МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕУЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических занятий по дисциплине

«Встраиваемые программируемые системы»

для студентов направления подготовки

11.04.04 Электроника и наноэлектроника

Направленность (профиль)

Интеллектуальные программируемые системы и устройства

Ставрополь
2026

Содержание

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Цель освоения дисциплины: формирование у студентов профессиональных компетенций, направленных на применение различных языков и инструментальных сред для программирования микропроцессорных устройств, и разработки встраиваемых систем управления на их основе.

Задачи освоения дисциплины:

1. Систематизация фундаментальных знаний по теории и практике применения различных языков программирования встраиваемых микропроцессорных систем.

2. Формирование системы знаний, умений и навыков по разработке фрагментов программного кода драйверов и программированию микроконтроллеров для решения типовых задач.

3. Формирование представлений по выбору языков программирования и архитектуры микропроцессорных систем для разработки компонентов системных программных продуктов

2. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-2. Способен разрабатывать специализированное программное обеспечение интеллектуальных программируемых систем и устройств	ИД-1$_{ПК-2}$ Интересуется современными тенденциями развития программных средств и языков программирования	Способен самостоятельно освоить новые языки и инструментальные средства программирования
	ИД-2$_{ПК-2}$ Использует теоретические знания и практические навыки, позволяющие строить алгоритмы, анализировать их работу при разных входных данных и реализовать их при помощи современных языков программирования	Способен программировать встраиваемые микропроцессорные устройства и интеллектуальные программируемые системы с использованием современных языков программирования
	ИД-3$_{ПК-2}$ Использует языки программирования высокого уровня и профессиональные системы программирования, инструментальные средства при решении профессионально-прикладных задач в	Разрабатывает интеллектуальные системы и устройства, а также их программное обеспечение с применением языков программирования высокого уровня и профессиональных систем программирования и инструментальных средств

	профессиональной сфере	
ПК-3. Способен использовать современные интегрированные среды разработки и программирования микропроцессорных систем	ИД-1$_{ПК-3}$ Использует средства автоматизации проектирования и конструирования приборов, схем и устройств различного функционального назначения.	Способен использовать средства автоматизации проектирования и конструирования интеллектуальных систем и устройства
	ИД-3$_{ПК-3}$ Корректно использует нормативные и справочные данные при разработке проектно-конструкторской документации, готовит проектно-конструкторскую документацию в соответствии со стандартами, техническими условиями и другими нормативными документами	Способен применять правила и нормативную документацию при разработке встраиваемых программируемых микропроцессорных устройств управления

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №1
ПРОГРАММИРОВАНИЕ ВСТРАИВАЕМЫХ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ НА ЯЗЫКЕ АССЕМБЛЕРА И В МАШИННЫХ КОДАХ

Цель работы: Исследование структуры восьмиразрядного микропроцессора и приобретение навыков программирования типовых задач управления на языке ассемблера и в машинных кодах

Учебные вопросы:

1. Ввод разовых команд и их индикация
2. Реализация вывода информации на семисегментный индикатор
3. Ввод данных с клавиатуры
4. Вывод данных на светодиодный индикатор в динамическом режиме

1. Ввод разовых команд и их индикация

Восьмиразрядные универсальные микропроцессоры, такие как Intel 8080 или КР580ВМ80А послужили архитектурной основой многих следующих поколений микропроцессоров. Особенностью низкоуровневого программирования является необходимость знания состава и структуры микропроцессора на уровне программно доступных регистров. Микропроцессор КР580ВМ80А может адресовать до 256 портов ввода и столько же портов вывода. Микропроцессорная лаборатория „Микролаб" имеет порты ввода/вывода, которые реализованы на программируемом периферийном интерфейсе (микросхема КР580ВВ55, рисунок 1). Порты этого интерфейса могут быть либо портами вывода, либо портами ввода, в зависимости от того, как интерфейс запрограммирован.

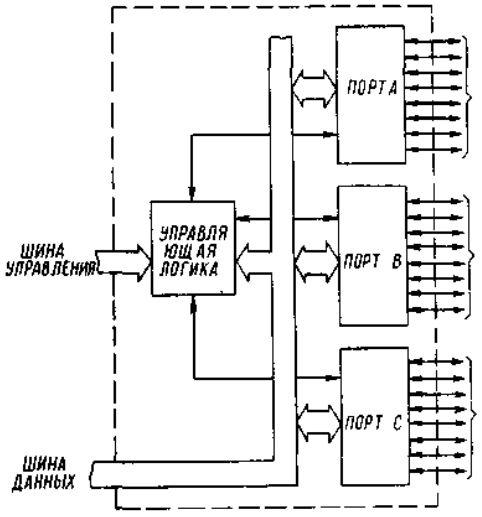


Рисунок 1- Программируемый параллельный интерфейс

Регистры портов имеют следующие адреса в шестнадцатеричном формате: Порт А - F8, порт В - F9, порт С – FA и управляющий регистр – FB. К трем разрядам порта С этого интерфейса подключены тумблеры. Этот порт будет программироваться как порт ввода. А к 8 разрядам порта В подключены светодиодные индикаторы. Этот порт будет программироваться как порт вывода, т. е. микропроцессор (МП) сможет считывать данные с тумблеров и выводить данные на светодиодные индикаторы. Для программирования микросхемы используется первый формат управляющего слова, назначение разрядов которого приведено на рисунке 2.

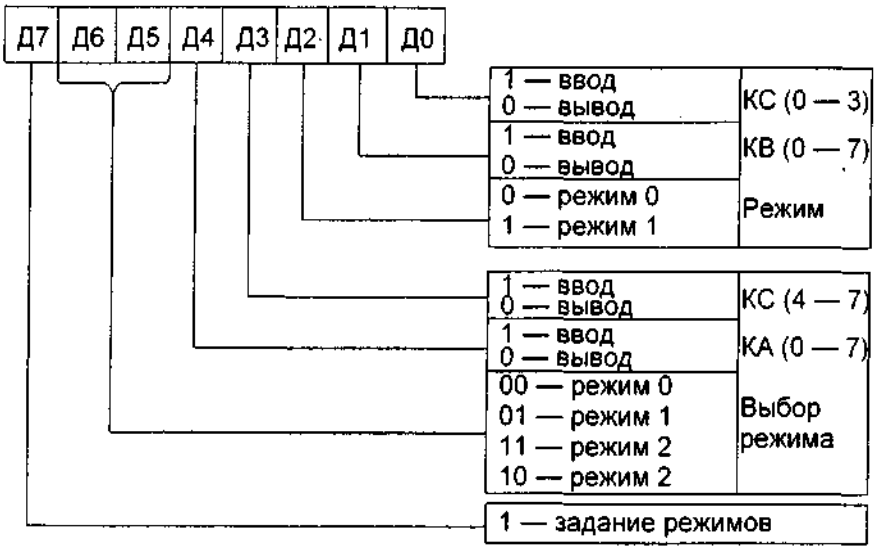


Рисунок 2- Формат 1 управляющего слова

На рисунке 3. показана схема алгоритма, демонстрирующая использование этих портов. В данном примере данные считываются из порта ввода С и записываются в порт вывода В. Листинг программы соответствующий алгоритму, представленному на рисунке 3, приведен в таблице 1.

Чтобы запрограммировать интерфейс, как было предусмотрено (С — порт ввода, В — порт вывода), необходимо подать управляющее слово, имеющее значение 81 по адресу управляющего регистра FB. Первая команда MVI A, 81 означает загрузку аккумулятора кодом 81. Следующая команда OUT FB записывает содержимое аккумулятора в порт по адресу FB, т. е. порт (интерфейс) программируется. Далее идет команда IN FA, обеспечивающая запись данных из порта с адресом FA, присвоенным порту С, в аккумулятор. Команда OUT F9 означает вывод данных из аккумулятора в порт вывода по адресу F9. Поскольку этот адрес в нашем случае присвоен порту В, происходит вывод данных из аккумулятора в порт В (на светодиодные индикаторы). Таким образом, программа пересылает данные из порта ввода в порт вывода. Последняя команда безусловного перехода JMP передает управление на начало цикла. Цикл выполняется непрерывно до нажатия кнопки СБРОС.

Для выполнения программы выполните следующие действия:

1. Введите программу, указанную в таблице 1.
2. Начните выполнение программы с адреса 8000.

3. Поставьте тумблер порта ввода в любое положение. Верхнее положение соответствует 1, нижнее 0.

