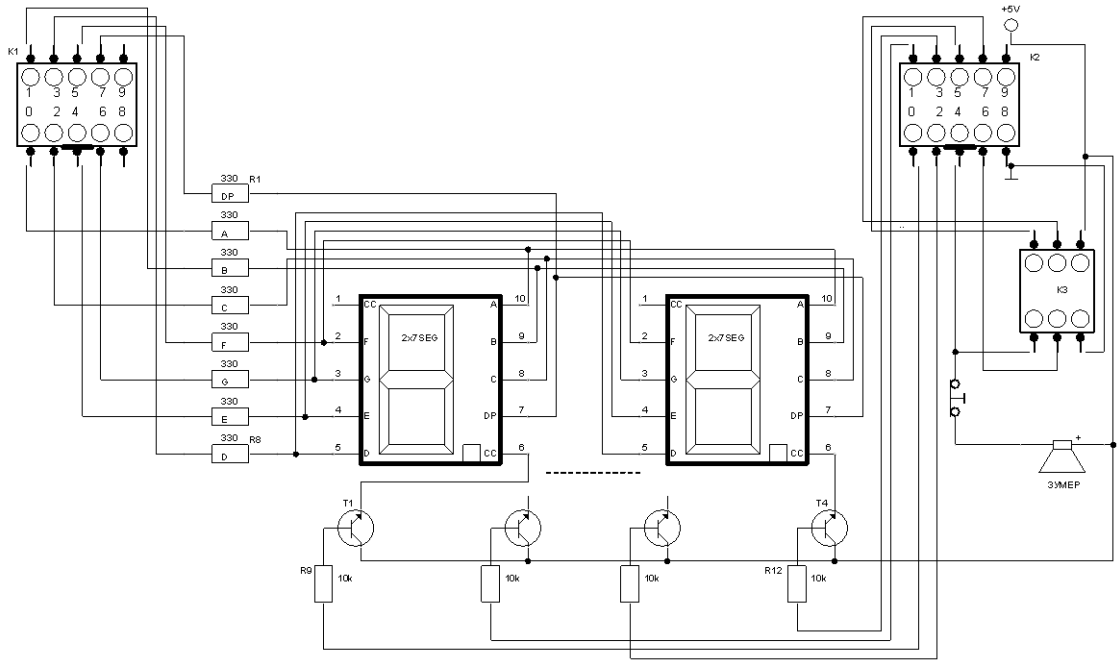
В модула има също пасивен зумер, свързан към извод чрез джъмпер. За генериране на звук е нужно да се извеждат импулси към него със звукова честота (между 20 Hz и 20 kHz).



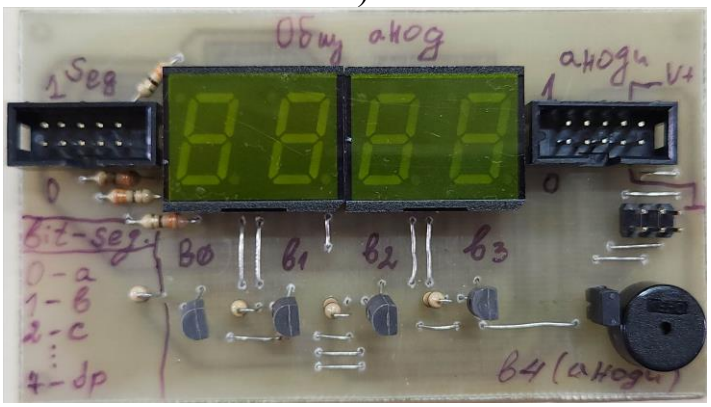
а)



б)



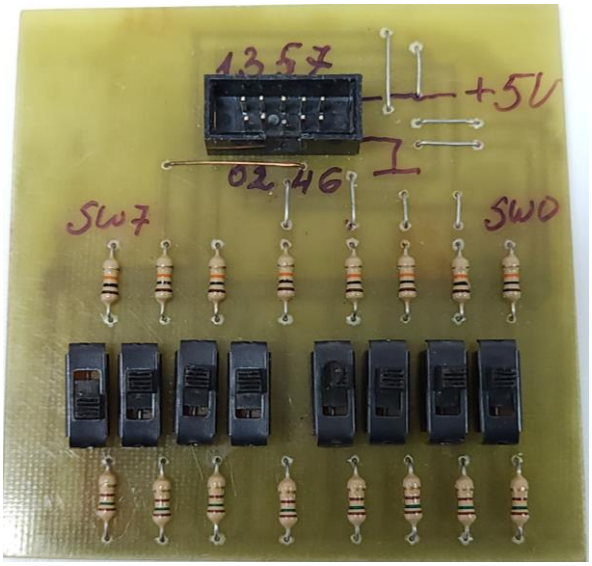
в)



г)

Фиг. 8.3. Модул 7-сегментни индикации и зумер

На фиг. 8.4 е представен външния вид на модул с 8 превключвателя, който може да се използва за статично извеждане на избрана комбинация към STK500 или друга периферия.



Фиг. 8.4. Модул с 8 превключвателя

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Свържете общите аноди на периферната платка със седем сегментния дисплей към рейката на PortA от развойната платка STK500, а сегментите към PortB. Съставете програма за изписване на числото 1. Променете програмата, така че да се визуализират върху дисплея числата 1, 2, 3 и 4 с използване на динамичния метод с програмно времезакъснение и апаратно времезакъснение с обслужване на прекъсване от таймер-брояча.
2. Свържете периферната платка с превключвателите към PortD. Съставете програма за преобразуване на двоичните значения, задавани от превключвателите в шестнадесетичен код, който да се извежда върху седем сегментния дисплей.
3. Като използвате метода за индиректно адресиране на SRAM паметта за данни съставете програма, реализираща десетичен брояч, визуализиран на седем сегментния дисплей.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Какви методи за управление на 7-сегментни индикатори познавате? Каква е разликата между тях, относно необходимите хардуерни ресурси и програмно осигуряване?
2. Какво ще се случи, ако при динамичния метод за извеждане времето за изчакване за смяна на активния индикатор е прекалено малко?

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 9
Управление на буквено-цифров течнокристален дисплей (LCD)

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да се научат да използват буквено-цифров дисплей с течни кристали (LCD), свързан към едночипов микроконтролер по 4 / 8-битов интерфейс.

