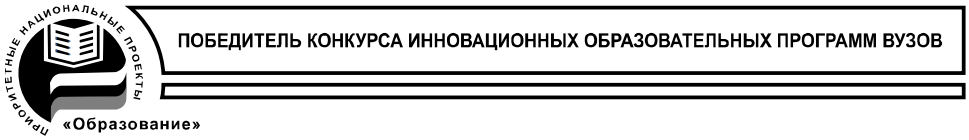
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ



Ю.В. Китаев

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ ATMEGA128 И 68HC908

Санкт-Петербург

2007

УДК 681.32

Китаев Ю.В. Основы программирования микроконтроллеров ATMega128 и 68hc908. Учебное пособие: СПб: СПбГУ ИТМО, 2007, 107 с.

Рассмотрены основные функциональные узлы микроконтроллеров ATMega128 и 68hc908 и программирование типовых периферийных устройств.

Для студентов, обучающихся по направлениям “Приборостроение” и “Оптотехника”

Рекомендовано к печати Советом ИФФ от 02 октября 2006г., протокол №2.

В 2007 году СПбГУ ИТМО стал победителем конкурса инновационных образовательных программ вузов России на 2007–2008 годы. Реализация инновационной образовательной программы «Инновационная система подготовки специалистов нового поколения в области информационных и оптических технологий» позволит выйти на качественно новый уровень подготовки выпускников и удовлетворить возрастающий спрос на специалистов в информационной, оптической и других высокотехнологичных отраслях экономики.

ОГЛАВЛЕНИЕ

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА ATMega128

1. ЦЕЛЬ РАБОТЫ

Целью работы является изучение схем подключения и/или программирования следующих типовых устройств:

1. 8-ми сегментного дисплея,
2. пьезоизлучателя звуковых сигналов,
3. датчика, подключенного к АЦП,
4. таймера0 в асинхронном режиме работы от кварцевого резонатора 32768Гц,
5. клавиатуры 4x3,
6. записи/чтения в/из EEPROM.

2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1. Сформировать секундный временной интервал с подачей короткого звукового сигнала и переключением светодиода.
2. Обеспечить непрерывное измерение напряжения (угла поворота) с одновременным отображением на 8-ми сегментном дисплее.
3. При нажатии на цифровые клавиши высвечивать их код.
4. Две клавиши: “*” и “#” использовать в качестве управляющих. Клавиша “*” переводит устройство в режим измерения напряжения, а клавишей “#” подается длительный звуковой сигнал.
5. Переключение задач производится с помощью клавиатуры.

3. СТРУКТУРА МИКРОКОНТРОЛЛЕРА ATMega128

3.1 НАЗНАЧЕНИЕ ВЫВОДОВ

На рис.1.1 изображен корпус и приведено назначение выводов микроконтроллера. В скобках указана альтернативная функция вывода.

Микроконтроллер ATmega128 включает следующие функциональные блоки:

- 8-разрядное арифметическо-логическое устройство (АЛУ);
- внутреннюю флэш-память программ объемом 128 Кбайт с возможностью внутрисистемного программирования через последовательный интерфейс;
- 32 регистра общего назначения;
- внутреннюю EEPROM память данных объемом 4 Кбайт;

- внутреннее ОЗУ данных объемом 4 Кбайт;
- 6 параллельных 8-разрядных портов;
- 4 программируемых таймера-счетчика;
- 10-разрядный 8-канальный АЦП и аналоговый компаратор;
- последовательные интерфейсы UART0, UART0, TWI и SPI;
- блоки прерывания и управления (включая сторожевой таймер).

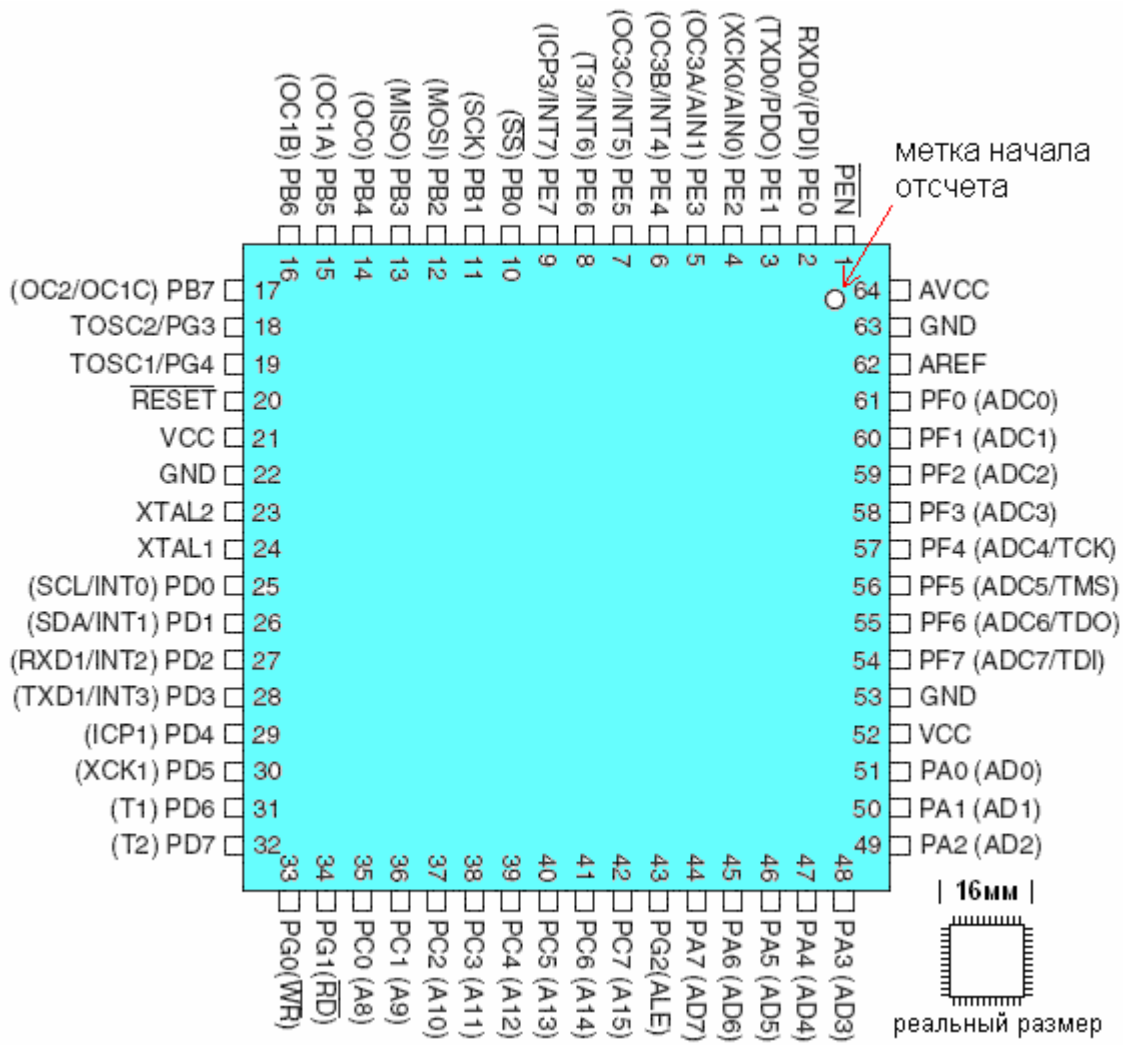


Рис.1.1. Вид корпуса и обозначение выводов микроконтроллера ATmega128.

Port A (PA7..PA). 8-разрядный двунаправленный порт. К выводам порта могут быть подключены встроенные нагрузочные резисторы (отдельно к каждому разряду). Выходные буферы обеспечивают ток 20 мА и способность прямо управлять светодиодным индикатором. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт А при наличии внешней памяти данных используется для организации мультиплексируемой шины адреса/данных.

Port B (PB7..PB0). 8-разрядный двунаправленный порт со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА. При использовании выводов порта в качестве входов и установке внешнего

сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт B используется также при реализации специальных функций.

Port C (PC7..PC0). Порт C является 8-разрядным выходным портом. Выходные буферы обеспечивают ток 20 мА. Порт C при наличии внешней памяти данных используется для организации шины адреса.

Port D (PD7..PD0). 8-разрядный двунаправленный порт со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт D используется также при реализации специальных функций.

Port E (PE7..PE0). 8-разрядный двунаправленный порт со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, вытекающий через них ток обеспечивается только при подключенных встроенных нагрузочных резисторах. Порт E используется также при реализации специальных функций.

Port F (PF7..PF0). 8-разрядный входной порт. Входы порта используются также как аналоговые входы аналого-цифрового преобразователя.

#RESET. Вход сброса. Для выполнения сброса необходимо удерживать низкий уровень на входе более 50 нс.

XTAL1, XTAL2. Вход и выход инвертирующего усилителя генератора тактовой частоты.

TOSC1, TOSC2. Вход и выход инвертирующего усилителя генератора таймера/счетчика.

#WR, #RD. Стробы записи и чтения внешней памяти данных.

ALE. Строб разрешения фиксации адреса внешней памяти. Строб ALE используется для фиксации младшего байта адреса с выводов AD0-AD7 в защелке адреса в течение первого цикла обращения. В течение второго цикла обращения выводы AD0-AD7 используются для передачи данных.

AVCC. Напряжение питания аналого-цифрового преобразователя. Вывод подсоединяется к V_{CC} через низкочастотный фильтр.

AREF. Вход опорного напряжения для аналого-цифрового преобразователя. На этот вывод подается напряжение в диапазоне между AGND и AVCC.

AGND. Это вывод должен быть подсоединен к отдельной аналоговой земле, если она есть на плате. В ином случае вывод подсоединяется к общей земле.

#PEN. Вывод разрешения программирования через последовательный интерфейс. При удержании сигнала на этом выводе на низком уровне после включения питания, прибор переходит в режим программирования по последовательному каналу.

V_{CC}, GND. Напряжение питания и земля.

3.2 ОРГАНИЗАЦИЯ ПАМЯТИ И ПОРТОВ ВВОДА/ВЫВОДА

Микроконтроллеры AVR имеют раздельные пространства адресов памяти программ и данных (гарвардская архитектура). Организация памяти МК ATMega128 показана на рис. 1.2.

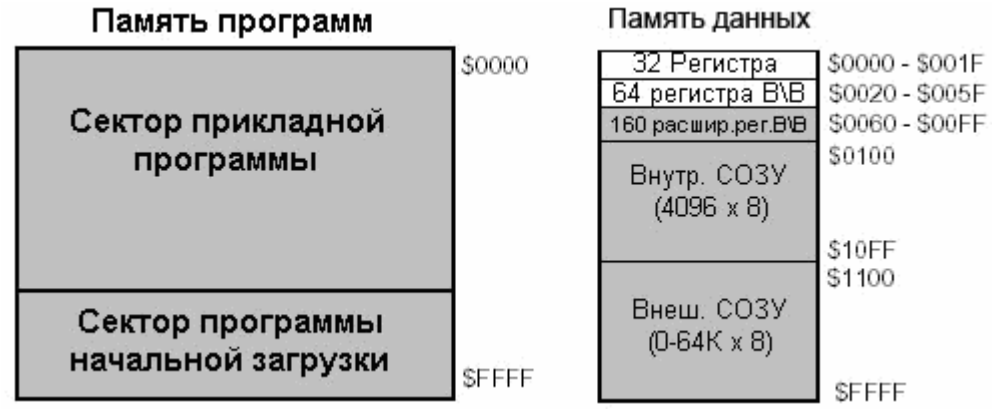


Рис.1.2. Организация памяти микроконтроллера ATmega128

Высокие характеристики семейства AVR обеспечиваются следующими особенностями архитектуры:

- В качестве памяти программ используется внутренняя флэш-память. Она организована в виде массива 16-разрядных ячеек и может загружаться программатором, либо через порт SPI;
- 16-разрядные память программ и шина команд вместе с одноуровневым конвейером позволяют выполнить большинство инструкций за один такт синхрогенератора (50 нс при частоте F_{OSC}=20 МГц);
- память данных имеет 8-разрядную организацию. Младшие 32 адреса пространства занимают регистры общего назначения, далее следуют 64 адреса регистров ввода-вывода, затем внутреннее ОЗУ данных объемом до 4096 ячеек. Возможно применение внешнего ОЗУ данных объемом до 60 Кбайт;
- внутренняя энергонезависимая память типа EEPROM объемом до 4 Кбайт представляет собой самостоятельную матрицу, обращение к которой осуществляется через специальные регистры ввода-вывода.

Как видно из рис. 1.2 и 1.3, 32 регистра общего назначения (РОН) включены в сквозное адресное пространство ОЗУ данных и занимают младшие адреса. Хотя физически регистры выделены из памяти данных, такая организация обеспечивает гибкость в работе. Регистры общего назначения прямо связаны с АЛУ. Каждый из регистров способен работать как аккумулятор. Большинство команд выполняются за один такт, при этом из регистров файла могут быть выбраны два операнда, выполнена операция и результат возвращен в регистровый файл. Старшие шесть регистров могут использоваться как три 16-разрядных регистра, и выполнять роль, например, указателей при косвенной адресации.

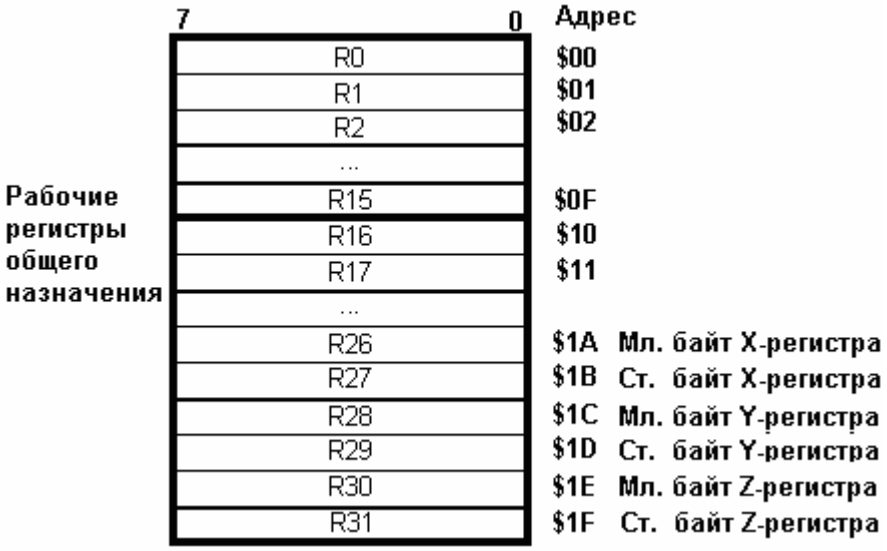


Рис.1.3. Регистры общего назначения микроконтроллера ATmega128

Следующие 64 адреса за регистрами общего назначения занимают регистры ввода-вывода (регистры управления/состояния и данных). В этой области сгруппированы все регистры данных, управления и статуса внутренних программируемых блоков ввода-вывода. При использовании команд IN и OUT используются адреса ввода-вывода с $00 по $3F. Но к регистрам ввода-вывода можно обращаться и как к ячейкам внутреннего ОЗУ. При этом к непосредственному адресу ввода-вывода прибавляется $20. Адрес регистра как ячейки ОЗУ приводится далее в круглых скобках. Регистры ввода-вывода с $00 ($20) по $1F ($3F) имеют программно доступные биты. Обращение к ним осуществляется командами SBI и CBI, а проверка состояния – командами SBIS и SBIC. В таблице В1 приведен список регистров ввода-вывода.

Таблица 1.1. Некоторые регистры ввода-вывода микроконтроллера ATmega128 (желтым цветом выделены регистры явно использующиеся в лабораторной работе).

Название	Функция
PORTG	Регистр данных порта G
DDRG	Регистр направления данных порта G
PING	Выводы порта G
PORTF	Регистр данных порта F
DDRF	Регистр направления данных порта F
SREG	Регистр состояния
SPH	Указатель стека, старший байт
SPL	Указатель стека, младший байт
TIMSK	Регистр маски прерываний от таймеров/счетчиков
TIFR	Регистр флагов прерываний от таймеров/счетчиков
MCUCR	Регистр управления микроконтроллером

MCUCSR	Регистр управления и состояния микроконтроллера
TCCR0	Регистр управления таймером/счетчиком T0
TCNT0	Счетный регистр таймера/счетчика T0
OCR0	Регистр совпадения таймера/счетчика T0
ASSR	Регистр состояния асинхронного режима
TCCR1A	Регистр управления A таймера/счетчика T1
PORTA	Регистр данных порта A
DDRA	Регистр направления данных порта A
PINA	Выводы порта A
PORTB	Регистр данных порта B
DDRB	Регистр направления данных порта B
PINB	Выводы порта B
PORTC	Регистр данных порта C
DDRC	Регистр направления данных порта C
PINC	Выводы порта C
PORTD	Регистр данных порта D
DDRD	Регистр направления данных порта D
PIND	Выводы порта D
SPDR	Регистр данных SPI
SPSR	Регистр состояния SPI
SPCR	Регистр управления SPI
ACSR	Регистр управления и состояния аналогового компаратора
ADMUX	Регистр управления мультиплексором АЦП
ADCSRA	Регистр управления и состояния АЦП
ADCH	Регистр данных АЦП, старший байт
ADCL	Регистр данных АЦП, младший байт
PORTE	Регистр данных порта E
DDRE	Регистр направления данных порта E
PINE	Выводы порта E
PINF	Выводы порта F

4. СХЕМА ПОДКЛЮЧЕНИЯ И ПРОГРАММИРОВАНИЕ, НЕОБХОДИМЫХ В РАБОТЕ, ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

4.1 НАСТРОЙКА ПОРТОВ ВВОДА/ВЫВОДА

Для работы МК с внешними устройствами (клавиатура, светодиод, 8-ми сегментный дисплей, динамик и др.) необходимо задать направление

обмена данными через соответствующие выводы, т.е. настроить их в качестве входов или выходов.

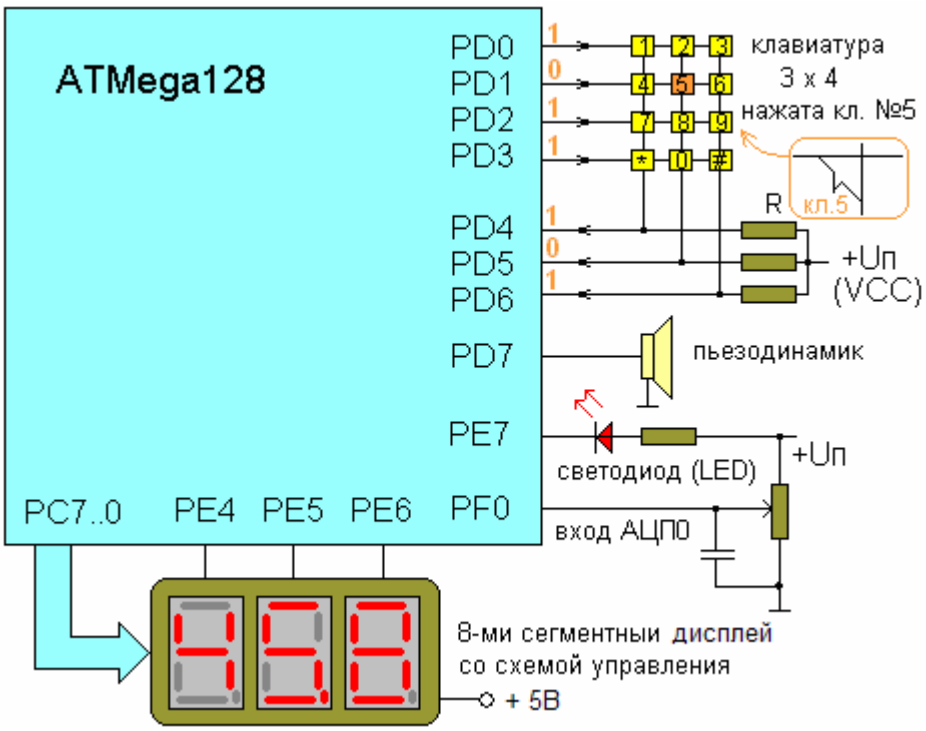


Рис.1.5. Схема подключения периферийных устройств

При подаче напряжения на МК или по сигналу #RESET все порты автоматически настраиваются **на ввод (в регистры направления портов DDRx записаны нули)**, поэтому направление сигналов через некоторые выводы портов необходимо переопределить. Для этого в соответствующие портам регистры направления DDRx нужно записать единицы. Из схемы на рис.1.5 и 1.5.1 видно, что порт PC служит для вывода инверсных 8-ми сегментных кодов, поэтому в регистр направления DDRC (Data Direction Register C) необходимо записать код FF.

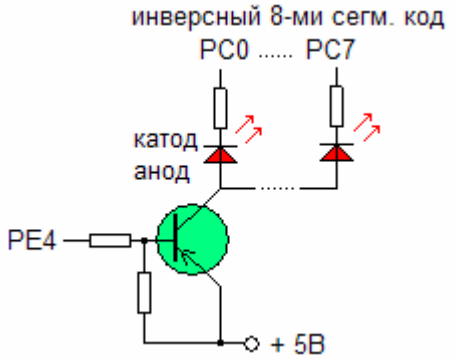


Рис.1.5.1. Схема включения левого 8-ми сегментного индикатора

Четыре старших бита порта PE служат для подачи напряжения (рис. 1.5.1) на аноды трех 8-ми сегментных индикаторов и катод светодиода, поэтому 4 старших бита регистра направления DDRE также д.б. равны единице. 4 младших бита порта PD предназначены для вывода “бегущего нуля” в 4

строки матрицы клавиатуры (3 вывода PD6..4 служат линиями возврата), а по линии порта PD7 выводится сигнал на пьезодинамик, т.е. в биты 3..0 и 7 регистра направления DDRD нужно также записать единицы. Порт PF (АЦП) при включении питания настроен на ввод, поэтому записывать в регистр DDRF нули не обязательно.

4.2 ПРОГРАММИРОВАНИЕ НАПРАВЛЕНИЯ ПОРТОВ ВВОДА/ВЫВОДА

Соответствующий фрагмент программы написанной на Pascal’е будет выглядеть следующим образом:

```
procedure Init_Ports; //== задаем направления передачи данных через порты,
begin                 //== а также начальные значения
  DDRD:=%10001111; //== PD0..PD3 выводим "бегущий 0",
       //== PD4..PD6 - считываем код возврата
       //== линия PORTD.7 подключена к пъезодинамику, поэтому настроим
       //== ее на вывод
  DDRC:=$FF; //== порт C на вывод 8-ми сегментного кода
  DDRE:=%11110000; //== бит7 порта E (LED) на вывод (по RESET'у все
порты настроены на ввод)
       //== 6,5,4 биты на базы транзисторов, коллекторы к общим анодам
       //== индикаторов
  PORTE:=%11111111; //== гасим индикаторы, подавая на базы ключевых
       //== транзисторов единицы (т.е. на аноды светодиодов – нули)
end;
```

4.3 ФОРМИРОВАНИЕ ИНТЕРВАЛОВ ЗАДАННОЙ ДЛИТЕЛЬНОСТИ И СИГНАЛИЗАЦИЯ С ПОМОЩЬЮ СВЕТОДИОДА И ПЬЕЗОДИНАМИКА

Для этой цели может быть выбран любой таймер (см. другие лабораторные работы курса). В нашем примере остановимся на асинхронном режиме работы 8-ми разрядного“таймера 0”, когда импульсы на вход счетчика таймера поступают не от общего источника CLKi/o, а от дополнительного генератора к входам, которого (TOSC1,2) подключен кварцевый резонатор с частотой генерации 32768Гц (2^{15}). Сигнал этого генератора не синхронизирован с сигналом CLKi/o, т.е. является асинхронным. Структурная схема таймера0 (таймера/счетчика 0) приведена на рис.1.6.

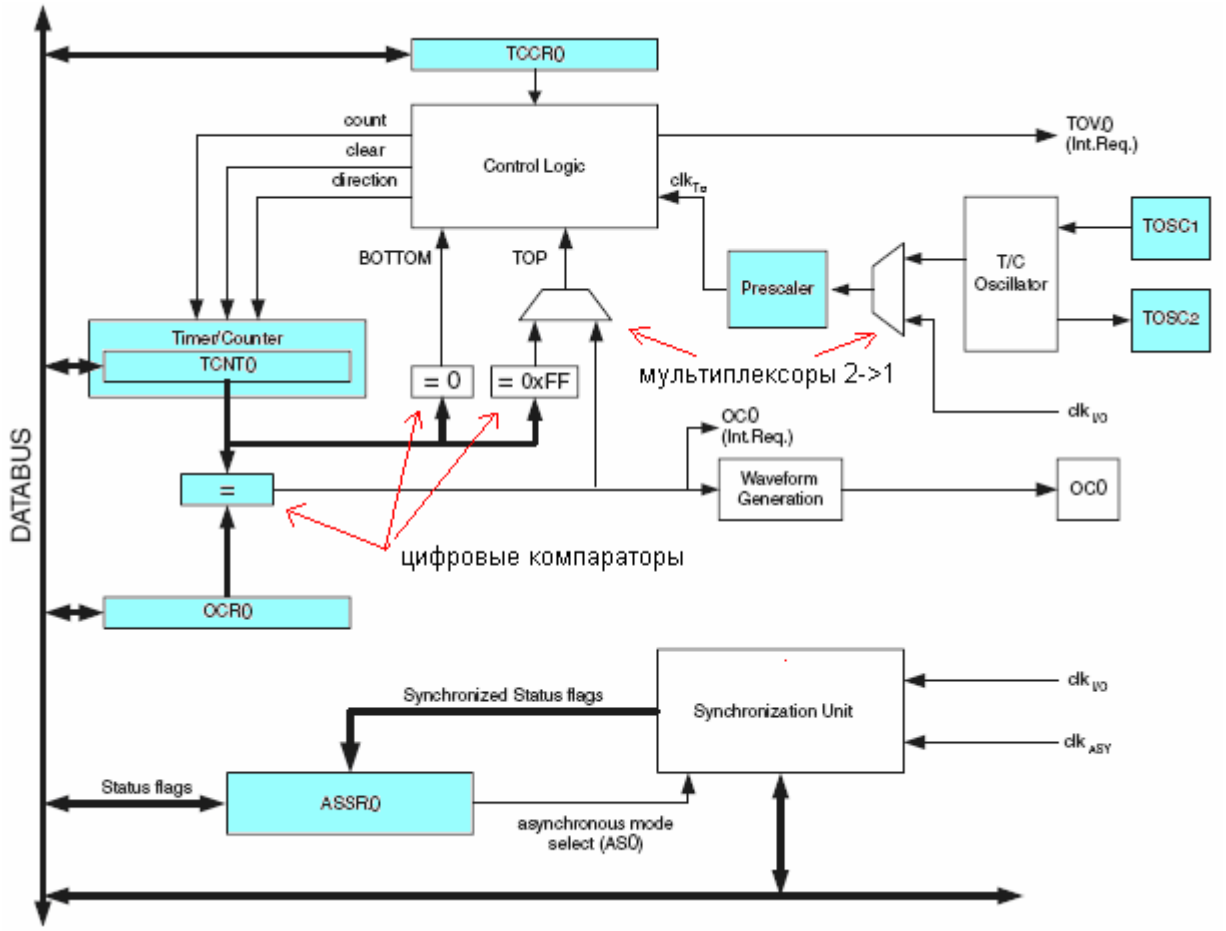


Рис.1.6. Структурная схема таймера0

Режим работы таймера0 определяется определяется информацией, записываемой в регистр управления таймером/счетчиком – TCCR0 (рис.1.6.1.).

Разряд	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Исходное значение	0	0	0	0	0	0	0	0

Рис.1.6.1. Регистр управления таймером/счетчиком - **TCCR0**

Исходные нулевые значения записываются в регистры при включении питания или подаче сигнала #RESET и в дальнейшем программист должен их переопределить в соответствии с задачей.

Номер режима	WGM01	WGM00	Режим работы таймер/счетчика
0	0	0	Нормальный режим
1	0	1	ШИМ с коррекцией фазы
2	1	0	Сброс при совпадении
3	1	1	Быстрый ШИМ

Рис.1.6.2. Биты **WGM00,WGM01** задают режим работы таймер/счетчика.

COM01	COM00	Описание
0	0	Таймер/счетчик отсоединен от выходного вывода OC0
0	1	Состояние выходной линии OC0 меняется на противоположное
1	0	Сброс выходной линии OC0 (установка в состояние 0)
1	1	Установка выходной линии OC0 (установка в состояние 1)

Рис.1.6.3. Биты **COM01, COM00** определяют режим работы блока сравнения.

CS02	CS01	CS00	Источник тактирования в зависимости от бита AS0 в регистре ASSR	
			AS0=0	AS0=1
0	0	0	таймер/счетчик T/C0 остановлен	
0	0	1	СК	TOSC1
0	1	0	СК / 8	TOSC1/8
0	1	1	СК / 32	TOSC1/32
1	0	0	СК / 64	TOSC1/64
1	0	1	СК / 128	TOSC1/128
1	1	0	СК / 256	TOSC1/256
1	1	1	СК / 1024	TOSC1/1024

Рис.1.6.4. Биты **CS02, CS01, CS00** - выбор источника тактового сигнала.

Бит **FOC0** - принудительное изменение состояния вывода OC0. Желтым цветом выделены названия режимов работы и значения разрядов регистра управления TCCR0, используемые в лабораторной работе.

Для формирования секундного интервала частоту 32768Гц можно поделить несколькими способами, например сначала на 128 и затем на 256 (32768/128/256=1Гц). Тогда биты CS02 и CS00 д.б. равны 1 и предделитель (prescaler) на рис.1.6. сформирует сигнал clk_{T0} с частотой 256Гц. Теперь можно поступить двумя способами: или запустить счетчик в нормальном режиме (WGM00=WGM01=0) с модулем счета 256(8 рарядов) или использовать регистр сравнения OCR0 текущего кода счетчика с кодом, записанным в этот регистр. Второй случай предпочтительней, т.к. позволяет использовать различные коэффициенты деления, а не только 256. Выберем второй способ и запишем в регистр сравнения код FF. Счетчик/таймер будет считать от 0 до FF, т.е. его модуль счета будет равен 256. Из сказанного следует, что биты и CS02=CS00=1, CS01=0, а биты WGM00=0 и WGM01=1.

Разряд	7	6	5	4	3	2	1	0
	-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB

Рис.1.6.5. Регистр состояния асинхр. режима таймера/счетчика T/C0 - **ASSR**

Бит AS0*:* Разрешение асинхронного режима таймера/счетчика T/C0. При установленном (= 1) бите на вход предделителя таймера/счетчика 0 поступают импульсы с внешнего кварцевого генератора.

Теперь нужно разрешить прерывания при совпадении кодов OCR0 и TCNT0 (рис.1.6). Для этого нужно записать 1 в бит OCIE0 регистра масок прерываний TIMSK

Разряд	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

Рис.1.6.6. Регистр масок прерываний для таймеров/счетчиков – **TIMSK**.

В соответствии с изложенным процедура инициализации таймера0 будет выглядеть следующим образом:

```
procedure Init_Timer0Async; //== настройка Таймера0 для работы от кварца с
                                //== частотой 32768Гц
begin
  TCCR0:=$0D;//== бит CS00=CS02=1(предделитель=128), WGM01=1(сброс
счетчика при совп.)
  OCR0:=$FF; //== регистр кода совпадения (256-1): 32768/128/256=ровно 1Гц
              //== режим совпадения предпочтительнее режима
              //== переполн., т.к. вместо FF можно выбирать другие значения
  ASSR.3:=1;  //== бит AS0=1 - переходим в асинхронный режим Timer0 от
              //== кварца с резонансной частотой 32768Гц
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при совпадении
   //== текущего значения таймера0 с кодом в регистре совпадения OCR0
end;
```

Переключение светодиода и подача звукового сигнала будут производиться в процедуре обработчике прерывания с символическим именем (адресом) TIMER0COMP:

```
var b: boolean;
var i,t0: byte;
interrupt TIMER0COMP; //== обработчик прерывания при совпадении кодов
//== таймера0 (счетчик TCNT0) и регистра OCR0
begin
  b:=not b;    //== инвертируем бит ‘b’
  PORTE.7:= b; //== и выводим его на катод светодиода
  for t0:=0 to 6 do //== теперь "тикаем"
    sound:=not sound; //== формируем меандр _|`|_|`|_|`|  (инвертируя бит sound)
    PORTD.7:=sound; //== и выводим его на пьезодинамик
    for i:=0 to 255 do endfor; //== длительность полуволны  "тика"
  endfor;
```

end;

Ниже приведена таблица 1.2 некоторых векторов прерываний МК ATMega128 и пример расположения вектора прерывания при совпадении кодов таймера0 по адресу 1E и процедуры обработчика с символическим адресом TIMER0COMP.