4. Посмотрите на светодиоды 2-4, считая справа. Они показывают те же данные, которые установлены на тумблерах порта ввода.

5. Измените положение тумблеров порта ввода. Данные порта вывода должны измениться соответственно.

6. Нажмите кнопку СБРОС. Программа остановится, управление перейдет к монитору.

7. Измените положение тумблеров порта ввода. Данные порта вывода не изменятся, так как введенная программа больше не выполняется.

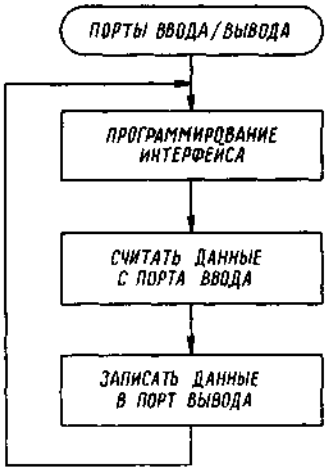


Рисунок 3- Схема алгоритма для пересылки данных из порта ввода в порт вывода

Таблица 1- Листинг программы для переписи данных из порта ввода в порт вывода

Адрес	Содержимое	Метка	Команда	Комментарии
8000	3E		MVI A, 81	Запись в аккумулятор данных для
8001	81			программирования интерфейса
8002	D3		OUT FB	Программирование интерфейса
8003	FB			
8004	DB	START:	IN FA	Чтение данных из порта ввода
8005	FA			
8006	D3		OUT F9	Запись данных в порт вывода
8007	F9			
8008	C3		JMP START	Цикл
8009	04			
800A	80			

Следует отметить, что, при желании, перед записью данных в порт вывода, можно произвести над ними любые действия, предусмотренные системой команд МП. Например, можно считать данные из порта ввода, проинвертировать их и передать в порт вывода. Для этого в программу (см. табл. 1) необходимо добавить команду, обеспечивающую инвертирование данных (CMA).

ПЗУ микролаборатории содержит программу монитора, которая считывает данные с клавиатуры, выполняет выбранную операцию и управляет дисплеем. Микролаборатория все время выполняет программу монитора, за исключением случая, когда она выполняет программу пользователя. Когда нажимается кнопка ПУСК, программа монитора заставляет процессор перейти к адресу, указанному на дисплее „Микролаб". Когда нажимается кнопка СБР, микролаборатория возвращается к программе монитора. Программа монитора позволяет проверять содержимое регистров в шаговом режиме после выполнения каждой команды (т. е. после каждого шага) На третьем и четвертом индикаторах, считая справа, после выполнения каждой команды (шаговый режим) высвечивается содержимое аккумулятора. Кроме того, после каждого шага команды программа монитора записывает содержимое регистров в специальные ячейки ОЗУ. Следовательно, можно проверить содержимое регистров на каждом шаге, просмотрев соответствующие ячейки ОЗУ (таблица 2).

Таблица 2 - Адреса регистров МП

Адрес	Регистр
83EB	Аккумулятор
83EA	Регистр признаков
83E9	B регистр
83E8	C регистр
83E7	D регистр
83E6	E регистр
83E5	H регистр
83E4	L регистр
83E3	Указатель стека (старший байт)
83E2	Указатель стека (младший байт)
83E1	Программный счетчик (старший байт)
83EO	Программный счетчик (младший байт)

2 Реализация вывода информации на семисегментный индикатор

В микролаборатории для индикации используются восемь семисегментных цифровых индикаторов. Каждому индикатору соответствует ячейка памяти, где хранится семисегментный код, управляющий свечением сегментов индикатора. Информация из этих ячеек посылается на индикаторы при помощи специальной схемы или программы, которая обеспечивает динамический режим индикации.

Задача программного обеспечения сводится к подготовке семисегментных кодов и засылке их в восемь ячеек памяти с адресами начиная с 83F8 по 83FF. Ячейка 83F8 соответствует левому индикатору, 83FF — правому

Монитор микролаборатории содержит программу SEGCG (адрес=01C0), которая преобразует шестнадцатеричные коды 0, 1, 2... E, F в семисегментные и заносит их в ячейки памяти 83F8...83FF. Исходные данные для этой программы должны находиться в ячейках 83F4...83F7. Каждый байт из этих ячеек соответствует паре индикаторов: содержимое ячейки 83F4 после преобразования высветится на двух левых индикаторах, содержимое ячейки 83F7 — на

двух правых.

В таблице 3 приведена программа показа информации, использующая подпрограмму SEGCG. Введите программу и проверьте ее работу. На индикаторах должны последовательно высвечиваться символы 1, 2, 3, 4, A, B, C, D.

Таблица 3- Программа вывода информации на симисегментные индикаторы

Адрес	Содержимое	Метки	Команда	Комментарии
8000	3E		MVI A, 12	12→ [A]
8001	12			
8002	32		STA 83F4	A→ [83F4]
8003	F4			
8004	83			
8005	3E		MVI A, 34	34→A
8006	34			
8007	32		STA 83F	A → [83F5]
8008	F5			
8009	83			
800A	3E		MVI A, AB	AB →A
800B	AB			
800C	32		STA 83F6	A →[83F6]
800D	F6			
800E	83			
800F	3E		MVI A, CD	CD→A
8010	CD			
8011	32		STA 83F7	A→ [83F7]
8012	F7			
8013	83			
8014	CD		CALL SEGCG	Вызов SEGCG
8015	C0			
8016	01			
8017	76		HLT	Останов

В предыдущей программе была использована подпрограмма SEGCG. Но эта подпрограмма оперирует только с шестнадцатеричными символами — 0 ... F, т. е. не полностью использует возможности семисегментных индикаторов. Можно формировать семисегментный код и получать на индикаторах любые символы, которые позволяют получить используемые в микролаборатории индикаторы. Как уже говорилось, семисегментные коды должны засылаться в определенные ячейки памяти. Каждый бит в этих ячейках соответствует определенному сегменту (рисунок 4). Если бит равен 1, то соответствующий сегмент будет „гореть", и — наоборот.

Бит	7	6	5	4	3	2	1	0
Сегмент	H	G	F	E	D	C	B	A

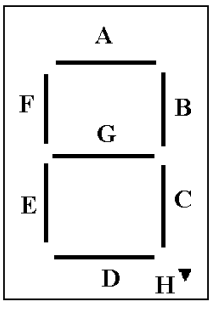


Рисунок 4- Соответствие разрядов и сегментов индикатора

В таблице 4 приведен листинг программы, которая выводит на индикаторы надпись „ScFU". Изменяя семисегментные коды, выведите на индикацию надпись соответствующую номеру учебной группы, например Ibt191.

Таблица 4- Программа вывода символов на индикаторы без подпрограммы SEGCG

Адрес	Содержимое	Метка	Команда	Комментарии
8000	3E		MVI A, 6d	Семисегментный код буквы S→A
8001	6d			
8002	32		STA 83F8	A → [83F8]
8003	F8			
8004	83			
8005	3E		MVI A, 58	Семисегментный код буквы c→A
8006	78			
8007	32		STA 83F9	A → [83F9]
8008	F9			
8009	83			
800A	3E		MVI A, 71	Семисегментный код буквы F→A
800B	30			
800C	32		STA 83FA	A → [83FA]
800D	FA			
800E	83			
800F	3E		MVI A, 3E	Семисегментный код буквы U→A
8010	6d			
8011	32		STA 83FB	A → [83FB]
8012	FB			
8013	83			
8014	76		HLT	Останов

3 Ввод данных с клавиатуры

Микро-ЭВМ имеет на передней панели 25 клавиш и переключатели режима работы, которые обеспечивают прямую и удобную связь с микропроцессорной системой без каких либо дополнительных аппаратных средств. Клавиатура имеет матричную организацию, кроме кнопки СБРОС. Опрос клавиатуры и обработка ее кодов производится программой монитора с помощью программируемого параллельного интерфейса (ППИ). Микросхема ППИ содержит три программируемых 8-разрядных шины ввода/вывода. Для опроса клавиатуры 4 старших бита шины порта С программируются на вывод, а порт А — на ввод. Программа монитора выдает по очереди на С-выходы микросхемы ППИ импульсы низкого уровня, считывая после каждого вывода информацию с порта А (рисунок 5). Каждый разряд порта С поступает на 8 кнопок, составляющих ряд. Каждый вход порта А является сборкой трех кнопок из разных рядов. В случае, если ни одна из кнопок ряда, на которой подан низкий уровень, не нажата, в порт А поступит код FF_{16}, если же нажата хотя бы одна кнопка, то в соответствующем разряде порта А появится лог. 0.