КРАТКА ТЕОРИЯ

1. Буквено-цифрови (знакови) дисплеи с течни кристали

Най–общо LCD дисплеите се състоят от две части: течнокристален матричен дисплей и схема за управление. Схемата за управление се нарича още LCD драйвер и представлява интегрална схема, чрез която върху дисплея се изобразяват желаните символи. Вида на символите, които се визуализират и позицията им върху дисплея се указват чрез комуникацията на драйвера със свързания към LCD дисплея микроконтролер. Следователно, тук управлението се осъществява на две нива: драйвер – течнокристален дисплей и микроконтролер – драйвер. На практика, при използване на LCD дисплеи, потребителят осъществява само управление на ниво микроконтролер – драйвер. За реализиране на това управление е необходимо да се познават особеностите на драйвера, използван в конкретния LCD дисплей. Изводите на драйвера, чрез който се осигурява връзка с микроконтролера и постоянно токовото му захранване са изведени като външни изводи за самия дисплей. Въпреки различията в драйверите, различията в брой редове и брой символи на ред, в повечето случаи разположението на изводите е по начина, показан в таблица 9.1.

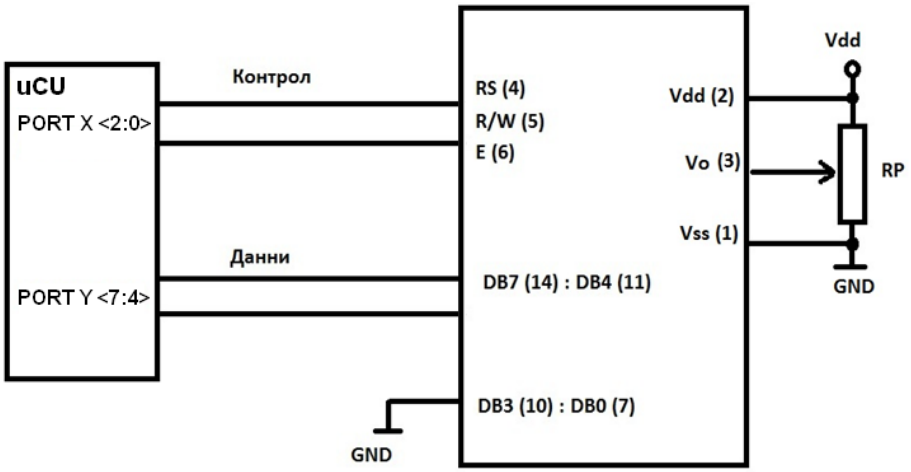
Таблица 9.1. Разположение на изводите на дисплей с течни кристали

Извод	Сигнал	Описание
1	GND	Маса
2	Vcc	Захранване
3	V0	Напрежение, определящо поляризацията кристалите на дисплея (на практика – ъгълът под който трябва да се гледа дисплея)
4	RS	Register Select – избор на регистър 0 – Регистър за инструкции (записи) 1 – Регистър за данни (запис/четене)
5	R /$\overline{W}$	Read/Write – четене/запис 0 – запис 1 – четене
6	E	Enable – разрешение

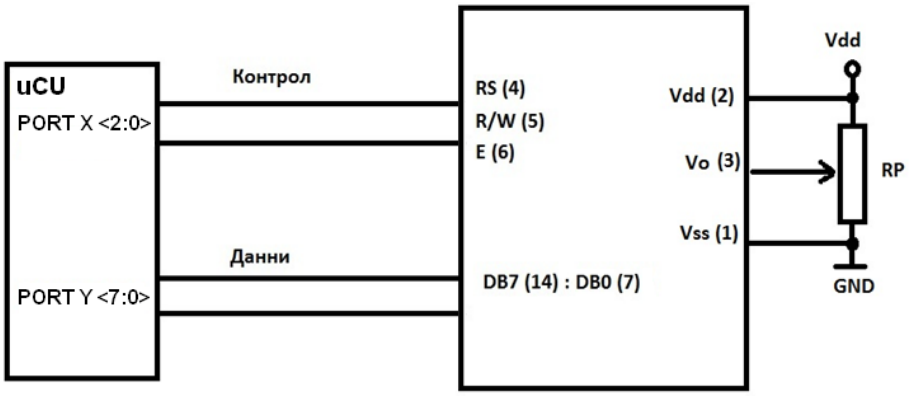
7 – 10	DB0-DB3	8 – битова комуникация – Данни (битове 0-3) 4 – битова комуникация – Не се използва
11 – 14	DB4-DB7	8 – битова комуникация – Данни (битове 4-7) 4 – битова комуникация – Данни (битове 0-3)

В случай, че LCD-дисплея разполага с подсветка (осветление) изводите за нейното захранване са с номера 15 и 16, както следва: захранващо напрежение +5V (15) и маса (16).

За управление на дисплея (респективно драйвера) се използва управляваща шина, включваща изводи RS, R/W, E и шина за данни - изводи DB7 - DB0. Чрез последната се осъществява комуникация на микроконтролера с LCD-дисплея, респективно драйвера. В зависимост от избрания режим, комуникацията може да се осъществи по четири (фиг. 9.1) или 8-битов интерфейс (фиг. 9.2).



Фиг. 9.1. Свързване на микроконтролер с LCD дисплей по 4-битов интерфейс



Фиг. 9.2. Свързване на микроконтролер с LCD дисплей по 8-битов интерфейс

С потенциометъра RP се осигурява поляризиращо напрежение за дисплея. Стойността на това напрежение зависи основно от физическите

свойства на кристала, като основната цел е, чрез настройка да се получи максимален контраст на символите при конкретен ъгъл на наблюдение.

2. Описание на LCD драйвер HD44780U

HD44780U представлява буквено-цифров LCD драйвер, с възможност за работа с 4 или 8-битов интерфейс. Той е произведен от HITACHI, това е една от първите компании разработващи LCD драйвери. Тъй като компанията е водеща в тази област, нейните продукти са се наложили като еталон, който следва от останалите производители на LCD драйвери. Това е едната причина за разглеждане точно на този драйвер, другата е, че на него са базирани голяма част от най-често срещаните LCD модули.