Таблица 1.2

Vector No.	Program Address	Source	описание прерывания
1	$0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	$001E	TIMER0 COMP	Timer/Counter0 совпадение кодов
17	$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	преобразование АЦП завершено
23	$002C	EE READY	EEPROM Ready

В момент совпадения кодов счетчика TCNT0 и регистра OCR0 схема управления (Control Logic) на рис.1.6 сформирует признак переполнения TOV0 и запрос на прерывание. МК по запросу автоматически обратится к ячейке с адресом 1E, куда программист (или транслятор) заносит команду перехода JMP TIMER0COMP к процедуре обработки прерывания с символическим адресом TIMER0COMP. Осуществив этот переход МК начнет выполнять подпрограмму interrupt TIMER0COMP.... (см. выше), в которой в нашем примере переключается светодиод и подаются звуковые отметки.

Для чего это нужно? В процессе разработки программы ее размер и относительное положение отдельных ее фрагментов постоянно меняется, поэтому указать сразу окончательное значение физического адреса, соответствующего символическому имени (TIMER0COMP) невозможно. Адрес же ячейки памяти, где будет храниться адрес перехода (или сама команда перехода) определяется структурой МК и известен из справочника. Содержимое такой ячейки(еек) памяти называют вектором прерывания.

На рисунке 1.7 показан фрагмент кода после очередной трансляции и компоновки, из которого видно, что в ячейку памяти с адресом 1E компоновщик поместил команду перехода JMP 03A9h к подпрограмме TIMER0COMP, которая на данном этапе разработки программы располагается по адресу 0003A9. Аналогично действует механизм

прерывания и для других источников, в частности для АЦП (вектор по адресу 2A).

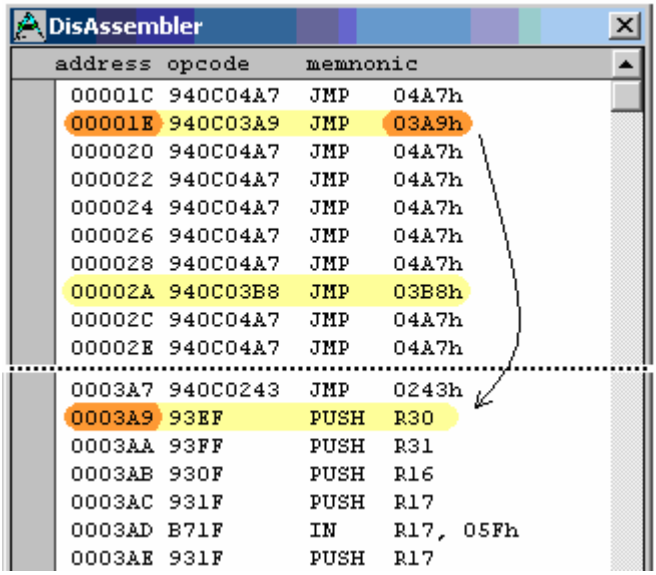


Рис.1.7. Фрагмент кода, сгенерированного компоновщиком

4.4 ПРОГРАММИРОВАНИЕ ВСТРОЕННОГО В МК АЦП

ATmega128 содержит 10-разрядный АЦП последовательного приближения (рис.1.8). Вход АЦП связан с выходом 8-канального аналогового мультиплексора. Эти 8 однополярных каналов АЦП связаны с линиями порта F. Входы АЦП могут объединятся попарно для ввода дифференциальных напряжений. Два дифференциальных входа (ADC1, ADC0 и ADC3, ADC2) содержат каскад со ступенчатым программируемым усилением: 0 дБ (1x), 20 дБ (10x), или 46 дБ (200x). Если выбрано усиление 10x, то точность преобразования ограничивается 8-ю битами, а если 200x, то 7-ю.

АЦП преобразовывает входное аналоговое напряжение в 10-разр. код методом последовательных приближений. Минимальное значение соответствует уровню GND (0), а максимальное уровню AREF равное 2,56 В в лабораторном стенде.

Канал аналогового ввода и каскад дифференциального усиления выбираются путем записи битов MUX в регистре ADMUX. В качестве однополярного аналогового входа АЦП может быть выбран один из входов ADC0…ADC7, а также GND и выход источника опорного напряжения 1,22 В.

Работа АЦП разрешается путем установки бита ADEN в ADCSRA. Выбор опорного источника и канала преобразования не возможно выполнить до установки ADEN.

АЦП генерирует 10-разрядный результат, который помещается в пару регистров данных АЦП ADCH и ADCL. По умолчанию результат преобразования размещается в младших 10-ти разрядах 16-разр. слова

(выравнивание справа), но может быть размещен в старших 10-ти разрядах (выравнивание слева) путем установки бита ADLAR в регистре ADMUX.

Практическая полезность представления результата с выравниванием слева существует, когда достаточно 8-разрядное разрешение, т.к. в этом случае необходимо считать только регистр ADCH. В другом же случае необходимо первым считать содержимое регистра ADCL, а затем ADCH, чем гарантируется, что оба байта являются результатом одного и того же преобразования. Как только выполнено чтение ADCL блокируется доступ к регистрам данных со стороны АЦП. Это означает, что если считан ADCL и преобразование завершается перед чтением регистра ADCH, то ни один из регистров не может модифицироваться и результат преобразования теряется. После чтения ADCH доступ к регистрам ADCH и ADCL со стороны АЦП снова разрешается.

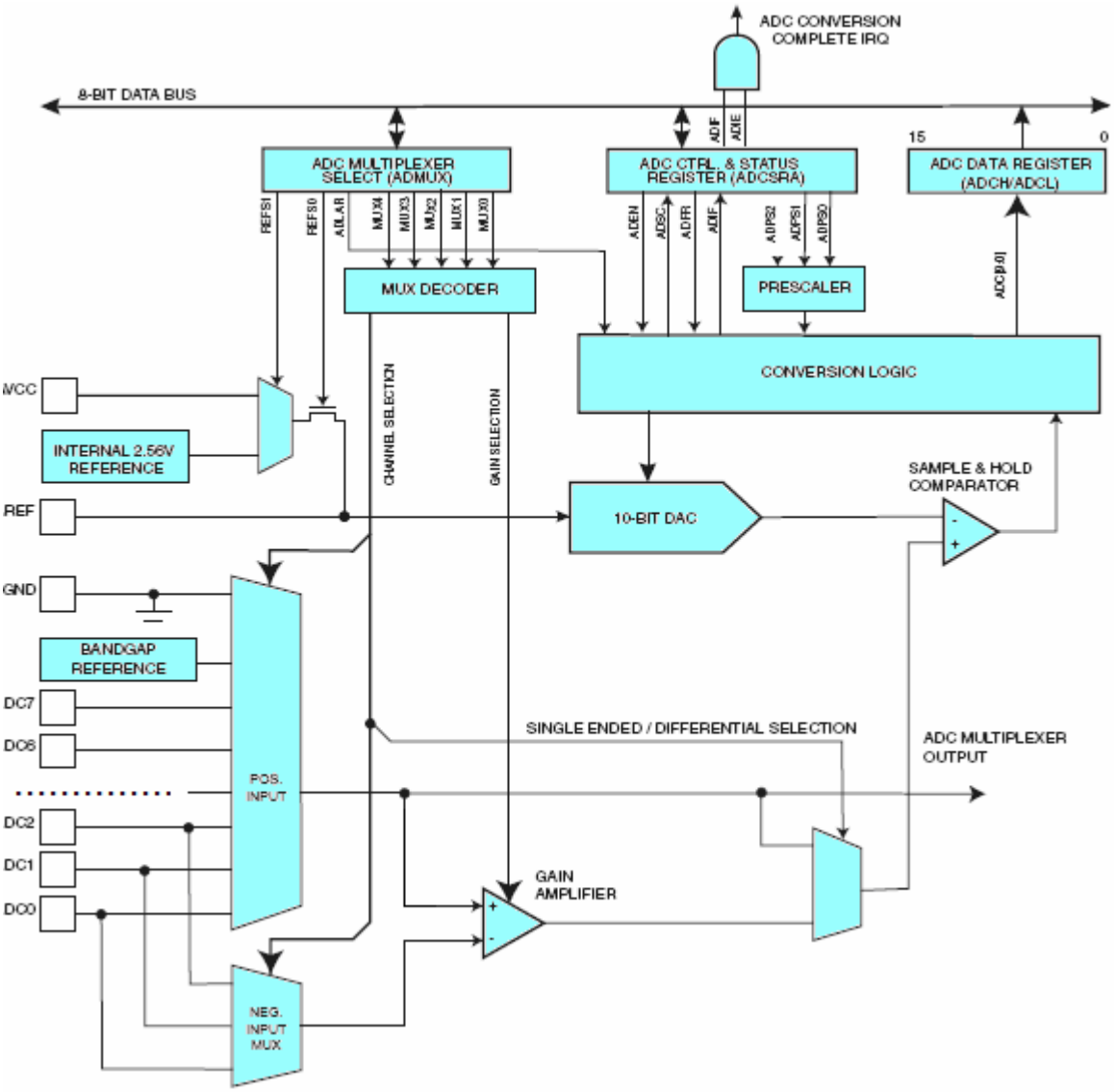


Рис.1.8. Структурная схема АЦП

Одиночное преобразование запускается путем записи лог. 1 в бит запуска преобразования АЦП ADSC. Данный бит остается в высоком состоянии в процессе преобразования и сбрасывается по завершении

преобразования. Если в процессе преобразования переключается канал аналогового ввода, то АЦП автоматически завершит текущее преобразование прежде, чем переключит канал.

В режиме автоматического перезапуска АЦП непрерывно оцифровывает аналоговый сигнал и обновляет регистр данных АЦП. Данный режим задается путем записи лог. 1 в бит ADFR регистра ADCSRA. Первое преобразование инициируется путем записи лог. 1 в бит ADSC регистра ADCSRA. В данном режиме АЦП выполняет последовательные преобразования, независимо от того сбрасывается ли флаг прерывания АЦП ADIF.

4.4.1 РЕГИСТРЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ АЦП

Регистр управления и статуса (состояния) АЦП – **ADCSRA** приведен на рис.1.9.

Разряд	7	6	5	4	3	2	1	0
Название	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Чтение/запись	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З
Исх. значение	0	0	0	0	0	0	0	0

Рис.1.9. Регистр управления и состояния **ADCSRA**

Бит **ADEN**: Разрешение работы АЦП. Запись в данный бит лог. 1 разрешает работу АЦП. Если в данный бит записать лог. 0, то АЦП отключается, даже если он находился в процессе преобразования.

Бит **ADSC**: Запуск преобразования АЦП. В режиме одиночного преобразования установка данного бита начинает преобразование. В режиме автоматического перезапуска установкой этого бита инициируется только первое преобразование, а все остальные выполняются автоматически. Первое преобразование после разрешения работы АЦП, инициированное битом ADSC, выполняется по расширенному алгоритму и длится 25 тактов синхронизации АЦП, вместо обычных 13 тактов. Это связано с необходимостью инициализации АЦП.

Бит **ADFR**: Выбор режима автоматического перезапуска АЦП. Если в данный бит записать лог. 1, то АЦП перейдет в режим автоматического перезапуска. В этом режиме АЦП автоматически выполняет преобразования и модифицирует регистры результата преобразования через фиксированные промежутки времени. Запись лог. 0 в этот бит прекращает работу в данном режиме.

Бит **ADIF**: Флаг прерывания АЦП. Данный флаг устанавливается после завершения преобразования АЦП и обновления регистров данных. Если

установлены биты ADIE и I (регистр SREG), то происходит прерывание по завершении преобразования. Флаг ADIF сбрасывается аппаратно при переходе на соответствующий вектор прерывания. Альтернативно флаг ADIF сбрасывается путем записи лог. 1 в него. Обратите внимание, что при выполнении команды "чтение-модификация-запись" с регистром ADCSRA ожидаемое прерывание может быть отключено. Данное также распространяется на использование инструкций SBI и CBI.

Бит **ADIE**: Разрешение прерывания АЦП. После записи лог. 1 в этот бит, при условии, что установлен бит I в регистре SREG, разрешается прерывание по завершении преобразования АЦП.

Биты **ADPS**2..0: Биты управления предделителем (Prescaler) АЦП. Данные биты определяют на какое значение будет поделена тактовая частота МК перед подачей на вход синхронизации АЦП. Значения коэффициентов деления указаны в таблице 1.3. Если требуется максимальная разрешающая способность (10 разрядов), то частота синхронизации должна быть в диапазоне 50…200 кГц. Если достаточно разрешение менее 10 разрядов, но требуется более высокая частота преобразования, то частота на входе АЦП может быть установлена свыше 200 кГц.

Таблица 1.3 – Управление предделителем АЦП

ADPS2	ADPS1	ADPS0	Коэффициент деления
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Разряд	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Чт./зап.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.

Рис.1.10. Регистр управления Регистр мультиплексором АЦП – **ADMUX**

Бит **REFS**1..0: Биты выбора источника опорного напряжения. Данные биты определяют, какое напряжение будет использоваться в качестве опорного для

АЦП. При работе с лабораторным стендом ЛС-2 нужно использовать только внешнее опорное напряжение (биты REFS0 и REFS1 должны быть равны 0).

Таблица 1.4 – Выбор опорного источника АЦП

REFS1	REFS0	Опорный источник
0	0	AREF, внутренний ИОН отключен
0	1	AVCC с внешним конденсатором на выводе AREF
1	0	Зарезервировано
1	1	Внутренний источник опорного напряжения 2.56В с внешним конденсатором на выводе AREF

Бит **ADLAR**: Бит управления представлением результата преобразования. Бит ADLAR влияет на представление результата преобразования в паре регистров результата преобразования АЦП. Если ADLAR = 1, то результат преобразования будет иметь левосторонний формат (старший значащий бит занимает место старшего бита старшего байта результата, т.е. сдвинуты влево до упора), в противном случае - правосторонний. Действие бита ADLAR вступает в силу сразу после изменения, независимо от выполняющегося параллельно преобразования.

Биты **MUX**4..0: Биты выбора аналогового канала и коэффициента усиления. Данные биты определяют, какие из имеющихся аналоговых входов подключаются к АЦП. Кроме того, с их помощью можно выбрать коэффициент усиления для дифференциальных каналов (см. табл. 1.5). Если значения бит изменить в процессе преобразования, то механизм их действия вступит в силу только после завершения текущего преобразования (после установки бита ADIF в регистре ADCSRA).

Таблица 1.5 – Выбор входного канала

MUX4..0	Однополярный вход
00000	ADC0
……	……
00111	ADC7

Регистры результата преобразования **– ADCL** и **ADCH**

При **ADLAR** = 0**:**

Разряд	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	ADC9	ADC8	**ADCH**
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	**ADCL**
	7	6	5	4	3	2	1	0	

При **ADLAR** = 1:

Разряд	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	**ADCH**
	ADC1	ADC0	-	-	-	-	-	-	**ADCL**
	7	6	5	4	3	2	1	0	

По завершении преобразования результат помещается в этих двух регистрах.

Левосторонний формат ADLAR=1 представления результата удобно использовать, если достаточно 8 разрядов. В этом случае 8-разрядный результат хранится в регистре ADCH и, следовательно, чтение регистра ADCL можно не выполнять. При правостороннем формате необходимо сначала считать ADCL, а затем ADCH.

4.4.2 ПРОГРАММИРОВАНИЕ АЦП

Прежде всего необходимо разрешить работу АЦП в режиме прерывания, т.е. записать в биты ADEN и ADIE единицу. Для 10-ти битного разрешения АЦП необходимо выбрать значения частоты его синхронизации. Т.к. она должна в этом случае находиться в диапазоне 50..200КГц, то при общей частоте синхронизации МК равной 6МГц коэффициент деления можно взять равным 64 (6000000/64 = 93750Гц) и биты ADSP2, ADSP1, ADSP0=110. Таким образом, в программе при инициализации АЦП необходимо записать ADCSRA=8E.

Из рис.1.5 видно, что источник сигнала (средний вывод потенциометра) подключен к нулевому входу ADC0 (вывод PF0), поэтому биты MUX4..0 регистра управления ADMUX должны быть равны нулю. В этом же регистре положим ADLAR=1, т.е. выравняем результат влево. В результате в регистр ADMUX необходимо записать код 20h. Запуск преобразования АЦП производится записью 1 в бит ADSC регистра ADCSRA в теле основного блока программы и в самом обработчике для нового запуска.

Также в теле основного блока программы необходимо разрешить прерывания записью единицы в бит **I** регистра состояния (флагов) SREG.

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C

Рис.1.11. Регистр состояния **SREG.**

Бит **I** – Разрешение всех прерываний. Для разрешения прерываний этот бит должен быть установлен (I=1). Разрешение конкретного прерывания выполняется регистрами маски прерывания EIMSK и TIMSK. Если этот бит очищен (I=0), то ни одно из прерываний не обрабатывается. Бит аппаратно очищается после возникновения прерывания и устанавливается (разрешая последующие прерывания) командой ассемблера RETI. В программе на Pascal'е этот бит устанавливается командой EnableInts;

Процедура инициализации АЦП и обработчик прерывания при завершении АЦ преобразования могут выглядеть следующим образом:

```
procedure Init_ADC; //== настройка АЦП
begin
  ADCSRA:=$8E; //== настройка АЦП  делитель прескалера = 64
                    //== (6000000/64=93750Гц - 50..100000)
  ADMUX:=$20;//== бит ADLAR=1-биты АЦП выравнены влево и читать
                  //== можно один ADCH
end;

interrupt ADCRDY; //== обработчик прерывания при окончании АЦ
                      //== преобразования
begin
  adccode:= word(ADCH); //== читаем только ст. байт кода АЦП
                            //== (т.е. 8-ми битный АЦП)
  ADCSRA.6:=1; //== новый запуск АЦП (бит ADSC=1)
end;
```

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

5.1 СОЗДАНИЕ ШАБЛОНА ПРОГРАММЫ

Откройте интегрированную среду разработки E-LAB PED32 для МК семейства AVR. В открывшемся рабочем пространстве из главного меню выберите п. "File | New Project". В появившемся окне диалога на странице "New-Edit-Account" заполните отмеченные стрелками поля (вместо папки gr8888 создайте папку с номером СВОЕЙ группы), затем нажмите на кнопки "OK" и "Save" и "Exit" (рис.1.12).

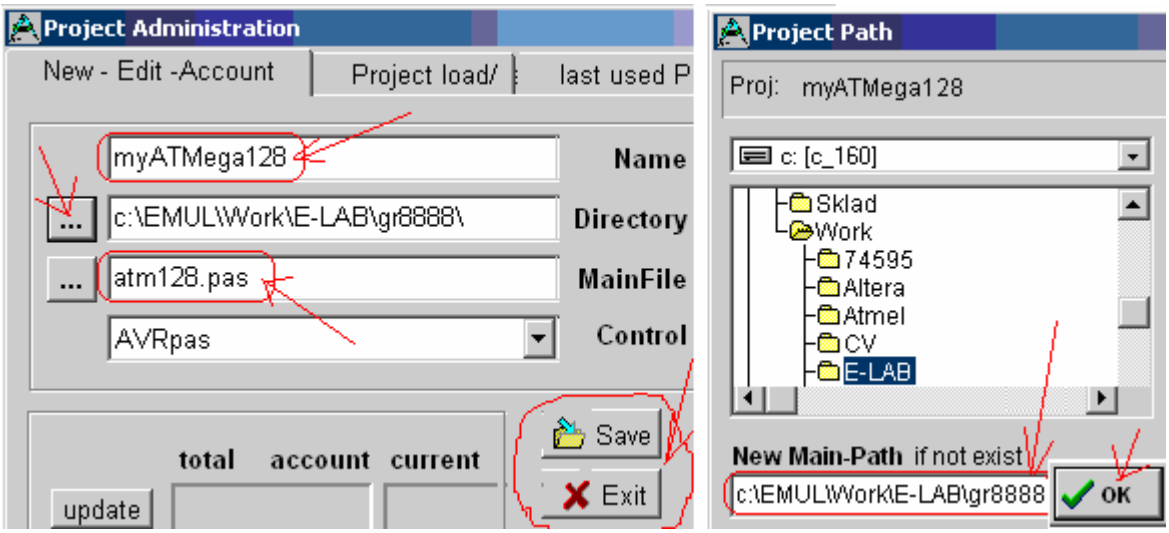


Рис. 1.12. Создание рабочей папки

Появится первая страница "мастера или генератора шаблона программы" (рис.1.13).

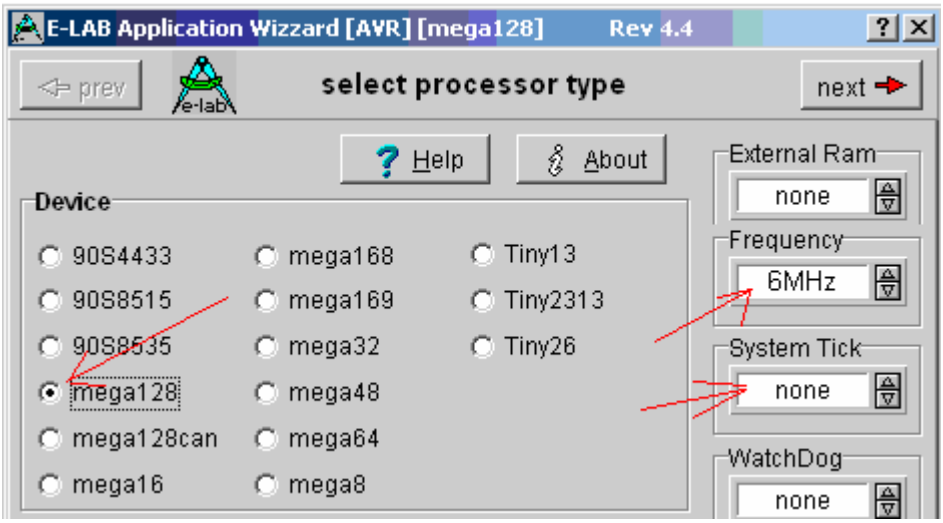


Рис.1.13. Генератор шаблонов программы

Выберем тип микроконтроллера "mega128", частоту задающего генератора и отметим отсутствие программного системного таймера "System Tick --> none".После заполнения полей жмем на кнопку "next" в правом верхнем углу. Проскакиваем все остальные страницы генератора шаблонов с помощью той же кнопки "next" и только на последней 16-ой странице жмем на кнопку 

и затем на "Store" и "Exit".

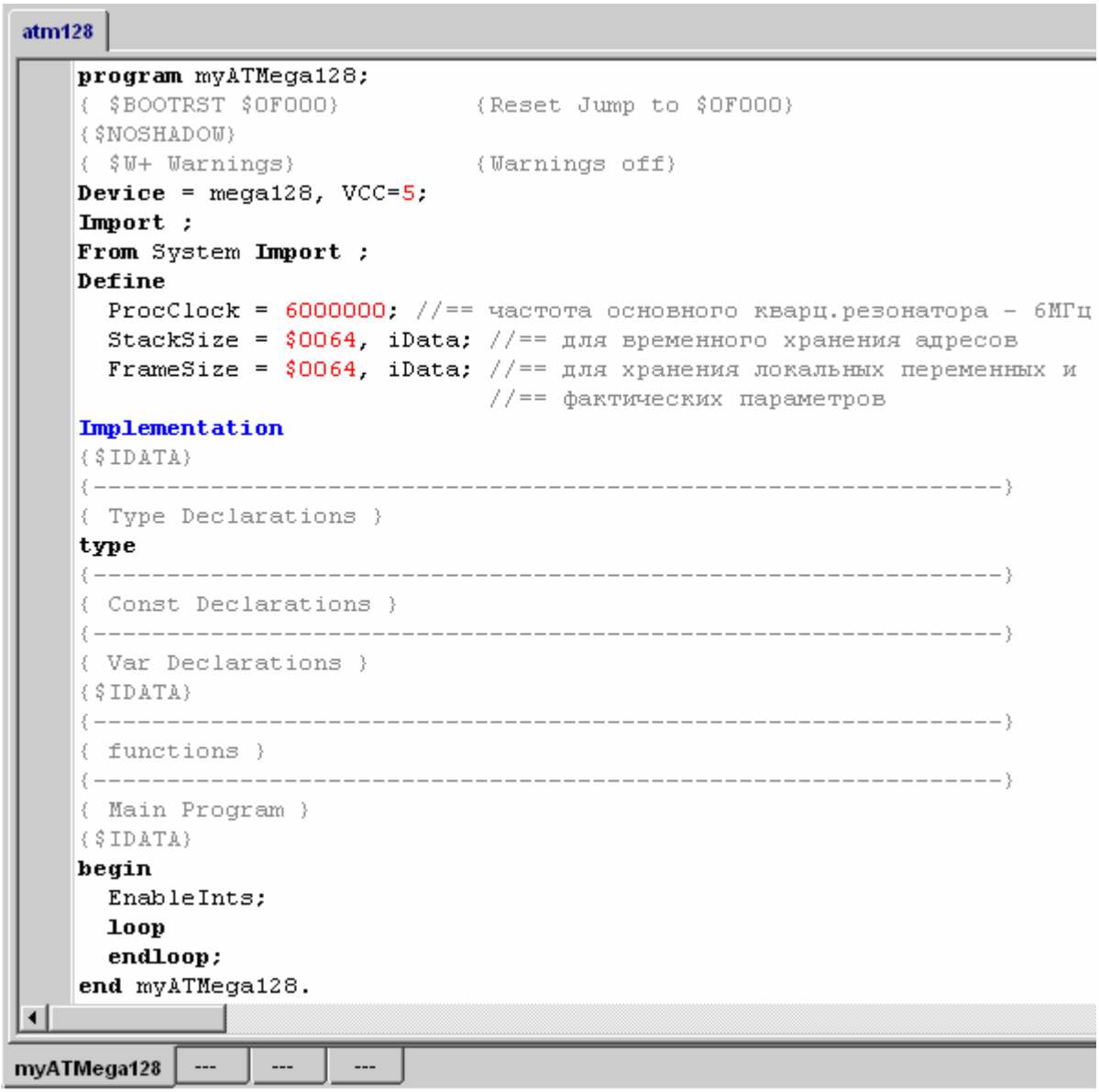


Рис.1.14. Шаблон программы

В окне редактора E-LAB PED32 появится сконструированный "Wizard'ом" шаблон нашей программы на языке "Pascal" (рис.1.14).

Теперь необходимо добавить операторы, обеспечивающие функциональность нашей программы в соответствие с техническим заданием.

5.2 РАЗРАБОТКА И ОТЛАДКА ПРОГРАММЫ

Если окно редактора открыто - продолжайте работу. Если нет, снова запустите приложение E-LAB PED32  . В открывшемяся рабочем пространстве из главного меню выберите п. "Project | Load Project". В

появившемся окне диалога на странице "Project load/delete" выберите проект "myATMega128" и нажмите на кнопку "Load" (рис.1.15) .

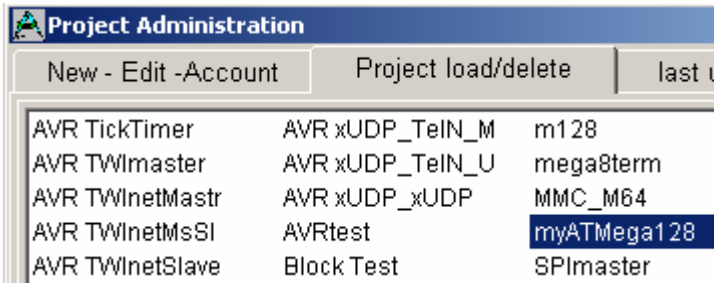


Рис.1.15. Окно управления проектом

В редакторе должен появиться исходный текст программы. Если текст не появился, выберите п. меню "File | Open MainFile". Теперь текст программы появится.

5.2.1 ИНИЦИАЛИЗАЦИЯ ПОРТОВ ВВОДА/ВЫВОДА

Добавим в программу операторы, инициализирующие порты ввода/вывода в соответствии с нашими задачами (рис.1.16).

```
{ functions }
{-------------------------------------------------------------}
procedure Init_Ports; //== задаем направления передачи данных
begin                 //== через порты, а также начальные значения
  DDRD:=%10001111;//== PD0..PD3 выводим "бегущий 0",
                  //==  PD4..PD6 считываем код возврата
                  //== линия PORTD.7 подключена к пьезодинамику,
                  //== поэтому настроим ее на вывод
   DDRC:=$FF; //== порт C на вывод 8-ми сегментного кода
   DDRE:=%11110000;//== бит7 порта E(LED)на вывод(по RESET'у все
                   //==  порты настроены на ввод)
                   //== 6,5,4 биты на базы транзисторов
   PORTE:=%11111111;//== гасим индикаторы,
                    //== подавая на базы транзисторов 111
end;
{ Main Program }
{$IDATA}
begin
  Init_Ports;
  EnableInts;
```

Рис.1.16. Инициализация портов

Для этого, как показано на рисунке 1.16, нужно вписать отмеченные фрагменты в указанные места программы.

Компилируем, полученную на этом этапе программу, для чего в редакторе E-LAB PED32 нажмем на кнопку “Make Project”  . Если ошибок при наборе программы не было, то компиляция закончится без предупреждающего сообщения и в статусной строке справа внизу появится

сообщение- 

. ВАЖНОЕ ПРИМЕЧАНИЕ! После каждой редакции текста выполняйте сборку проекта кнопкой  . Иначе выявить все накопившиеся ошибки будет трудней.

5.2.2 НАСТРОЙКА ТАЙМЕРА “0”

Теперь добавим в программу операторы, настраивающие таймер0 для формирования временных отрезков длительностью 1сек. и осуществляющие сигнализацию светодиодом.

```
  PORTE:=%11111111;//== гасим индикаторы
end;
procedure Init_Timer0Async;//== настройка Таймера0 для работы
begin                       //== от кварца 32768Гц
  TCCR0:=$0D;//== бит CS00=CS02=1(предделитель=128),WGM01=1
             //== (сброс счетчика при совп.)
  OCR0:=$FF; //== регистр кода совпадения (256-1):
             //== 32768/128/256=ровно 1Гц
  ASSR.3:=1; //== бит AS0=1 - переходим в асинхронный режим
             //== Timer0 от кварца 32768Гц
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания
             //== при совпадении текущего значения таймера0
             //== с кодом в регистре совпадения OCR0
end;
{ Main Program }
{$IDATA}
begin
  Init_Ports;
  Init_Timer0Async;
  EnableInts;
```

Рис.1.16. Настройка таймера0

Далее необходимо добавить процедуру – обработчик прерывания, которая будет вызываться ровно через 1 секунду. В процедуре TIMER0COMP предусмотрим управление светодиодом (рис.1.17).

```
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при
             //== совпадении текущего значения таймера0
end;         //== с кодом в регистре совпадения OCR0
var b: boolean;
interrupt TIMER0COMP;//== обработчик прерывания при
                     //== переполнении таймера0 ровно через
            //== 1 сек от TOSC=32768Гц в асинхронном режиме
begin
  b:=not b;     //== мигаем
  PORTE.7:= b; //== светодиодом
end;
{ Main Program }
{$IDATA}
```

Рис.1.17. Обработчик прерывания при совпадении кодов

Снова компилируем, полученную на этом этапе программу  . В этот момент она должна иметь следующий вид:

```
program myATMega128;
{ $BOOTRST $0F000}        {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}           {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000; //== частота основного кварц.резонатора - 6МГц
  StackSize = $0064, iData; //== для временного хранения адресов возврата и
др.
  FrameSize = $0064, iData; //== для хранения локальных переменных и
                                //== фактических параметров
Implementation
{$IDATA}
{-------------------------------------------------------------}
{ Type Declarations }
type
{-------------------------------------------------------------}
{ Const Declarations }
{-------------------------------------------------------------}
{ Var Declarations }
{$IDATA}
{-------------------------------------------------------------}
{ functions }
{-------------------------------------------------------------}
procedure Init_Ports; //== задаем направления передачи данных через порты,
begin                 //== а также начальные значения
  DDRD:=%10001111;//== PD0..PD3 выводим "бегущий 0", PD4..PD6
считываем код возврата
                //== линия PORTD.7 подключена к пъезодинамику, поэтому
настроим ее на вывод
  DDRC:=$FF; //== порт C на вывод 8-ми сегментного кода
  DDRE:=%11110000;//== бит7 порта E (LED) на вывод(по RESET'у все
порты настроены на ввод)
                    //== 6,5,4 биты на базы транзисторов, коллекторы к общим
анодам индикаторов
  PORTE:=%11111111;//== гасим индикаторы, подавая на их аноды нули
через инверторы
end;
```

```
procedure Init_Timer0Async;//== настройка Таймера0 для работы от кварца
32768Гц
begin
  TCCR0:=$0D;//== бит CS00=CS02=1(предделитель=128),WGM01=1(сброс
счетчика при совп.)
  OCR0:=$FF; //== регистр кода совпадения (256-1): 32768/128/256=ровно 1Гц
!
                  //== режим совпадения предпочтительнее режима
переполнения, т.к. вместо FF
                  //== можно выбирать другие значения
  ASSR.3:=1; //== бит AS0=1 - переходим в асинхронный режим Timer0 от
кварца 32768Гц
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при совпадении
текущего
end; //== значения таймера0 с кодом в регистре совпадения OCR0

var b: boolean;
interrupt TIMER0COMP;//== обработчик прерывания при переполнении
таймера0
                              //==  ровно через 1 сек от TOSC=32768Гц в
асинхронном режиме
begin
  b:=not b;          //== мигаем
  PORTE.7:= b; //== светодиодом
end;
{ Main Program }
{$IDATA}
begin

  Init_Ports;
  Init_Timer0Async;
  EnableInts;//== разрешить прерывания
  loop
  endloop;
end myATMega128.
```

5.2.3 ЗАГРУЗКА ПРОГРАММЫ ВО ФЛЭШ ПАМЯТЬ МК

Проверим ход выполнения промежуточного варианта программы, для чего запишем ее во флэш память микроконтроллера ATMega128.