Учитывая номер ряда и разряд порта А, в котором при вводе был лог. 0, монитор формирует код нажатой кнопки и обращается в соответствии с ним к

нужной подпрограмме, выполняющей соответствующую этой кнопке операцию.

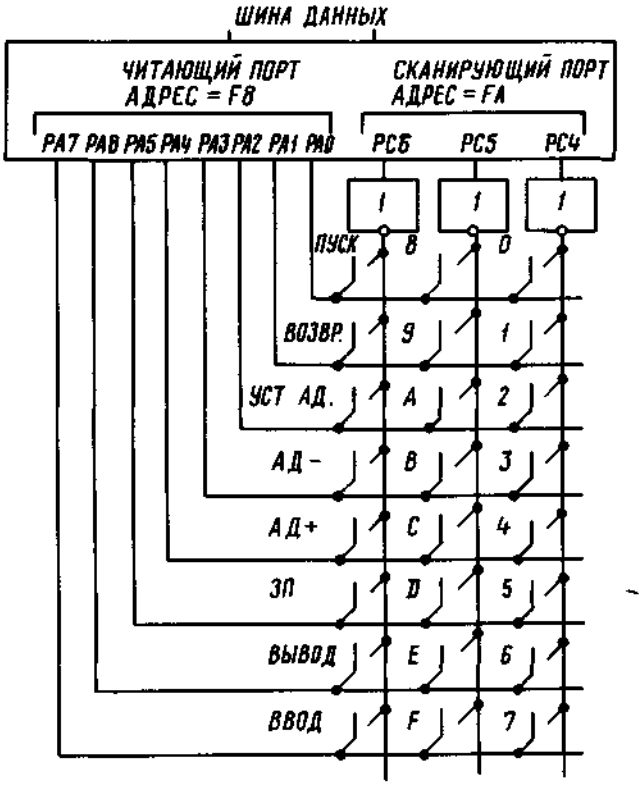


Рисунок 5- Упрощенная схема клавиатуры

Считываемые данные преобразуются в код, соответствующий нажатой кнопке, с помощью программы монитора KEYIN (адрес 0216h). Эта программа вызывается с помощью команды CALL KEYIN. После возврат программы KEYIN в аккумуляторе будет содержаться код нажатой кнопки (таблица 5). Эта мониторная программа может использоваться для чтения клавиатуры.

В таблице 6 показан листинг программы, которая вызывает звуковой сигнал, если нажата кнопка 7. Клавиатура читается с помощью вышеописанной программы KEYIN, которая заносит код нажатой клавиши в аккумулятор.

Таблица 5- Коды кнопок микролаборатории

Кнопка	Код (Hex)
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09

A	0A
B	0B
C	0C
D	0D
E	0E
F	0F
ПУСК	10
ВОЗВР	11
УСТ АД	12
АД -	13
АД +	14
ЗП	15
ВЫВОД	16
ВВОД	17

Команда CPI 07 устанавливает нулевой флаг процессора, если содержимое аккумулятора равно 07. Команда JNZ READ вызывает возврат программы на начало, если нулевой флаг не установлен. Если нулевой флаг установлен, то будет вызываться подпрограмма звукового сигнала BEEP (адрес 0350). Затем процесс повторяется. Таким образом, звуковой сигнал будет генерироваться при нажатии кнопки 7.

Введите листинг программы в микролабораторию и проверьте правильность ввода. Запустите программу. Нажмите кнопку 7 — динамик гудит. Нажмите любую другую кнопку — звуковой сигнал отсутствует.

Составьте программу, которая генерирует звуковой сигнал, если нажаты в определенной последовательности три определенные кнопки, например 7, 3, 9. Эту программу можно взять за основу при создании электронного замка. Составление программы начните со структурной схемы. Введите программу в микролабораторию и проверьте ее выполнение.

Сканирование клавиатуры осуществляется подпрограммой READ циклически. За один цикл считываются данные с одного ряда из восьми кнопок (см. рис. 5). Для простоты рассмотрим только один ряд, т. е. кнопки 0, 1, 2 ... 7. Чтение ряда кнопок представляет собой двухшаговую операцию: на первом шаге выполняется запись данных в сканирующий порт для выбора кнопок, а на втором, чтение байта данных читающим портом.

Таблица 6 - Программа вызова звукового сигнала

Адрес	Содержимое	Метка	Команда	Комментарии
8000	CD	READ	CALL KEYIN	Вызов программы, чтения данных с клавиатуры
8001	16			
8002	02			
8003	FE		CPI 07	Сравнение кода ключа с 07
8004	07			
8005	C2		JNZ READ	Возврат, если кнопка 7 не нажата
8006	00			
8007	80			
8008	CD		CALL BEEP	Звуковой сигнал, если кнопка 7 нажата
8009	50			

800A 800B 800C 800D	03 C3 00 80		JMP READ	Повторение программы

Чтобы выбрать определенный ряд кнопок, соответствующий бит сканирующего порта (PC4, PC5 или PC6) устанавливается в лог. 1, другие биты остаются равными лог. 0. Таким образом, чтобы выбрать ряд кнопок 0 ... 7, нужно послать в порт FA код 1001 1111 (9Fh), чтобы выбрать кнопки 8 ... F - код 1010 1111 (AFh) и чтобы выбрать кнопки ПУСК . . . ВВОД, - код 1100 1111 (CFh). Биты PC0...PC3, PC7 при сканировании клавиатуры не используются и устанавливаются равными лог.1. Итак, чтобы считать данные с ряда 0 ... 7, в порт выводится байт 9F. Затем производится чтение входным портом для сбора информации о состоянии кнопок. При нажатии определенной кнопки соответствующий бит читающего порта будет устанавливаться в 0 (таблица 7).Обратите внимание, что коды кнопок, приведенные в таблицах 5 и 7, не совпадают. Кнопка из других рядов при чтении выбранного ряда не влияет на результаты чтения.

Таблица 7- Коды, читаемые с клавиатуры

Кнопка	Читаемый код (двоичный)	Читаемый код (HEX)
Нет нажатых кнопок	11111111	FF
0	11111110	FE
1	11111101	FD
2	11111011	FB
3	11110111	F7
4	11101111	EF
5	11011111	DF
6	10111111	BF
7	01111111	7F

Листинг, представленный в таблице 8, показывает программу, реализующую только что описанный процесс чтения.

Таблица 8- Программа определения состояния кнопок

Команда	Комментарии
MVI A,9F OUT FA IN F8	Выбор ряда кнопок 0 ... 7 Чтение состояния кнопок

Программа монитора KEYIN не только сканирует клавиатуру, но и преобразует коды, читаемые с колонок (в зависимости от выбранного ряда), в коды, приведенные в таблице 5. В таблице 9 приведена программа для проверки состояния кнопки 2, без использования подпрограммы KEYIN. Если кнопка 2 нажата, генерируется звуковой сигнал. Наберите программу на „Микролаб" и проверьте ее работу.

Таблица 9- Программа проверки состояния кнопки 2

Адрес	Содержимое	Метка	Команда	Комментарии
8000	3E		MVI A, 90	Программирование порта А на ввод данных, а порта С на вывод
8001	90			
8002	D3		OUT FB	
8003	FB			
8004	3E	READ:	MVI A, 9F	Выбор ряда кнопок 0 ... 7
8005	9F			
8006	D3		OUT FA	
8007	FA			
8008	DB		IN F8	Чтение состояния колонок
8009	F8			
800A	FE		CPI FB	Сравнение аккумулятора с кодом (кнопка 2)
800B	FB			
800C	C2		JNZ READ	Читать снова, если кнопка 2 не нажата
800D	04			
800E	80			
800F	CD		CALL BEEP	Если нажата кнопка 2, появляется звуковой сигнал
8010	50			
8011	03			
8012	C3		JMP READ	Повторение программы
8013	04			
8014	80			

4 Вывод данных на светодиодный индикатор в динамическом режиме

Индикаторы выходного порта (светодиоды) могут использоваться для отображения разовых (битовых) команд. На этом принципе строится имитатор „бегущих огней". Контроллер бегущих огней должен регулировать следующую последовательность чередования горения светодиодов:

1. Горят светодиоды 1, 4, 7, остальные погашены.
2. Задержка времени горения установленных светодиодов.
3. Горят светодиоды 2, 5, 8; остальные погашены.
4. Задержка времени горения установленных светодиодов.
5. Горят светодиоды 3, 6; остальные погашены.
6. Задержка времени горения установленных светодиодов. .
7. Повторение процесса (переход к шагу 1).