Основните характеристики на HD44780U са:

- захранващо напрежение: 2,7 – 5,5 V;
- формат на изобразяване на символите: 5x8 или 5x10 точки;
- 9920 бита ROM знаков генератор (CGROM – Character Generator ROM);
- 80x8 бита RAM памет за визуализиране на символите (DDRAM – Display Data RAM);
- 64x8 бита RAM памет за дефиниране на допълнителни символи (CGRAM – Character Generator RAM, при 5x8 точки – 8 символа, а при 5x10 – 4 символа);
- Максимална скорост на интерфейса: 2 MHz, при захранване 5 V;
- Комуникация с микроконтролера чрез 4 или 8-битов интерфейс.

Блоковете имащи отношение към управлението микроконтролер – драйвер са: – регистър за инструкции (IR – Instruction Register) и регистър за данни (DR – Data Register). Изборът на регистър се осъществява, чрез извода RS (Register Select). Комбинациите за този избор са представени в таблица 9.2.

Таблица 9.2. Избор на регистър

RS	R / $\overline{W}$	Действие
0	0	Подаваната команда се записва в регистъра за инструкции
0	1	Чете се флаг BUSY(DB7) и броячът на адреси (DB0 – DB6)
1	0	Запис в регистъра за данни
1	1	Четене от регистъра за данни

IR съхранява кода на инструкциите, изпратен от микроконтролера (изчистване на дисплея, изместване на курсора и др.) или адреса за четене/запис от/в DDRAM или CGRAM. Регистъра за данни служи за временно съхранение на данните, които ще се четат или записват от(в) DDRAM или CGRAM. Данните, записани в DR, автоматично се записват в предварително указания чрез IR адрес от DDRAM или CGRAM. При четене от DDRAM или CGRAM, съдържанието на клетката с адрес, указан

в IR, временно се съхранява в DR. Така, след това, то може да бъде прочетено от микроконтролера. Операциите четене/запис, от/в регистрите IR и DR, се осъществяват в зависимост от нивото на извода R / $\overline{W}$ (таблица 9.2).

2.1. Флаг BUSY(Зает)

Този флаг отразява моментното състояние на драйвера.

BF = 1 (BF – Busy Flag) означава, че драйверът е „зает" с изпълнение на предходната инструкция и ако бъде подадена следваща такава, тя ще се игнорира. Когато RS = 0 и R / $\overline{W}$ = 1, състоянието на този флаг се извежда на извод DB7 (таблица 9.2). Подаване на следваща инструкция към драйвера трябва да става, когато BF = 0.

2.2. Адресен брояч (AC – Address Counter)

Адресния брояч съдържа адреса за четене/запис от/в DDRAM или CGRAM. Когато инструкцията, указваща адреса, се запише в IR, битовете, съдържащи адресна информация, автоматично се записват в AC. Изборът на паметта (DDRAM или CGRAM), за която се указва адреса, става в самата инструкция.

Когато се извършва четене или запис, съдържанието на AC (респективно адреса) автоматично се увеличава или намалява с единица, в зависимост от избрания режим. При RS = 0 и R/ $\overline{W}$ = 1 (таблица 9.2), съдържанието на адресния брояч се извежда на изводите (DB6 – DB0).

2.3. Памет за визуализиране на символите (DDRAM – Display Data RAM)

В тази памет се записват 8-битовите кодове на символите, които ще се изобразяват на дисплея. Тези кодове всъщност представляват адресът на символа от знаковия генератор (CGRAM). Капацитетът на DDRAM е 80x8 бита, което дава възможност на нея да се запишат кодове за 80 символа. Адресите от паметта, които не се използват за визуализиране на символи, могат да бъдат използвани от микроконтролера като памет с общо предназначение. Максималния брой символи, които могат да се визуализират е 8 при едноредов режим, а при двуредов 2x8 (16 символа). Чрез този драйвер може да се визуализира и 2x16 символа, но за целта към него трябва да се включат специални допълнителни драйвери. По този начин се осигурява необходимия брой изходи за управление на течнокристалния дисплей.

2.4. Знаков генератор (CGROM – Character Generator ROM)

Това е ROM памет, в която са дефинирани (от производителя) всички символи, които могат да се изпишат върху дисплея. В зависимост

от вида на символите, дефинирани в CGROM, конкретния LCD драйвер се произвежда в две разновидности: HD44780UA00 и HD44780UA02.

Таблица 9.3. Връзка между вида на символа в CGROM и кода, чрез който се визуализира

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	Ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	₺	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	■

В таблица 9.3 са дадени символите, дефинирани в CGROM и кодовете, чрез които те могат да бъдат визуализирани. Използваните от производителя кодове са аналогични на ASCII кода, използван в персоналните компютри. Таблица 9.3 се отнася за драйвер със суфикс A00. Както се вижда от таблицата, с този драйвер не може да се изобразяват символи на кирилица. Тези символи са заложени в HD44780UA02, но за съжаление сравнително рядко се срещат LCD модули, които го използват.

2.5. Инструкции поддържани от HD44780U

Инструкциите, които се поддържат от разгледания драйвер са дадени в таблица 9.4.

Таблица 9.4. Инструкции на драйвер HD44780U

Инструкция	Код на инструкцията										Описание	Време за изпълнение на инструкция (fosc=270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Записва "20H" в DDRAM. Мести указателя към "00H" от AC	1.53 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Връща указателя към адрес "00H" на DDRAM от AC и връща курсора към оригиналната си позиция, ако е бил изместен. Съдържанието на DDRAM не се изменя.	1.53 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Задава посоката на движение на курсора и разрешава превключването на дисплея	39 us
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Настройка на дисплея (D), курсора (C), мигането на курсора (B)	39 us
Cursor Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Настройка на посоката на движение на курсора и бита за управление на превключването на дисплея, без промяна на съдържанието на DDRAM.	39 us
Function Set	0	0	0	0	1	DL	N	F	-	-	Настройка на дължината на интерфейсната дума (DL: 8 bit / 4 bit), брой редове (N: 2 линии / 1 линия) и типа на шрифта на дисплея (F 5 x 11 точки / 5 x 8 точки).	39 us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Зареждане на адреса от CGRAM в адресния брояч.	39 us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Зареждане на адреса от DDRAM в адресния брояч.	39 us
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Четенето на флаговете може да се извърши или по време на операцията или чрез прочитане на BF. Съдържанието на адресния брояч също може да бъде прочетено.	0 us
Write Data To RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Запис на информация във вътрешната RAM памет (DDRAM / CGRAM)	43 us
Read Data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Четене на информация от вътрешната RAM (DDRAM / CGRAM)	43 us