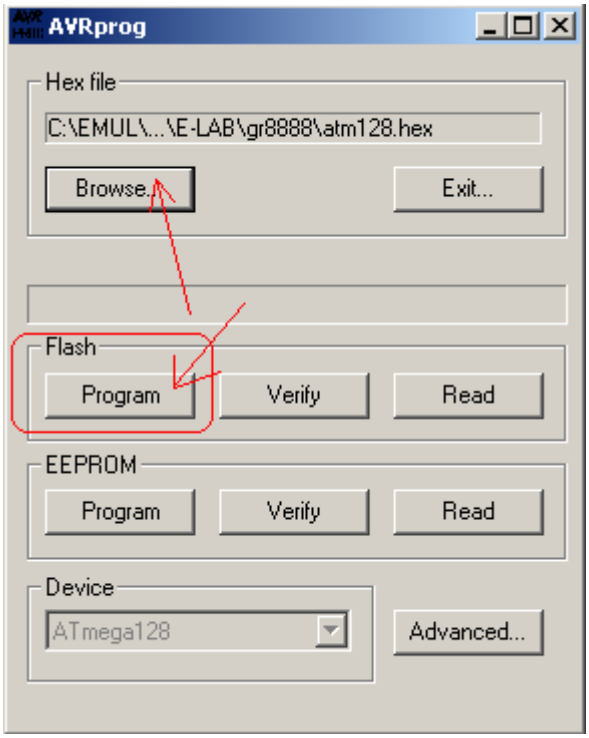


Рис.1.18. Панель программатора AVRprog

В процессе компиляции программная оболочка E-LAB PED32 создает файл с выполнимым кодом программы в специальном интеловском HEX-формате с расширением *.hex (в нашем примере это файл - atm128.hex). Этот файл должен быть доставлен на плату с микроконтроллером и записан в его флэш память. Для этой цели служит специальный кабель с программатором и свободно распространяемая программа AVRprog (рис.1.18).

Запустите на выполнение программу AVR PROG. Затем кнопкой “Browse..” найдите загрузочный файл “C:\EMUL\Work\E-LAB\gr8888\atm128.hex” и нажмите кнопку Flash -> Program. Шкала прогресса покажет момент окончания загрузки (рис.1.19). Одновременно начнет выполняться записанная во Flash память программа и светодиод начнет “мигать”, как и было задумано. Результат покажите преподавателю.

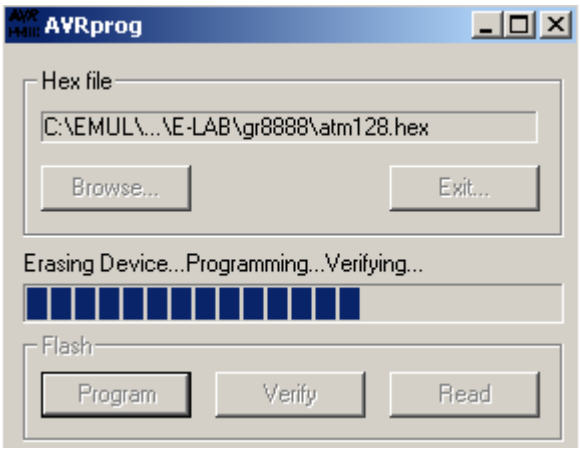


Рис.1.19. Шкала прогресса

5.2.4 ПОДКЛЮЧЕНИЕ КЛАВИАТУРЫ И 8-МИ СЕГМЕНТ. ДИСПЛЕЯ

Следующим усовершенствованием нашей программы будет работа с клавиатурой и отображение цифровых кодов нажатых клавиш на 8-ми сегментном дисплее. Для этого, во-первых, добавим в программу ASCII коды нажатых клавиш (“01…9”) и их 8-ми сегментные эквиваленты. Введем в программу переменные **var**, которые потребуются для дальнейшей работы (рис.1.20). Также введем подпрограмму-функцию однократного сканирования клавиатуры ScanKeyOnce (рис. 1.20-1) и в основном блоке программы добавим операторы проверки фактов нажатия и отпускания клавиши с последующим отображением кода на 8-ми сегментном индикаторе (рис. 1.20-2).

```
{ Const Declarations }
const keyASCII: string ='123456789*0#';//== ASCII коды клавиш
const key8segm: string = #$F9+#$A4+#$B0+#$99+#$92+#$82+#$F8+
//==                        1    2    3    4    5    6    7
                          #$80+#$90+#$7F+#$C0+#$B6;
//==                         8    9    *    0    #
//== инверсные 8-ми сегментные коды клавиш
{-------------------------------------------------------------
{ Var Declarations }
{$IDATA}
var sound :boolean;
    s:string[5];//== для хранения 8-ми сегм. цифр
    maskind,j,key,kn,e2temp: byte;
    adccode:word;
{-------------------------------------------------------------
{ functions }
```

Рис.1.20. Глобальные константы и переменные

```
{ functions }
{-------------------------------------------------------------}
function ScanKeyOnce (var kn: byte):boolean;//== однокр. сканир. клав-ры
var row,col: byte;  //== var kn - параметр переменная № клавиши
var delay:word;
begin
  for col:=0 to 3 do //== сканируем по столбцам матрицы клавиатуры
    PORTD:=not(1 shl col);//== 11111110,11111101,11111011,11110111 бегущий0
    for row:=0 to 2 do //== сканируем по строкам матрицы клавиатуры
      if ((PIND and ($10 shl row))=0) then //== если обнаружен ноль, то
        kn := col*3+row;//== вычисляем порядковый № клавиши (0..11)
        for delay:=0 to 5000 do endfor;//== задержка на дребезг контактов
        return (true); //== возвращаем бит "обнаружено нажатие клавиши"
      endif;
    endfor;
  endfor;
  return (false); //== возвращаем бит "ни одна из клавиш не нажата"
end;
procedure Init_Ports; //== задаем направления передачи данных через порты,
```

Рис.1.20-1 Функция однократного сканирования клавиатуры.

```
  EnableInts;
  loop //== бесконечный цикл
    while not ScanKeyOnce(kn) do endwhile;//== ожидаем нажатия на клавишу
    key:=byte(keyASCII[kn+1]);//== +1,т.к.в 0-м байте-длина строки!
    while ScanKeyOnce(kn)do endwhile;//== ожидаем отпускание клавишы
    case key of //== выполняем действия в соотв-ии с кодом клавишы(0..9)
      '0'..'9':
        PORTE:=%10111111;//== подаем на анод правого инд-ра напряж-е
        PORTC:=byte(key8segm[kn+1]);//== выводим 8-ми сегм.код клавиши
                                    //== на LED индикатор
      | //== этот разделитель обязателен
    endcase;
  endloop;
end myATMega128.
```

Рис.1.22. Отображение кода нажатой клавиши на ЖКД

Снова компилируем, полученную на этом этапе программу  . В этот момент она должна иметь следующий вид

```
program myATMega128;
{ $BOOTRST $0F000}        {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}           {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000; //== частота основного кварц.резонатора - 6МГц
  StackSize = $0064, iData; //== для времен. хранения адресов возврата и др.
  FrameSize = $0064, iData; //== для хранения локальных переменных и
                    //== фактических параметров
Implementation
{$IDATA}
{-------------------------------------------------------------}
{ Type Declarations }
type
{-------------------------------------------------------------}
{ Const Declarations }
{-------------------------------------------------------------}
const keyASCII: string = '123456789*0#';//== ASCII коды клавиш/цифр
const key8segm: string = #$F9 + #$A4 + #$B0 + #$99 + #$92 + #$82 + #$F8 +
#$80 + #$90 + #$7F + #$C0 + #$B6; //== инверсные 8-ми сегм. коды цифр
{ Var Declarations }
{$IDATA}
var sound :boolean;
   s:string[5];//== для хранения 8-ми сегм. Цифр
   maskind,j,key,kn,e2temp: byte;
   adccode:word;
```

```
{------------------------------------------------------------}
{ functions }
{------------------------------------------------------------}
function ScanKeyOnce (var kn: byte):boolean;//== однокр. сканирование клав-
ры
var row,col: byte;    //== var kn - параметр переменная № клавиши
var delay:word;
begin
  for col:=0 to 3 do //== сканируем по столбцам матрицы клавиатуры
    PORTD:=not(1 shl col);//== 11111110,11111101,11111011,11110111 бегущий
                          //== ноль)
    for row:=0 to 2 do //== сканируем по строкам матрицы клавиатуры
      if ((PIND and ($10 shl row))=0) then //== если обнаружен ноль, то
        kn := col*3+row;//== вычисляем порядковый № клавиши (0..11)
        for delay:=0 to 5000 do endfor;//== задержка на дребезг контактов
        return (true); //== возвращаем бит "обнаружено нажатие клавиши"
      endif;
    endfor;
  endfor;
  return (false); //== возвращаем бит "ни одна из клавиш не нажата"
end;
procedure Init_Ports; //== задаем направления передачи данных через порты,
begin                 //== а также начальные значения
  DDRD:=%10001111; //== PD0..PD3 выводим "бегущий 0", PD4..PD6
                   //== считываем код возврата
    //== линия PORTD.7 подключена к пъезодинамику, поэтому настроим ее
    //== на вывод
  DDRC:=$FF; //== порт C на вывод 8-ми сегментного кода
  DDRE:=%11110000; //== бит7 порта E (LED) на вывод(по RESET'у все
                   //== порты настроены на ввод)
    //== 6,5,4 биты на базы транзисторов, коллекторы к общим анодам
  //== индикаторов
  PORTE:=%11111111; //== гасим индикаторы, подавая на их аноды нули
                    //== через инверторы
end;
procedure Init_Timer0Async; //== настройка Таймера0 для работы от кварца
                            //== 32768Гц
begin
  TCCR0:=$0D; //== бит CS00=CS02=1(предделитель=128),WGM01=1(сброс
              //== счетчика при совп.)
  OCR0:=$FF; //== регистр кода совпадения (256-1): 32768/128/256=ровно 1Гц
  //== режим совпадения предпочтительнее режима переполнения, т.к. вместо
  //== FF можно выбирать другие значения
  ASSR.3:=1; //== бит AS0=1 - переходим в асинхронный режим Timer0 от
             //== кварца 32768Гц
```

```
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при совпадении
//== текущего значения таймера0 с кодом в регистре совпадения OCR0
end;
var b: boolean;
interrupt TIMER0COMP; //== обработчик прерывания при переполнении
                                  //== таймера0
                  //== ровно через 1 сек от TOSC=32768Гц в асинхронном режиме
begin
  b:=not b;    //== мигаем
  PORTE.7:= b; //== светодиодом
end;
{ Main Program }
{$IDATA}
begin
  Init_Ports;
  Init_Timer0Async;
  EnableInts; //== разрешить прерывания
  loop //== бесконечный цикл
    while not ScanKeyOnce(kn) do endwhile;//== ожидаем нажатия на клавишу
    key:=byte(keyASCII[kn+1]); //== +1, т.к. в паскале в 0-м байте -длина
                                        //== строки!
    while ScanKeyOnce(kn) do endwhile;//== ожидаем отпускание клавишы
    case key of //== выполняем действия в соотв-ии с нажатой клавишей
                  //== (#,*,0..9)
      '0'..'9':
        PORTE:=%10111111; //== подаем на анод правого 8-ми сегм. инд-ра
                                    //== напряж-е
        PORTC:=byte(key8segm[kn+1]); //== выводим 8-ми сегм. код нажатой
                                              //== клавиши на LED индикатор

      | //== этот разделитель '|' обязателен
    endcase;
  endloop;
end myATMega128.
```

Снова запустите на выполнение программу AVR PROG и кнопкой Flash -> Program запишите усовершенствованный код программы в ПЗУ. Затем, нажимая на клавиши 0,1,2…9 удостоверьтесь в том, что соответствующие цифры отображаются на LED индикаторе. Результат покажите преподавателю.

5.2.5 ПРОГРАММИРОВАНИЕ АЦП

Введем в программу код, позволяющий измерять напряжение на выходе датчика с помощью АЦП (рис.1.22). В качестве датчика используется потенциометр, поэтому код на выходе АЦП пропорционален углу поворота рукоятки и в этом качестве можно считать, что АЦП измеряет не просто напряжение, а угол поворота. Для начала запишем в программу две процедуру обработчик прерывания, возникающего при завершении цикла преобразования и процедуру настройки АЦП на заданный режим работы.

```
  b:=not b;     //== мигаем
  PORTE.7:= b; //== светодиодом
end;
procedure Init_ADC; //== настройка АЦП
begin
  ADMUX:=$20;//== бит ADLAR=1-биты АЦП выравнены влево и читать можно
             //== один ADCH
  ADCSRA:=$8E; //== настройка АЦП  делитель (предделителя)прескалера = 64
               //== (6000000/64=93750Гц - 50..100000)
end;
interrupt ADCRDY; //== обработчик прерывания при окончании АЦ преобразования
begin
  adccode:= word(ADCH);//== читаем только ст. байт кода АЦП
                       //== (т.е. 8-ми битный АЦП)
  ADCSRA.6:=1;//== новый запуск АЦП
end;
{ Main Program }
{$IDATA}
begin
  Init_Ports; //== инициализация портов ввода/вывода
  Init_Timer0Async; //== инициализация таймера0
  Init_ADC; //== инициализация АЦП
  EnableInts; //== разрешение прерываний
  loop //== бесконечный цикл
```

Рис.1.22. Инициализация АЦП и обработчик прерывания

Также нам понадобятся две вспомогательные процедуры (рис. 1.23):

```
procedure Display8seg; //== высвечивание на индикаторе информации (8-ми сегм.
var i,j:byte;          //== код)
begin
  maskind:=%11110111; //== начальное значение маски для позиции индикатора
  for i:=2 downto 0 do //== сканируем 3 линии порта E (PORTE.6 .. PORTE.4)
    maskind:=maskind rol 1; //== готовим вывод в следующий индикатор
    if not b then maskind:=maskind and $7F;endif;//== если LED (PORTE.7)
                                                 //== "горит", то и должен гореть
    PORTC:=byte(s[i+1]);//== выводим инверсн. 8-ми сегмент. код цифры на индик-р
    PORTE:=maskind; //== подаем на анод текущего 8-ми сегм. инд-ра высокий
    for j:=0 to 255 do endfor; //== уровень для увеличения яркости
  endfor;
end;
{ Main Program }
{$IDATA}
```

Рис.1.23. Процедура Display8seg

Функция Dec2_8seg_Str преобразует двоичный код в строку 8-ми сегментных символов, процедура Display8seg выводит их на 8-ми сегментный дисплей.

```
  ADCSRA.6:=1;//== новый запуск АЦП
end;
function Dec2_8seg_Str (decn:word):string[5];//== 10 бит = 1023max
var s:string[5];
  temp:char;
  remn:word;
begin
  s:='';
  repeat
    remn:=decn mod 10; //== вычисляем очередную цифру числа
    decn:=decn div 10; //== путем последовательного деления
    if remn<>0 then    //== результата на 10
      temp:=key8segm[remn];//== извлекаем из таблицы код цифры от 1 до 9
    else
      temp:=key8segm[remn+11]; //== цифра 0 не на "своем" месте
    endif;                     //== на клавиатуре
    s:=s+temp; //== помещаем код цифры в строку 's'
   until decn<1;
  return(s); //== возвращаем строку (например "123" для кода 321)
end;
procedure Display8seg; //== высвечивание на индикаторе информации
var i,j:byte;          //== (8-ми сегм. код)
```

Рис.1.24. Процедура Dec2_8seg_Str

Следующий фрагмент нужно поместить в тело основного блока программы (рис.1.25).

```
while ScanKeyOnce(kn) do endwhile;//== ожидаем отпускание клавишы
case key of //== действия в соотв-ии с нажатой клавишей (*,0..9)
  '*': //== пуск АЦП (считыв.кода по прерыванию от бита готовн.)
    ADCSRA.6:=1; //== первый инициализационный пуск АЦП (
                 //== остальные пуски в обработчике прерывания)
    loop  //== бесконечный цикл
     s:=Dec2_8seg_Str(adccode);//== 8 старших бит АЦП при ADLAR=1
     s:=s+#$FF+#$FF+#$FF; //== FF - гасит неиспольз. индикаторы
                          //== например "__8" или "_48"
     Display8seg; //== отображаем код АЦП
     if ScanKeyOnce(kn)then //== если нажата клавиша,
       ADCSRA.6:=0; //== то остановить АЦП
       exitloop; //== и выйти из цикла измерения напряжения
     endif;
    endloop;
  | //== этот разделитель обязателен
  '0'..'9':
    PORTE:=%10111111;//== подаем на анод правого 8-ми сегм. инд-ра
                     //== напряж-е
```

Рис.1.25. Варианты действий при нажатии на клавишу

В этот момент программа должна иметь следующий вид. Снова компилируем, полученную на этом этапе программу.

```
program myATMega128;
{ $BOOTRST $0F000}        {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}           {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000; //== частота основного кварц.резонатора - 6МГц
  StackSize = $0064, iData; //== для времен. хранения адресов возврата и др.
  FrameSize = $0064, iData; //== для хранения локальных переменных и
                      //== фактических параметров
Implementation
{$IDATA}
{-------------------------------------------------------------}
{ Type Declarations }
type
{-------------------------------------------------------------}
{ Const Declarations }
{-------------------------------------------------------------}
const keyASCII: string = '123456789*0#';//== ASCII коды клавиш/цифр
const key8segm: string =
#$F9+#$A4+#$B0+#$99+#$92+#$82+#$F8+#$80+#$90+#$7F+#$C0+#$B6;
{ Var Declarations }
{$IDATA}
var sound :boolean;
   s:string[5];//== для хранения 8-ми сегм. Цифр
   maskind,j,key,kn,e2temp: byte;
   adccode:word;
{-------------------------------------------------------------}
{ functions }
{-------------------------------------------------------------}
function ScanKeyOnce (var kn: byte):boolean;//== однокр. сканирование клав-
ры
var row,col: byte;    //== var kn - параметр переменная № клавиши
var delay:word;
begin
  for col:=0 to 3 do //== сканируем по столбцам матрицы клавиатуры
    PORTD:=not(1 shl col);//== 11111110,11111101,11111011,11110111 бегущий
ноль)
    for row:=0 to 2 do //== сканируем по строкам матрицы клавиатуры
      if ((PIND and ($10 shl row))=0) then //== если обнаружен ноль, то
        kn := col*3+row;//== вычисляем порядковый № клавиши (0..11)
        for delay:=0 to 5000 do endfor;//== задержка на дребезг контактов
        return (true); //== возвращаем бит "обнаружено нажатие клавиши"
```

```
     endif;
    endfor;
  endfor;
  return (false); //== возвращаем бит "ни одна из клавиш не нажата"
end;
procedure Init_Ports; //== задаем направления передачи данных через порты,
begin                 //== а также начальные значения
  DDRD:=%10001111;//== PD0..PD3 выводим "бегущий 0", PD4..PD6 Ъ
                              //== считываем код возврата
    //== линия PORTD.7 подключена к пъезодинамику, поэтому настроим ее
//== на вывод
  DDRC:=$FF; //== порт C на вывод 8-ми сегментного кода
  DDRE:=%11110000;//== бит7 порта E (LED) на вывод(по RESET'у все
    //== порты настроены на ввод)
    //== 6,5,4 биты на базы транзисторов, коллекторы к общим анодам
    //== индикаторов
  PORTE:=%11111111;//== гасим индикаторы, подавая на их аноды нули
 //== через инверторы
end;
procedure Init_Timer0Async;//== настройка Таймера0 для работы от кварца
32768Гц
begin
  TCCR0:=$0D;//== бит CS00=CS02=1(предделитель=128),WGM01=1(сброс
//== счетчика при совп.)
  OCR0:=$FF; //== регистр кода совпадения (256-1): 32768/128/256=ровно 1Гц
  //== режим совпадения предпочтит. режима переполнения, т.к. вместо FF
  //== можно выбирать другие значения
  ASSR.3:=1; //== бит AS0=1 - переходим в асинхронный режим Timer0 от
//== кварца 32768Гц
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при совпадении
//== текущегозначения таймера0 с кодом в регистре совпадения OCR0

end;                var b: boolean;
interrupt TIMER0COMP;//== обработчик прерывания при переполнении
                              //== таймера0, ровно через 1 сек от TOSC=32768Гц в
                              //==асинхронном режиме
begin
  b:=not b;    //== мигаем
  PORTE.7:= b; //== светодиодом
end;
procedure Init_ADC; //== настройка АЦП
begin
  ADMUX:=$20;//== бит ADLAR=1-биты АЦП выравнены влево и читать
//==можно один ADCH
  ADCSRA:=$8E; //== настройка АЦП  делитель прескалера = 64
```

```
            //== (6000000/64=93750Гц - 50..100000)
end;
interrupt ADCRDY; //== обработчик прерывания при окончании АЦ
//==преобразования
begin
  adccode:= word(ADCH);//== читаем только ст. байт кода АЦП (т.е. 8-ми
//==битный АЦП)
  ADCSRA.6:=1;//== новый запуск АЦП
end;
function Dec2_8seg_Str (decn:word):string[5];//== 10 бит = 1023max
var s:string[5];
  temp:char;
  remn:word;
begin
  s:='';
  repeat
    remn:=decn mod 10; //== вычисляем очередную ДЕСЯТИЧНУЮ цифру
числа
    decn:=decn div 10; //== путем последовательного деления результата на 10
    if remn<>0 then
      temp:=key8segm[remn]; //== извлекаем из таблицы код цифры от 1 до 9
    else
      temp:=key8segm[remn+11]; //== цифра 0 не на "своем" месте на
//==клавиатуре
    endif;
    s:=s+temp; //== помещаем код цифры в строку 's' (задом наперед для E-
//==LAB)
   until decn<1;
  return(s); //== возвращаем строку (например "123" для кода 321)
end;
procedure Display8seg; //== высвечивание на индикаторе информации (8-ми
//==сегм. код)
var i,j:byte;
begin
  maskind:=%11110111; //== начальное значение маски для позиции индик-ра
  for i:=2 downto 0 do //== сканируем 3 линии порта E (PORTE.6 .. PORTE.4)
    maskind:=maskind rol 1; //== готовим вывод в следующий индикатор
    if not b then maskind:=maskind and $7F;endif;//== если LED (PORTE.7)
//=="горит", то и должен гореть
    PORTC:=byte(s[i+1]); //== выводим инверсный 8-ми сегментный код
//==цифры на индикатор
    PORTE:=maskind; //== подаем на анод текущего 8-ми сегм. инд-ра высокий
//==уровень"
    for j:=0 to 255 do endfor; //== для увеличения яркости свечения
  endfor;
```

```
end;
{ Main Program }
{$IDATA}
begin
  Init_Ports; //== инициализация портов ввода/вывода
  Init_Timer0Async; //== инициализация таймера0
  Init_ADC; //== инициализация АЦП
  EnableInts; //== разрешение прерываний
  loop //== бесконечный цикл
    while not ScanKeyOnce(kn) do endwhile;//== ожидаем нажатия на клавишу
    key:=byte(keyASCII[kn+1]);//== +1, т.к. в паскале в 0-м байте -длина
//==строки!
    while ScanKeyOnce(kn) do endwhile;//== ожидаем отпускание клавишы
    case key of //== выполняем действия в соотв-ии с нажатой клавишей (*,0..9)
      '*': //== пуск АЦП (считывание кода по прерыванию от бита готовности)
        ADCSRA.6:=1; //== первый инициализационный пуск АЦП (остальные
//==пуски в обработчике прерывания)
        loop  //== бесконечный цикл
         s:=Dec2_8seg_Str(adccode);//== 8 старших бит АЦП при ADLAR=1
         s:=s+#$FF+#$FF+#$FF; //== FF - гасит неиспольз. индикаторы,
//==например "__8" или "_48"
         Display8seg; //== отображаем код АЦП
         if ScanKeyOnce(kn)then //== если нажата клавиша,
           ADCSRA.6:=0; //== то остановить АЦП
           exitloop; //== и выйти из цикла измерения напряжения
         endif;
        endloop;
      | //== этот разделитель обязателен
      '0'..'9':
        PORTE:=%10111111;//== подаем на анод правого 8-ми сегм. инд-ра
//==напряж-е
        PORTC:=byte(key8segm[kn+1]);//== выводим 8-ми сегм. код нажатой
//==клавиши  на LED индикатор
      | //== этот разделитель обязателен
    endcase;
  endloop;
end myATMega128.
```

Снова запустите на выполнение программу AVR PROG и кнопкой “Flash -> Program” запишите усовершенствованный код программы в ПЗУ микрокронтроллера. Затем, нажмите на клавишу “*” и вращая ручку потенциометра удостовертесь в том, что код соответствующий напряжению отображается на светодиодном дисплее. Результат покажите преподавателю.

5.2.6 ЗАПИСЬ И ЧТЕНИЕ В/ИЗ EEPROM

Следующее усовершенствование программы касается записи и чтения данных в/из EEPROM. Для этого добавляем раздел данных типа EEPROM и модифицируем код между меткой ‘0’..’9’ и endcase, как показано на рис. 1.26.

```
Implementation
{$EEPROM} //== добавить самостоятельно
var
  e2prom: array[0..9]of byte; //== для долговременного (>10лет)
                              //== хранения данных
{$IDATA}
      '0'..'9':
        e2prom[key-'0']:=byte(key8segm[kn+1]); //== пишем в EEPROM
        e2temp:=e2prom[key-'0'];//== читаем из EEPROM
        //== теперь в AVRPROG можно прочитать EEPROM и убедиться,
        //== что коды клавиш сохранены даже после выключения
        //== напряжения питания стенда
        PORTE:=%10111111;//== подаем на анод правого 8-ми сегм.
                         //== инд-ра напряж-е
        PORTC:=e2temp;//== выводим 8-ми сегм. код нажатой клавиши
                      //== на LED индикатор
      | //== этот разделитель обязателен
    endcase;
```

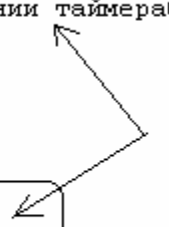
Рис.1.26. Действия с EEPROM

5.2.7 ОКОНЧАТЕЛЬНЫЙ ТЕКСТ ПРОГРАММЫ

Факультативно добавлен фрагмент генерации продолжительного звукового сигнала и модифицирован обработчик TIMER0COMP для подачи звуковых отметок.

```
case key of //== выполняем действия в соотв-ии с нажатой клавишей (#,*,0..9)
  '#': //== подача звукового сигнала
    PORTE:=%11101111; //== подаем на анод левого 8-ми сегм. инд-ра напряж-е
    PORTC:=byte(key8segm[11+1]);//== выводим 8-ми сегм. код клавиши '#'
                                //== на LED индикатор
    loop //== бесконечный цикл
      sound:=not sound; //== инвертируем "полуволну"
      PORTD.7:=sound; //== выводим ее на динамик
      for j:=0 to 255 do endfor;//== задает программно частоту звук. сигнала
      if ScanKeyOnce(kn)then exitloop;endif; //== если нажата клавиша, выйти
    endloop;                                 //== из бесконечного цикла
  | //== этот разделитель обязателен
  '*': //== пуск АЦП (считывание кода по прерыванию от бита готовности)
```

```
var b: boolean;
var i,t0: byte;//== v.4.09 позволяет сделать эти переменные локальными !!
interrupt TIMEROCOMP;//== обработчик прерывания при переполнении таймера0
                      //== ровно через 1 сек от TOSC=32768Гц
                      //== в асинхронном режиме
begin
  b:=not b;    //== мигаем
  PORTE.7:= b; //== светодиодом
  for t0:=0 to 5 do //== "тикаем"
    sound:=not sound; //== формируем меандр _|`|_|`|_|`|
    PORTD.7:=sound;
    for i:=0 to 255 do endfor; //== длительность полуволны "тика"
  endfor;
end;
```