На рисунке 6 приведен алгоритм работы контроллера.

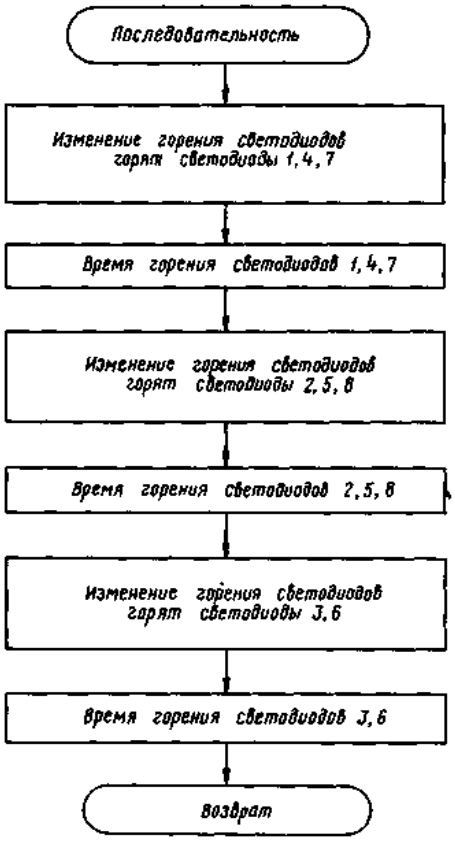


Рисунок 6- Алгоритм программы последовательности горения светодиодов

Выходные индикаторы (светодиоды YD1...YD8) подключаются к шине данных с помощью программируемого интерфейса. В таблице 10 показаны используемые комбинации сигналов имитирования „бегущих огней" на светодиодах. Единица в двоичном коде сигнала означает включенный индикатор.

Таблица 10- Значение разовых команд для реализации "бегущих огней"

VD1	VD2	VD3	VD4	VD5	VD6	VD7	VD8	Код
1	0	0	1	0	0	1	0	92_{16}
0	1	0	0	1	0	0	1	49_{16}
0	0	1	0	0	1	0	0	24_{16}

Если реализовать программу установки разовых команд последовательно одну за другой без некоторой задержки, частота горения светодиодов будет очень большой и эффекта бегущих огней наблюдаться не будет. Поэтому необходимо разработать подпрограмму, вызывающую задержку заданной длительности. Простейший способ генерирования задержки заключается в циклическом выполнении команд. Для увеличения длительности задержки используются регистровые пары и вложенные циклы.

В программе контроллера „бегущие огни", для получения задержки организуется счетчик в регистре D. Число, первоначально записанное в регистр D, определяет, сколько раз выполняется задержка длительностью в 0,786 с. Эта задержка формируется внутренним циклом с использованием регистровой

пары B-C. Максимальная задержка, полученная двумя циклами, таким образом, равна 256x0,786 =201 с.

В таблице 11 приведена программа задержки, используемая для контролера „бегущие огни". Команда LXI B, 0 заносит 0 в пару регистров B, C. Далее вырабатывается задержка 0,786 с. Регистр D используется для счета основной задержки.

Таблица 11-. Программа задержки

Метки	Команды	Комментарии
DELAY: LOOP:	LXI B, 0 DCX B MOV A,B ORA C JNZ LOOP DCR D RET	Начало счета внутренней петли Петля задержки 0,786 с Уменьшение основного счетчика и работа по петле, если ≠ 0

Основная программа последовательности довольно проста (таблица 12). Сначала программа устанавливает в регистр A число 92 — код выходных индикаторов, при котором светодиоды 1, 4, 7 — горят, а светодиоды 2, 3, 5, 6 погашены. Затем вызывает подпрограмму задержки. Для чего программа последовательности задает величину задержки в регистр D, после чего вызывается сама подпрограмма задержки. Далее эта последовательность повторяется для других кодов выходных сигналов светодиодов.

Для проверки программы:

1. Введите программу из таблицы 12 в ОЗУ.

2. Удостоверьтесь, что она правильно записана в память.

3. Выполняйте основную программу, начинающуюся по адресу 8000.

4. Значения временных задержек можно заменить на другие значения, для уменьшения скорости „бегущих огней по адресам 8009, 8012, 801B, 8023, 8024.

Изменяя значения кодов, добейтесь противоположного направления движения "бегущих огней".

Выполнить корректировку текста программы таким образом, чтобы скорость перемещения "бегущих огней" изминилась в большую и меньшую стороны.

Таблица 12- Программа контроллера "бегущие огни"

Адреса	Коды	Метки	Команды	Комментарии
8000	3E	FIRE:	MVI A, 81	Программирование интерфейса
8001	81			
8002	D3		OUT FB	
8003	FB			
8004	3E	SEQ:	MVI A, 92	Установка кода горения 1,4, 7 светодиодов
8005	92			
8006	D3		OUT F9	

8007	F9			
8008	16		MVI D,01	Горение данной последовательности
8009	01			
800A	CD		CALL DELAY	Вызов подпрограммы задержки
800B	22			
800C	80			
800D	3E		MOV A, 49	Установка кода горения 2, 5, 8 светодиодов
800E	49			
800F	D3		OUT F9	
8010	F9			
8011	16		MVI D, 01	Горение данной последовательности
8012	01			
8013	CD		CALL DELAY	
8014	22			
8015	80			
8016	3E		MOV A, 24	Установка кода горения 3, 6 светодиодов
8017	24			
8018	D3		OUT F9	
8019	F9			
801A	16		MVI D, 01	Горение данной последовательности
801B	01			
801C	CD		CALL DELAY	
801D	22			
801E	80			
801F	C3		JMP SEQ	Возврат в начало основной программы „бегущие огни"
8020	04			
8021	80			
8022	01	DELAY:	LXI B,0000	Начало внутренней петли задержки
8023	00			
8024	00			
8025	OB	LOOP:	DCX B	Внутренняя петля, генерирующая задержку 0,786 с
8026	78		MOV A, B	
8027	B1		ORA C	
8028	C2		JNZ LOOP	
8029	25			
802A	80			
802B	15		OCR D	Основная петля задержки
802C	C2		JNZ DELAY	
802D	22			
802E	80			
802F	C9		RET	Возврат

Задание для самостоятельной работы

Выполнить все примеры программ. По всем пунктам лабораторной работы необходимо составить отчет, содержащий текст программы на языке ассемблера и в машинных кодах с необходимыми комментариями. Сделать выводы о технологии программирования с использованием языка ассемблера и машинных кодов.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №2
ПРОГРАММИРОВАНИЕ ПРОЦЕДУР ВВОДА-ВЫВОДА БИТОВОЙ ИНФОРМАЦИИ

Цели лабораторной работы:

- познакомиться с платформой Arduino Uno и интегрированной средой программирования Arduino IDE;
- изучить состав и возможности интегрированной среды программирования Atmel Studio 7.

Учебные вопросы:

1. Технология программирования микроконтроллеров с использованием отладочной платы Arduino Uno и среды программирования Arduino IDE
2. Состав и возможности интегрированной среды программирования Atmel Studio 7

1. Технология программирования микроконтроллеров с использованием отладочной платы Arduino Uno и среды программирования Arduino IDE

Для эффективного изучения основ программирования в среде Arduino необходимо изучить реализацию основных алгоритмов в среде Arduino и представить примеры написания программного.

В основу языка программирования, используемого в проектах Arduino, положены языки C и C++ —самые широко используемые языки программирования, поддерживающие как работу с низкоуровневыми командами, так и построение сложных объектов.

Программирование контроллеров Arduino удобно осуществлять в специальной среде Arduino IDE, поскольку в нее включен основной функционал для работы с ними.

Установка Arduino IDE. Изучать основы программирования на платформе Arduino можно как на линейке физических устройств, так и в виртуальной онлайн среде Electronics Lab, предоставляемой компанией Autodesk на сайте https://circuits.io. Arduino IDE можно скачать в Интернете по адресу основного сайта проекта: www.arduino.cc/en/Main/Software, программное обеспечение свободно распространяется и для скачивания нужно выбрать в разделе Download the Arduino IDE вариант Windows Installer (рисунок 1), а затем JUST DOWNLOAD (рисунок 2).