Времето за изпълнение на всяка инструкция зависи от тактовия генератор, вграден в LCD драйвера. Въпреки че генератора е част от драйвера, честотата му се определя от външен резистор и при повечето

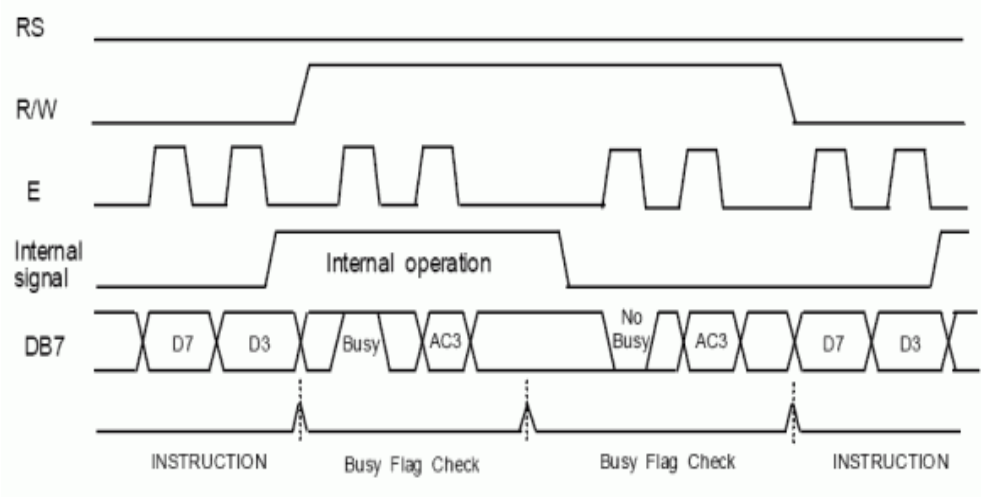
LCD модули тя е 270 kHz. За тази честота се отнасят и времената, дадени в таблица 9.4. Тези времена са много важни, защото те определят, колко време драйверът ще е „зает” с изпълнение на инструкцията. В случай че не се знае честотата, трябва да се следи флаг „Busy”.

Таблица 9.5. Значения на отделните битове от таблица 4

I/D	„0” – Намаляване с 1 „1” – Увеличаване с 1	DL	„0” – 4 битов интерфейс „1” – 8 битов интерфейс
S	„1” – Преместване на дисплейния прозорец при запис и четене	N	„0” – 1 ред „1” – 2 реда
S/C	„0” – Преместване на курсора „1” – преместване на дисплейния прозорец	F	„0” – 5x8 точки символна матрица „1” – 5x10 точки символна матрица
R/L	„0” – Преместване на ляво „1” – Преместване на дясно	BF	„0” – готовност за следваща инструкция „1” – изпълнява се вътрешна операция
Acg – адрес в CGRAM , Add – Адрес в DDRAM, AC – Брояч на адресите RD – Данни за четене, WD – Данни за запис			

3. Комуникация с микроконтролера

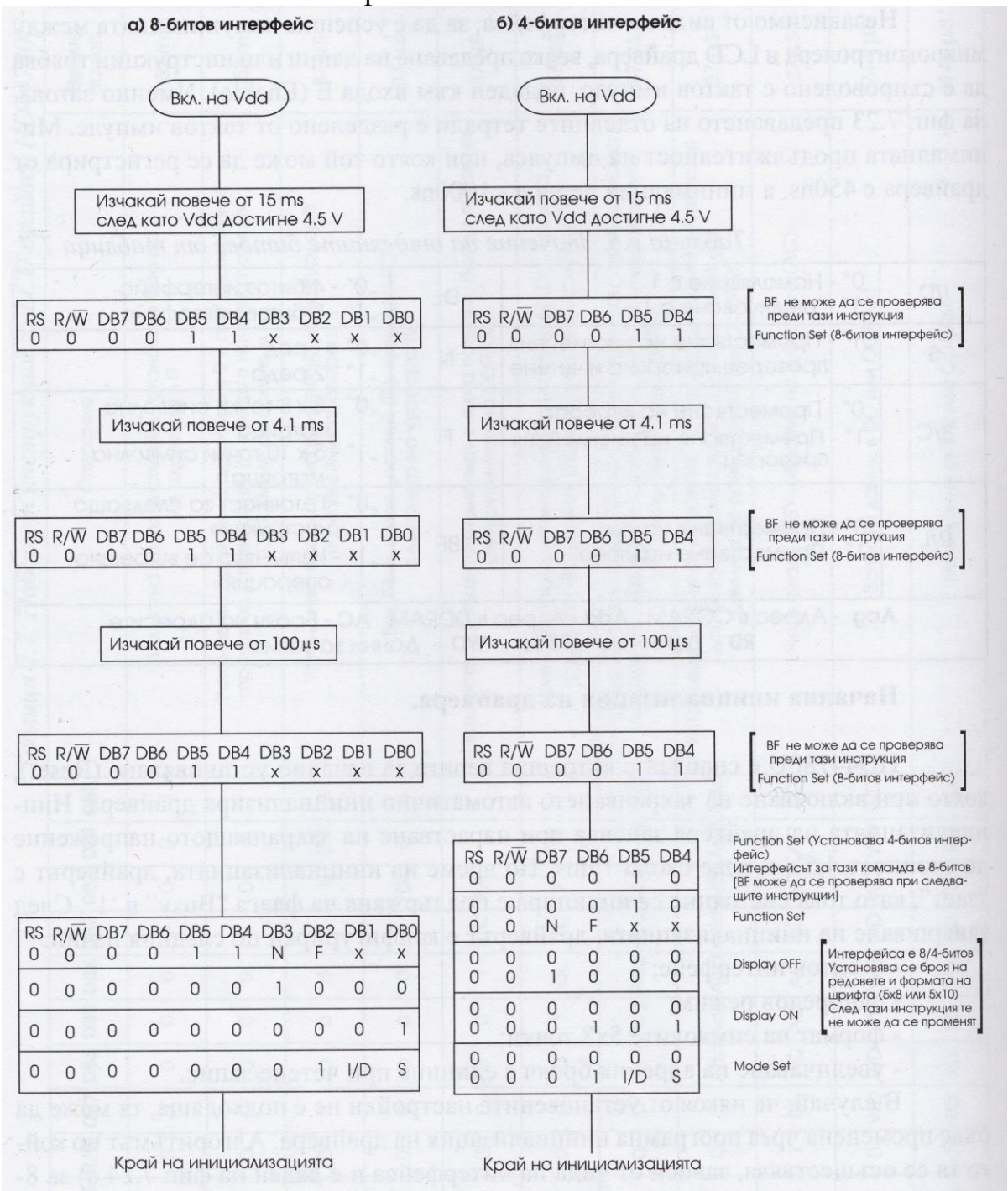
Както стана ясно, с използване на драйвер HD44780U, комуникацията с микроконтролера може да се осъществи чрез 4 или 8-битов интерфейс. При 8-битовият интерфейс се използват всички изводи (DB7-DB0) от шината за данни, а при 4-битовият връзката се осъществява само чрез изводите DB7 – DB4. В този случай предаването на 8-битовите данни и инструкции става на два цикъла (фиг. 9.3). В първия цикъл се предава старшата тетрада (битове 7:4) от 8-битовата дума, а във втория младшата (битове 3:0).