Последний раз откомпилируйте программу и убедитесь в подаче звуковых сигналов.

```
program myATMega128;
//== рабочая программа для микроконтроллнра ATMega128, в которой
изучется:
//== 1.работа с 8-ми сегм. дисплеем, 2.подача звуковых сигналов, 3.работа с
АЦП,
//== 4.асинхронная работа таймера0 от кварцевого резонатора 32768Гц,
//== 5.работа с клавиатурой, 6.запись/чтение в/из EEPROM
{$NOSHADOW}
{ $W+ Warnings}          {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000; //== частота основного кварц.резонатора - 6МГц
  StackSize = $0064, iData; //== для времен. хранения адресов возврата и др.
  FrameSize = $0064, iData; //== для хранения локальных переменных и
                     //== фактических параметров
Implementation
{$EEPROM} //== добавить самостоятельно
var
  e2prom: array[0..9]of byte; //== для долговремен. (>10лет) хранения данных
{$IDATA}
{-------------------------------------------------------------}
{ Type Declarations }
type
{-------------------------------------------------------------}
{ Const Declarations }
{-------------------------------------------------------------}
const keyASCII: string = '123456789*0#';//== ASCII коды клавиш/цифр
```

```
const key8segm: string =
#$F9+#$A4+#$B0+#$99+#$92+#$82+#$F8+#$80+#$90+#$7F+#$C0+#$B6;
{ Var Declarations }     //== инверсные 8-ми сегментные коды клавиш/цифр
{$IDATA}
var sound :boolean;
   s:string[5];//== для хранения 8-ми сегм. Цифр
   maskind,j,key,kn,e2temp: byte;
   adccode:word;
{-------------------------------------------------------------}
{ functions }
{-------------------------------------------------------------}
function ScanKeyOnce (var kn: byte):boolean;//== однокр. сканир. клав-ры
var row,col: byte;    //== var kn: параметр-переменная № клавиши
var delay:word;
begin
  for col:=0 to 3 do //== сканируем по столбцам матрицы клавиатуры
    PORTD:=not(1 shl col);//== 11111110,11111101,11111011,11110111 бегущий
ноль)
    for row:=0 to 2 do //== сканируем по строкам матрицы клавиатуры
      if ((PIND and ($10 shl row))=0) then //== если обнаружен ноль, то
        kn := col*3+row;//== вычисляем порядковый № клавиши (0..11)
        for delay:=0 to 5000 do endfor;//== задержка на дребезг контактов
        return (true); //== возвращаем бит "обнаружено нажатие клавиши"
      endif;
    endfor;
  endfor;
  return (false); //== возвращаем бит "ни одна из клавиш не нажата"
end;
procedure Init_Ports; //== задаем направления передачи данных через порты,
begin                 //== а также начальные значения
  DDRD:=%10001111;//== PD0..PD3 выводим "бегущий 0", PD4..PD6
считываем код возврата
    //== линия PORTD.7 подключена к пъезодинамику, поэтому настроим ее
на вывод
  DDRC:=$FF; //== порт C на вывод 8-ми сегментного кода
  DDRE:=%11110000;//== бит7 порта E (LED) на вывод(по RESET'у все
порты настроены на ввод)
    //== 6,5,4 биты на базы транзисторов, коллекторы к общим анодам
индикаторов
  PORTE:=%11111111;//== гасим индикаторы, подавая на их аноды нули
через инверторы
end;
procedure Init_Timer0Async;//== настройка Таймера0 для работы от кварца
32768Гц
begin
```

```
  TCCR0:=$0D;//== бит CS00=CS02=1(предделитель=128),WGM01=1(сброс
счетчика при совп.)
  OCR0:=$FF; //== регистр кода совпадения (256-1): 32768/128/256=ровно 1Гц
!
  //== режим совпадения предпочтительнее режима переполнения, т.к. вместо
FF
  //== можно выбирать другие значения
  ASSR.3:=1; //== бит AS0=1 - переходим в асинхронный режим Timer0 от
кварца 32768Гц
  TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при совпадении
текущего
end;                //== значения таймера0 с кодом в регистре совпадения OCR0
var b: boolean;
var i,t0: byte;//== v.4.09 позволяет сделать эти переменные локальными !!
interrupt TIMER0COMP;//== обработчик прерывания при совпадении кодов
таймера0
                    //== ровно через 1 сек от TOSC=32768Гц в асинхронном режиме
begin
  b:=not b;    //== мигаем
  PORTE.7:= b; //== светодиодом
  for t0:=0 to 6 do //== "тикаем"
    sound:=not sound; //== формируем меандр _|`|_|`|_|`|
    PORTD.7:=sound;
    for i:=0 to 255 do endfor; //== длительность полуволны "тика"
  endfor;
end;
procedure Init_ADC; //== настройка АЦП
begin
  ADCSRA:=$8E; //== настройка АЦП  делитель (предделителя)прескалера =
64
              //== (6000000/64=93750Гц - 50..100000)
  ADMUX:=$20;//== бит ADLAR=1-биты АЦП выравнены влево и читать
можно один ADCH
end;
interrupt ADCRDY; //== обработчик прерывания при окончании АЦ
преобразования
begin
  adccode:= word(ADCH);//== читаем только ст. байт кода АЦП (т.е. 8-ми
битный АЦП)
  ADCSRA.6:=1;//== новый запуск АЦП
end;
function Dec2_8seg_Str (decn:word):string[5];//== 10 бит = 1023max
var s:string[5];
  temp:char;
  remn:word;
```

```
begin
  s:='';
  repeat
    remn:=decn mod 10; //== вычисляем очередную ДЕСЯТИЧНУЮ цифру
числа
    decn:=decn div 10; //== путем последовательного деления результата на 10
    if remn<>0 then
      temp:=key8segm[remn]; //== извлекаем из таблицы код цифры от 1 до 9
    else
      temp:=key8segm[remn+11]; //== цифра 0 не на "своем" месте на
клавиатуре
    endif;
    s:=s+temp; //== помещаем код цифры в строку 's' (задом наперед для E-
LAB)
   until decn<1;
  return(s); //== возвращаем строку (например "123" для кода 321)
end;
procedure Display8seg; //== высвечивание на индикаторе информации (8-ми
сегм. код)
var i,j:byte;
begin
  maskind:=%11110111; //== начальное значение маски для позиции
индикатора
  for i:=2 downto 0 do //== сканируем 3 линии порта E (PORTE.6 .. PORTE.4)
    maskind:=maskind rol 1; //== готовим вывод в следующий индикатор
    if not b then maskind:=maskind and $7F;endif;//== если LED (PORTE.7)
"горит",
                                        //== то и должен гореть
    PORTC:=byte(s[i+1]); //== выводим инверсный 8-ми сегментный код
цифры на индикатор
    PORTE:=maskind; //== подаем на анод текущего 8-ми сегм. инд-ра высокий
уровень"
    for j:=0 to 255 do endfor; //== для увеличения яркости
  endfor;
end;
{ Main Program }
{$IDATA}
begin  //== начало основного блока программы
  Init_Ports; //== инициализация портов ввода/вывода
  Init_Timer0Async; //== инициализация таймера0
  Init_ADC; //== инициализация АЦП
  EnableInts; //== разрешение прерываний
  loop //== бесконечный цикл
    while not ScanKeyOnce(kn) do endwhile;//== ожидаем нажатия на клавишу
```

```
   key:=byte(keyASCII[kn+1]);//== +1, т.к. в паскале в 0-м байте -длина
строки!
   while ScanKeyOnce(kn) do endwhile;//== ожидаем отпускание клавишы
   case key of //== выполняем действия в соотв-ии с нажатой клавишей
(#,*,0..9)
     '#': //== (нажата клавиша #) подача звукового сигнала
       PORTE:=%11101111; //== подаем на анод левого 8-ми сегм. инд-ра
напряж-е
       PORTC:=byte(key8segm[11+1]);//== выводим 8-ми сегм. код клавиши '#'
                          //== на LED индикатор
       loop //== бесконечный цикл
         sound:=not sound; //== инвертируем "полуволну"
         PORTD.7:=sound; //== выводим ее на динамик
         for j:=0 to 255 do endfor;//== задает программно частоту звук. сигнала
         if ScanKeyOnce(kn)then exitloop;endif; //== если нажата клавиша, выйти
       endloop;                          //== из бесконечного цикла
     | //== этот разделитель обязателен
     '*': //== пуск АЦП (считывание кода по прерыванию от бита готовности)
       ADCSRA.6:=1; //== первый инициализационный пуск АЦП (остальные
пуски ]
                  //== в обработчике прерывания)
       loop  //== бесконечный цикл
        s:=Dec2_8seg_Str(adccode);//== 8 старших бит АЦП при ADLAR=1
        s:=s+#$FF+#$FF+#$FF; //== FF - гасит неиспольз. индикаторы
                        //== например "__8" или "_48"
        Display8seg; //== отображаем код АЦП
        if ScanKeyOnce(kn)then //== если нажата клавиша,
          ADCSRA.6:=0; //== то остановить АЦП
          exitloop; //== и выйти из цикла измерения напряжения
        endif;
       endloop;
     | //== этот разделитель обязателен
     '0'..'9': //== нажата клавиша (0..9)
       e2prom[key-'0']:=byte(key8segm[kn+1]); //== пишем в EEPROM
(E2PROM)
       e2temp:=e2prom[key-'0'];//== читаем из EEPROM
  //== теперь в AVRPROG можно прочитать EEPROM и убедиться, что коды
  //== клавиш сохранены даже после выкл. напряжения питания стенда
       PORTE:=%10111111;//== подаем на анод правого 8-ми сегм. инд-ра
напряж-е
       PORTC:=e2temp;//== выводим 8-ми сегм. код нажатой клавиши на LED
индикатор
     | //== этот разделитель обязателен
   endcase;
 endloop;
```

```
end myATMega128.
```

5.2.8 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Пояснить цель работы и техническое задание.
2. Структурная схема МК ATMega128 и назначение выводов (раздел 1.1).
3. Организация памяти МК ATMega128 (рис.1.3).
4. Настройка портов (рис.1.5).
5. Таймер/счетчик0 (регистры TCNT0, OCR0, TCCR0).
6. АЦП (регистры ADCSRA, ADMUX, ADCH/ADCL).
7. Работа клавиатуры и 8-ми сегментного дисплея.
8. Комментарии к программе

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА 68HC908

6. ЦЕЛЬ РАБОТЫ

Целью работы является изучение схем подключения к МК и программирования следующих типовых устройств:

1. Жидкокристаллического - ЖК (LCD) дисплея с 4-х битным интерфейсом (экономия 4 линий портов по сравнению с 8-ми битным интерфейсом),
2. Синхронного последовательного интерфейса SPI на примере температурного датчика (ТД) DS1722,
3. Таймера в режиме широтно-импульсной модуляции - ШИМ (PWM) для управления яркостью свечения светодиода и дополнительно - организация часов,
4. Встроенного АЦП.
5. Работу с клавиатурой по прерыванию (без поллинга, т.е. без сканирования).
6. Функционирование устройств перечисленных в пп. 1-4 полностью АВТОМАТИЧЕСКОЕ – управление от клавиатуры этими устройствами не предусмотрено.

7. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1. Сформировать ШИМ сигнал для управления яркостью светодиода.
2. Обеспечить непрерывное измерение напряжения (угла поворота) с одновременным отображением напряжения и кода на ЖК дисплее.
3. Производить непрерывное измерение температуры внутри лабораторного стенда с отображением результата на ЖКД.
4. Обеспечить индикацию минут и секунд, прошедших с момента запуска программы.
5. Организовать работу клавиатуры по прерыванию в момент нажатия на клавишу и выводить ASCII код клавиши на ЖКД.

8. СТРУКТУРА МИКРОКОНТРОЛЛЕРА ATMega128

8.1 НАЗНАЧЕНИЕ ВЫВОДОВ

Микроконтроллер MC68HC908GP32 содержит 8-разрядный процессор CPU08, Flash-память емкостью 32 Кбайт, ОЗУ данных емкостью 512 байт и набор служебных и периферийных модулей.

Процессор CPU08 выполняет обработку 8-разрядных операндов и реализует набор из 90 команд. Он содержит пять программно-доступных регистров: 8-разрядные аккумулятор A и регистр признаков (флагов) CCR, 16-разрядные индексный регистр H:X, указатель стека SP и программный счетчик PC. В состав служебных модулей входят:

- генератор тактовых импульсов CGM08,
- модуль системной интеграции SIM08,
- модуль контроля напряжения питания LVI08,
- модуль прерывания в контрольной точке BREAK08,
- модуль управления внешним прерыванием IRQ08,
- сторожевой таймер COP08,
- базовый таймер TBM08.

Модуль генератора импульсов CGM08 генерирует импульсные сигналы, на базе которых модуль системной интеграции SIM08 формирует тактовые импульсы. Выходные сигналы модуля CGM08 определяют частоту тактовых импульсов для работы процессора и периферийных модулей.

Модуль системной интеграции SIM08 выполняет ряд функций, обеспечивающих совместную работу различных модулей микроконтроллера. Он работает совместно с другими служебными модулями: CGM08, LVI08, IRQ08, BREAK08, COP08, выполняя формирование тактовых импульсов, запуск микроконтроллера, организацию обслуживания прерываний.

Модуль прерывания в контрольной точке BREAK08 обеспечивает останов выполнения программы в заданной контрольной точке и используется в процессе отладки программного обеспечения.

Модуль управления внешним прерыванием IRQ08 принимает внешний запрос прерывания, поступающий на вход IRQ#, и обеспечивает различные варианты его обслуживания.

Сторожевой таймер COP08 осуществляет контроль выполнения текущей программы.

Модуль LVI08 вырабатывает сигнал перезапуска микроконтроллера при снижении его напряжения питания ниже порогового уровня.

Модуль базового таймера TBM08 обеспечивает периодическое формирование запросов прерывания.

Периферийные модули обеспечивают обмен данными и совместную работу микроконтроллера с другими устройствами, входящими в состав системы управления.

Микроконтроллер MC68HC908GP32 содержит следующие периферийные модули:

- пять параллельных портов A, B, C, D, E для ввода-вывода данных,
- асинхронный последовательный порт SCI08,
- синхронный последовательный порт SPI08,
- модуль контроля клавиатуры KBI08,
- 8-разрядный аналого-цифровой преобразователь ADC08,
- два таймерных модуля TIM08.

Двунаправленные порты A, B, C, D, E обеспечивают параллельный обмен данными с внешними устройствами. Порты A, B имеют по 8 линий ввода-вывода, порт E - 2 линии, а порты C, D - от 5 до 8 линий в зависимости от числа выводов корпуса, в котором смонтирован микроконтроллер.

Выводы параллельных портов A, B, D, E совмещены с выводами других периферийных модулей - KBI08, ADC08, TIM08-1, TIM08-2, SPI08, SCI08. При работе вышеуказанных модулей соответствующие выводы параллельных портов служат для передачи сигналов, необходимых для функционирования модуля, и не могут использоваться для параллельного ввода-вывода данных.

Последовательные порты SCI08, SPI08 реализуют соответственно последовательный асинхронный и синхронный обмен данными между микроконтроллером и внешними устройствами.

Таймерный модуль TIM08 выполняет широкий набор функций, включая фиксацию времени поступления входных сигналов, выдачу выходных сигналов в заданный момент времени, формирование последовательности импульсов заданной частоты и длительности.

Модуль аналого-цифрового преобразования ADC08 производит преобразование значения потенциала, поступающего на один из 8 аналоговых входов, в 8-разрядное двоичное число.

Модуль контроля клавиатуры KBI08 обеспечивает формирование запроса прерывания при поступлении сигнала на определенные входы параллельных портов, которые обычно используются для подключения клавиатуры.

8.2 ОРГАНИЗАЦИЯ ПАМЯТИ И ПОРТОВ ВВОДА/ВЫВОДА

Микроконтроллеры семейства 68HC08/908 адресуют 64 Кбайт внутренней памяти (адреса $0000-FFFF). Распределение адресного пространства задается определяется объемом внутренней памяти и набором периферийных устройств, входящим в состав данной модели микроконтроллера. В таблице 2.1 приведена карта адресного пространства для микроконтроллеров MC68HC908GP32.

Таблица 2.1

$0000 $003F	**Регистры периферийных и служебных модулей (64 байт)**
$0040 $023F	**ОЗУ данных (512 байт)**
$0080 $7FFF	**Не используется (32 192 байт)**
$8000 $FDFF	**Flash-память (32 256 байт)**
$FE00	**Регистр SBSR (модуль BREAK08)**
$FE01	**Регистр SRSR (указывает причину запуска)**
$FE02	**Резервировано**
$FE03	**Регистр SBFCR (модуль BREAK08)**
$FE04	**Регистр INT1 (запросы прерывания)**
$FE05	**Регистр INT2 (запросы прерывания)**
$FE06	**Регистр INT3 (запросы прерывания)**
$FE07	**Резервировано**
$FE08	**Регистр FLCR (управление Flash-памятью)**
$FE09	**Регистр BRKh (модуль BREAK08)**
$FE0A	**Регистр BRKl (модуль BREAK08)**
$FE0B	**Регистр BRKSCR (модуль BREAK08)**
$FE0C	**Регистр LVISR (модуль LVI08)**
$FE0D $FE1F	**Не используются (19 байт)**
$FE20 $FE52	**ПЗУ – монитор отладки (307 байт)**
$FE53 $FF7D	**Не используются (43 байт)**
$FF7E	**Регистр FLBPR (управление Flash-памятью)**
$FF7F $FFDB	**Не используются (93 байт)**
$FFDC $FFFF	**Вектора запуска и прерываний (36 байт)**

В адресном пространстве имеется ряд неиспользуемых позиций, которые соответствуют ячейкам памяти, отсутствующим в данной модели микроконтроллеров. При обращении к этим адресам производится перезапуск микроконтроллера.

Младшие 64 позиции адресного пространства (адреса $000-$003F) занимают регистры служебных и периферийных модулей (в табл. 2.2 приведены не все регистры). Отметим, что 16-разрядные регистры таймерных модулей TCNT, TMOD, TCHx занимают по две позиции адресного пространства: младший байт с суффиксом l (строчная L – не путать с единицей), старший байт с суффиксом h. Регистры отмеченные желтым цветом явно используются в лабораторной работе.

Таблица 2.2

Адрес	Регистр	Назначение
$0000	PTA	Регистры данных портов A, B, C, D
$0001	PTB	
$0002	PTC	
$0003	PTD	
$0004	DDRA	Регистры направления передачи портов A, B, C, D
$0005	DDRB	
$0006	DDRC	
$0007	DDRD	
$0008	PTE	Регистр данных порта E
$000D	PTAPUE	Регистр управления подтягивающими резисторами порта A
$0010	SPCR	Регистры синхронного порта SPI08
$0011	SPSCR	
$0012	SPDR	
$001A	INTKBSCR	Регистры модуля KBI08
$001B	INTKBIER	
$001C	TBCR	Регистр базового таймера TBM08
$001D	INTSCR	Регистр прерываний
$001E	CONFIG2	Регистры конфигурации
$001F	CONFIG1	
$0020	T1SC	Регистры таймерного модуля TIM08-1
$0021-22	T1CNTh-l	
$0023-24	T1MODh-l	
$0025	T1SC0	
$0026-27	T1CH0h-l	
$0028	T1SC1	
$0029-2A	T1CH1h-l	
$003B	PMDS	
$003C	ADSCR	Регистры модуля ADC08
$003D	ADR	
$003E	ADCLK	

В адресном пространстве ОЗУ располагаются ячейки стека, которые адресуются с помощью указателя стека SP. При установке микроконтроллера в начальное состояние (запуске) содержимое SP принимает значение $00FF, адресуя ячейку ОЗУ с данным адресом. В процессе выполнения программы можно установить любое значение указателя стека с помощью команды TXS, которая загружает в SP содержимое индексного регистра H:X, уменьшенное на 1. После записи байта в стек содержимое SP уменьшается на 1, адресуя следующую незаполненную ячейку стека. Таким образом, стек заполняется в направлении уменьшения адресов. Адрес вершины стека (последней заполненной ячейки стека) можно загрузить в регистр H:X с помощью команды TSX.

Микроконтроллер MC68HC908GP32 имеет внутреннюю Flash-память, содержимое которой может стираться и записываться при работе в режиме отладки или в процессе выполнения прикладной программы. Допускается до 10000 циклов стирания-программирования, время хранения информации составляет более 10 лет. Необходимое для программирования повышенное напряжение обеспечивается внутренним преобразователем, поэтому не требуется подключение внешнего источника. Специальный механизм защиты позволяет предотвратить случайное стирание содержимого Flash-памяти. Наличие байтов секретности позволяет предотвратить несанкционированное считывание информации.

На кристалле микроконтроллера содержится 512 байт статической оперативной памяти, ячейки которой имеют адреса в диапазоне $0040-$023F. Обычно ОЗУ используется для хранения переменных и реализации стека.

Часть адресного пространства занята ячейками служебного ПЗУ, в котором содержится программа-монитор, которая реализует необходимые процедуры при работе микроконтроллера в режиме отладки, обеспечивая возможность контроля его внутреннего состояния. Это масочно-программируемое ПЗУ, содержимое которого записывается в процессе изготовления микроконтроллера.

В старших позициях адресного пространства располагаются вектора начального запуска и прерываний. Для размещения таблицы векторов прерываний в микроконтроллерах семейства 68HC08/908 зарезервированы старшие адреса. В таблице 2.3 показано размещение некоторых векторов прерываний.

Таблица 2.3

Запрос модуля ADC08	$FFDE-DF
Запрос модуля KBI08	$FFE0-E1
Запрос канала 0 таймера TIM08-2	$FFF0-F1
Переполнение таймера TIM08-1	$FFF2-F3
Запрос канала 1 таймера TIM08-1	$FFF4-F5
Запрос канала 0 таймера TIM08-1	$FFF6-F7
Прерывание по внешнему запросу IRQ#	$FFFA-FB
Команда ***SWI*** (программное прерывание)	$FFFC-FD
Вектор начального запуска (Reset)	$FFFE-FF

Поступление запросов прерывания проверяется микроконтроллером после выполнения каждой команды программы. Если поступило нескольких запросов, то в первую очередь обслуживается запрос с более высоким приоритетом. Запросы имеют фиксированные приоритеты в соответствии с порядком их расположения в таблице векторов прерываний (табл. 2.3) - чем больше адрес, тем выше приоритет.

9. СХЕМА ПОДКЛЮЧЕНИЯ И ПРОГРАММИРОВАНИЕ, НЕОБХОДИМЫХ В РАБОТЕ, ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

9.1 НАСТРОЙКА ПОРТОВ ВВОДА/ВЫВОДА

Для работы МК с внешними устройствами (клавиатура, светодиод, ЖК дисплей, термодатчик и др.) необходимо задать направление обмена данными через соответствующие выводы, т.е. настроить их в качестве входов или выходов (рис.2.1).

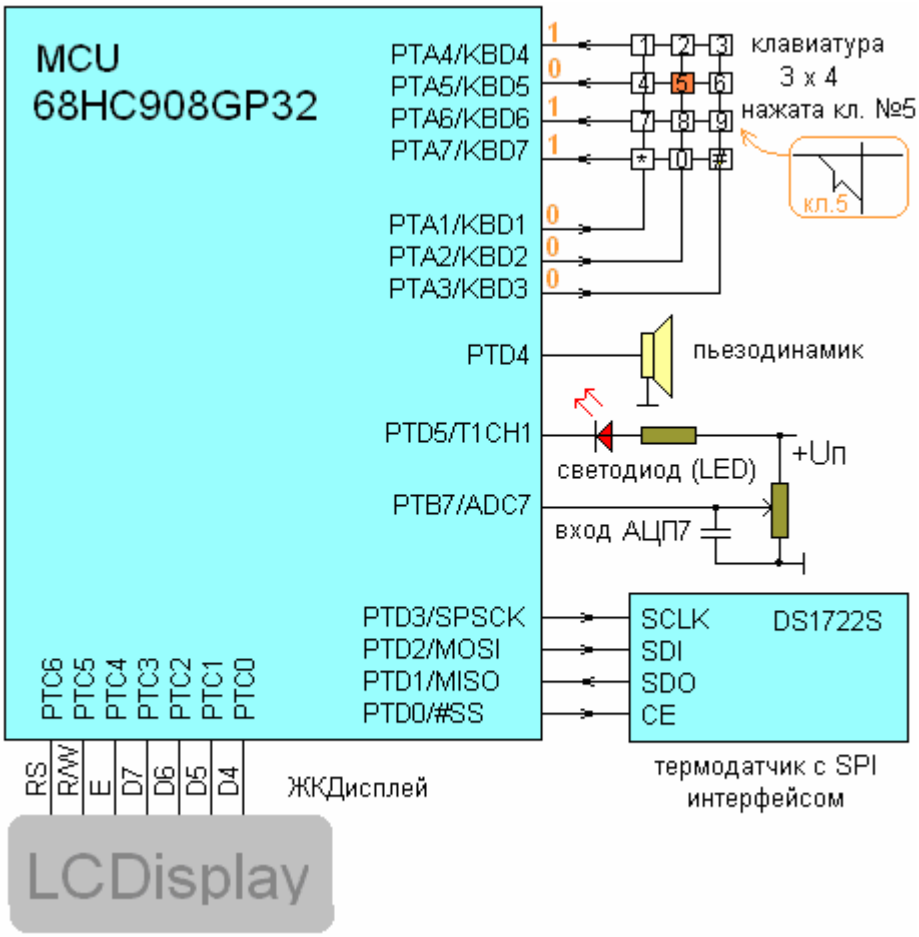


Рис. 2.1. Схема подключения периферийных устройств

При подаче напряжения на МК или по сигналу #RESET все порты автоматически настраиваются на ввод (в регистры направления портов DDRx записаны нули), поэтому направление сигналов через некоторые выводы портов необходимо переопределить. Для этого в соответствующие портам регистры направления DDRx нужно записать единицы. Из схемы на рис. 2.1 видно, что порт PTC служит для управления ЖК дисплеем, поэтому в регистр направления DDRC (Data Direction Register C) необходимо записать код 7F. Два бита 4 и 5 порта PTD служат для вывода сигналов на светодиод и динамик, поэтому соотв. биты регистра направления DDRD также д.б. равны единице. 4 старших бита порта PTA предназначены для считывания состояния клавиатуры. На 3 вывода PTA3..1 выводятся нули, поэтому в регистр направления DDRA необходимо записать код 0E.

Управление линиями порта PTD3..1 в альтернативном режиме (интерфейс SPI) производится автоматически. Поэтому явно записывать в регистр DDRD направление нужно только для вывода #SS.

9.2 НАСТРОЙКА РЕГИСТРОВ СПЕЦИАЛЬНЫХ ФУНКЦИЙ МОДУЛЯ КЛАВИАТУРЫ

Одним из наиболее распространенных устройств ввода команд и данных в цифровую систему является клавиатура. В стенде 12-кнопочная клавиатура (4 ряда по 3 кнопки) подключена к выводам PTA1-PTA7 порта А. Считывать код можно путем опроса клавиатуры или по запросу прерывания. Схема подключения выводов клавиатуры к выводам PTA приведена на рис.2.2.

Линии порта А (PTAi/ KBDi), имеющие альтернативную функцию для подключения клавиатуры могут индивидуально настраиваться на ввод или вывод. В отличие от многих других МК, эти линии могут быть запрограммированы таким образом, что появление нуля или отрицательного перепада на них вызовет прерывание. Таким образом, появляется возможность обойтись без периодического сканирования клавиатуры (опроса или поллинга) и высвободить время, требуемое на опрос для решения других задач. Этот метод используется в лабораторной работе.

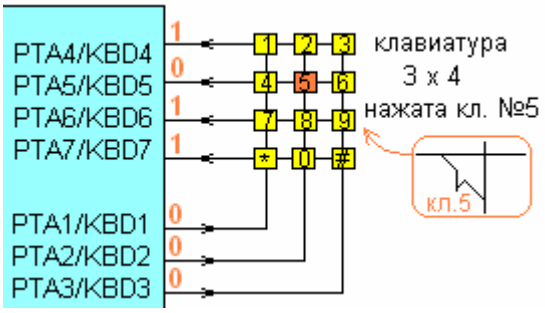


Рис.2.2. Подключение клавиатуры

В этом случае алгоритм определения нажатой клавиши включает подачу нулей на все столбцы KBD3..1 и записи единиц на входах KBD7..4. Если не нажата ни одна из клавиш, на всех входах PTA7..4 будет высокий потенциал (код F). В момент замыкания контактов любой клавиши нулевой сигнал (перепад 1-->0) поступит на один из входов PTA7..4 и вызовет прерывание “запрос модуля KBI08” см. таблицу 2.3. В примере нажата кл.”5”, поэтому на входах PTA7..4 будет зафиксированн код 1101=13. В обработчике прерывания последовательно подавая единицу на выходы PTA3..1 зафиксируем момент, когда ноль на входе (в примере PTA5) снова сменится на единицу. Теперь можно вычислить номер нажатой клавиши по формуле keynum=col+row*3 (0..11) в таблице key ASCII кодов клавиш (char key[ ] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#',"};). В примере номер столбца col=1 и номер строки row=1, поэтому keynum=4 будет соответствовать клавише “5” (элементы массивов, в т.ч. key в Си всегда индексированы с нуля).

В отличие от других схем (см. лабор. работы 13 и 15) для формирования единиц на входах PTA7..4 будем использовать не внешние резисторы, подключенные к источнику питания, а внутренние резисторы порта A (подтягивающие резисторы – PullUp) рис.2.3.

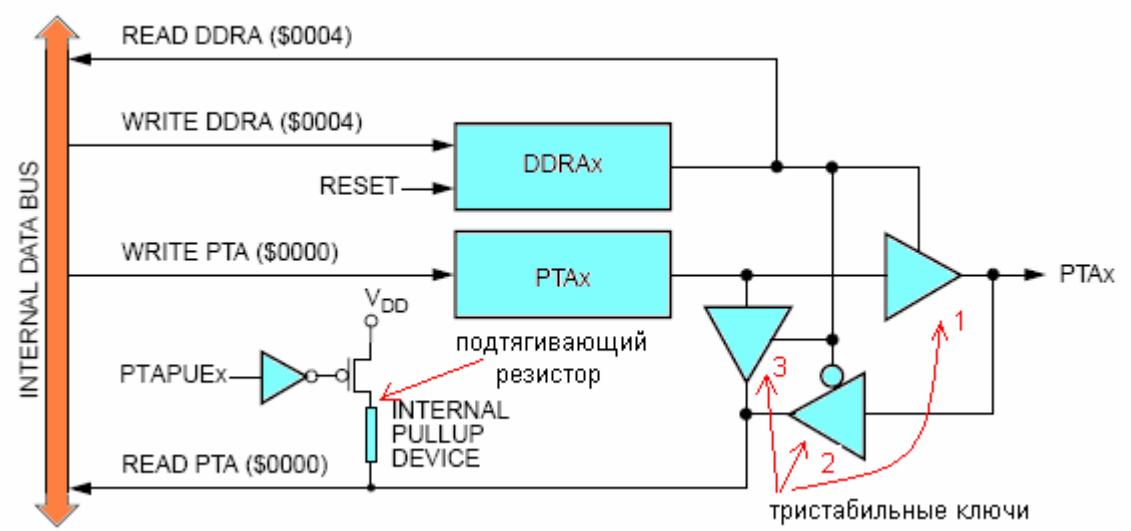


Рис.2.3. Схема одного вывода “x” порта A

Для подключения этих резисторов к высокому потенциалу Vdd необходимо в регистр управления подтягивающими резисторами PTAPUE записать единицы для входов PTA7..4 (PTAPUE=0xF0).

Для выбранного режима работы порта A четыре старших бита DDRA равны нулю, поэтому ключи 1 и 3 разомкнуты (находятся в третьем состоянии), а ключ 2 замкнут и выводы PTA7..4 используются как входы, к которым подключены внутренние резисторы.

Завершим инициализацию модуля клавиатуры разрешением прерываний при нажатии на клавишу для чего нужно записать единицы в четыре старших бита в регистр INTKBIER (рис. 2.4).

Read: Write:	KBIE7	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
Reset:	0	0	0	0	0	0	0	0

Рис. 2.4. Регистр управления прерываниями от клавиатуры - INTKBIER

Также необходимо указать тип события для прерывания по нулевому уровню или отрицательному перепаду на входах (бит MODEK=0 – по перепаду) и разрешить прерывания от клавиатуры (бит IMASKK=0). Т.к. по сигналу RESET эти биты обнуляются, явную запись в регистр INTKBSCR производить не будем.

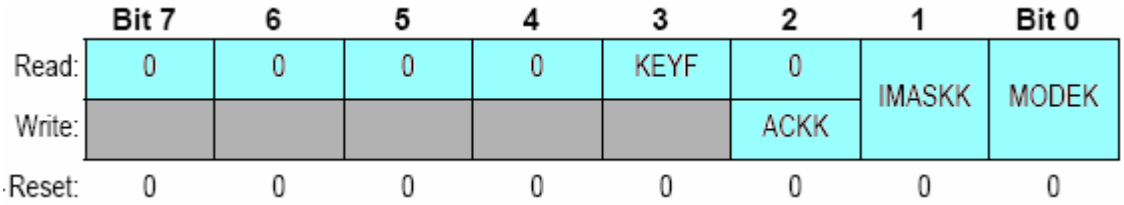


Рис. 2.5. Регистр управления и состояния модуля клавиатуры - INTKBSCR

Соответствующий фрагмент (процедура KBDinit) инициализации модуля клавиатуры будет выглядеть следующим образом:

```
KBDinit(){//== установка интерфейса клавиатуры в исходное состояние
  DDRA=0x0E;//== 4 ст. бита - на ввод (линии возврата), 3 бита PTA1..3 - на вывод
  PTAPUE=0xF0;//== включаем 4 подтягивающих резистора (PullUp) на входах PTA4..7
  //== если не нажата ни одна из клавищ, на ВСЕХ линиях возврата - будут единицы
  PTA=0xF1;//== на все выходы PA1..PA3 подаем нули (без сканирования)
  INTKBIER=0xF0;//== разрешить прер-я от отрицат. фронта (``|__) на входах PTA4..7
 }
```

В обработчике прерывания KBD_int (см. внизу) при нажатии на клавишу необходимо подтвердить обработку установив бит ACKK=1, который автоматически обнуляется при возникновении прерывания. Как обычно, предусматриваем задержку на время возможного “дребезга” контактов клавиши. Затем читаем код на входах PTA7..4 (r=(PTA&0xF0)>>4)) и выбираем из таблицы rown номер ряда (row=rown[r]). Маска F0 нужна, т.к. считываются все биты порта целиком, но для нас здесь имеют значения только четыре старших. Номер колонки находим последовательно подавая единицы на выходы PTA3..1 (PTA|=0x08>>i) и повторно считывая код на входах PTA7..4. Если ноль в колонке нажатой клавиши “перекрыт”, т.е. 4 старших бита равны 1111=F (if((PTA&0xF0)==0xF0)), записываем номер колонки (col=2-i) и вычисляем номер нажатой клавиши (keynum=col+row*3). Оставшиеся операции достаточно откомментированы.

```
char key[]={'1','2','3','4','5','6','7','8','9','*','0','#',''};//== ASCII коды клавиш
char keynum, keypressed=0; //== номер клавиши и признак "клавиша была нажата"
char rown[]={0,0,0,0,0,0,0,3,0,0,0, 2, 0, 1, 0, 0};//== таблица номеров рядов клавиш
                                                    7        11    13 14
```

```
    void KBD_int(void){//== обработчик нажатия на клавишу
    char row,col,temp,r,i; //== col - номер колонки слева, row - номер ряда
                           //== сверху
     INTKBSCR|=(1<<ACKK); //== подтверждаем обработку прерывания
     Delay(800); //== задержка на время возможного дребезга при нажатии
     if((r=(PTA&0xF0)>>4)==0x0F)goto exit; //== r=0111(7), 1011(11),
1101(13), 1110(14)
     //== "goto exit" игнорирует возможный дребезг при отпускании
клавиши
     //== и появление кода r = 1111 вместо 0111..1110
     row=rown[r]; //== выбираем из таблицы номер ряда = (0,1,2,3)
     for(i=0;i<3;i++){//== ищем в какой колонке находится нажатая
клавиша
       PTA|=0x08>>i; //== последовательно "перекрываем" единицей
колонки
                     //== PTA3..1(col=2..0)
         if((PTA&0xF0)==0xF0){col=2-i;break;}//== если 1 перекрыла 0 –
                                             //== закончить проверку
     }
     keynum=col+row*3; //== вычисляем номер нажатой клавиши (0..11) в
                       //== таблице key
    exit:
     PTA=0b11110001; //== возвращаем начальное состояние выводов
порта A
     INTKBSCR|=(1<<ACKK); //== сбрасываем возможные запросы при
                          //== манипулир. с битами PTA
     keypressed=1; //== устанавливаем признак (флаг) нажатия клавиши
(этот
    //== признак можно использовать в основной программе - main())
    }
```

9.3 МОДУЛЬ АЦП

Модуль 8-разрядного аналого-цифрового преобразователя ADC08 (рис. 2.6), входящего в состав MC68HC908GP32, содержит входной мультиплексор, осуществляющий выбор одного из восьми входов аналогового сигнала, АЦП последовательного приближения, регистр управления-состояния ADSCR, регистр настройки тактовой частоты ADCLK и регистр результата преобразования ADR. Входы AD0-AD7 модуля ADC08 совмещены с выводами PTB0-PTB7 параллельного порта B. Так как в один и тот же момент времени может использоваться только один вход АЦП, то остальные выводы порта B доступны для параллельного ввода-вывода.