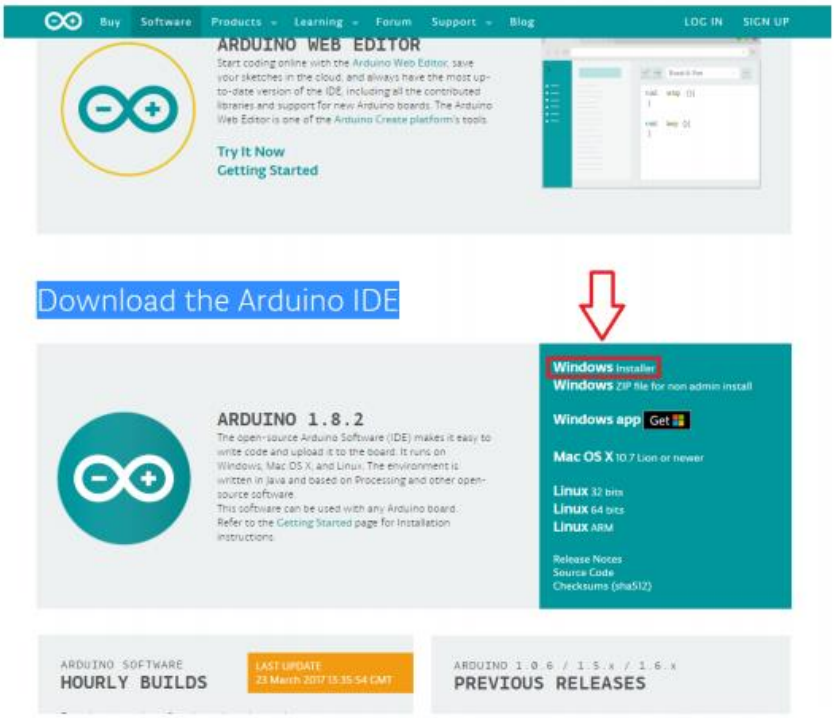


Рисунок 1 — Окно скачивания программы Arduino IDE

Рисунок 2 — Кнопка загрузки установочного файла Arduino IDE

На компьютер необходимо инсталлировать программу Arduino IDE — в настоящий момент это файл arduino-1.8.13-windows.exe. Его надо запустить на выполнение с административными полномочиями, принять условия лицензии GNU LESSER GENERAL PUBLIC LICENSE и согласиться с предложенным вариантом установки. На все предупреждения Windows в процессе установки следует отвечать утвердительно (продолжать установку). Когда установщик предложит установить драйверы порта, также ответить утвердительно.

Начало работы с Arduino IDE. Среда разработки имеет интуитивно понятный русскоязычный интерфейс. При запуске установленной Arduino IDE откроется окно (рисунок 3), в котором уже содержится заготовка программы. Она состоит из двух функций: setup и loop. Функция setup содержит команды, выполняемые одни раз при включении Arduino, — это установка номеров портов ввода/вывода для управления мониторами и установки скорости обмена данными между Arduino и компьютером. Функция loop выполняется бесконечное число раз — до тех пор, пока мы не отключим питание, фактически она

зациклена, алгоритмически это изображено на рисунке 4. Интерфейс позволяет реализовать ранее изученные базовые алгоритмы.

Рисунок 3 — Окно Arduino IDE

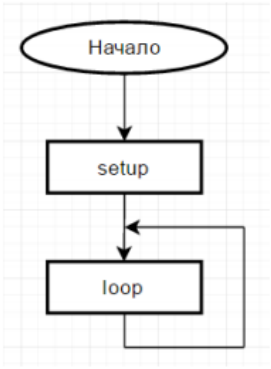


Рисунок 4 — Типовой алгоритм программы Arduino IDE

Подключение контроллера Arduino к ПК. Возможности среды позволяют в зависимости от контроллера Arduino использовать, разные USB-кабели (рисунок 5):

– для Arduino UNO R3 и Mega — это кабель с разъемом под USB принтер (Type B) с одной стороны и стандартным USB-разъемом (Type A) с другой;

– для контроллера Nano требуется кабель с разъемом mini-USB;

– для Arduino Micro — это micro-USB, а для программирования контроллера Arduino Mini и Pro Mini потребуется программатор, так как у него отсутствует стандартный интерфейс для подключения его к компьютеру.

Рисунок 5 — Варианты USB-кабелей: а) USB type B для Arduino UNO и Mega; б) кабель для Arduino Nano; в) micro-USB для Arduino Mini

Некоторые контроллеры требуют для своей работы нестандартный драйвер. Они требуют установки отдельного драйвера, не входящего в комплект Arduino IDE, — его название ch341ser.exe. Найти его не трудно, после установки драйвера проблема будет решена.

Контроллер подключается к ПК через USB интерфейс и устанавливает связи между ним и оболочкой Arduino IDE. Для этого нужно задать номер порта, к которому подключен контроллер (рисунок 6). Если портов много, и найти нужный сложно, рекомендуется запомнить все имеющиеся, а затем физически отсоединить Arduino от кабеля, и снова проанализировать список портов, — тот, который исчез, и есть нужный. После подключения необходимо выбрать нужный порт — для этого нужно установить соответствующий ему флажок. Иногда для появления порта в списке требуется некоторое время, за которое операционная система компьютера анализирует и проверяет подключенное устройство, так что следует немного подождать до завершения этого процесса.

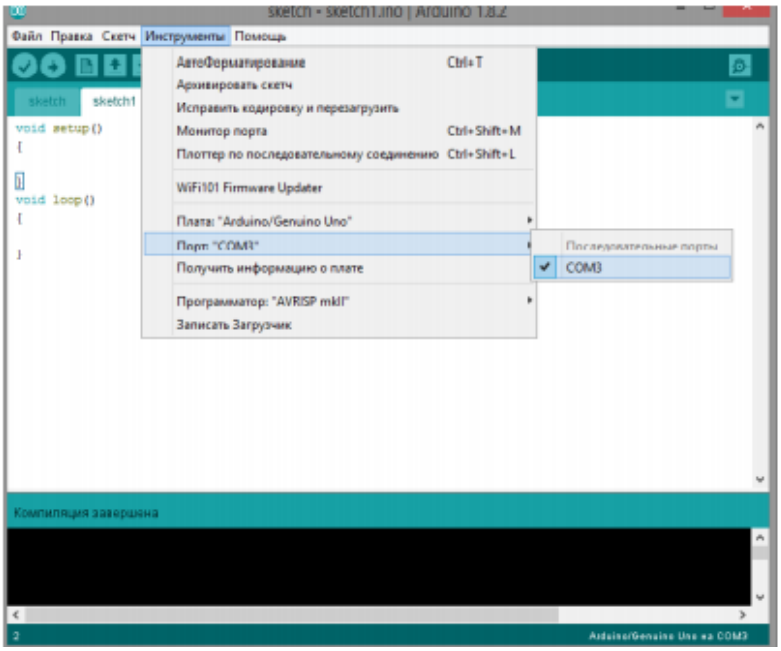


Рисунок 6 — Выбор нужного порта

Затем необходимо выбрать тип контроллера Arduino. На рисунке 7 можно видеть, что выбран Arduino/ Genuino Uno.

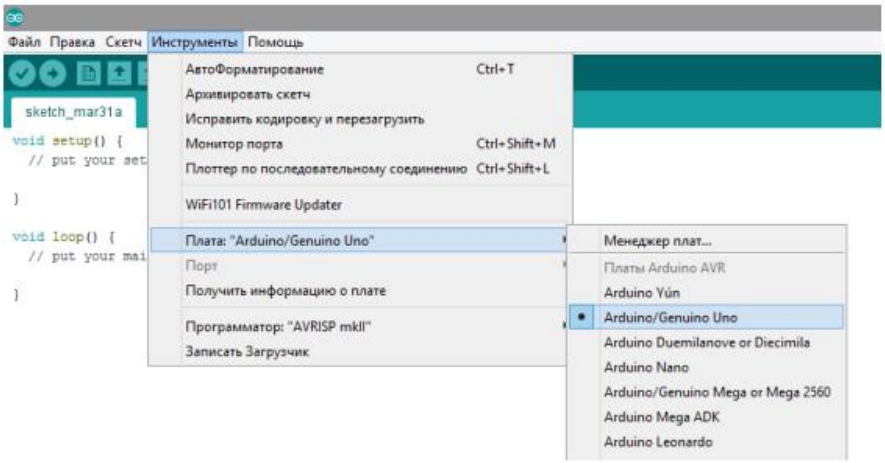


Рисунок 7 — Выбор типов контроллера Arduino

Если контроллер выбран системой автоматически, необходимо проверить правильность этого выбора. Для некоторых контроллеров необходимо еще выбрать подвид микроконтроллера, на котором реализована плата Arduino. Название микроконтроллера можно найти на самой его микросхеме — это, как правило, самая большая микросхема платы и расположена она в ее центре.