Фиг. 9.3. Комуникация с микроконтролера по 4-битов интерфейс

4. Програмна инициализация на драйвера HD44780U

Алгоритъмът по който програмната инициализация се осъществява за HD44780U е показан на фиг. 9.4.



Фиг. 9.4. Програмна инициализация на LCD драйвер HD44780U

Независимо от вида на интерфейса, за да е успешна комуникацията между микроконтролера и LCD драйвера, всяко предаване на данни или инструкции трябва да е съпроводено с тактов импулс, подаден към входа E (Enable). Минималната продължителност на импулса, за да се отчете от драйвера е 450 ns, а минималния период 1000 ns.

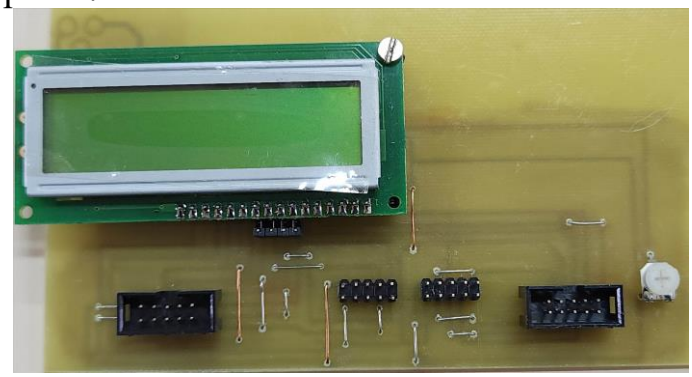
Драйвер HD44780U е снабден с вътрешна верига за начално установяване (Reset), която при включване на захранването автоматично инициализира драйвера. Инициализацията на драйвера започва при нарастване на захранващото напрежение до стойност 4,5 V и трае около 15ms. По време на инициализацията драйверът е „зает”. След завършване драйверът е конфигуриран по следния начин:

- 8-битов интерфейс;
- Едноредов режим;
- Формат на символите 5x8 точки;
- Увеличаване на адресния брояч при четене/запис.

В случай, че някоя от установените настройки не е подходяща, тя може да бъде променена чрез програмна инициализация на драйвера (фиг. 9.4).

5. Модул с LCD и свързване към STK500

На фиг. 9.5 е представен външния вид на LCD модул 2 реда по 16 символа с изведени изводи за свързване към STK500 и потенциометър за контрол на контраста.



Фиг. 9.5. LCD модул с изведени изводи за свързване към STK500

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Съставете програма за изписване на двуредов стринг с използване на 8-битов интерфейс за връзка на LCD дисплея с микроконтролера.

2. Съставете програма за изписване на двуредов стринг с използване на 4-битов интерфейс за връзка на LCD дисплея с микроконтролера.

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Колко и какви са възможните режими на обмен на данни с LCD модул, базиран на драйвер HD44780U? Какви са разликите между тях?

2. Помислете и предложете решение за извеждане на движещ се надпис, който е с повече от 16 символа на ред? Тествайте го практически.

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 10
Връзка с отдалечен компютър по сериен интерфейс RS-232 чрез модул USART

ЦЕЛ НА УПРАЖНЕНИЕТО

Целта на упражнението е студентите да се запознаят с основния принцип за реализиране на сериен обмен на данни с компютър (или друг контролер) по сериен интерфейс RS-232 и използване на функционален блок USART в едночипов микроконтролер ATmega8515.

КРАТКА ТЕОРИЯ

1. Характеристики на Универсален синхронно-асинхронен приемо-предавател (УСАПП) – USART – Universal Synchronous-Asynchronous Receiver / Transmitter

Сериен (последователен) интерфейс: данните между предавателя и приемника се обменят във всеки момент по една комуникационна линия, а във времето – бит след бит т.е. последователно по битове.

Паралелен интерфейс: данните между предавателя и приемника се обменят във всеки един момент по k линии, т.е. в думи (групи) от по k бита едновременно, т.е. паралелно по битове, а във времето – дума след дума, т. е. последователно по думи.

"Асинхронен" обмен означава, че между предавателя и приемника не се предава в явен вид синхронизиращ (тактов) сигнал. Като следствие, за него не се заделя отделна линия. Поради това тактовият сигнал трябва да се регенерира "от отсрещната страна". За целта се използват специални служебни битове, вмъквани в потока на обмен (казва се: "във форма́та на кадъра" за обмен).

"Синхронен" обмен означава, че синхронизиращият (тактовият) сигнал между предавателя и приемника се предава в явен вид по отделна обособена допълнителна линия за връзка.

Асинхронният метод на обмен е по-икономичен по апаратни разходи (по броя на комуникационните линии), но използва обема на канала по-неефективно т.е. освен битовете за данни се предават и служебни битове за тактуване на обмена.