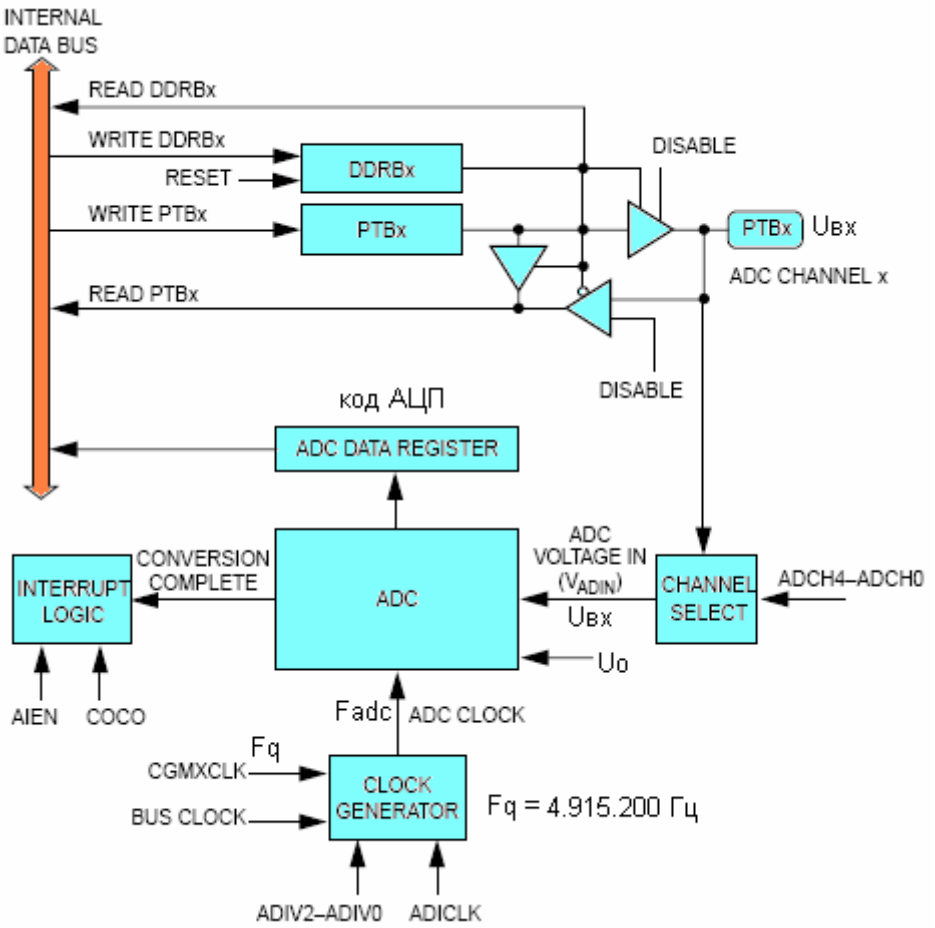


Рис. 2.6 Схема порта B и структура АЦП

Функционирование АЦП определяется содержимым регистра управления и состояния АЦП **ADSCR**, который содержит следующие биты:

7	6	5	4	3	2	1	0
COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0

Бит COCO - признак окончания преобразования, доступный только для чтения, если установлено значение бита AIEN = 0; принимает значение COCO = 1 после выполнения очередного цикла преобразования,

Бит AIEN =1 разрешает формирование запроса прерывания после каждого цикла преобразования;

Бит ADCO - определяет режим работы АЦП: однократное преобразование при значении ADCO=0, непрерывная работа преобразователя при ADCO = 1,

Биты ADCH4-ADCH0 - осуществляют выбор аналогового входа мультиплексора в соответствии с таблицей 2.4.

Таблица 2.4

Значение ADCH4-ADCH0	Подключаемый аналоговый вход
00000	AD0 (PTB0)
00001	AD1 (PTB1)
00111	AD7 (PTB7)
1000-11100	Не используются
11101	Потенциал Vrf
11110	Аналоговая «земля»
11111	Отключение АЦП

Запуск АЦП осуществляется автоматически после выбора определенного аналогового входа мультиплексора, путем записи соответствующего значения битов ADCH4-ADCH0 в регистре ADSCR.

Значение результата преобразования в регистре ADR линейно зависит от значения входного потенциала Uвх, при этом максимальное значение Uвх соответствует числу $FF, а минимальное - $00. Поступающий на аналоговый вход потенциал Uвх должен находиться в диапазоне 0 < Uвх < Uo, где Uo - величина опорного напряжения, подаваемого на специальный вывод микроконтроллера. В лабораторном макете Uo = 5В, таким образом, разрешающая способность АЦП в лабораторном стенде ЛС-1 составляет около 20 мВ, а погрешность преобразования ±10 мВ.

В состав модуля ADC08 входит схема выбора тактовой частоты АЦП, настройка которой осуществляется в регистре **ADCLK**, содержащем следующие биты:

7	6	5	4	3	2	1	0
ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0

ADICLK - определяет выбор сигнала тактовой частоты для АЦП: системная тактовая частота Ft при установке значения ADICLK = 1 или частота кварцевого резонатора Fq при ADICLK = 0;

ADIV2-ADIV0 - задают коэффициент деления частоты входного сигнала Kadc для формировании тактовых сигналов АЦП (табл. 2.5).

Таблица 2.5

ADIV2-ADIV0	Kadc
000	1
001	2
010	4
011	8
1xx	16

Оптимальным значением тактовой частоты аналого-цифрового преобразователя Fa является частота порядка 1МГц. Эта частота обеспечивается путем деления тактовой частоты Ft (при значении бита ADICLK = 1) или частоты кварцевого резонатора Fq (при значении бита ADICLK = 0). Необходимое значение коэффициента деления Kadc = Ft/Fadc или Kadc = Fq/Fadc задается установкой битов ADIV2-ADIV0 в регистре ADCLK в соответствии с таблицей 2.5. Для выполнения одного цикла преобразования требуется 17 тактов, поэтому в случае Fa = 1 МГц время преобразования составляет 17 мкс, а в общем случае Тпреобр = 17 / Fadc.

Результат преобразования помещается в регистр данных АЦП - **ADR**:

7	6	5	4	3	2	1	0
AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

9.4 НАСТРОЙКА РЕГИСТРОВ СПЕЦИАЛЬНЫХ ФУНКЦИЙ АЦП

В лабораторной работе датчик напряжения Uвх (он же датчик угла поворота) подключен к 7-му входу АЦП (рис. 2.7), поэтому значения битов ADCH4..0 регистра ADSCR равны 00111(BIN)=7(DEC).

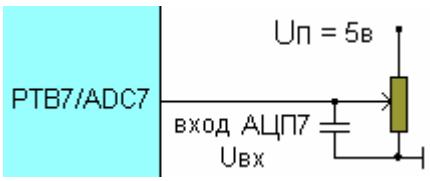


Рис. 2.7. Подключение потенциометра к 7-му каналу АЦП

Чтение кода напряжения и запуск АЦП (кроме первого запуска) будем производить в обработчике прерывания, поэтому для разрешения прерываний в бит AIEN запишем 1. Также установим однократное преобразование (повторные пуски АЦП в обработчике) бит ADCO=0. В соответствии с этим значение регистра ADSCR должно быть равно 47(HEX).

Тактовую частоту АЦП Fadc (рис. 2.6) выберем не превышающую 1МГц. В качестве источника частоты выберем частоту кварцевого резонатора Fq (сигнал CGMXCLK). В стенде Fq=4.915.200Гц, поэтому коэффициент деления Kadc сделаем равным 8-ми. При этом биты ADIV2..0 = 011(BIN), а бит ADICLK=0. Т.е. в регистр ADCLK нужно записать код 60(HEX). Соответствующий фрагмент процедуры инициализации и обработчик прерывания будут выглядеть следующим образом:

```
InitDevices(){//== установка режимов работы портов и периферии
  ..............................
  ADCLK=0x60; //== задаем тактовую частоту АЦП -  ADIV1,0=1(Fq/8),
```

```
                    //== ADICLK=0, Fq = 4.915.200 Гц / 8 ~= 0.6 Мгц
  ADSCR=0x47; //== AIEN=1(разр.прерыв.),
ADCH4..ADCH0=00111(канал
                    //== AD7(PTB7) и пуск)
.....................................
}

char ad;//== глобальная переменная для 8-ми битного кода АЦП
void ADC_int(void){//== обработчик прерывания по завершению АЦ
                    //== преобразования
  ad=ADR; //== читаем код напряжения из регистра данных АЦП)
  ADSCR|=0x07; //== снова запускаем АЦП (биты ADCH4..ADCH0 –
                    //== принуд. обнуляются)
}
```

9.5 МОДУЛЬ ТАЙМЕРА 1

На рис. 2.8 приведена структурная схема таймерного блока.

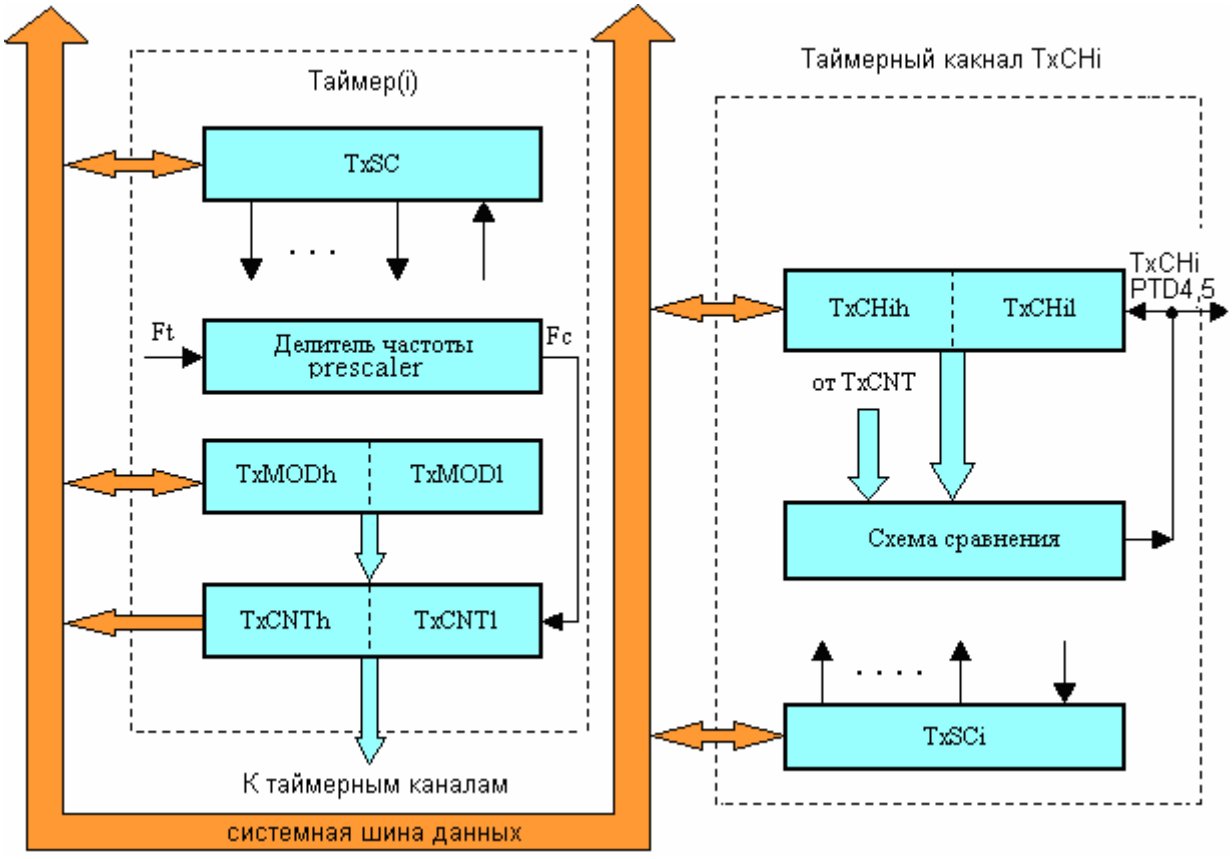


Рис. 2.8. Схема таймерного блока

Таймеры выполняют широкий набор функций, наиболее распространенными из которых являются функция счета, функция захвата

события (перепада сигнала на специальном входе), функция совпадения и функция формирования сигналов с широтно-импульсной модуляцией (ШИМ) сигналов.

Микроконтроллер MC68HC908GP32 имеет два таймерных модуля TIM08, именуемых соответственно TIM1 и TIM2. Модуль TIM08 (рис. 2.8) содержит блок таймера/счетчика и 2 таймерных канала, выводы которых служат для ввода внешних сигналов или для выдачи сигналов управления в заданные моменты времени.

Каждый блок таймера/счетчика содержит 16-разрядный счетчик TxCNT, 16-разрядный регистр модуля счета TxMOD и 8-разрядный регистр управления-состояния TxSC. В именах регистров символ ‘x’ принимает значение 1 или 2 при обращении к соответствующему модулю TIM1 или TIM2. Каждый таймерный канал содержит 16-разрядный регистр данных TxCHi и 8-разрядный регистр управления-состояния канала TxSCi, где в качестве символа ‘i’ указывается номер соответствующего канала: 0 или 1. Входы-выходы таймерных каналов T1CH0 и T1CH1 модуля TIM1 совмещены с выводами PTD4 и PTD5 параллельного порта D соответственно, а таймерных каналов T2CH0 и T2CH1 модуля TIM2 с выводами PTD6 и PTD7.

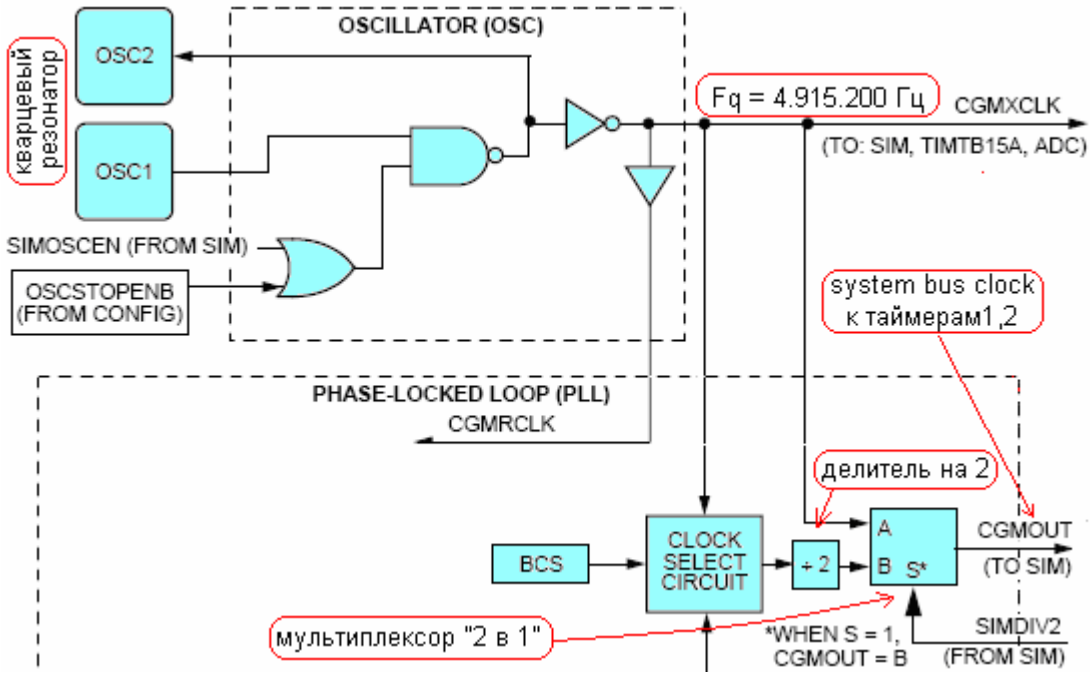


Рис. 2.9. Формирование сетки частот для модулей МК

Функционирование таймера/счетчика в модуле TIM08 определяется содержимым регистра **TxSC.**

Биты выделенные желтым цветом явно используются в лабораторной работе.

7	6	5	4	3	2	1	0
TOF	TOIE	TSTOP	TRST	-	PS2	PS1	PS0

TOF - признак переполнения таймера, доступный только для чтения; принимает значение TOF=1, если содержимое счетчика TxCNT достигает максимального значения, заданного содержимым регистра TxMOD;

TOIE - разрешает при значении TOIE = 1 формирование запроса прерывания при переполнении счетчика (признак TOF = 1);

TSTOP - вызывает останов работы счетчика TxCNT при установке значения бита TSTOP = 1;

TRST - вызывает при установке значения TRST = 1 сброс содержимого счетчика TxCNT в состояние $0000 и установку значения битов PS2-PS0 = 000; этот бит доступен только для записи и устанавливается в 0 после сброса TxCNT;

PS2-PS0 - определяют коэффициент деления Kd=Ft/Fc частоты переключения счетчика TxCNT (рис. 2.8). В таблице 2.6 приведены значения коэффициента деления частоты Kd для модуля TIM08.

таблица 2.6

PS2-0	Kd
0 0 0	1
0 0 1	2
0 1 0	4
0 1 1	8
1 0 0	16
1 0 1	32
1 1 0	64
1 1 1	Не используется

При запуске микроконтроллера счетчик TxCNT устанавливается в нулевое состояние, после чего осуществляется переключение счетчика с частотой Fc = Ft/Kd, где коэффициент деления Kd определяется значением битов PS2-PS0 в регистре TxSC (табл. 2.6). Когда содержимое счетчика TxCNT достигает значения, записанного в регистре TxMOD, счетчик сбрасывается в нулевое состояние. При этом устанавливается значение признака переполнения TOF = 1, что вызывает прерывание процессора, если в регистре TxSC значение бита TOIE = 1. Модуль счета Mc, задаваемый содержимым регистра TxMOD, может принимать значения от 2 до 65535 (при TxMOD=0xFFFF). При запуске МК все биты TxMOD устанавливаются в 1. Останов и последующий запуск счетчика производится путем установки соответствующего значения бита TSTOP в регистре TxSC.

Функционирование i-го таймерного канала определяется содержимым его регистра управления-состояния TxSCi. Биты выделенные желтым цветом явно используются в лабораторной работе.

7	6	5	4	3	2	1	0
CHiF	CHiIE	MSiB	MSiA	ELSiB	ELSiA	TOVi	CHiMAX

CHiF - признак срабатывания i-го канала, доступный только для чтения; принимает значение CHiF = 1, если канал фиксирует событие в режиме захвата или формирует выходной сигнал в режиме совпадения;

CHiIE - разрешает при значении CHiIE = 1 формирование запроса прерывания при срабатывании i-го канала (признак CHiF = 1);

MSiB, MSiA - определяют режим работы i-го канала (табл.1.7);

ELSiB, ELSiA - определяют тип события в режиме захвата или уровень выходного сигнала в режиме совпадения (табл.8.3);

TOVi - задает вариант изменения сигнала на выходе TxCHi канала, работающего в режиме совпадения или режиме формирования ШИМ-сигналов, при переполнении таймера/счетчика: сохранение текущего состояния при значении TOVi = 0, изменение состояния на противоположное при значении TOVi = 1;

CHiMAX - определяет выбор коэффициента заполнения Km в режиме формирования ШИМ-сигналов: при значении CHiMAX=0 коэффициент Km определяется содержимым регистра данных i-го канала TxCHi, при CHiMAX=1 коэффициент Km= 1.

Каждый канал таймера может работать в режиме захвата, совпадения или формирования ШИМ-сигналов в зависимости от значения битов MSiB-MSiA и ELSiB-ELSiA в регистре TxSCi (табл.2.7 тоько режим совпадения кодов).

Таблица 2.7. Режимы работы таймерных каналов модуля TIM08.

Режим канала	MSiB-A	ELSiB-A	Производимые действия
Режим совпадения или выдачи ШИМ-сигналов (без буферизации)	0 1	0 1	Изменение уровня на выводе TxCHi при совпадении
	0 1	1 0	Установка 0 на выводе TxCHi при совпадении
	0 1	1 1	Установка 1 на выводе TxCHi при совпадении
Режим совпадения или выдачи ШИМ-сигналов (с буферизацией)	1 X	0 1	Изменение уровня на выводе TxCHi при совпадении
	1 X	1 0	Установка 0 на выводе TxCHi при совпадении
	1 X	1 1	Установка 1 на выводе TxCHi при совпадении

В режиме совпадения (значения битов MSiB-MSiA = 01) в регистр данных TxCHi соответствующего канала предварительно записывается код Kc, задающий время срабатывания. Когда содержимое счетчика TxCNT становится равным этому коду, на выходе TxCHi соответствующего канала формируется определенный сигнал, а в регистре TxSCi устанавливается значение признака CHiF = 1. Вид выдаваемого на выходе TxCHi сигнала

(логический 0, логическая 1 или изменение уровня сигнала на противоположный) определяется значением битов ELSiB-ELSiA (табл. 1.7). Установка признака CHiF = 1 вызывает формирование запроса прерывания таймера, если значение бита разрешения прерывания CHiIE = 1.

9.6 НАСТРОЙКА РЕГИСТРОВ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ ТАЙМЕРА 1

9.6.1 ФОРМИРОВАНИЕ СИГНАЛА С ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИЕЙ

В лабораторной работе необходимо сформировать сигнал с широтно-импульсной модуляцией (ШИМ) для управления яркостью светодиода, подключенного к выходу1 таймерного канала1 PTD4/T1CH1 (рис. 2.10).

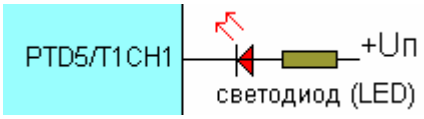


Рис.2.10. Подключение светодиода

Сигнал с ШИМ имеет постоянный период повторения - Tc, и переменную длительность (ширину) – Tp (рис. 2.11).

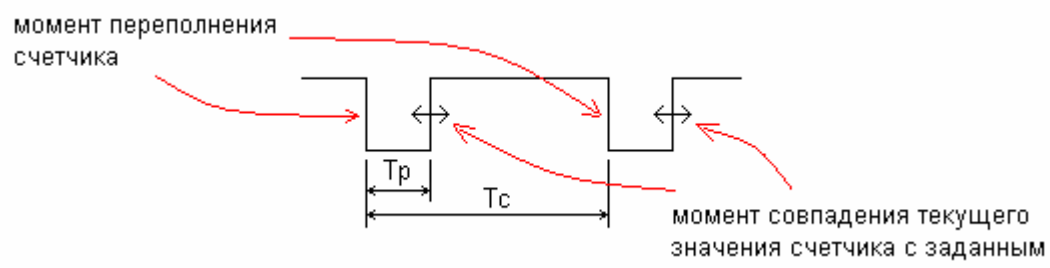


Рис. 2.11. Широтно-импульсная модуляция

Широтно-импульсная модуляция часто используется для построения ЦАП. В этом случае сформированную импульсную последовательность подают на вход фильтра нижних частот (ФНЧ), например RC – цепочку. На выходе ФНЧ будет выделена постоянная составляющая сигнала пропорциональная отношению Tp / Tc. В лабораторной работе в качестве ФНЧ будут наши глаза, которые при достаточно высокой частоте повторения 1/Tc будут фиксировать усредненное значение яркости свечения за период повторения Tc.

Определим необходимые значения разрядов регистра управления и состояния таймера1 T1SC. Конкретные значения модуля счета, длительности импульса ШИМ и значение коэффициента предварительного деления Kd

могут быть в нашей задаче любыми допустимыми, т.к. оценка результата будет производиться буквально “на глаз”.

Коэффицент деления системной частоты (CGMOUT) Kd=Ft/Fc (рис. 2.8 и 2.9) выберем равным 64. Этому соответствуют биты предварительного делителя (prescaler’a) PS2,PS1,PS0=110. Таким образом, на счетный вход счетчика таймера1 будут поступать импульсы с частотой 4.915.200Гц/2/64 = 38400Гц. Частота кварцевого генератора принудительно делится в два раза (рис.1.5-1).

Для разрешения прерываний при достижении счетчиком заданного модуля счета необходимо установить бит TOIE=1. Модуль счета таймера1 выберем равным T1MOD=400. В результате таймер будет срабатывать с частотой 38400Гц/400=96Гц (меньшая частота будет приводить к заметному мельканию светодиода). В результате, в регистр T1SC нужно записать значение 0x46.

Теперь необходимо задать значения разрядов регистра регистра управления-состояния T1SC1 первого канала таймера1.

Сначала необходимо разрешить прерывания при совпадении текущего кода в регистре счетчика T1CNT и кода записанного в регистр T1CH1 (рис.1.5). Для этого бит CH1IE должен быть равен 1.

Далее необходимо устанавливать 1 на выходе PTD5/T1CH1 при совпадении кодов. Для этого биты MS1B=0, MS1A=1; ELS1B=ELS1A=1. Для возврата нуля на этом выходе при переполнении бит TOV1 должен быть равен 1. При этом будет сформирован отрицательный импульс длительностью Tp.

```
#define LED  5 //== 5-й бит порта D (светодиод)
#define Tc 400 //== модуль счета таймера1 (38400/Tc целое число)
  ..................................................
InitDevices(){//== установка режимов работы портов и периферии
  ..................................................
  //========= настройка первого канала таймера1
  T1SC1=0x40;//== CH1IE=1 - разр. прерывания при совпадении
             //== кодов канала1 таймера1
  T1SC1|=0x10|0x0C;//== MS1B=0,MS1A=1; ELS1B=ELS1A=1
                   //== (выход T1CH1=1 при совп-нии)
  T1SC1|=0x02;//== бит TOV1=1 при переполнении смена значения
              //== на выходе T1CH1(снова 0)
  T1CH1=1;//== начальная длительность имп-са ШИМ (Tp)
  //========= настройка таймера1
  T1SC=0x46;//== TOIE=1(разр. прер.),4.915.200Гц/2/64=38400Гц
            //== (PS2..PS0=110(bin)->(64))
  //== на таймер1 поступают имп-сы с част. Fosc/2=4.915.200Гц/2 Гц
  T1MOD=Tc;//== загружаем модуль счета для режима ШИМ (Tc=400)
  //========= настройка линии порта D для управления светодиодом
  DDRD=1<<LED;//== 5-й вывод настраиваем, как выход для светодиода
  ..................................................
}
```

Рис. 2.12. Настройка таймера на заданный режим работы

Таким образом в регистр T1SC1 необходимо записать: T1SC1 = 0x40 | 0x10 | 0xC | 0x2 = 0x5E. Начальное значение регистра совпадения кодов установим равным T1CH1=1. В обработчике прерывания оно будет линейно увеличиваться до 200.

Соответствующий фрагмент программы инициализации таймера1 и таймерного канала1 приведен на рис. 2.12.

ПРИМЕЧАНИЕ: В процессе изменения (перезагрузки) содержимого регистра TxCHi, возможен пропуск момента совпадения. Если загружаемый в TxCHi код Kc оказывается меньше, чем текущее содержимое счетчика TxCNT в момент загрузки, то совпадение будет зафиксировано только в следующем цикле работы счетчика (после его переполнения и сброса в нулевое состояние). Таким образом, в течение текущего цикла работы счетчика на выход TxCHi не будет выдан сигнал совпадения, а в случае формирования ШИМ-сигналов возможен пропуск одного импульса. Исключить такие ошибки можно путем настройки канала на работу в режиме совпадения с буферизацией.

Режим совпадения с буферизацией включается при установке в регистре TxSCi значения бита MSiB = 1 (табл.1.7). Для реализации этого режима используются регистры данных TxCHi двух таймерных каналов одновременно, поэтому каналы объединяются в пару 0-1. В паре канал с номером 0 работает в режиме совпадения с буферизацией, используя регистр данных канала с номером 1 в качестве буфера для загрузки нового кода Kc. Регистр TxSC0 задает режим работы канала. Таким образом, для выдачи сигналов совпадения или ШИМ-сигналов в этом режиме может использоваться только выход TxCH0.

Так как в лабораторном стенде светодиод подключен к выходу T1CH1, режим с буферизацией не используется.

9.6.2 ФОРМИРОВАНИЕ СИГНАЛОВ ТОЧНОГО ВРЕМЕНИ

Как уже было рассчитано, прерывания по переполнению происходят с частотой 38400Гц/400=96Гц. Поэтому в обработчике необходимо отсчитывать с помощью вспомогательной переменной 96 прерываний и увеличивать число секунд на 1. Отсчитав 60 секунд с помощью переменной ‘sec’ увеличиваем число минут (переменная ‘min’).

Соответствующий фрагмент программы с подпрограммами обработки прерываний по переполнению таймера1 (TIM1_ovf) и совпадению кодов таймерного канала1 (TIM1_cmp) приведен на рис. 2.13:

```
#define TOF    7 //== 7-й бит байта T1SC (флаг переполнения таймера1)
.....
#pragma interrupt_handler TIM1_ovf
unsigned int d;//== вспомогательная глобальная переменная
char sec=0,min=0,m=0;
void TIM1_ovf(void){//== обработчик прерывания по переполнению таймера1
  T1SC&=~(1<<TOF); //== программно обнуляем флаг переполнения TOF таймера1
  d=T1CH1;//== читаем текущую длительность импульса Tp ШИМ
  d++;//== увеличиваем ее
  if (d==201)d=1;//= восстанавливаем начальное значение длительности
                 //= импульса Tp
  T1CH1=d;//== увеличим длительность имп-са ШИМ на выходе T1CH1
          //= в следующем периоде
  if(++m==(38400/Tc)){m=0;//== заодно отмеряем секунды (прошла секунда)
    if(++sec==60){sec=0;if(++min==60)min=0;}//== и минуты
  }
}
#pragma interrupt_handler TIM1_cmp
void TIM1_cmp(void){//==обработчик прерывания по совпадению кодов таймера1
  T1SC1&=~(1<<CH1F);//==программно обнуляем флаг совпадения CH1F таймера1
}
```

Рис. 2.13. Обработчики прерываний по переполнению счетчика и при совпадении кодов

9.7 СИНХРОННЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ ИНТЕРФЕЙС И ТЕРМОДАТЧИК DS1722S

Обмен данными по интерфейсу SPI производится между двумя устройствами, одно из которых является ведущим (master), а другое ведомым (slave) – рис. 2.14.

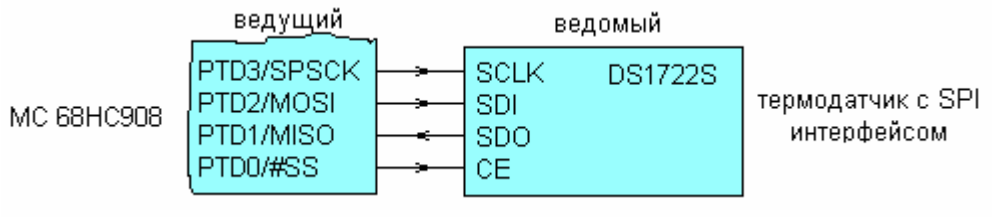


Рис. 2.14. Термодатчик с SPI интерфейсом

В SPI используются четыре сигнальных линии, которые выполняют следующие функции:

" SPSCK - выход синхросигнала ведущего и вход SCLK синхросигнала ведомого модуля;

" MOSI - выход данных для ведущего и вход SDI данных для ведомого модуля;

" MISO - вход данных для ведущего и выход SDO данных для ведомого модуля;

" SS# - вход сигнала выбора ведомого модуля: на этот вход для ведомого модуля необходимо подать сигнал SS#=0, для ведущего модуля - сигнал SS#=1. ВНИМАНИЕ: В микросхеме DS1722S вход выбора ведомого модуля -

прямой и имеет обозначение CE (Chip Enable, тоже что и ChipSelect), поэтому для активизации температурного датчика на этот вход нужно подавать единицу. В микроконтроллере MC68HC908GP32 выводы модуля SPI08 совмещены с выводами PTD3..0 порта D.