Базовые правила синтаксиса языка C\C++ .Прежде чем начать изучение языка C\C++, необходима интегрированная среда разработки — IDE. Язык C\C++ представляет собой набор команд. Этот самый набор команд иными словами называется кодом или исходным кодом.

Загрузка программы. Написанная программа в Arduino IDE загружается в контроллер нажатием кнопки со стрелкой вправо (см. рисунок 3). Оболочка проверит программу на наличие ошибок, а затем переведет ее в двоичный код данных и команд выбранного микроконтроллера и запишет в Arduino. Если после появления слова Загрузка в нижней строке окна Arduino IDE замигали светодиоды TX и RX на плате Arduino, то загрузка программы в плату началась успешно. Если после этого фраза Загрузка завершена не появилась, то вероятнее всего неверно выбран тип платы Arduino. Если же диоды TX и RX не замигали вообще, то проблемы заключаются в выборе порта платы Arduino. Если все пойдет штатно, появится надпись Загрузка завершена, и программа автоматически начнет выполняться. Если загружена пустая программа, то ничего происходить не будет, — пустой код будет бесконечно повторяться, пока к плате подключено электропитание.

Программирование Arduino рассмотрим на примере программы управления встроенным светодиодом, закрепленным за 13-м выводом разъема платы (рисунок 8).

Цифровые порты могут получать и передавать информацию только в виде последовательности нулей и единиц. В приведенном примере как раз и передается на порт 13 подобная последовательность — визуально это отслеживается по морганию светодиода на плате контроллера. Особенностью сце-

нария Arduino IDE является использование встроенных подпрограмм и функций, а также возможность использования логических имен регистров микроконтроллера.

```
void setup() // настройка
{
  pinMode (13, OUTPUT); // перевод 13-го порта в состояние вывода информации
}

void loop() // основная программа
{
  digitalWrite (13, 1);  // включение светодиода на плате
  delay (1000);          // задержка в 1 секунду
  digitalWrite (13,0);   // выключение светодиода на плате
  delay (1000);          // задержка в 1 секунду
}
```

Рисунок 8 — Программа мигания светодиода

Когда светодиод горит, на порту высокое электрическое напряжение 5В, а когда потушен — низкое (0В).

Первая встроенная подпрограмма определяет назначение 13 вывода разъема, размещенного на отладочной плате (рисунок 9).

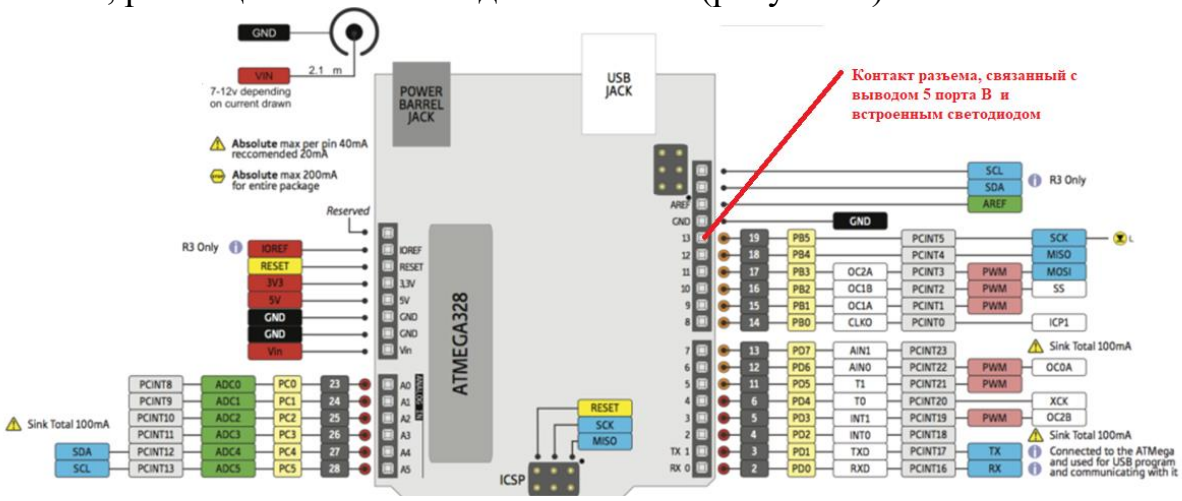


Рисунок 9 – Назначение выводов отладочной платы Arduino Uno

Синтаксический анализатор, встроенный в Arduino IDE, выделяет голубым цветом известные логические имена переменных и служебные слова, а красным цветом – имена встроенных подпрограмм и функций.

Назначение 5 вывода порта В на выдачу бита информации (OUTPUT) можно осуществить и следующим образом:

DDRB|=1<<5;

В программе, написанной с использованием Atmel Studio выводы портов, предназначенные для приема информации , назначаются по умолчанию. В Arduino IDE необходимо выполнить подпрограмму pinMode с аргументом INPUT, например для PORTD7:

pinMode(7,INPUT).

Для вывода и приема битовой информации используются подпрограммы digitalWrite и digitalRead с аргументами в виде номера ранее настроенного вывода и значения для вывода. Состояние вывода можно задать с использованием зарезервированных переменных HIGH, для установки единицы и LOW для установки нуля.

Встроенная подпрограмма delay(1) осуществляет задержку на 1 мс.

Мониторинг работы программы. Мониторинг программы позволяет выполнить отладку устройства. Для обмена информацией персонального компьютера и микроконтроллера в процессе выполнения программы используется библиотечные подпрограммы Serial. Настройка последовательного пртокола обмена производится установкой в разделе setup подрограммы Serial.begin(9600), которая указывает, с какой скоростью происходит обмен информацией с ПК. Для обмена информацией требуется, чтобы приемник и передатчик осуществляли обмен на одинаковой скорости, а так как контроллер Arduino достаточно медленный, то наиболее подходящей для обмена с ним является скорость 9600 бод. Обмен информацией на более высоких скоростях также возможен, но задействует большее количество ресурсов контроллера, при этом чаще происходят прерывания текущей программы, и основной код может выполняться медленнее. Команда Serial.println() передает с контроллера на ПК значение, указанное в скобках. Результат работы приведен на рисунке 10. Для получения на экране компьютера полученных результатов после запуска программы потребуется открыть вкладку Инструменты | Монитор порта и скорректировать скорость порта (9600 бод) — в окне Монитор порта скорость задается в правом нижнем углу выбором из всплывающего списка.

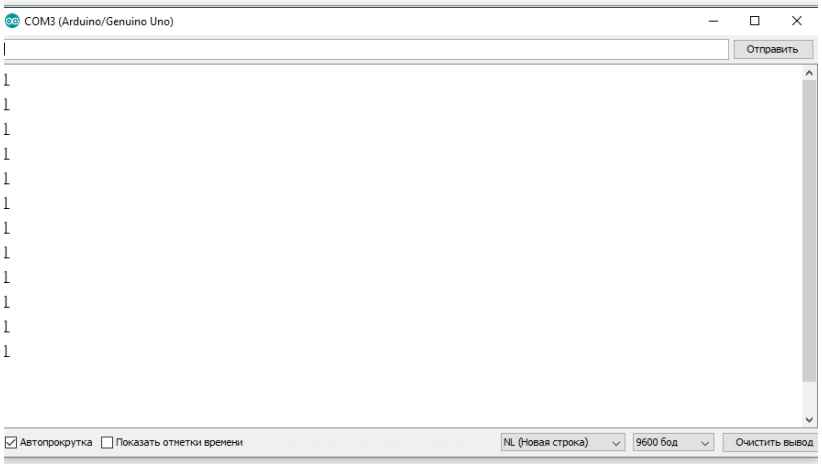


Рисунок 10 — Вывод результатов на экран компьютера с использованием последовательного протокола

Переменные. Важной составляющей синтаксиса языков программирования являются переменные. Переменные — это области памяти, имеющие имя, которое иначе называют идентификатором. До начала работы с переменными их нужно объявить, как и в любой другой среде программирования, например Turbo Pascal. В языке C/C++ все переменные должны быть объявлены. Это означает, что, во-первых, в начале каждой программы или функции

должен быть приведен список всех используемых переменных, а во-вторых, указан тип каждой из них (рисунок 11).

```
// переменные видны в программе везде
int a;
float b;

// переменная видна только внутри функции setup
void setup() // настройка
{
  int c;
  c=10;
  Serial.begin(9600); // скорость порта связи Arduino - ПК
  a=c*5; //значение переменной a изменится на 10*5
}

void loop() // основная программа
{
  // следующая строка сожержит ошибку
  c=a+5;        // переменная c не может быть видна за пределами функции setup
  delay(a*100); // задержка 10*5*100 миллисекунд
  Serial.println(millis()); // передача на ПК время работы в миллисекундах
  delay(1000); // задержка 1 секунду
}
```

Рисунок 11 — Определение переменных. Зоны видимости. Сообщение об ошибке

Рисунок 11 в учебных целях содержит специально добавленную в него ошибку объявления переменной, и при попытке его компилировать об этой ошибке будет выведено сообщение: ‘c’ was not declared in this scope. Переменные могут отличаться по типу данных, т.е. по размеру ячеек памяти (в байтах), зарезервированных за этими переменными (таблица 1).