УСАПП в микроконтролер ATmega8515 предоставя следните възможности:

- Реализира сериен (т.е. побитов) обмен на данни;
- Вгражда два канала за обмен: един за предаване и един за приемане: това се отразява в наименованието му – "приемо-предавател";

Може да се конфигурира програмно и за **двата метода за синхронизация на обмен**:

- за асинхронен обмен на данни - тогава той е просто UART (Asynchronous);
- за синхронен обмен на данни - USRT (Synchronous).

Тази би-функционалност също е фиксирана в наименованието му - "Универсален синхронно-асинхронен".

УСАПП се използва по-често в качеството си на асинхронен приемо-предавател и преди всичко - за реализация на стандартизирания сериен интерфейс RS-232.

Буфериране на данните в УСАПП:

- В предавателя – еднократно (един байт в предавателния регистър);
- В приемника – двукратно (два байта в приемния буфер).

В/И изводи в ATmega8515, използвани за **асинхронен сериен интерфейс**:

- Извод PD0: RxD (Received Data) - извод за приети данни (постъпващи в микроконтролера);
- Извод PD1: TxD (Transmitted Data) - извод за предадени данни (изпращани от микроконтролера).

2. В/И регистри за работа с USART в ATmega8515

Таблица 10.1. В/И регистри за работа с USART в ATmega8515

Адрес	Име	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
$20 ($40) 2 рег. / 1 адр.	UBRRH	URSEL	-	-	-	UBRR [11 : 8]			
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
$0C ($2C)	UDR	USART I/O Data Register							
$0B ($2B)	UCSRA	RXC **	TXC **	UDRE	FE	DOR	PE	U2X	MPCM
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN *	TXEN *	UCSZ2	RXB8	TXB8
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte							
* Активиране на функцията USART за изводи PD0 (RxD) и PD1 (TxD) от порт D на ATmega8515									
** Статусни битове за сканиране (Polling) на готовността на приемника (RXC) и предавателя (TXC)									

2.1. Управляващи / Статусни регистри A, B и C:

UCSRA, UCSRB, UCSRC; "U" от USART; "CS" от Control / Status; "R" от Register.

Управляващите битове за активиране на функция USART по изводи PD0 (за RxD) и PD1 (за TxD) от порт D на ATmega8515 и подтискане на нормалната им входно-изходна функция, са два:

- RXEN (Receive Enable) – разрешаване на работата на приемната част на USART: от момента, когато в този бит програмата впише 1;
- TXEN (Transmit Enable) – разрешаване на работата на предавателната част на USART: от момента, когато в този бит програмата впише 1.

Двата указани бита могат да се установяват в 1 или да се нулират независимо един от друг. Така може да се активира само приемникът, само предавателят, двата едновременно или никой от двата блока на USART.

Статусните битове за сканиране (Polling) на готовността на приемника и предавателя в USART са следните два:

- **RXC (Receive Complete)** – когато, след прочитане на регистър UCSRA, този бит се намери в състояние 1, това означава, че в приемния буфер се съдържа поне един непрочетен байт (а буферът с размерност два байта може да съдържа и два – последните два приети байта);
- **TXC (Transmit Complete) –** когато, след прочитане на регистър UCSRA, този бит се намери в състояние 1, това означава, че предавателният буфер е празен* и програмата вече може да копира в предавателния регистър с име UDR (USART Data Register) следващият символ за изпращане със сериализиране по битове. Ако той се намери в състояние 0, програмата трябва да изчака вдигането му в 1, като го чете от време на време (проверявайки дали е в 1), преди да копира следващият символ в UDR.

* Определението "празен" означава, че данните от приемния регистър, който е с име UDR (USART Data Register) – същото е името и за предавателния регистър за данни – са били вече прочетени, т.е. не са актуални. В действителност, приемният регистър "е пълен", ако се има предвид, че той съдържа неразрушени последните приети, но неактуални данни.

3. Предавател за асинхронен обмен на данни

Предавателят изпраща по еднопроводната линия за обмен на данни "порция" от данни с определен брой даннови битове. "Порцията" данни се нарича кадър от данни (аналог на "дума"). Битовете от всеки кадър се изтласкват в линията един след друг.

Това определя наименованието на типа на предаване, а именно – "Сериен (или Последователен) обмен".

Броят на битовете в кадъра се определя от неговия формат. Форматът най-често включва 5 – 9 бита за данни.

Предавателят може да издава кадрите в линията, долепени плътно един след друг или да оставя между два последователни кадъра паузи с произволна продължителност. В такава ситуация отдалеченият приемник, който се намира на другия край на линията за връзка, се намира постоянно

в очакване на нов кадър. Неговата задача е по-трудна от тази на предавателя, тъй като той трябва да открива непрекъснато момента на началото на всеки нов кадър и да се синхронизира по него.

3.1. Старт-стопен принцип за сериен обмен на данни

Началото на всеки кадър от данни се маркира още в предавателя с вмъкване на един допълнителен бит St, наричан "Стартов", който предшества по време (както и в линията) първия бит от предаваните данни. Стартовият бит има значение логическа нула, ниво обратно на това по линията за обмен в състояние на покой (което е 1).

Наличието в кадъра на бит за старт облекчава задачата на приемника, защото при тези условия приемникът трябва да открива само момента на пристигане на стартов бит (приемане на 0, след като линията е била в 1).

За да може приемникът да раздели стартовият бит за текущия кадър от последния бит на предшестващия кадър, когато този бит е бил също нула, предавателят вмъква в края на всеки кадър още един бит Sp, наричан "Стопов", като разделител на кадрите.

Битът за стоп (или за край на кадъра) е винаги единица, т. е. съвпада с логическото ниво на линията в покой. Така началото на следващ кадър се идентифицира с пристигане на първия нулев бит след бита за стоп – именно този бит се интерпретира от приемника като стартов бит за пристигащ по линията кадър.

Битът за Старт се проявява за приемника като спадащ фронт на нивото в линията за обмен, ако тя се е намирала в състояние на покой. Спадащите фронтове между битовете приети данни по време на приемане на кадъра не се възприемат като сигнал за начало на нов кадър, защото след откриване на стартов бит приемникът третира състоянието на линията като състояние, което не е покой.