9.7.1 МОДУЛЬ СИНХРОННОГО ПОСЛЕДОВАТЕЛЬНОГО ИНТЕРФЕЙСА

Модуль SPI08 содержит два отдельных регистра данных, которые имеют общее имя SPDR. Один из этих регистров доступен только для записи и является буфером данных передатчика. Второй регистр доступен только для чтения и служит буфером данных приемника. Управление работой модуля SPI08 осуществляется с помощью регистра управления SPCR и регистра состояния-управления SPSCR, форматы содержимого которых приведены на рисунках 2.15 и 2.16.

SPCR

7	6	5	4	3	2	1	0
SPRIE	0	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE

Рис. 2.15. Регистр управлениия SPI

SPSCR

7	6	5	4	3	2	1	0
SPCRF	ERRIE	VRF	MODF	SPTE	MODFEN	SPR1	SPR0

Рис. 2.16. Регистр управления и состояния SPI

Разряды регистра SPCR имеют следующее назначение:
SPRIE - разрешает при значении SPRIE=1 формирование запроса прерывания после окончания приема данных (в регистре SPSCR признак SPRF=1);
SPMSTR - устанавливает режим работы модуля: в качестве ведущего при значении SPMSTR=1 или в качестве ведомого при значении SPMSTR=0;
CPOL, CPHA - определяют полярность и фазу синхросигналов обмена;
SPWOM - задают режим работы выходных буферных каскадов на выводах SPSCK, MOSI, MISO: обычный двухфазный выходной каскад при значении SPWOM=0, выход с "открытым стоком" при SPWOM=1;
SPE - разрешает при SPE=1 или запрещает при SPE=0 работу модуля SPI08;
SPTIE - разрешает при значении SPTIE=1 формирование запроса прерывания, когда буфер передатчика готов к приему данных (в регистре SPSCR признак SPTE=1).

Назначение разрядов регистра SPSCR:
SPCRF - признак завершения приема данных, принимает значение SPCRF=1 после ввода последнего бита данных и их перезаписи в буферный регистр приемника;

OVRF - признак переполнения, принимает значение OVRF=1, если в сдвиговый регистр поступают новые данные в то время, как ранее принятые данные еще не считаны из буферного регистра приемника;
MODF - признак ошибки режима, принимает значение MODF=1, если на ведущий модуль подан сигнал SS#=0, или на ведомый модуль в процессе передачи поступает сигнал SS#=1;
SPTE - признак освобождения буфера передатчика, принимает значение SPTE=1, когда данные из буфера передатчика переписываются в сдвиговый регистр для реализации обмена.
В регистре SPSCR содержатся также управляющие биты, доступные для записи-чтения:
ERRIE - разрешает при значении EERIE=1 формирование запроса прерывания при установке признаков OVRF=1 или MODF=1;
MODFEN - разрешает при значении MODFEN=1 контроль режима работы модуля SPI08 путем установки признака MODF;
SPR1..0 - задают значения коэффициента деления Kd, определяющего частоту синхросигналов обмена

SPR1-0	Kd
00	2
01	8
10	32
11	128

В процессе обмена данными сдвиговые регистры ведущего и ведомого модулей соединяются в кольцо. При этом в каждом такте обмена производится ввод бита в младший разряд сдвигового регистра ведущего и ведомого модулей и вывод старшего бита из этого регистра. После окончания передачи 8-ми битного символа в регистре SPSCR устанавливается признак завершения обмена SPCRF=1. При этом формируется запрос прерывания, если в регистре SPCR установлен бит разрешения прерывания SPRIE=1.

В исходном состоянии на выходе SPSCK ведущего SPI поддерживается постоянный уровень 0, если в регистре SPCR установлено значение бита полярности синхросигналов CPOL=0, или уровень 1, если CPOL=1. Обмен производится 8-разрядными символами, которые поступают в сдвиговый регистр после записи данных в регистр SPDR ведущего модуля SPI08. При этом на выход SPSCK ведущего модуля поступают синхросигналы, положительный или отрицательный фронт которых определяет начало передачи очередного бита и момент его ввода в принимающий регистр. Выбор активного фронта синхросигнала определяется значением бита фазы синхронизации CPHA, который задает формат передачи данных.

9.7.2 РАБОЧИЕ РЕГИСТРЫ ТЕРМОДАТЧИКА DS1722

В лабораторном стенде имеется цифровой термометр DS1722 (термодатчик), который поддерживает обмен данными по интерфейсу SPI в режиме ведомого. После включения питания DS1722 находится в режиме ожидания. Для того чтобы активировать термометр, необходимо дать команду на выполнение одного цикла преобразования температуры или разрешить непрерывную работу термометра. В последнем случае внутренняя схема DS1722 начинает непрерывный цикл определения температуры, результат которого сохраняется во внутренних регистрах устройства. Результат преобразования представляется в виде 8- (точность 1°C), 9-, 10-, 11- или 12-разрядного (точность 0.0625°C) числа. Чем точнее результат, тем больше времени требуется на преобразование температуры в цифровой код.

Двухбайтовый код температуры хранится во внутренних регистрах DS1722 с адресами $01 для младшего и $02 для старшего байтов. Значение температуры дается в дополнительном коде: старший байт содержит целую часть значения температуры, а младший - дробную.

Управляющий байт, определяющий режим работы термометра и точность преобразования находится во внутреннем регистре управления DS1722. Этот регистр имеет адрес $00 для считывания и $80 для записи. Содержимое регистра управления представлено на рисунке 2.17:

7	6	5	4	3	2	1	0
1	1	1	1SHOT	R2	R1	R0	SD

Рис. 2.17. Регистр управления термодатчиком

1SHOT - если бит SD = 1, то установка 1SHOT = 1, приведет к выполнению одного цикла преобразования, после чего бит 1SHOT будет сброшен в 0.
R0-R2 - данные биты определяют точность преобразования в соответствии с таблицей 2.8.

Таблица 2.8

R2	R1	R0	Точность преобразования, бит	Время преобразования, с
0	0	0	8	0.075
0	0	1	9	0.15
0	1	0	10	0.3
0	1	1	11	0.6
1	x	x	12	1.2

SD - данный бит определяет режим работы DS1722 по окончании очередного цикла преобразования. Если SD = 0, то термометр находится в режиме непрерывного преобразования, если SD = 1, то после текущего цикла преобразования DS1722 перейдет в режим ожидания.
Три старших бита D7..5 имеют предопределенное значение – “1”.

9.7.3 НАСТРОЙКА SPI ДЛЯ РАБОТЫ С ТЕРМОДАТЧИКОМ

В процедуре инициализации InitDevices необходимо настроить вывод 0 порта D в качестве выхода сигнала #SS, поэтому в соответствующий бит регистра DDRD записываем 1 (не изменяя значения остальных разрядов этого регистра). Начальное значение выхода PTD0 = CE = 0, т.е. при инициализации термометр будет в пассивном состоянии. В биты SPMSTR и SPE регистра SPCR записываем 1, настраивая микроконтроллер в качестве ведущего устройства и разрешая работу интерфейса SPI. Соответствующий фрагмент процедуры InitDevices будет выглядеть следующим образом (Рис. 2.18):

```
#define SS 0 //== 0-бит портаD(вывод #SS SlaveSelect) использ.как ChipSelect
  .....
InitDevices(){//== установка режимов работы портов и периферии
  .....
  //========= настройка последовательного периферийного интерфейса
  //========= SPI (порт D)
  DDRD|=1<<SS;//== вывод #SS настраиваем в качестве выхода управления
              //== (ChipEnable)
  PTD&=~(1<<SS);//== #SS=CE=0 (термодатчик - не выбран по входу CE(CS)
  SPCR=0x2A;//== SPMSTR=1(MCU - ведущий (мастер)), SPE=1(разрешение SPI)
  .....
}
```

Рис. 2.18. Настройка SPI

9.7.4 НАСТРОЙКА ТЕРМОДАТЧИКА НА ЗАДАННЫЙ РЕЖИМ РАБОТЫ И ЧТЕНИЕ КОДА ТЕМПЕРАТУРЫ

В лабораторной работе будем использовать циклическое однократное измерение температуры, поэтому биты 1SHOT = SD = 1. Температуру будем измерять с точностью до 1°C (разряды R2,R1,R0 = 0). Управляющий байт для термометра в этом случае равен 11110001 = F1. Запись режима работы термодатчика оформим в виде процедуры SPIctrl() (рис. 2.19-1), а передачу байта от МК к термодатчику с помощью функции Send_SPI_Byte() (рис. 2.19-2):

```
SPIctrl(){//== передаем адрес регистра управления и управл.байт DS1722
  PTD|=1<<SS;//== активируем термодатчик(ТД) по входу CE(он же ChipSelect)
  Send_SPI_Byte(0x80);//== 80-адрес рег-ра управл. ТД для записи
                      //== (00-для чтения)
  Send_SPI_Byte(0xF1);//== 1SHOT=SD=1(однокр.пуск),
                      //==  R2=R1=R0=0(точность t-ры 8 бит)
  //== |1|1|1|1SHOT|R2|R1|R0|SD| - формат управл. байта ТД
  PTD&=~(1<<SS);//== деактивируем термодатчик по входу CE
  Delay(150);//== необходимая задержка
}
```

Рис. 2.19-1. Режим работы термодатчика

```
Send_SPI_Byte(char c){//== передача байта от ведущего к ведомому
  SPDR=c;//== при записи в регистр SPDR, байт передается по линии MOSI
  //== в ведомое устройство (термодатчик)автоматически
  while((SPSCR & (1<<SPCRF))==0)continue;//== проверяем бит готовности
  c=SPDR;//== нужно обязательно прочитать !!!! (и отбросить)
  //== т.к. регистры данных ведущего и ведомого SPI устр-в замкнуты в кольцо
}
```

Рис. 2.19-2. Запуск термодатчика

Передача адресов старшего и младшего байтов температуры производится в процедуре SPIdata(), а чтение и индикацию температуры будем осуществлять в бесконечном цикле основной функции main().

```
SPIdata(){//== передаем адреса регистров температуры
  PTD|=(1<<SS);//== активируем термодатчик (ТД) по входу CE
  Send_SPI_Byte(0x02);//== передаем адрес регистра старшего байта t-ры
  Send_SPI_Byte(0x02);//== при 8-ми битной t-ре СНОВА ст.байт
  //== при 9..12-ти битной t-ре СНАЧАЛА 0x01(мл.байт), ЗАТЕМ 0x02(ст.)
  PTD&=~(1<<SS);//== снова деактивируем термодатчик по входу CE
}
.....

  SPIctrl();//== передаем адрес регистра управления и управляющий байт ТД
  .....
  while(8888){//== бесконечный цикл
    SPIdata();//== передаем адреса регистров данных термодатчика
    LCDNxy(6,1); //== переходим к позиции LCD, где будет отображаться t-ра
    LCDN_ReadSPIValue();//== читаем и выводим t-ру на LCD
  .....
  }
LCDN_ReadSPIValue(){//== чтение и отображение принятого байта
char c,s[10];
  c=SPDR; //== читаем принятый байт из внутр. буфера SPI
  sprintf(s,"%d",(signed char)c);//== преобразуем целый тип со знаком
                                 //== в строку ASCII
  LCDNstrn(s);//== отображаем температуру на ЖКИ
}
```

Рис. 2.20. Считывание кода температуры из датчика и ее отображение на ЖК дисплее

9.8 ПРОГРАММИРОВАНИЕ ЖК ДИСПЛЕЯ С 4-Х БИТНЫМ ИНТЕРФЕЙСОМ

В лабораторном стенде управляющие выводы дисплея RS(~C/D), RW(R/~W) и E подключены к выводам порта C (PTC6..4). 8-ми битный код символа выводится на дисплей поочередно двумя тетрадами через 4-ре старших бита шины данных ЖК дисплея (рис. 2.21).

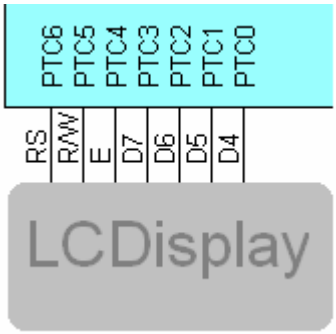


Рис. 2.21. 4-х битный интерфейс ЖКД.

9.8.1 СПРАВОЧНЫЕ СВЕДЕНИЯ ДЛЯ ПРОГРАММИРОВАНИЯ ЖКД

Символьные ЖКД служит для отображения символов ASCII и др. с помощью матрицы точек 5x8 или 5x10. ЖКД имеет встроенный контроллер и три блока памяти: DDRAM (Data Display RAM) – память данных, сюда пользователь записывает коды, отображаемых на дисплее символов, CGROM (Character Generator ROM) – ПЗУ знакогенератор, здесь хранятся образы символов и CGRAM (Character Generator RAM) – ОЗУ знакогенератор, область, в которую программист может записывать коды дополнительных (отсутствующих в CGROM) символов.

Объем памяти данных ЖКД (DDRAM) составляет 80 байт (80 символов). В двухстрочном 16-ти и 20-ти символьных ЖКД адрес крайней левой позиции верхней строки ЖКД (знакоместа) равен 00h, адрес крайней левой позиции нижней строки = 40h (таблица 2.9).

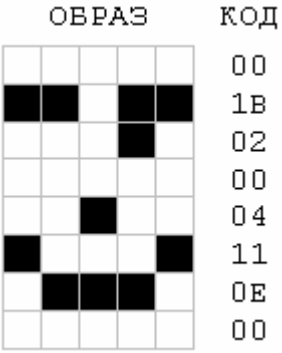
Таблица 2.9 (соответствие между адресами DDRAM и позициями ЖКД)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0f	1	1	1	1
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e		0	1	2	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4f	5	5	5	5
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e		0	1	2	3

В таблице 2.10 приведены образы символов размером 5 х 8 точек, хранящиеся в ПЗУ (CGROM) ЖКД, используемого в УМК. Цветом выделены 8 ячеек ОЗУ (CGRAM) ЖКД, в которые пользователь может записать коды дополнительных символов. Другие 8 ячеек CGRAM, разработчики отобразили на первые 8 адресов. Поэтому попытка разместить в CGRAM’е 16 пользовательских символов не увенчается успехом.

Таблица 2.10

Upper 4 bit / Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	CG RAM (2)															
LLHL	CG RAM (3)															
LLHH	CG RAM (4)															
LHLL	CG RAM (5)															
LHLH	CG RAM (6)															
LHHL	CG RAM (7)															
LHHH	CG RAM (8)															
HLLL	CG RAM (1)															
HLLH	CG RAM (2)															
HLHL	CG RAM (3)															
HLHH	CG RAM (4)															
HHLL	CG RAM (5)															
HHLH	CG RAM (6)															
HHHL	CG RAM (7)															
HHHH	CG RAM (8)															



CGROM может хранить образы символов размером 5 x 8 и 5 x 10 точек на основе нестандартного 8-ми битного кода. Латинские буквы, цифры, знаки препинания, арифметические знаки и некоторые другие соответствуют стандартным ASCII кодам, другие не соответствуют. Это значит, что записав в программе в DDRAM символ ‘F’ мы увидим в соответствующей позиции ЖКД  (ASCII код = LHLL LHHL = 0100 0110 = 46h). Буквы русского алфавита находятся “не на своих местах”. Например буква ‘Б’ в таблице CGROM имеет код = HLHL LLLL = 1010 0000 = A0h, который не соотвествует ни DOS ни WIN(CP-1251) кодировкам буквы ‘Б’. Например в WIN кодировке код ‘Б’ = c1h. Поэтому при записи в программе вывода на ЖКД буквы ‘Б’, на дисплее отобразится символ  , в соответствии с кодировкой буквы ‘ш’ в CGROM. В таких случаях, в программе для вывода на дисплей символов с нестандартными кодами, например буквы ‘Б’ придется записать не символ ‘Б’, а его CGROM код ‘\xa0’ (по правилам языка Си).

Помимо приведенных в таблице CGROM символов, пользователь может “нарисовать” до 8-ми своих собственных и разместить их в CGRAM. Например для “смайла” (см. таблицу 1.9.1) пользователь должен записать 8 указанных байтов (в которых значащими являются только 5 младших битов – матрица 5x8) по одному из адресов CGRAM (0..7)

Большинство ЖКД имеют встроенный контроллер типа 44780 (Hitachi). Команды (управляющие байты) контроллера приведены в таблице 2.11. В максимальной конфигурации управление производится с помощью 8-ми выводов: RS(~Control/Data), RW(Read/~Write) и E, а также D7..D0. По линии E – подается строб-импульс. RS определяет, чем будут обмениваться контроллер и МК: управляющей информацией при RS=0 или байтом данных RS=1. Сигнал RW или точнее R/~W, естественно определяет, что производится : запись или чтение.

Таблица 2.11

RS(~C/D)	R/~W	D7	D6	D5	D4	D3	D2	D1	D0	Команда/функции
0	0	0	0	0	0	0	0	0	1	Очистка дисплея
0	0	0	0	0	0	0	0	1	x	Возвращение курсора в исходное состояние
0	0	0	0	0	0	0	1	ID	S	Задание направления перемещения курсора
0	0	0	0	0	0	1	D	C	B	Разрешение отображения курсора
0	0	0	0	0	1	SC	RL	x	x	Смещение курсора / сдвиг изображ. на дисплее
0	0	0	0	1	DL	N	F	x	x	Сброс/задание параметров интерфейса
0	0	0	1	A	A	A	A	A	A	Перевод курсора в CGRAM
0	0	1	A	A	A	A	A	A	A	Перевод курсора на экран дисплея по адресу A..A
1	0	H	H	H	H	H	H	H	H	Запись символа в текущую позиц. курсора
0	1	BF	x	x	x	x	x	x	x	Проверка признака "занято"
1	1	H	H	H	H	H	H	H	H	Считывание символа, указываемого курсором

- ID - автоинкрементирование позиции курсора
- S - сдвиг изображения после записанного байта
- D - включение / отключение дисплея (1/0)
- C - разрешение / запрет использования курсора (1/0)
- B - задание / отмена зежима мерцания курсора (1/0)
- SC - разрешение / запрещение сдвига изображения (1/0)
- RL - направление сдвига: вправо / влево(1/0)
- DL - разрядность данных: 8/4 бита (1/0)
- N - число строк изображения: 1/2 (0/1)
- F - размер знакоместа: 5x10/5x8 (1/0)
- BF - флаг занятости: равен 1 в процессе обработки данных ЖК дисплеем
- A – адрес
- H – данные

9.8.2 ПРОГРАММИРОВАНИЕ ЖКД

Прежде всего необходимо настроить выводы порта C в качестве выходов (рис. 2.22):

```
InitDevices(){//== установка режимов работы портов и периферии
  ....................................................
  //========= настройка порта C для работы с ЖКИ
  DDRC=0x7F;//== линии 0..6 порта C настроены на вывод в LCD
  PTC=0;//== начальное значение
}
```

Рис. 2.22. Настройка порта ЖКД

Затем нужно написать подпрограмму вывода 4-х бит (тетрада, ниббл) в порт PTC3..0. Заодно определим 4 полезных константы (рис. 2.23).

```
#define LCD_CTRL PTC //== PORTC
#define LCD_DATA PTC
#define LCD_RS 6 //== RS(~C/D) 6-й бит порта C
#define LCD_E  4 //== строб E (4-й бит порта C)
....................................................
LCDNibble(nib){//== выводим младшую тетраду байта nib в порт C (на LCD дисплей)
  PTC&=0xF0; //== не разрушая 3 управляющих бита обнуляем 4 старых бита данных
  PTC|=nib & 0x0F; //== вписываем 4 новых бита данных
  PTC|=1<<LCD_E; //== формируем передний фронт строба E  _┌┐_
  PTC&=~(1<<LCD_E);//== формируем задний фронт строба E  _┌┐_
  Delay(1);//== 100us (нужно >= 40)
}
```

Рис. 2.23. Подпрограмма вывода одной тетрады

Теперь можно ввести подпрограммы вывода в LCD управляющего байта (таблица 2.11), байта с кодом отображаемого символа и вывода целой строки (рис. 2.24):

```
LCDNchar(char ch){//== вывод символа в ЖК дисплнй
  LCD_DATA|=1<<LCD_RS;//== будем писать байт данных (бит RS=~C/D=1)
  LCDNibble(ch>>4);//== сначала выводим СТАРШУЮ тетраду (nibble)
  LCDNibble(ch&0x0F);//== затем выводим МЛАДШУЮ тетраду
}
LCDNctrl(char ch){//== вывод управляющего байта в ЖК дисплнй
  LCD_CTRL&=~(1<<LCD_RS);//==  будем писать управл. байт (бит RS=~C/D=0)
  LCDNibble(ch>>4);//== то же самое,
  LCDNibble(ch&0x0F);//== что и в предыдущем случае
}
LCDNstrn(char *s){ //== вывод строки в ЖК дисплей
  while(*s!=0)LCDNchar(*s++);//== выводить символы, пока *s не равно 0
}
```

Рис. 2.24. Подпрограммы вывода одного символа и строки

Заодно определим процедуры очистки дисплея и позиционирования курсора (рис 2.25):

```
LCDNclr(){LCDNctrl(1);Delay(200);}//== очистка дисплея
LCDNxy(char x,char y){//== позиционирование курсора
  //== x = 0..15 позиция символа в строке, y=0/1 - верхн./нижн. строка
  LCDNctrl(0x80+(x+((y)?0x40:0)));
}
```

Рис. 2.25. Подпрограммы очистки дисплея и позиционирования курсора

Строка “LCDNctrl(0x80+(x+((y)?0x40:0)));” означает, что в управляющем байте 1AAAAAAA (см. таблицу 2.11) адрес AAAAAAA = 1000000 = 40h соответствует нижней строке дисплея, а адрес AAAAAAA = 0000000 = 0 соответствует верхней строке дисплея (см. таблицу 2.9).

10. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

10.1 СОЗДАНИЕ ШАБЛОНА НОВОГО ПОЕКТА

Откройте интегрированную среду разработки ImageCraft IDE icc08  . для МК семейства HC08. В открывшемяся рабочем пространстве из главного меню выберите п. "Project | New ". В появившемся окне диалога перейдите в папку “C:\EMUL\Work\” и создайте в ней (кликнув правой кнопкой мыши) свою папку с именем “68hc08_grXXXX”, причем часть имени “68hc08_gr” обязательна. Вместо XXXX запишите номер своей группы, например 8888. Также впишите имя проекта, например “myHC908”. Закончите операцию создания файла проекта, нажав на кнопки “Открыть” и затем “Сохранить” (рис. 2.26).

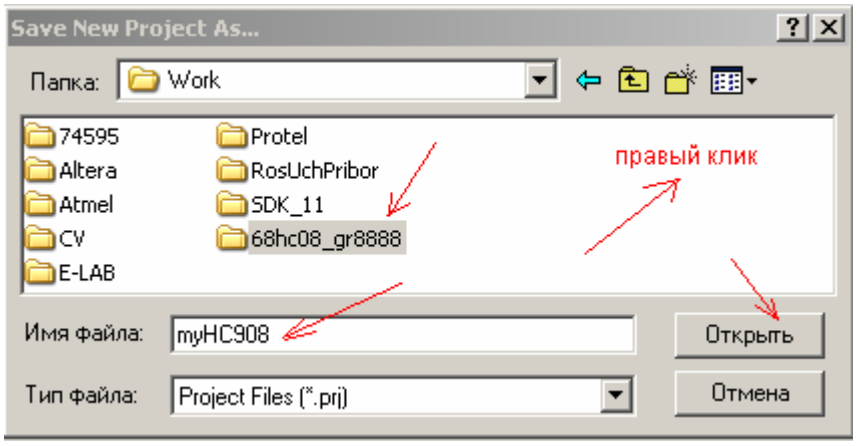


Рис. 2.26. Сохранение нового проекта

Теперь выберите п. меню “File | New”. Появится пустое окно с именем “Untitled-1”. С помощью п. меню “File | Save As” сохраните в своей папке “68hc08_gr8888” пустой пока файл под любым именем, но с раширением “c”, например my08.c (рис. 2.27).

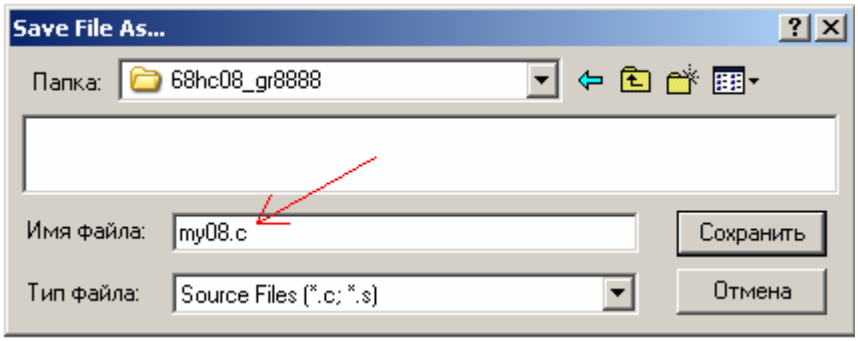
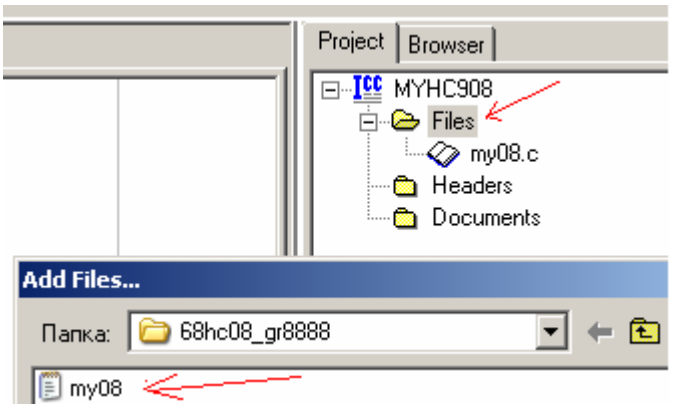


Рис. 2.27 Сохранение нового рабочего файла

Затем необходимо добавить вновь созданный файл в проект, для чего кликните правой кнопкой по узлу “Files” страницы “Project”, расположенной на панели справа и добавьте к проекту (“Add File(s)…”) свой файл “my08.c”.

Если этой панели нет, то из меню “View” отметьте птичкой “Project File Window” и панель появится..



Теперь необходимо уточнить параметры нашего проекта (рис. 2.28), для чего кликните по кнопке на панели инструментов или выберите п. меню “Project | Options…”. В открывшемся диалоговом окне на двух страницах “Target” и “Compiler” нужно отметить указанные на рисунке позиции. Причем GP32 означает тип МК в семействе 68HC08. Так как в программе понадобится стандартная функция sprintf(…) форматирования данных в формате с плавающей точкой (напряжение), то отметим тип данных “float”. Формат мотороловского выходного файла для загрузки в МК похож по структуре на интеловский HEX файл и имеет стандартное расширение *.S19.

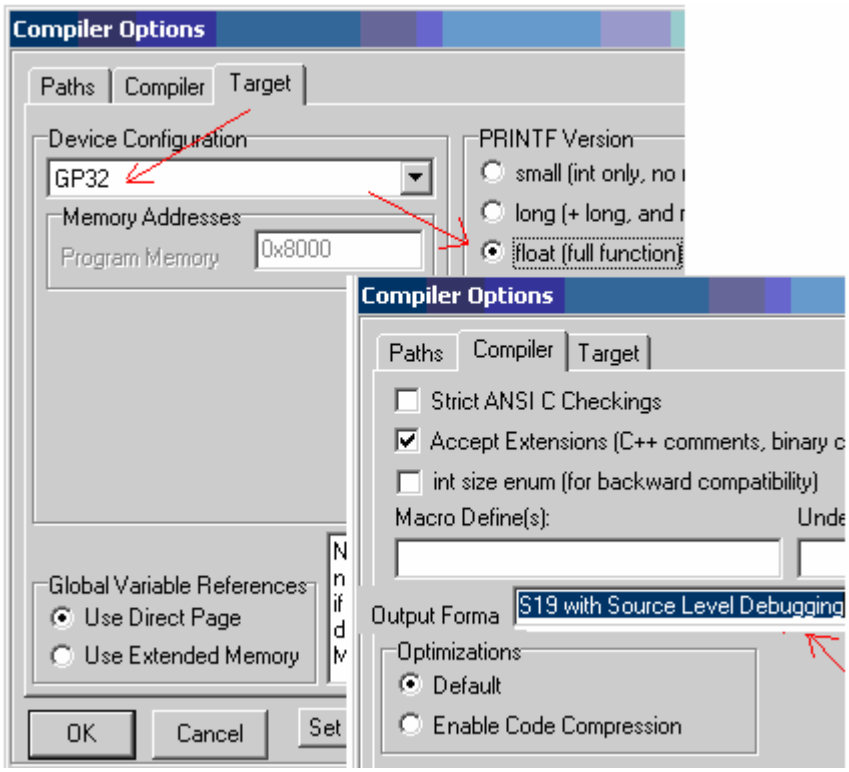


Рис. 2.28. Настройка компилятора

Нажатием на кнопку “OK” завершим создание основы проекта.

10.2 РАЗРАБОТКА И ОТЛАДКА ФУНКЦИОНАЛЬНОЙ ЧАСТИ ПРОГРАММЫ

Прежде всего необходимо подключить к проекту файл с определением портов ввода/вывода и регистров специальных функций “iogp32.h” и короткий файл с несколькими макросами языка Си “hc08def.h”. Также запишем в программу тело основной функции main. На этом этапе программа должна иметь следующий вид (рис. 2.29).

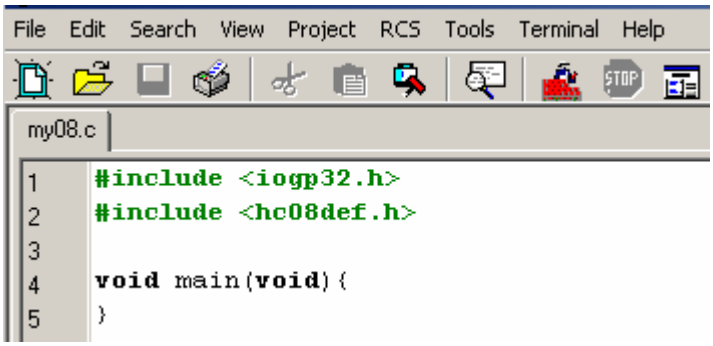
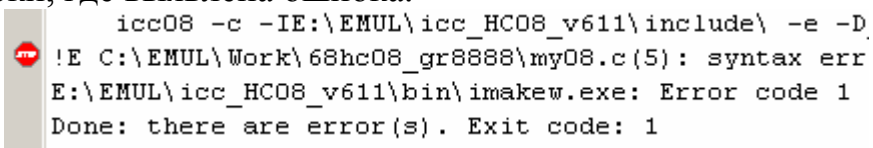


Рис. 2.29. Шаблон программы

Проведем пробную компиляцию и сборку проекта кнопкой  на панели инструментов. Так как умудриться наделать ошибок в столь короткой программе затруднительно, то компиляция должна пройти безошибочно, в противном случае в статусном окне появится предупреждающий знак с номером строки, где выявлена ошибка.



ВАЖНОЕ ПРИМЕЧАНИЕ! После каждой вставки очередного блока в текст программы выполняйте сборку проекта кнопкой  . Иначе, потом выявить все накопившиеся ошибки будет трудней.