Таблица 1 — Типы данных

Обозначение в программе	Название	Принимаемые значения	Размер памяти
boolean	Логический	True/False 1/0	1 байт
char	Символьный	Код ASCII	1 байт
byte	Короткий беззнаковый	0-255	1 байт
int	Целый	-32768…32768	2 байта
long	Длинное целое	-2147483648… 2147483648	4 ьайта
float	С плавающей точкой	-3,4028235 $x10^{38}$- 3,4028235 $x10^{38}$	4 байта

При работе с переменными следует понимать, что они не являются идеальным хранилищем информации — так, например, целочисленные перемен-

ные могут переполняться. Это происходит в тех случаях, когда значение, которое следует записать в переменную, больше максимально возможного для этого типа данных. Переменные с плавающей точкой подвержены другой проблеме — они округляют свои значения при сложении большого числа с малым. Так же Arduino IDE не всегда корректно работает с преобразованием типов данных. На следующем примере можно продемонстрировать небольшую программу, осуществляющую получение данных от компьютера через порт ввода/вывода (рисунок 12).

```
void setup() // настройка
{
 Serial.begin(9600); //скорость порта связи Arduino - ПК
}

void loop() // основная программа
{
 char symbol; // переменная символьного типа данных
 if (Serial.available() > 0)
 {
  symbol=Serial.read(); // считывание символов из переменной
  Serial.println(symbol); // вывонд на ПК то, что было считано
 }
}
```

Рисунок 12 — Получение данных от компьютера через порт ввода/вывода данных

Скомпилировав и загрузив эту программу, можно ввести в верхней части окна Монитор порта слова «Hello, World!» и отправить их на порт. Программа отправит сообщение обратно на ПК посимвольно (рисунок 13).

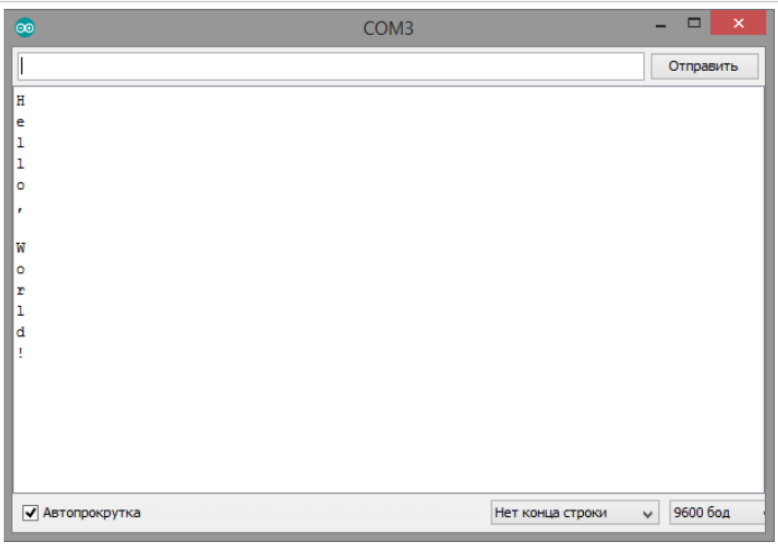


Рисунок 13 — Вывод сообщения на ПК

2. Состав и возможности интегрированной среды программирования Atmel Studio 7

Интегральная среда разработки Atmel studio загружается как обычное Windows приложение. Внешний вид заставки Atmel studio актуальной на сегодня седьмой версии представлен на рисунке 14.

Рисунок 14 – Внешний вид программы Atmel Studio7 при загрузке

После загрузки открывается рабочее окно программы в котором угадываются знакомые для всех Windows приложений средства работы с файлами File – New- Project (рисунок 14).

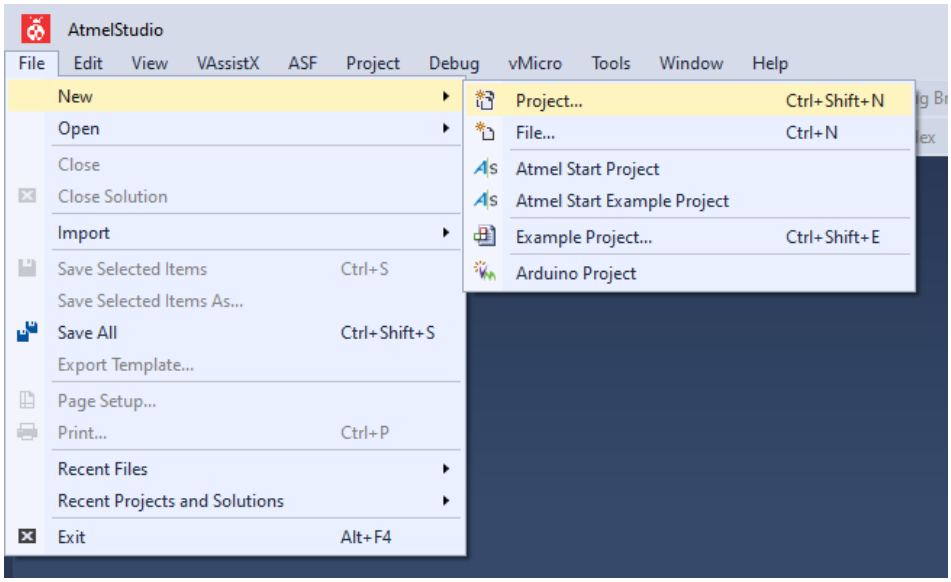


Рисунок 14 – Создание нового проекта

В открывшемся окне можно выбрать разные виды проектов. В виде сценария (Arduino Sketch), исполняемой программы на языке С или С++ или подпрограммы для библиотеки подпрограмм (рисунок 15).