3.2. Асинхронно запускане на приемния тактов генератор (ТГ)

Запускането на приемния тактов генератор се извършва в средата на стартовия бит, след като той е бил открит надеждно по линията. УСАПП в ATmega8515 поддържа два режима за възстановяване на честотата на приемния ТГ по честотата на битовата скорост в канала: 16-пъти битовата скорост в канала (режим 16х-) и 8-пъти битовата скорост в канала (режим 8х-). По-неточна е синхронизация във 2-я случай (режим 8х-). За определяне на логическото ниво в средата на всеки сканиран бит, в т. ч. и на нивото на стартовия бит, в чипа е реализирана логика за мажоритарен вот по 3 дискрети около средната точка на бита, идващ по приемния вход от линията.

Необходимото условие за надеждно откриване на стартовия бит и за запускане на ТГ е поне две от трите стойности, сканирани около средата

на битовото време, или всичките три да бъдат 0. И обратно – ако 2 или всички 3 от 3-те сканирани стойности са 1, то се счита, че е имало случайно запускане от шум, генерирал падащ фронт. В такива случаи приемният ТГ не се запуска.

3.3. Скорост на обмен – [Bd] и [бит/сек]

Мерната единица бит/сек се използва за измерване на скоростта за обмен на данни в цифровата част на канала.

За измерване на скоростта в аналоговата част на канала се използва мерната единица "Baud", "Бод" [Bd], в чест на Baudot – френски изобретател на 5-битов код за предаване на данни.

Когато в аналоговия тракт на линията за връзка блокове от N бита в кадъра се кодират с 2^N нива на аналоговия сигнал, тогава за един времеви квант и с едно аналогово ниво се предава не един, а N бита. В такива случаи бодовата скорост [Bd] е различна от битовата скорост (бит/сек).

- ❖ Скоростта в [Bd] е броят на промените в аналоговото ниво на сигнала за една секунда.

Оттук и скоростта, измерена в [***Bd***] в аналоговата част на канала, е ***най-малко N пъти по-ниска*** от скоростта в цифровата му част, измервана в [***бит/сек***].

Двете мерки за скорост на обмен ***съвпадат в случаите,*** когато между приемника и предавателя не се реализира съответното аналогово кодиране по групи от битове (по-точно, аналоговото кодиране се реализира за блокове от по един бит).

3.4. Задаване на битовата скорост [Bd] (за случая, когато бит/сек = Bd)

Изборът на коефициента на делене за получаване на желана "бодова" скорост [Bd] зависи от честотата на използвания ТГ за USART.

Коефициентът на делене се записва в 12-битовия регистър UBRR, състоящ се от два (по-точно – от един и половина) 8-битови регистри.

Избраният коефициент на делене задава една и съща скорост за обмен за локалния предавател и за локалния приемник. С други думи, не е възможно локалният предавал да предава на една скорост, а локалният приемник да приема на друга.

Първото условие за успешна комуникация по сериен интерфейс е идентичност на настройките на формата на кадъра на локалните предавател и приемник с тези за отдалечените приемник и предавател, което включва идентичност:

- по брой битове за данни,
- по брой битове за стоп,
- по брой битове за четност и
- по скорост на обмен [Bd].

При зададена честота на локалния ТГ и за желаната скорост на обмен се избира такъв коефициент на делене, който води до най-малка относителна грешка между битовите честоти на локалния предавател и на отдалечения приемник.

Фирма Atmel предлага табличен метод за избор на коефициентите, вписвани в регистър UBRR, в зависимост от желаната бодова скорост и от честотата на ТГ в микроконтролера.

3.4.1. Табличен метод за избор на коефициента на делене

Честотата на ТГ **Fosc = 3.6864 MHz** е най-често използваната в STK500. За този случай се предлага следната таблица (таблица 10.2) за избор на коефициент на делене при желана скорост на обмен.

Таблица 10.2. Избор на коефициент на делене

Baud Rate (bps)	f_{osc} = 3.6864 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%
4800	47	0.0%	95	0.0%
9600	23	0.0%	47	0.0%
14.4k	15	0.0%	31	0.0%
19.2k	11	0.0%	23	0.0%
28.8k	7	0.0%	15	0.0%
38.4k	5	0.0%	11	0.0%
57.6k	3	0.0%	7	0.0%
76.8k	2	0.0%	5	0.0%
115.2k	1	0.0%	3	0.0%
230.4k	0	0.0%	1	0.0%
250k	0	-7.8%	1	-7.8%
0.5M	–	–	0	-7.8%
1M	–	–	–	–
Max. (1)	230.4 kbps		460.8 kbps	

1. UBRR = 0, Error = 0.0%

Обграден е частният случай, когато с коефициент за делене 23, който трябва да се впише от програмата в регистър UBRR, се постига настройка за скорост 9600 бит/сек на тактовите генератори на приемника и предавателя.

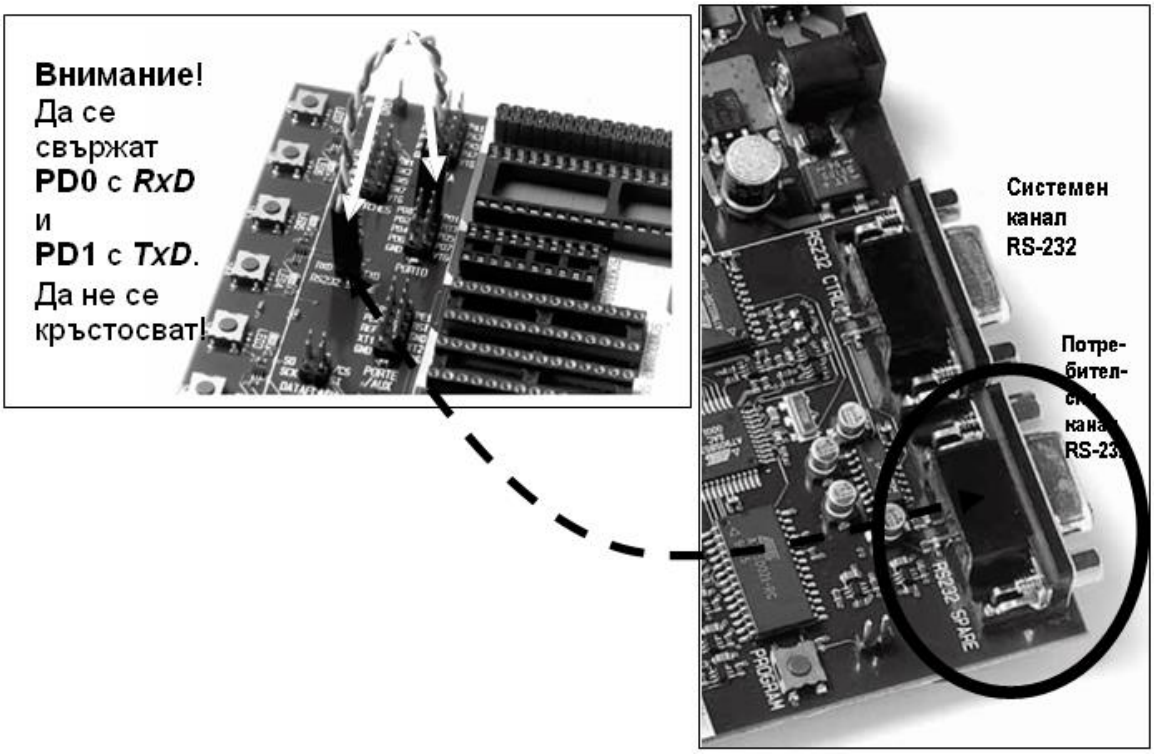
4. Описание на каналите за RS-232 в STK500