10.2.1 НАСТРОЙКА ПОРТОВ ВВОДА/ВЫВОДА

Добавим в программу процедуру InitDevices(), инициализирующую порты ввода/вывода в соответствии с техническим заданием и рис. 2.1, на котором приведена схема подключения внешних устройств к МК 68HC908GP32. Заодно определим все необходимые константы. Три новых фрагмента, указанных на рисунке 2.30 добавьте в программу .

```
#include <iogp32.h>
#include <hc08def.h>
#include <float.h>   //== для sprintf
#include <stdio.h>   //== для sprintf
#define LCD_CTRL PTC //== PORTC
#define LCD_DATA PTC
#define LCD_RS 6 //== RS(~C/D) 6-й бит порта C
#define LCD_E  4 //== строб E (4-й бит порта C)
#define LED    5 //== 5-й бит порта D (светодиод)
#define SS     0 //== 0-й бит порта D (вывод #SS)
#define SPCRF  7 //== 7-й бит байта SPSCR (флаг завершения приема SPI)
#define TOF    7 //== 7-й бит байта T1SC (флаг переполнения таймера1)
#define CH1F   7 //== 7-й бит байта T1SC1 (флаг совпадения кодов таймера1)
#define Tc   400 //== модуль счета таймера1 (38400/Tc - д.б. целое число)
#define ACKK   2 //== бит подтверждения прерывания от клавиатуры
#define IMASKK 1 //== если бит IMASKK=0, прерывания от клавиатуры разрешены

InitDevices(){//== установка режимов работы портов и периферии
  CONFIG1|=1;//== запретить сторожевой таймер COP (бит COPD=1)
  //== биты в регистры CONFIG записываются ОДНОКРАТНО,
  //== и в дальнейшем не могут переопределяться (до след. запуска)
  //========= настройка АЦП
  ADCLK=0x60;//== задаем тактовую частоту АЦП - ADIV1,0=1(Fq/8),ADICLK=0
  ADSCR=0x47;//== AIEN=1(разр.прерыв.),ADCH4..ADCH0=00111(канал AD7(PTB7)и пуск)
  //========= настройка первого канала таймера1
  T1SC1=0x40;//== CH1IE=1 - разр. прерывания при совпадении кодов
  T1SC1|=0x10|0x0C;//== MS1B=0,MS1A=1; ELS1B=ELS1A=1 (выход T1CH1=1 при совп)
  T1SC1|=0x02;//== бит TOV1=1 при переполнении смена значения на выходе T1CH1
  T1CH1=1;//== начальная длительность имп-са ШИМ (Tp)
  //========= настройка таймера1
  T1SC=0x46;//== TOIE=1(разр. прер.),4.915.200Гц/2/64=38400Гц
  //== на таймер1 поступают импульсы с частотой Fosc/2=4.915.200Гц/2 Гц
  T1MOD=Tc;//== загружаем модуль счета для режима ШИМ (Tc=400)
  //========= настройка линии порта D для управления светодиодом
  DDRD=1<<LED;//== 5-й вывод настраиваем, как выход для светодиода
  //========= настройка последовательного периферийного интерфейса SPI (порт D)
  DDRD|=1<<SS;//== вывод #SS настраиваем в качестве выхода управления(ChipEnable
  PTD&=~(1<<SS);//== #SS=CE=0 (термодатчик - не выбран по входу CE(CS))
  SPCR=0x2A;//== SPMSTR=1(MCU - ведущий (мастер)), SPE=1(разрешение SPI)
  //========= настройка порта C для работы с ЖКИ
  DDRC=0x7F;//== линии 0..6 порта C настроены на вывод в LCD
  PTC=0;//== начальное значение
}
void main(void){
  InitDevices();//== установка режимов работы портов и периферии
  while(8888){//== бесконечный цикл
  }
}
```

Рис. 2.30. Определение констант и инициализация портов

Бесконечный цикл необходим (while(8888)), т.к. иначе микроконтроллер закончив выполнение ф-ии main() будет бесконтрольно выбирать неинициализированные байты из свободной области памяти и выполнять их. Это приведет либо к “зависанию”, либо к перезагрузке системы, если МК снова доберется до нулевого адреса. В худшем случае,

если последовательность таких байтов образует вредоносный код, то лабораторный стенд может выйти из строя.

Строчки #include <float.h> и #include <stdio.h> необходимы для определения чисел с плавающей точкой и функции sprintf() преобразования выводимых данных в один из нескольких форматов: целые числа знаковые и беззнаковые в любой системе счисления, числа с плавающей точкой, буквенно-цифровые символы и строки символов.

Проведите компиляцию и сборку проекта кнопкой  на панели инструментов. Исправьте возможные на этом этапе синтаксические и грамматические ошибки и снова откомпилируйте проект. На этом этапе функция (main) программы не выполняет ни одного полезного действия, поэтому приступим к постепенному наполнению ее содержанием.

10.2.2 ПРОГРАММИРОВАНИЕ ТАЙМЕРА В РЕЖИМЕ ШИМ

```
#define ACKK    2//== бит подтверждения прерывания от клавиатуры
#define IMASKK 1//== если бит IMASKK=0, прерывания от клав.разрешены
//== определяем обработчики прерываний
#pragma interrupt_handler TIM1_ovf
unsigned int d;//== вспомогательная глобальная переменная
void TIM1_ovf(void){//== обработчик прерыв.по переполн. таймера1
  T1SC&=~(1<<TOF); //== программно обнуляем флаг переполнения TOF
  d=T1CH1;//== читаем текущую длительность импульса Tp ШИМ
  d++;//== увеличиваем ее
  if (d==201)d=1;//= восстанавл.начальное значение длит импульса Tp
  T1CH1=d;//== увеличим длительность имп-са ШИМ на выходе T1CH1
}
#pragma interrupt_handler TIM1_cmp
void TIM1_cmp(void){//== обработчик прерыв.по совп.кодов таймера1
  T1SC1&=~(1<<CH1F);//== программно обнуляем флаг совпадения CH1F
}
//== инициализируем таблицу векторов прерываний по соотв. адресам
#pragma abs_address:0xFFF2
void (*_vec_tim1_ovf)(void) = TIM1_ovf;
#pragma end_abs_address
#pragma abs_address:0xFFF4
void (*_vec_tim1_ch1)(void) = TIM1_cmp;
#pragma end_abs_address
InitDevices(){//== установка режимов работы портов и периферии
  CONFIG1|=1;//== запретить сторожевой таймер COP (бит COPD=1)
. . . . . .
void main(void){
  InitDevices();//== установка режимов работы портов и периферии
  ADSCR=0x1F; //== временно запрещаем работу АЦП
  //== AIEN=0(запр.прерыв.), ADCH4..ADCH0=11111(11111 - останов АЦП)
  CLI();//== разрешить прерывания,(SEI()-запретить)
  //== ВНИМАНИЕ: в других языках прогр. cli - запр., sei(sti)- разр.
  while(0x8888){//== бесконечный цикл
  }
}
```

Рис. 2.31. Программирование таймера в режиме ШИМ

В соответствии с техническим заданием добавим код управления таймером в

режиме широтно-импульсной модуляции - ШИМ (PWM) для управления яркостью свечения светодиода. На рисунке 2.31 два новых фрагмента программы помещены в рамки.

Откомпилируйте программу кнопкой  и исправьте возможные ошибки. Если компиляция прошла без ошибок, то в рабочей папке появится 16-ный мотороловский файл “myHC908.s19” в формате “S Record”. Этот файл является аналогом интеловского HEX-файла (см. лабор. работу №10) и включает машинный код рабочей программы с адресами. Следующим этапом будет загрузка этого кода через COM порт компьютера в целевую плату. Включите питание стенда (адаптер и переключатель SA1).

10.2.3 ЗАГРУЗКА И ЗАПУСК ПРОГРАММЫ

Среда разработки IDEicc08 для МК семейства HC08 не содержит встроенного загрузчика/отладчика, поэтому воспользуемся одной из многочисленных программ - WinIDE  , предназначенной для загрузки кода в формате “S Record”, а также для его отладки и выполнения. В открывшемся окне, на панели инструментов нажимаем на кнопку  “Programmer - программатор” (рис.2.32).

Рис.2.32. Панель управления Win IDE

Если появилось указанное ниже окно (рис. 2.33), значит вы не подали питание на стенд или “воткнули” интерфейсный кабель не в тот разъем. Проверьте подключение и нажмите на кнопку “Contact”.

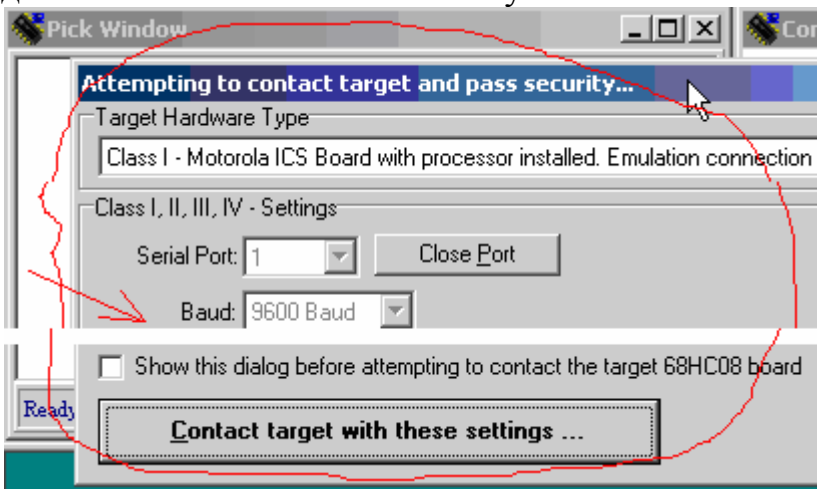


Рис. 2.33. Диалоговое окно с параметрами подключения

Теперь выберите алгоритм программирования, в соответствии с указаниями рисунка 2.34.

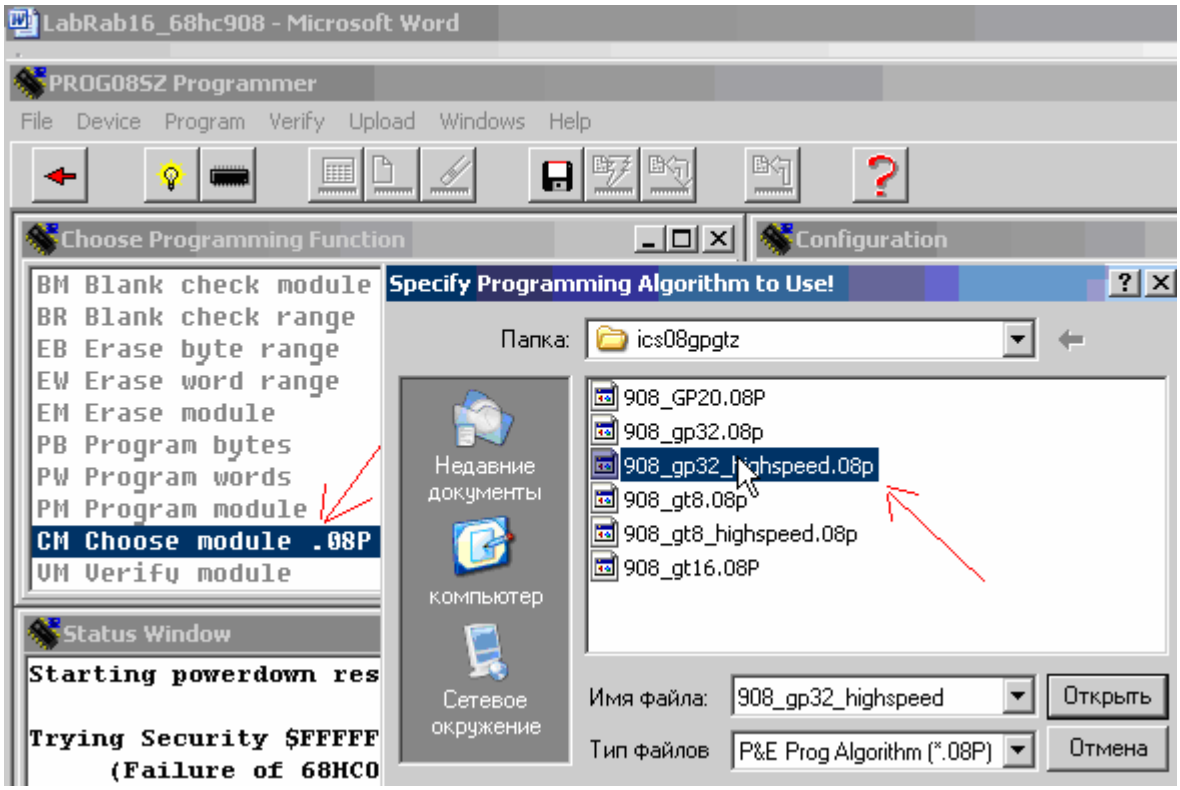


Рис. 2.34. Панель программатора

Дождитесь появления статусного сообщение “Done / Выполнено” (рис. 2.24-1).

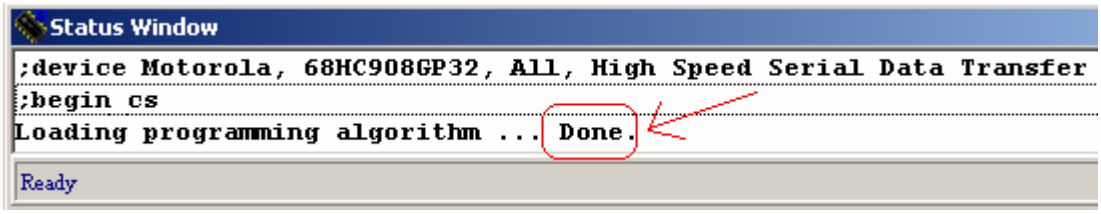


Рис. 2.24-1. Загрузка программы во флэш-память закончена

Укажите 16-ный файл рабочей (целевой) программы для загрузки в целевую плату (стенд). Файл *.s19 - мотороловский аналог интеловского HEX-файла *.hex должен находиться в вашей рабочей папке (рис. 2.35).

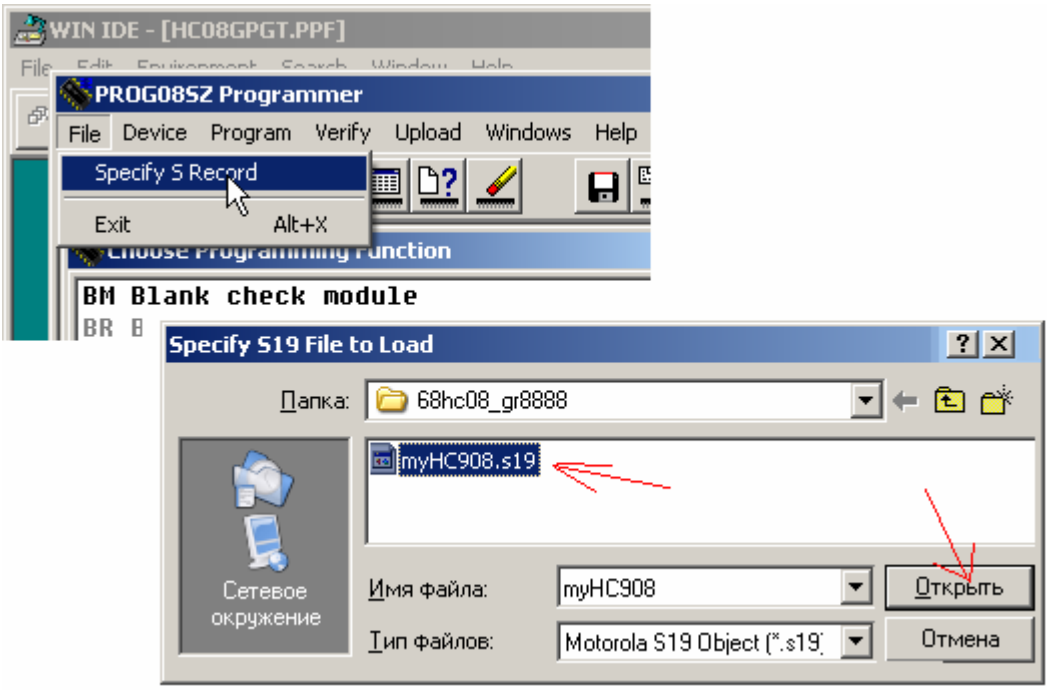


Рис. 2.35. Загрузка 16-ного рабочего файла программы

В окне конфигурации появится выбранный вами рабочий файл.

Теперь двойным кликом стираем программную флэш-память. Должно появиться сообщение “Module has been erased” (рис. 2.36).

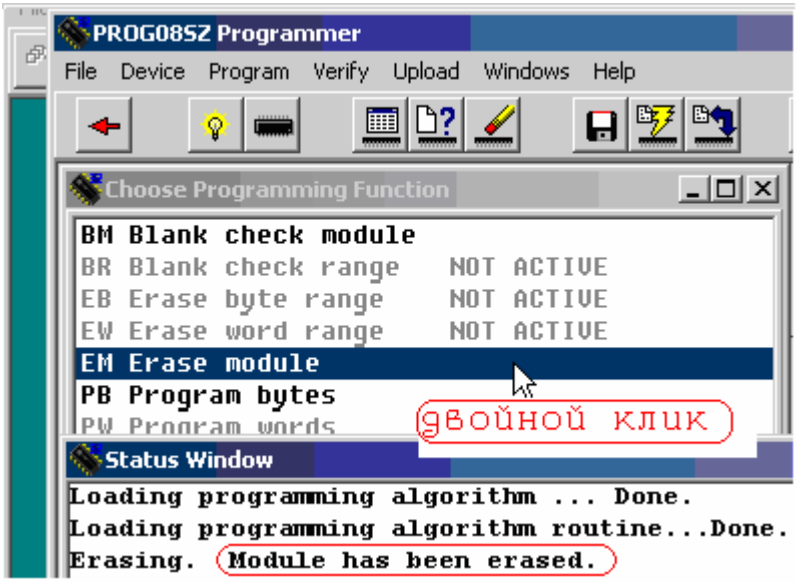


Рис. 2.36. Стирание флэш-памяти микроконтроллера

Также двойным щелчком запускаем процесс программирования микроконтроллера и дожидаемся статусного сообщения “Programmed” (рис. 2.37).

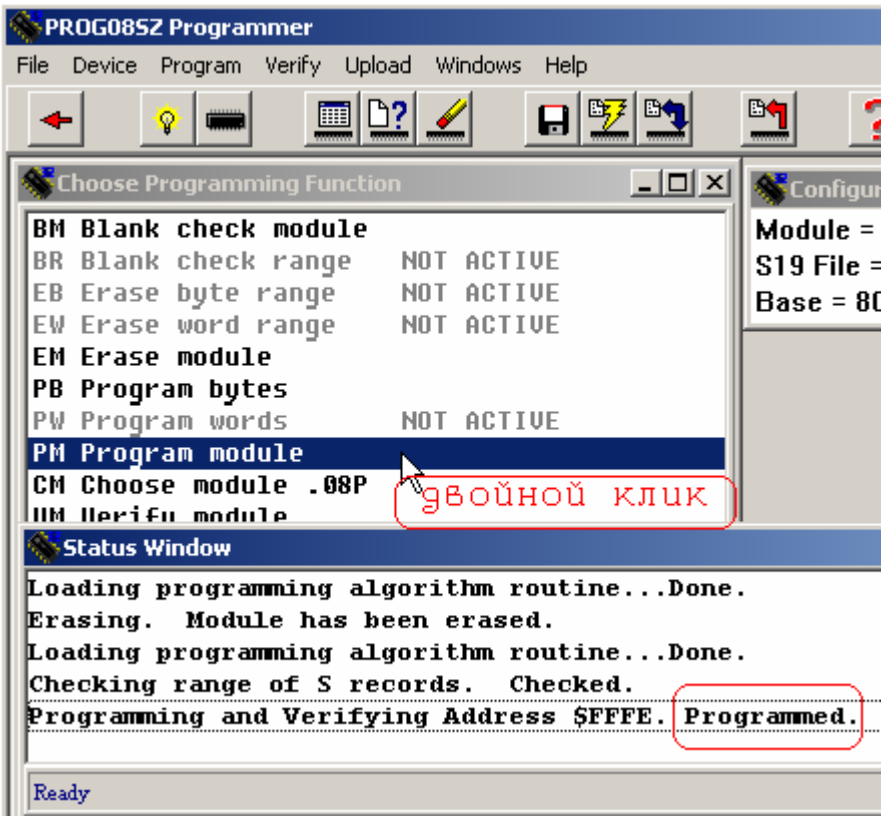
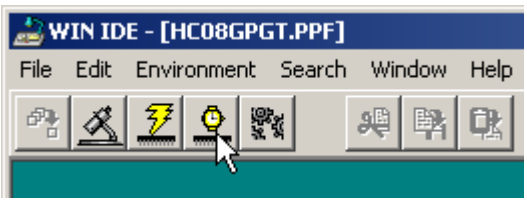


Рис. 2.37. Запись программы во флэш-память МК

Перед запуском программы необходимо выйти из программатора, как показано на рисунке (и программатор и отладчик используют один и тотже COM порт).

Из оболочки запускаем внутрисхемный отладчик  .



Появляется масса окон и панель с инструментами, на которой жмем на кнопку пуск  .

Начинает выполняться целевая программа и зеленый светодиод плавно меняет яркость свечения. Этот первый результат покажите преподавателю.

Остановите выполнение программы кнопкой “Reset”  . Так как в нашей программе не предусмотрена работа с COM портом, то появится следующее сообщение (рис. 2.38), в котором можно либо “сбросить” микроконтроллер в исходное состояние (Reset), либо выйти из отладчика (Halt). Если нет нужды в повторном запуске программы, то выйдите из отладчика.

Рис. 2.38. Диагностическое сообщение

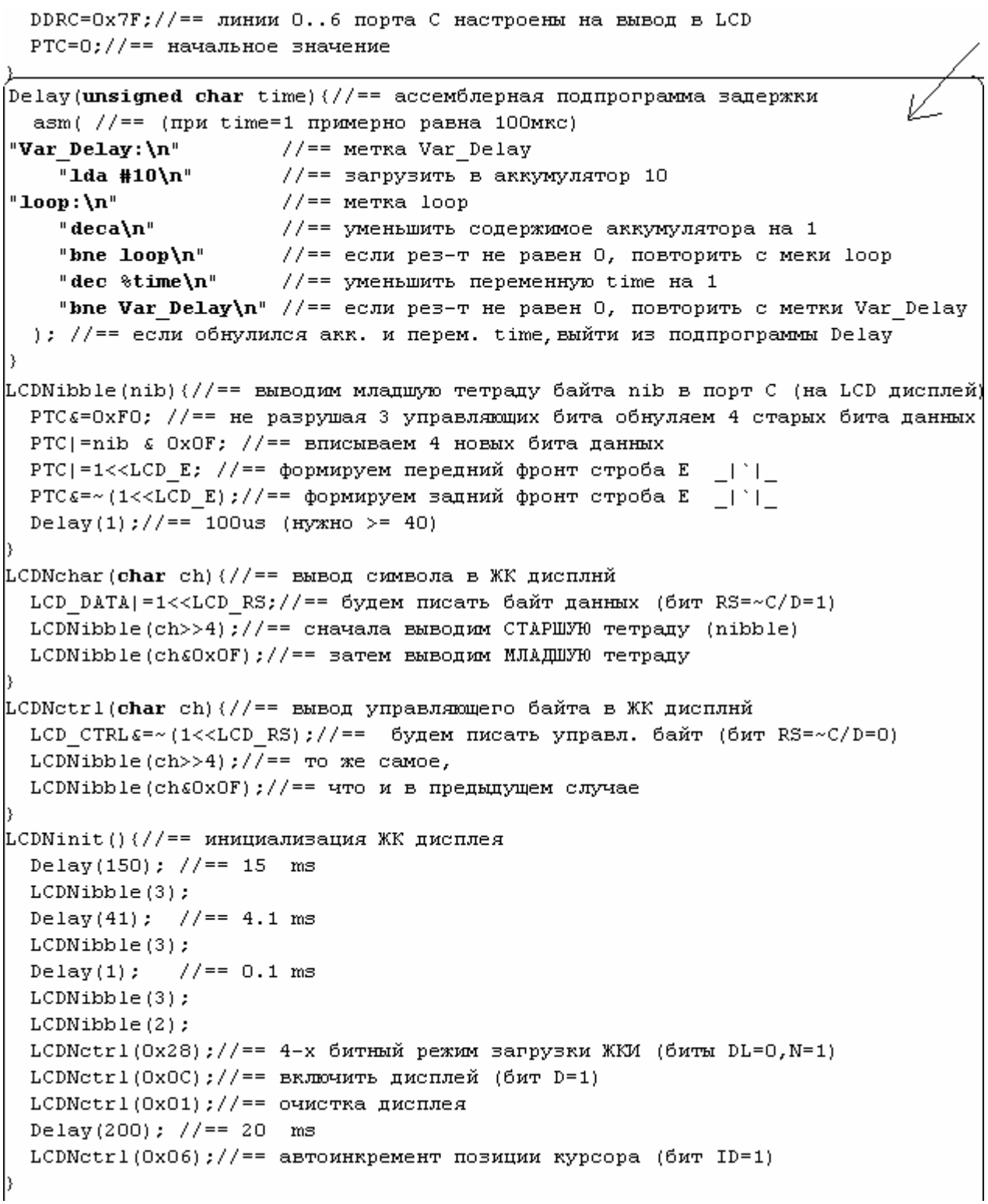
10.2.4 ПРОГРАММИРОВАНИЕ ЖК ДИСПЛЕЯ (LCD)

Теперь необходимо записать процедуры для работы с LCD дисплеем, т.к. все остальные модули программы выводят в него данные. Интерфейс обмена данными между МК и ЖКД четырехбитный, поэтому в первой процедуре LCDNibble(...) в соответствии со схемой на рис.2.1 передача полубайта (ниббла, тетрады) в ЖКД будет производиться через четыре младших бита порта C. Управляющие работой LCD сигналы передаются по старшим линиям порта C. Операнды 0xF0(11110000) и 0x0F(00001111) являются масками для обнуления/ выделения требуемой тетрады с помощью поразрядных логических команд (И - &) и (ИЛИ - |).

Далее понадобятся две процедуры записи в ЖКД кода отображаемого символа и управляющего байта в соответствии с таблицами 2.10 и 2.11. Процедура инициализации дисплея переводит его в 4-х битный режим, очищает дисплей и производит некоторые другие стандартные действия, как это видно из комментариев.

Подпрограммы LCDNstrn(), LCDNclr() и LCDNxy() осуществляют запись строки, очистку, позиционирование курсора и являются полезными дополнениями. Также нам понадобится вспомогательная подпрограмма

Delay() для задержки на небольшие интервалы времени. Для разнообразия напишем ее на встроенном ассемблере. Вернитесь в редактор проекта и введите в программу два фрагмента, которые показаны ниже на рисунке 2.39.



```
  DDRC=0x7F;//== линии 0..6 порта C настроены на вывод в LCD
  PTC=0;//== начальное значение
}
Delay(unsigned char time){//== ассемблерная подпрограмма задержки
  asm( //== (при time=1 примерно равна 100мкс)
"Var_Delay:\n"          //== метка Var_Delay
    "lda #10\n"         //== загрузить в аккумулятор 10
"loop:\n"               //== метка loop
    "deca\n"            //== уменьшить содержимое аккумулятора на 1
    "bne loop\n"        //== если рез-т не равен 0, повторить с меки loop
    "dec %time\n"       //== уменьшить переменную time на 1
    "bne Var_Delay\n" //== если рез-т не равен 0, повторить с метки Var_Delay
  ); //== если обнулился акк. и перем. time,выйти из подпрограммы Delay
}
LCDNibble(nib){//== выводим младшую тетраду байта nib в порт C (на LCD дисплей)
  PTC&=0xF0; //== не разрушая 3 управляющих бита обнуляем 4 старых бита данных
  PTC|=nib & 0x0F; //== вписываем 4 новых бита данных
  PTC|=1<<LCD_E; //== формируем передний фронт строба E  _|`|_
  PTC&=~(1<<LCD_E);//== формируем задний фронт строба E  _|`|_
  Delay(1);//== 100us (нужно >= 40)
}
LCDNchar(char ch){//== вывод символа в ЖК дисплнй
  LCD_DATA|=1<<LCD_RS;//== будем писать байт данных (бит RS=~C/D=1)
  LCDNibble(ch>>4);//== сначала выводим СТАРШУЮ тетраду (nibble)
  LCDNibble(ch&0x0F);//== затем выводим МЛАДШУЮ тетраду
}
LCDNctrl(char ch){//== вывод управляющего байта в ЖК дисплнй
  LCD_CTRL&=~(1<<LCD_RS);//==  будем писать управл. байт (бит RS=~C/D=0)
  LCDNibble(ch>>4);//== то же самое,
  LCDNibble(ch&0x0F);//== что и в предыдущем случае
}
LCDNinit(){//== инициализация ЖК дисплея
  Delay(150); //== 15  ms
  LCDNibble(3);
  Delay(41);  //== 4.1 ms
  LCDNibble(3);
  Delay(1);   //== 0.1 ms
  LCDNibble(3);
  LCDNibble(2);
  LCDNctrl(0x28);//== 4-х битный режим загрузки ЖКИ (биты DL=0,N=1)
  LCDNctrl(0x0C);//== включить дисплей (бит D=1)
  LCDNctrl(0x01);//== очистка дисплея
  Delay(200); //== 20  ms
  LCDNctrl(0x06);//== автоинкремент позиции курсора (бит ID=1)
}
```

```
LCDNstrn(char *s){ //== вывод строки в ЖК дисплей
  while(*s!=0)LCDNchar(*s++);//== выводить символы, пока *s не равно 0
}
LCDNclr(){LCDNctrl(1);Delay(200);}//== очистка дисплея
LCDNxy(char x,char y){//== позиционирование курсора
  //== x = 0..15 позиция символа в строке, y=0/1 - верхн./нижн. строка
  LCDNctrl(0x80+(x+((y)?0x40:0)));
}
void main(void){
  InitDevices();//== установка режимов работы портов и периферии
  LCDNinit();//== инициализируем ЖК дисплей
  LCDNchar('K');//== только для демонстрации работоспособности
```

Рис. 2.39. Программирование ЖКД

Снова откомпилируйте программу кнопкой  , далее откройте программатор  и затем отладчик  (см. п. 10.2.3). Запустите программу на выполнение  и покажите результат преподавателю (какой д.б. результат видно из функции main()).

10.2.5 ИЗМЕРЕНИЕ УГЛА ПОВОРОТА (ДАТЧИК НАПРЯЖЕНИЯ)

Добавим в программу процедуры и команды для измерения угла поворота датчика напряжения (рис. 2.40). Датчик подключен к 7-му входу порта B (ADC7 рис. 2.1). В отличие от других лабораторных работ, в этой предусмотрено автоматическое измерение напряжения, поэтому пуск АЦП, считывание кода, преобразование его в численное значение напряжения и отображение результатов производится без участия оператора.

```
#pragma interrupt_handler TIM1_cmp
void TIM1_cmp(void){//== обработчик прерывания по совпадению кодов таймера1
  T1SC1&=~(1<<CH1F);//== программно обнуляем флаг совпадения CH1F таймера1
}
#pragma interrupt_handler ADC_int
char ad;//== глобальная переменная - для 8-ми битного кода АЦП
void ADC_int(void){//== обработчик прерывания по завершению АЦ преобразования
  ad=ADR;//== читаем код напряжения из регистра ADR (регистр данных АЦП)
  ADSCR|=0x07;//== снова запускаем АЦП (7-ой канал),
}             //== бит ADCO=0 - однократный пуск
//== инициализируем таблицу векторов прерываний по соотв. адресам
#pragma abs_address:0xFFDE
void (*_vec_adc)(void) = ADC_int;
#pragma end_abs_address
#pragma abs_address:0xFFF2
void (*_vec_tim1_ovf)(void) = TIM1_ovf;
```

Рис. 2.40. Программирование АЦП

При завершении преобразования напряжения в код АЦП генерирует прерывание и микроконтроллер переходит по адресу FFDE, где размещается обработчик прерывания ADC_int(). Следует учесть, что так как запуск АЦП мы сделали однократным (бит ADCO регистра ADSCR равен 0), то для повторных измерениий требуется новая запись номера канала АЦП, что приводит приводит к очередному запуску АЦП, очередному обращению к обработчику прерывания и т.д..

Далее, в основной программе main() необходимо произвести следующие изменения (рис. 2.41): (ВНИМАНИЕ: не забудьте закомментировать или убрать указанные две строки: первая уже не нужна, а вот вторая просто заблокирует работу АЦП!)

```
void main(void){
  char sa[8];//== буфер для вывода данных на ЖКИ
  InitDevices();//== установка режимов работы портов и периферии
  LCDNinit();//== инициализируем ЖК дисплей
//  LCDNchar('K');//== только для демонстрации работоспособности
//   ADSCR=0x1F; //== временно запрещаем работу АЦП и вызов обработчика
  //== AIEN=0(запр.прерыв.), ADCH4..ADCH0=11111(останов АЦП)
  CLI();//== разрешить прерывания (asm("cli\n")),(SEI()-запретить(asm("sei\n"))
  //== ВНИМАНИЕ: в других языках прогр. cli - запрещает, sei(sti) - разрешает)
  LCDNinit();//== инициализируем ЖК дисплей (курсор слева и в верхней строке)
  LCDNstrn("U(Vt) = ");//== выводим надпись "напряжение(вольт) = ...."
  while(0x8888){//== бесконечный цикл
    LCDNxy(8,0);//== переходим к верхней строке и пишем в ней напряжение и код
    sprintf(sa,"%#1.2f %03d",((float)ad*(float)5)/(float)256,ad);
      //== #-все позиции иначе при U=0 будет отобр. только 0,
      //== а в позициях за ним - старые знач.!
    LCDNstrn(sa);//== отображаем напряжение и соотв. код
  }
}
```

Рис. 2.41. Отображение напряжения и соответствующего кода

Функция sprintf() принимает 2 аргумента – число с плавающей точкой (float) – расчетное значение напряжения и целое число – код ad. Первый аргумент отображается с одной цифрой до запятой (точки) и двумя цифрами после запятой. Второму целому аргументу также отводятся три цифры, но при необходимости слева дописываются нули. Преобразованные в соответствии со спецификацией аргументы записываются в кодировке ASCII в строку sa. Следующий оператор выводит на дисплей эту строку.