Развойната система STK500 реализира два канала за връзка RS-232.

Първият от тях е резервиран за системни цели – за комуникация на STK500 с интегрираната среда AVR Studio в компютъра PC. Той се управлява не от целевия потребителски чип, а от системния контролер 8535, служещ още и за програматор на целевия uCU (тук – ATmega8515). Използва се 9-изводен конектор, именован „RS-232 ***CTRL***”.

Вторият канал RS-232 може да бъде използван за потребителска комуникация от приложната целева (target) система, която е изградена от микроконтролер ATmega8515, поставен в съответен цокъл в система STK500. Комуникира се през 9-изводен конектор, обозначен на платката като „RS-232 ***SPARE***”.

За да стане вторият канал използваем, изводите на UART в потребителския микроконтролер трябва да се свържат физически чрез 2-жичен кабел към изводната рейка от платка STK500, означена като „RS232 ***SPARE***”.



Фиг. 10.1. Канали за RS-232 в развойна система STK500

ЗАДАЧИ ЗА ИЗПЪЛНЕНИЕ

1. Да се състави програмен код, чрез който да се изпрати от микроконтролера към PC съобщението „TU – Gabrovo Katedra KST”.

Указание:

Използвайте вградената в операционната система (Windows) терминална програма Hyperterminal, която ще интерпретира като ASCII-кодове символите на съобщението. За тестване на програмата е

необходимо да се извърши следната последователност от действия с цел апаратна и програмна настройка:

1. Изключете развойната платка STK500 от захранването.

Използвайки двужилния кабел от комплекта на STK500;

2. Свържете извод PD0 с клема RXD и извод PD1 с клема TXD от платката;

3. Свържете PortB със светодиодите на STK500 чрез 10-жилен кабел от комплекта;

4. Включете система STK500;

5. Програмирайте микроконтролера;

6. Отново изключете развойната платка STK500 от захранването;

7. Преместете серийния кабел от системния конектор RS232 CTRL към RS232 SPARE;

8. Стартирайте Hyperterminal в Windows;

9. Настройте Hyperterminal:

- задайте номера на използвания порт (COM1 или 2) и формат 9600, 8N1.

10. Включете захранването на платката STK500.

2. Да се състави програмен код, чрез който да се приемат и визуализират върху светодиодната индикация на платката STK500 ASCII-кодовете на символи изпращани от PC (чрез Hyperterminal).

КОНТРОЛНИ ВЪПРОСИ И ЗАДАЧИ

1. Какви са разликите между синхронен и асинхронен обмен на данни?

2. Какви мерни единици за скорост на обмен познавате при серийните комуникации и каква е разликата между тях?

3. При серийната комуникация чрез асинхронен режим на USART модула, по какъв начин приемникът разпознава началото и края на изпратен кадър данни, за да се синхронизира неговият вътрешен тактов генератор?

4. Свържете към STK500 по подходящ начин модул с LCD и реализирайте програма, чрез която да се извеждат приетите символи, изпращани от PC (чрез Hyperterminal).

Литературни източници

[1] Atmel Corporation, 8-bit Microcontroller with 8K Bytes In-System Programmable Flash ATmega8515 ATmega8515L, Rev. 2512G–AVR–03/05, 2005 (www.atmel.com/literature).

[2] Microchip Technology Inc., Downloads Archive for AVR and SAM MCUs/MPUs, https://www.microchip.com/en-us/tools-resources/archives/avr-sam-mcus - 09.2024.

[3] Eric B. Weddington, WinAVR™ User Manual – 20050214, https://winavr.sourceforge.net/WinAVR-user-manual.html, 09.2024.

[4] Atmel Corporation, AVR STK500 User Guide, 1925C–AVR–3/03, https://ww1.microchip.com/downloads/en/DeviceDoc/doc1925.pdf, 2003.

[5] Tjabo Kloppenburg. AVR delay loop generator v1.2, https://www.softpedia.com/get/Science-CAD/AVR-delay-loop-generator.shtml, 2023.

[6] Agilent Technologies, Inc., 14.2 mm (0.56 inch) General Purpose Two Digit Seven Segment Displays Technical Data HDSP-52xE Series HDSP-52xG Series HDSP-52xY Series, July 17, 2004, 5988-5378EN, https://www.farnell.com/datasheets/95204.pdf.

[7] Hitachi, Ltd., HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver), ADE-207-272(Z), '99.9, Rev. 0.0 (1998), https://www.sparkfun.com/datasheets/LCD/HD44780.pdf, 2021.

[8] Microchip Technology Inc., ATmega8515 Documentation, https://www.microchip.com/en-us/product/ATmega8515, 2024.

[9] Microchip Technology Inc., http://www.avrfreaks.net/ - форум с мото „всичко за 8 и 32-битови AVR микроконтролери“

СЪДЪРЖАНИЕ