Снова откомпилируйте программу кнопкой  , далее откройте программатор  и затем отладчик  (см. п. 10.2.3). Запустите программу на выполнение  и вращая ручку датчика убедитесь в смене кодов. Покажите результат преподавателю (примерно такой, как на рис. 2.42).

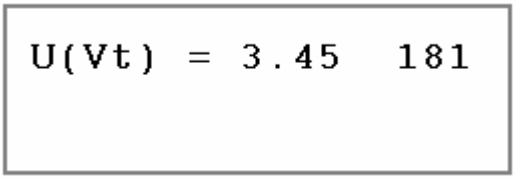


Рис. 2.42. Панель ЖК дисплея

10.2.6 ИЗМЕРЕНИЕ ТЕМПЕРАТУРЫ

Перейдем к программированию термодатчика. Добавим в программу процедуры для работы со стандартным SPI интерфейсом и конкретным термодатчиком. Действия команд подпрограмм достаточно откомментированы (рис. 2.43).

```
Send_SPI_Byte(char c){//== передача байта от ведущего к ведомому
  SPDR=c;//== при записи в регистр SPDR, байт передается по линии MOSI
  //== в ведомое устройство (термодатчик)автоматически
  while((SPSCR & (1<<SPCRF))==0)continue;//== проверяем бит готовности
  c=SPDR;//== нужно обязательно прочитать !!!! (и отбросить)
  //==т.к.регистры данных ведущегои ведомого SPI устр-в замкнуты в кольцо
}
LCDN_ReadSPIValue(){//== чтение и отображение принятого байта
char c,s[10];
  c=SPDR; //== читаем принятый байт из внутр. буфера SPI
  sprintf(s,"%d",(signed char)c);//== преобразуем целый тип со знаком
  LCDNstrn(s);//== в строку ASCII и отображаем температуру на ЖКД
}
SPIctrl(){//== передаем адрес регистра управления и управляющий байт DS1722
  PTD|=1<<SS;//== активируем термодатчик (ТД) по входу CE (он же ChipSelect)
  Send_SPI_Byte(0x80);//== 80-адрес рег-ра управл.ТД для записи(00-для чтения)
  Send_SPI_Byte(0xF1);//== 1SHOT=SD=1(однокр.пуск),R2=R1=R0=0(точн t-ры 8 бит)
  //== |1|1|1|1SHOT|R2|R1|R0|SD| - формат управл. байта ТД
  PTD&=~(1<<SS);//== деактивируем термодатчик по входу CE
  Delay(150);//== необходимая задержка
}
SPIdata(){//== передаем адреса регистров температуры
  PTD|=(1<<SS);//== активируем термодатчик (ТД) по входу CE
  Send_SPI_Byte(0x02);//== передаем адрес регистра старшего байта t-ры
  Send_SPI_Byte(0x02);//== при 8-ми битной t-ре СНОВА ст.байт
  //== при 9..12-ти битной t-ре СНАЧАЛА 0x01(мл.байт), ЗАТЕМ 0x02(ст.)
  PTD&=~(1<<SS);//== снова деактивируем термодатчик по входу CE
}
void main(void){
  char sa[8];//== буфер для вывода данных на ЖКИ
```

Далее необходимо добавить команды в основную функцию main() (рис. 2.44).

```
. . . . . .
LCDNstrn("U(Vt) = ");//== выводим надпись "напряжение(вольт) = ...."
LCDNxy(0,1);//== перейти на нижнюю строку LCD
LCDNstrn("T(oC)=");//== вывести "температура = "
SPIctrl();//== передаем адрес регистра управления и управляющий байт ТД
while(0x8888){//== бесконечный цикл
  LCDNxy(8,0);//== переходим к 8-ой позиции и пишем в ней напряжение и код
  sprintf(sa,"%#1.2f %03d",((float)ad*(float)5)/(float)256,ad);
   //== #-все позиции иначе, при U=0 будет отобр. только 0,
  //== а в позициях за ним - старые знач.!
  LCDNstrn(sa);//== отображаем напряжение и соотв. код
  SPIdata();//== передаем адреса регистров данных термодатчика
  LCDNxy(6,1); //== переходим к позиции LCD, где будет отображаться t-ра
  LCDN_ReadSPIValue();//== читаем и выводим t-ру на LCD
}
```

Рис. 2.44. Очередная модификация функции main()

Снова выполните цикл: трансляция, стирание, запись и запуск программы. В случае успеха экран ЖК дисплея будет выглядеть примерно так:

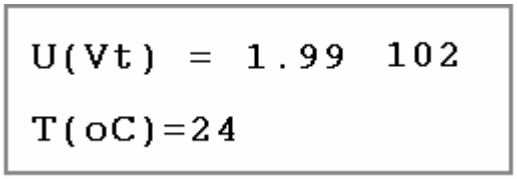


Рис. 2.45. Панель ЖК дисплея

10.2.7 ПРОГРАММИРОВАНИЕ КЛАВИАТУРЫ

Сначала добавим в программу обработчик прерывания и его абсолютный адрес (рис. 2.46).

```
void ADC_int(void){//== обработчик прерывания по завершению АЦ преобр
  ad=ADR;//== читаем код напряжения из регистра данных АЦП)
  ADSCR|=0x07;//== снова запускаем АЦП (7-ой канал), бит ADCO=0 - однокр пуск
}
#pragma interrupt_handler KBD_int
char key[]={'1','2','3','4','5','6','7','8','9','*','0','#',''};
char keynum, keypressed=0;//== номер клавиши и признак "клавиша нажата"
char rown[]={0,0,0,0,0,0,0,3,0,0,0, 2, 0, 1, 0, 0};//== таблица номеров
//==                          7        11    13 14    //== рядов клавиш
void KBD_int(void){//== обработчик нажатия на клавишу (БЕЗ сканирования!)
char row,col,temp,r,i;//==col - номер колонки слева, row - номер ряда сверху
  INTKBSCR|=(1<<ACKK);//== подтверждаем обработку прерывания
  Delay(800);//== задержка на время возможного дребезга при нажатии
  if((r=(PTA&0xF0)>>4)==0x0F)goto exit;//== r=0111(7),1011(11),(13),(14)
  //== "goto exit" игнорирует возможный дребезг при отпускании клавиши
  //==  и появление кода r = 1111 вместо 0111..1110
```

```
  row=r[rown];//== выбираем из таблицы номер ряда = (0,1,2,3)
  for(i=0;i<3;i++){//== ищем в какой колонке находится нажатая клавиша
    PTA|=0x08>>i;//== послед-но "перекрываем" единицей колонки(col=2..0)
    if((PTA&0xF0)==0xF0){col=2-i;break;}//== если 1 перекрыла 0
  }                                     //==  - закончить проверку
  keynum=col+row*3;//== вычисляем номер нажатой клавиши (0..11)в табл. key
exit:
  PTA=0b11110001;//== возвращаем начальное состояние выводов порта A
  INTKBSCR|=(1<<ACKK);//== сброс возможн. запросов при манипулир. с битами PTA
  keypressed=1;//== устанавливаем признак нажатия клавиши (этот признак
}              //== можно использовать в основной программе - main())
//== инициализируем таблицу векторов прерываний по соотв. адресам
#pragma abs_address:0xFFE0
void (*_vec_keyboard)(void) = KBD_int;
#pragma end_abs_address
#pragma abs_address:0xFFDE
void (*_vec_adc)(void) = ADC_int;
#pragma end_abs_address
. . . . . .
```

Рис. 2.46. Обработчик прерывания нажатой клавиши

Затем необходимо поместить в пролграмму процедуру инициализации и ее вызов. В бесконечном цикле запишите три строки для вывода кода нажатой клавиши в LCD дисплей.

```
KBDinit(){//== установка интерфейса клавиатуры в исходное состояние
  DDRA=0x0E;//== 4 ст. бита - на ввод (линии возврата),
            //== 3 бита PTA1..3 - на вывод
  PTAPUE=0xF0;//== включаем 4 подтягивающих резистора (PullUp)
              //== на входах PTA4..7,если не нажата ни одна из клавищ,
              //== на ВСЕХ линиях возврата - будут единицы
  PTA=0xF1;//== на все выходы PA1..PA3 подаем нули (без сканирования)
  INTKBIER=0xF0;//== разрешить прерывание от нуля на входах PTA4..7
}
void main(void){
  char sa[8];//== буфер для вывода данных на ЖКИ
  InitDevices();//== установка режимов работы портов и периферии
  LCDNinit();//== инициализируем ЖК дисплей
  KBDinit();//== инициализируем порт клавиатуры
  keynum=12;//==еще не нажата ни одна из клавиш(поэтому выведем-пусто'')
  CLI();//== разрешить прерывания (asm("cli\n"))
  . . . . . .
  while(0x8888){//== бесконечный цикл
    LCDNxy(9,1);//== переходим в девятую позицию нижней строки
    sprintf(sa,"%c",key[keynum]);//==записываем в строку sa ASCII код клавиши
    LCDNstrn(sa);//== выводим на ЖКИ ASCII код нажатой клавиши
  . . . . . .
```

Рис. 2.46. Инициализация порта клавиатуры

После запуска программы нажмите несколько раз на различные клавиши и убедитесь в правильности отображения кода (позовите преподавателя).

10.2.8 ОТОБРАЖЕНИЕ НА ДИСПЛЕЕ МИНУТ И СЕКУНД

Последний штрих (добавим отсчет минут и секунд с момента запуска программы, см. рис. 2.47).

```
#pragma interrupt_handler TIM1_ovf
unsigned int d;//== вспомогательная глобальная переменная
char sec=0,min=0,m=0;
void TIM1_ovf(void){//== обработчик прерыв. по переполнению таймера1
  T1SC&=~(1<<TOF); //== программно обнуляем флаг переполнения TOF
  d=T1CH1;//== читаем текущую длительность импульса Tp ШИМ
  d++;//== увеличиваем ее
  if (d==201)d=1;//= восстанавл. начальн. значение длит-сти имп-са Tp
  T1CH1=d;//== увеличим длительность имп-са ШИМ на выходе T1CH1
  if(++m==(38400/Tc)){m=0;//== отмеряем секунды(прошла секунда)
    if(++sec==60){sec=0;if(++min==60)min=0;}//== и минуты
  }
}
. . . . . .

  while(0x8888){//== бесконечный цикл
    LCDNxy(11,1);//== переходим на нижнюю строку ЖКИ в 11 позицию
    sprintf(sa,"%02d:%02d",min,sec);//== '0' - слева будут нули
    LCDNstrn(sa);//== выводим на LCD минуты:секунды
```

Рис. 2.47. Программирование отсчета и индикации секунд и минут

В окончательном варианте информация на дисплее должна выглядеть примерно так, как на рис. 2.48:

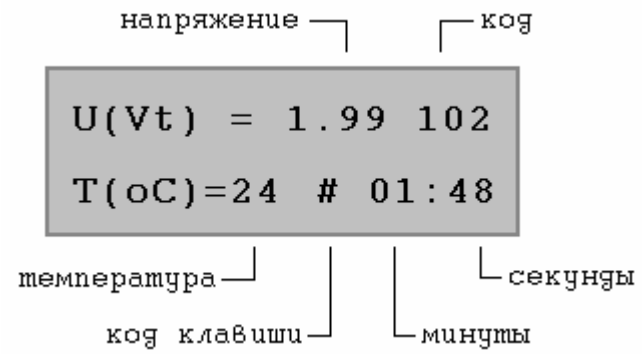


Рис. 2.48. Информация высвечиваемая на дисплее

Позовите преподавателя и, если осталось время, получите дополнительное задание ;-(

10.3 ОКОНЧАТЕЛЬНЫЙ ТЕКСТ ПРОГРАММЫ

```
#include <iogp32.h>
#include <hc08def.h>
#include <float.h>   //== для sprintf
#include <stdio.h>   //== для sprintf
#define LCD_CTRL PTC //== PORTC
#define LCD_DATA PTC
#define LCD_RS 6 //== RS(~C/D) 6-й бит порта C
#define LCD_E  4 //== строб E (4-й бит порта C)
#define LED    5 //== 5-й бит порта D (светодиод)
#define SS     0 //== 0-й бит порта D (вывод #SS SlaveSelect используем как ChipSelect)
#define SPCRF  7 //== 7-й бит байта SPSCR (флаг завершения приема SPI)
#define TOF    7 //== 7-й бит байта T1SC (флаг переполнения таймера1)
#define CH1F   7 //== 7-й бит байта T1SC1 (флаг совпадения кодов таймера1)
#define Tc   400 //== модуль счета таймера1 (38400/Tc - д.б. целое число)
#define ACKK   2 //== бит подтверждения прерывания от клавиатуры
#define IMASKK 1 //== если бит IMASKK=0, прерывания от клавиатуры разрешены
//== определяем обработчики прерываний
#pragma interrupt_handler TIM1_ovf
unsigned int d;//== вспомогательная глобальная переменная
char sec=0,min=0,m=0;
void TIM1_ovf(void){//== обработчик прерывания по переполнению таймера1
  T1SC&=~(1<<TOF); //== программно обнуляем флаг переполнения TOF таймера1
  d=T1CH1;//== читаем текущую длительность импульса Tp ШИМ
  d++;//== увеличиваем ее
  if (d==201)d=1;//= восстанавливаем начальное значение длительности импульса Tp
  T1CH1=d;//== увеличим длительность имп-са ШИМ на выходе T1CH1 в следующем периоде
  if(++m==(38400/Tc)){m=0;//== заодно отмеряем секунды (прошла секунда)
    if(++sec==60){sec=0;if(++min==60)min=0;}//== и минуты
  }
}
#pragma interrupt_handler TIM1_cmp
void TIM1_cmp(void){//== обработчик прерывания по совпадению кодов таймера1
  T1SC1&=~(1<<CH1F);//== программно обнуляем флаг совпадения CH1F таймера1
}
#pragma interrupt_handler ADC_int
char ad;//== глобальная переменная - для 8-ми битного кода АЦП
void ADC_int(void){//== обработчик прерывания по завершению АЦ
```

```
преобразования
  ad=ADR;//== читаем код напряжения из регистра ADR (регистр данных
АЦП)
  ADSCR|=0x07;//== снова запускаем АЦП (7-ой канал), бит ADCO=0 -
однократный пуск
}
#pragma interrupt_handler KBD_int
char key[]={'1','2','3','4','5','6','7','8','9','*','0','#',''};//== ASCII коды клавиш
char keynum, keypressed=0;//== номер клавиши и признак "клавиша была
нажата
char rown[]={0,0,0,0,0,0,0,3,0,0,0, 2, 0, 1, 0, 0};//== таблица номеров рядов
//==                      7        11    13 14    //== клавиш
void KBD_int(void){//== обработчик нажатия на клавишу (БЕЗ
сканирования!)
char row,col,temp,r,i;//== col - номер колонки слева, row - номер ряда сверху
  INTKBSCR|=(1<<ACKK);//== подтверждаем обработку прерывания
  Delay(800);//== задержка на время возможного дребезга при нажатии
  if((r=(PTA&0xF0)>>4)==0x0F)goto exit;//==
r=0111(7),1011(11),1101(13),1110(14)
  //== "goto exit" игнорирует возможный дребезг при отпускании клавиши и
появление
  //== кода r = 1111 вместо 0111..1110
  row=r[rown];//== выбираем из таблицы номер ряда = (0,1,2,3)
  for(i=0;i<3;i++){//== ищем в какой колонке находится нажатая клавиша
    PTA|=0x08>>i;//== последовательно "перекрываем" единицей колонки
PTA3..1(col=2..0)
        if((PTA&0xF0)==0xF0){col=2-i;break;}//== если 1 перекрыла 0 -
закончить проверку
  }
  keynum=col+row*3;//== вычисляем номер нажатой клавиши (0..11)в таблице
key
exit:
  PTA=0b11110001;//== возвращаем начальное состояние выводов порта A
  INTKBSCR|=(1<<ACKK);//== сбрасываем возможные запросы при
манипулир. с битами PTA
  keypressed=1;//== устанавливаем признак (флаг) нажатия клавиши (этот
признак
}             //== можно использовать в основной программе - main())
//== инициализируем таблицу векторов прерываний по соотв. адресам
(таблица 1.3)
#pragma abs_address:0xFFE0
void (*_vec_keyboard)(void) = KBD_int;
#pragma end_abs_address
#pragma abs_address:0xFFDE
void (*_vec_adc)(void) = ADC_int;
```

```
#pragma end_abs_address
#pragma abs_address:0xFFF2
void (*_vec_tim1_ovf)(void) = TIM1_ovf;
#pragma end_abs_address
#pragma abs_address:0xFFF4
void (*_vec_tim1_ch1)(void) = TIM1_cmp;
#pragma end_abs_address
InitDevices(){//== установка режимов работы портов и периферии
  CONFIG1|=1;//== запретить сторожевой таймер COP (бит COPD=1)
  //== биты в регистры CONFIG записываются ОДНОКРАТНО,
  //== и в дальнейшем не могут переопределяться (до след. запуска)
  //========= настройка АЦП
  ADCLK=0x60;//== задаем тактовую частоту АЦП -
ADIV1,0=1(Fq/8),ADICLK=0
  ADSCR=0x47;//== AIEN=1(разр.прерыв.), ADCH4..ADCH0=00111(канал
AD7(PTB7)и пуск)
  //========= настройка первого канала таймера1
  T1SC1=0x40;//== CH1IE=1 - разр. прерывания при совпадении кодов
канала1 таймера1
  T1SC1|=0x10|0x0C;//== MS1B=0,MS1A=1; ELS1B=ELS1A=1 (выход
T1CH1=1 при совп-нии)
  T1SC1|=0x02;//== бит TOV1=1 при переполнении смена значения на выходе
T1CH1(снова 0)
  T1CH1=1;//== начальная длительность имп-са ШИМ (Tp)
  //========= настройка таймера1
  T1SC=0x46;//== TOIE=1(разр. прер.), 4.915.200Гц/2/64 = 38400Гц (PS2..PS0
= 110(bin)->(64))
  //== на таймер1 поступают импульсы с частотой Fosc/2=4.915.200Гц/2 Гц
  T1MOD=Tc;//== загружаем модуль счета для режима ШИМ (Tc=400)
  //========= настройка линии порта D для управления светодиодом
  DDRD=1<<LED;//== 5-й вывод настраиваем, как выход для светодиода
  //========= настройка последовательного периферийного интерфейса SPI
(порт D)
  DDRD|=1<<SS;//== вывод #SS настраиваем в качестве выхода управления
(ChipEnable)
  PTD&=~(1<<SS);//== #SS=CE=0 (термодатчик - не выбран по входу CE(CS)
см. рис.)
  SPCR=0x2A;//== SPMSTR=1(MCU - ведущий (мастер)), SPE=1(разрешение
SPI)
  //========= настройка порта C для работы с ЖКИ
  DDRC=0x7F;//== линии 0..6 порта C настроены на вывод в LCD
  PTC=0;//== начальное значение
}
Delay(unsigned char time){//== ассемблерная подпрограмма задержки
  asm( //== (при time=1 примерно равна 100мкс)
```

```
"Var_Delay:\n"       //== метка Var_Delay
   "lda #10\n"      //== загрузить в аккумулятор 10
"loop:\n"           //== метка loop
   "deca\n"         //== уменьшить содержимое аккумулятора на 1
   "bne loop\n"     //== если рез-т не равен 0, повторить с меки loop
   "dec %time\n"    //== уменьшить переменную time на 1
   "bne Var_Delay\n" //== если рез-т не равен 0, повторить с метки
Var_Delay
  ); //== если обнулился акк. и перем. time выйти из подпрограммы Delay
}
LCDNibble(nib){//== выводим младшую тетраду байта nib в порт C (на LCD
дисплей)
  PTC&=0xF0; //== не разрушая 3 управляющих бита обнуляем 4 старых бита
данных
  PTC|=nib & 0x0F; //== вписываем 4 новых бита данных
  PTC|=1<<LCD_E; //== формируем передний фронт строба E      _|`|_
  PTC&=~(1<<LCD_E);//== формируем задний фронт строба E     _|`|_
  Delay(1);//== 100us (нужно >= 40)
}
LCDNchar(char ch){//== вывод символа в ЖК дисплнй
  LCD_DATA|=1<<LCD_RS;//== будем писать байт данных (бит RS=~C/D=1)
  LCDNibble(ch>>4);//== сначала выводим СТАРШУЮ тетраду (nibble)
  LCDNibble(ch&0x0F);//== затем выводим МЛАДШУЮ тетраду
}
LCDNctrl(char ch){//== вывод управляющего байта в ЖК дисплнй
  LCD_CTRL&=~(1<<LCD_RS);//==  будем писать управл. байт (бит
RS=~C/D=0)
  LCDNibble(ch>>4);//== то же самое,
  LCDNibble(ch&0x0F);//== что и в предыдущем случае
}
LCDNinit(){//== инициализация ЖК дисплея
  Delay(150); //== 15  ms
  LCDNibble(3);
  Delay(41);  //== 4.1 ms
  LCDNibble(3);
  Delay(1);   //== 0.1 ms
  LCDNibble(3);
  LCDNibble(2);
  LCDNctrl(0x28);//== 4-х битный режим загрузки ЖКИ (биты DL=0,N=1)
  LCDNctrl(0x0C);//== включить дисплей (бит D=1)
  LCDNctrl(0x01);//== очистка дисплея
  Delay(200); //== 20  ms
  LCDNctrl(0x06);//== автоинкремент позиции курсора (бит ID=1)
}
LCDNstrn(char *s){ //== вывод строки в ЖК дисплей
```

```
  while(*s!=0)LCDNchar(*s++);//== выводить символы, пока *s не равно 0
}
LCDNclr(){LCDNctrl(1);Delay(200);}//== очистка дисплея
LCDNxy(char x,char y){//== позиционирование курсора
  //== x = 0..15 позиция символа в строке, y=0/1 - верхн./нижн. строка
  LCDNctrl(0x80+(x+((y)?0x40:0)));
}
Send_SPI_Byte(char c){//== передача байта от ведущего к ведомому
  SPDR=c;//== при записи в регистр SPDR, байт передается по линии MOSI
  //== в ведомое устройство (термодатчик)автоматически
  while((SPSCR & (1<<SPCRF))==0)continue;//== проверяем бит готовности
  c=SPDR;//== нужно обязательно прочитать !!!! (и отбросить)
  //== т.к. регистры данных ведущего и ведомого SPI устр-в замкнуты в
кольцо
}
LCDN_ReadSPIValue(){//== чтение и отображение принятого байта
char c,s[10];
  c=SPDR; //== читаем принятый байт из внутр. буфера SPI
  sprintf(s,"%d",(signed char)c);//== преобразуем целый тип со знаком в
строку ASCII
  LCDNstrn(s);//== отображаем температуру на ЖКИ
}
SPIctrl(){//== передаем адрес регистра управления и управляющий байт
DS1722
  PTD|=1<<SS;//== активируем термодатчик (ТД) по входу CE (он же
ChipSelect)
  Send_SPI_Byte(0x80);//== 80-адрес рег-ра управл. ТД для записи (00-для
чтения)
  Send_SPI_Byte(0xF1);//==
1SHOT=SD=1(однокр.пуск),R2=R1=R0=0(точность t-ры 8 бит)
  //== |1|1|1|1SHOT|R2|R1|R0|SD| - формат управл. байта ТД
  PTD&=~(1<<SS);//== деактивируем термодатчик по входу CE
  Delay(150);//== необходимая задержка
}
SPIdata(){//== передаем адреса регистров температуры
  PTD|=(1<<SS);//== активируем термодатчик (ТД) по входу CE
  Send_SPI_Byte(0x02);//== передаем адрес регистра старшего байта t-ры
  Send_SPI_Byte(0x02);//== при 8-ми битной t-ре СНОВА ст.байт
  //== при 9..12-ти битной t-ре СНАЧАЛА 0x01(мл.байт), ЗАТЕМ 0x02(ст.)
  PTD&=~(1<<SS);//== снова деактивируем термодатчик по входу CE
}
KBDinit(){//== установка интерфейса клавиатуры в исходное состояние
  DDRA=0x0E;//== 4 ст. бита - на ввод (линии возврата),3 бита PTA1..3 - на
вывод
  PTAPUE=0xF0;//== включаем 4 подтягивающих резистора (PullUp) на
```

```
входах PTA4..7
  //== если не нажата ни одна из клавищ, на ВСЕХ линиях возврата - будут единицы
  PTA=0xF1;//== на все выходы PA1..PA3 подаем нули (без сканирования)
  INTKBIER=0xF0;//== разрешить прерывание от нуля на входах PTA4..7
}
void main(void){
  char sa[8];//== буфер для вывода данных на ЖКИ
  InitDevices();//== установка режимов работы портов и периферии
  LCDNinit();//== инициализируем ЖК дисплей
  KBDinit();//== инициализируем порт клавиатуры
  keynum=12;//== еще не нажата ни одна из клавиш (поэтому выведем - пусто ")
  CLI();//== разрешить прерывания (asm("cli\n")),(SEI()-запретить(asm("sei\n"))
  //== ВНИМАНИЕ: в других языках прогр. cli - запрещает, sei(sti) - разрешает)
  LCDNinit();//== инициализируем ЖК дисплей (курсор слева и в верхней строке)
  LCDNstrn("U(Vt) = ");//== выводим надпись "напряжение(вольт) = ...."
  LCDNxy(0,1);//== перейти на нижнюю строку LCD
  LCDNstrn("T(oC)=");//== вывести "температура = "
  SPIctrl();//== передаем адрес регистра управления и управляющий байт ТД
  while(0x8888){//== бесконечный цикл
        LCDNxy(11,1);//== переходим на нижнюю строку ЖКИ в 11 позицию
        sprintf(sa,"%02d:%02d",min,sec);//== '0' - слева будет дополнение нулями
        LCDNstrn(sa);//== выводим на LCD минуты:секунды
        LCDNxy(9,1);//== переходим в девятую позицию нижней строки
        sprintf(sa,"%c",key[keynum]);//== записываем в строку sa ASCII код клавиши
        LCDNstrn(sa);//== выводим на ЖКИ ASCII код нажатой клавиши
        LCDNxy(8,0);//== переходим к 8-ой позиции и пишем в ней напряжение и код
        sprintf(sa,"%#1.2f %03d",((float)ad*(float)5)/(float)256,ad);//== #-все позиции
        //== иначе при U=0 будет отобр. только 0, а в позициях за ним - старые знач.!
        LCDNstrn(sa);//== отображаем напряжение и соотв. код
        SPIdata();//== передаем адреса регистров данных термодатчика
        LCDNxy(6,1); //== переходим к позиции LCD, где будет отображаться температура
        LCDN_ReadSPIValue();//== читаем и выводим t-ру на LCD
  }
}
```

11. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Пояснить цель работы и техническое задание.
2. Структурная схема МК и назначение выводов.
3. Организация памяти МК.
4. Назначение и настройка портов (рис. 2.1).
5. Таймер/счетчик1 (регистры).
6. АЦП (регистры).
7. Работа клавиатуры и LCD дисплея.
8. Комментарии к программе.

СПИСОК ЛИТЕРАТУРЫ

1. Китаев Ю.В. Цифровые и микропроцессорные устройства. Учебник. (http://faculty.ifmo.ru/electron), 2003.
2. Китаев Ю.В. Дистанционные лабораторные и практические работы. (http://faculty.ifmo.ru/electron), 2004.
3. А.В.Евстифеев. Микроконтроллеры AVR семейств Tiny и Mega фирмы Atmel. Изд-во "Додека-XXI", М, 2005г.
4. М.С.Голубцов, А.В.Кириченкова. Микроконтроллеры AVR: от простого к сложному. Изд-во "СОЛОН-пресс", М, 2004г.
5. А.В.Евстифеев. Микроконтроллеры AVR семейства Classic фирмы Atmel. Изд-во "Додека-XXI", М, 2004г.
6. В.Н.Баранов. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. Изд-во "Додека-XXI", М, 2004г.
7. В.Б.Бродин, А.В.Калинин. Системы на микроконтроллерах и БИС программируемой логики. Изд-во "ЭКОМ", М., 2002г.
8. И.И.Шагурин. Современные микроконтроллеры и микропроцессоры Motorola. Изд-во "Горячая линия-Телеком", М., 2004г.
9. В.В.Сташин и др. Проектирование цифровых устройств на однокристалльных микроконтроллерах. Изд-во "Энергоатомиздат",М., 1990г.
10. Однокристальные микроЭВМ. Справочник. Изд-во "Бином", 1994г.
11. Майкл Предко. Справочник по PIC-микроконтроллерам. Изд-во "Додека", М, 2002г.
12. Кристиан Тавернье. PIC-микроконтроллеры.Практика применения. Изд-во "ДМК", М, 2003г.
13. В.С.Яценков. Микроконтроллеры MicroCHIP. Изд-во "Горячая линия-Телеком", М., 2005г.
14. А.В.Белов. Конструирование устройств на МК. Изд-во "НИТ", СПб, 2005г.
15. А.В.Фрунзе. Микроконтроллеры? Это же просто! Изд-во "СКИМЕН", М, 2003г.
16. П.П.Редькин. Прецизионные системы сбора данных семейства MSC12xx. Изд-во "Додека-XXI", М, 2006г.

В 2007 году СПбГУ ИТМО стал победителем конкурса инновационных образовательных программ вузов России на 2007–2008 годы. Реализация инновационной образовательной программы «Инновационная система подготовки специалистов нового поколения в области информационных и оптических технологий» позволит выйти на качественно новый уровень подготовки выпускников и удовлетворить возрастающий спрос на специалистов в информационной, оптической и других высокотехнологичных отраслях экономики.

КАФЕДРА ЭЛЕКТРОНИКИ

Заведующий кафедрой: д.т.н., проф. Г.Н. Лукьянов.

Кафедра Электроники (первоначальное название “Радиотехники”) была основана в 1945 году. Первым руководителем кафедры был С.И. Зилитинкевич известный в стране и за рубежом ученый в области физической электроники и радиотехники, активный работник высшей школы, заслуженный деятель науки и техники РСФСР, доктор технических наук, профессор ЛИТМО с 1938 г., инициатор создания в ЛИТМО инженерно-физического и радиотехнического факультетов (1946 г.). С.И. Зилитинкевич заведовал кафедрой с 1945 до 1978 года. Под его научным руководством аспирантами и соискателями выполнено более 50 кандидатских диссертаций, многие его ученики стали докторами наук.

В дальнейшем, с 1978 г. по 1985 г. кафедру возглавил к.т.н., доцент Е.К. Алахов, один из учеников С.И. Зилитинкевича.

С 1985 года по 2006 год руководителем кафедры стал д.т.н., профессор В.В. Тогатов, известный специалист в области силовой электроники и приборов для измерения параметров полупроводниковых структур.

Начиная с 2006 г. кафедрой заведует д.т.н., профессор Г.Н. Лукьянов.

Основные направления кафедры связаны с разработкой приборов для лазерной и медицинской техники, приборов для измерения параметров полупроводниковых структур, а также встраиваемых цифровых и микропроцессорных устройств.

Под руководством В.В. Тогатова было разработано и изготовлено большое число приборов различного назначения:

- Измеритель параметров ультрабыстрых диодов;

- Универсальное устройство для исследования переходных процессов в силовых полупроводниковых структурах;
- Измеритель времени жизни заряда в слаболегированных областях диодных, тиристорных и транзисторных структур;
- Универсальный разрядный модуль для накачки твердотельных лазеров;
- Импульсный источник токов для накачки лазерных линеек;
- Высокочастотный разрядный модуль для систем накачки твердотельных лазеров и импульсных источников света;
- Программируемый источник света для питания галогенных ламп;
- Блок управления затвором с нарушением полного внутреннего отражения;
- и много других.

На кафедре написаны и размещены на сайте ЦДО следующие материалы для дистанционного обучения (автор Ю.В. Китаев):

- Конспект лекций по дисциплине “Электроника и микропроцессорная техника”;
- свыше 600 вопросов к обучающим и аттестующим тестам;
- 18 дистанционных лабораторных и практических работ

На кафедре имеются следующие компьютеризированные учебные лаборатории:

- АРМС – полупроводниковые приборы;
- Устройства на полупроводниковых приборах;
- Цифровая техника;
- Микропроцессорная техника
- Моделирование электронных устройств.