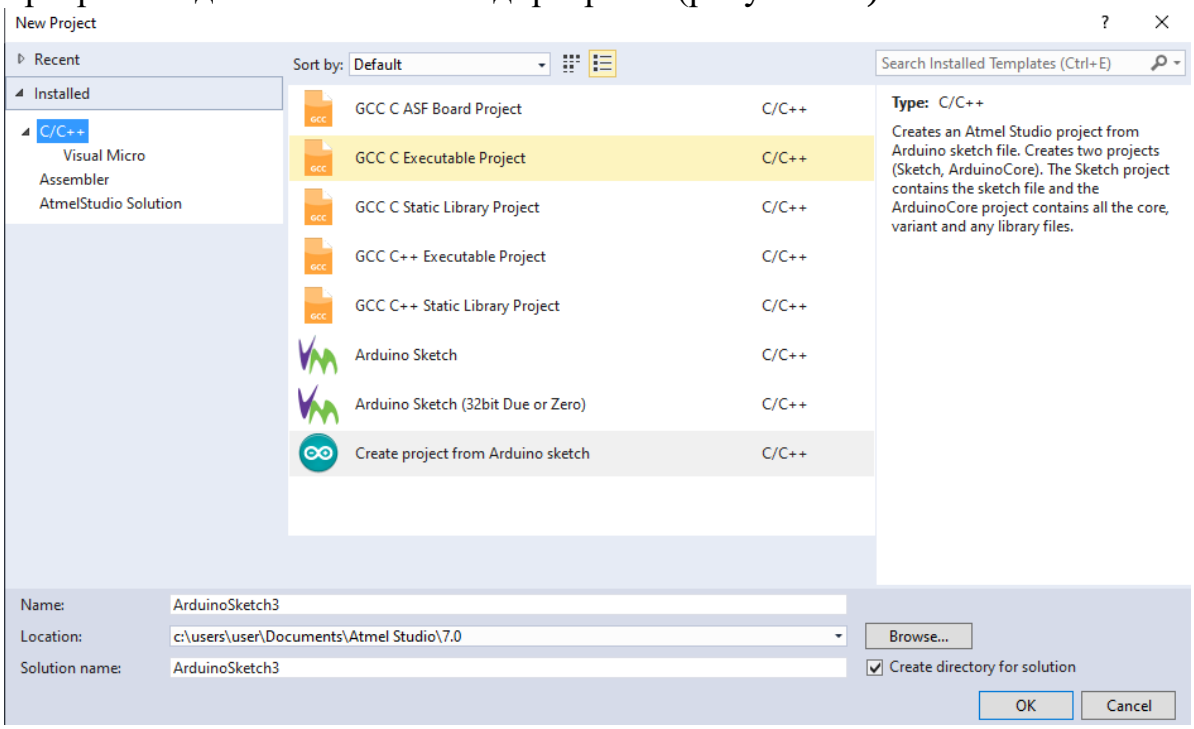


Рисунок 15 – Выбор варианта проекта

При выполнений лабораторных работ необходимо выбирать проект исполняемого файла на языке С. После его выбора открывается окно выбора типа микроконтроллера. В нашем случае Atmega 328 (рисунок 16).

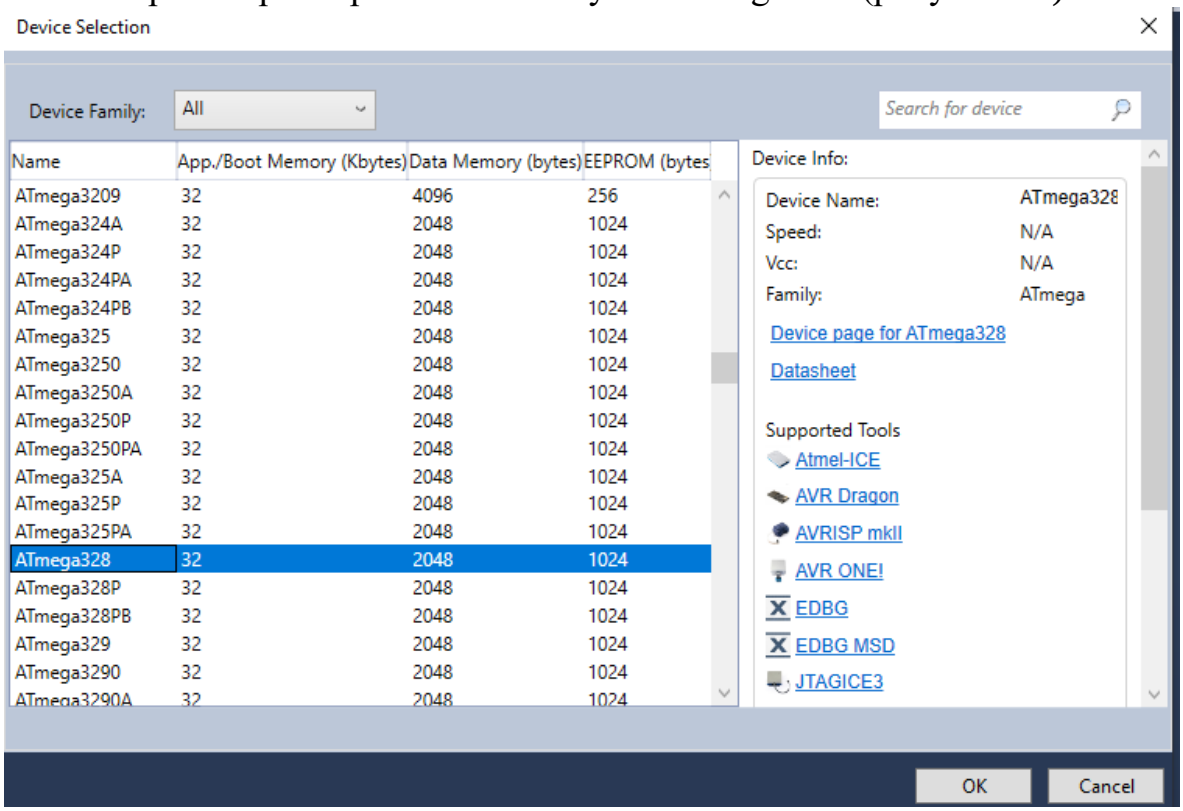


Рисунок 16 – Выбор типа микроконтроллера

После выбора микроконтроллера к новому проекту добавляются необходимые библиотеки в которых произведена замена физических адресов регистра на их огические наименования, более понятные человеку.

Окно шаблона файла на языке С представлено на рисунке 17.

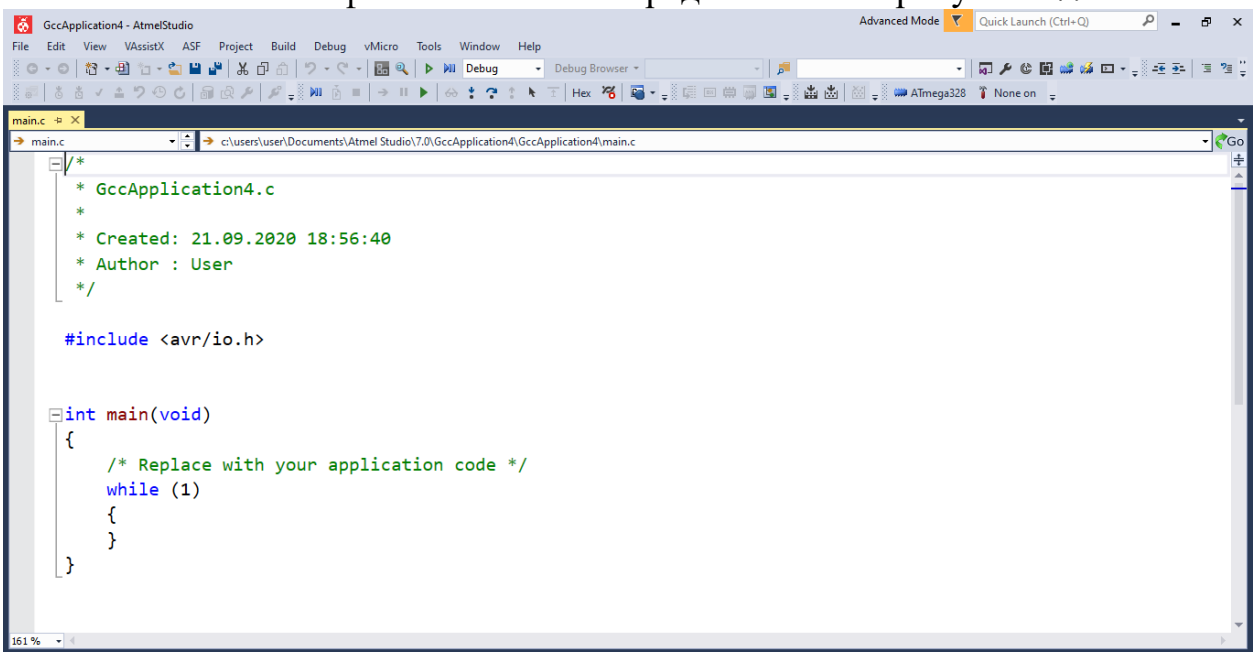


Рисунок 17 – Шаблон программы на С

На рисунке 6 представлена программа управления светодиодом платы Arduino Uno, закрепленного за пятым выводом порта B (PB5). Дополните текст программы, так как показано на рисунке 18.

```
 *
 * Created: 21.09.2020 18:56:40
 * Author : User
 */

#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>

int main(void)
{
    DDRB |=1<<5;
    PORTB &=~1<<5;
    while (1)
    {
        PORTB|=1<<5;
        _delay_ms(500);
        PORTB&=~1<<5;
        _delay_ms(20);
    }
}
```

Рисунок 18 – Программа управления морганием светодиодом

Найдите в папке в которой размещена интегрированная среда Atmel Studio файлы `io.h` и iom328p.h и ознакомьтесь с их содержанием. Возможный путь размещения файлов `C:\Program Files (x86)\Atmel\AVR Tools\avr8-gnu-toolchain-win32_x86\avr\include\avr`

Выполните компиляцию программы (преобразуйте текст программы в шестнадцатеричный код). Для этого пункт панели инструмента Build (кнопка F7) в меню Project (Рисунок 19).

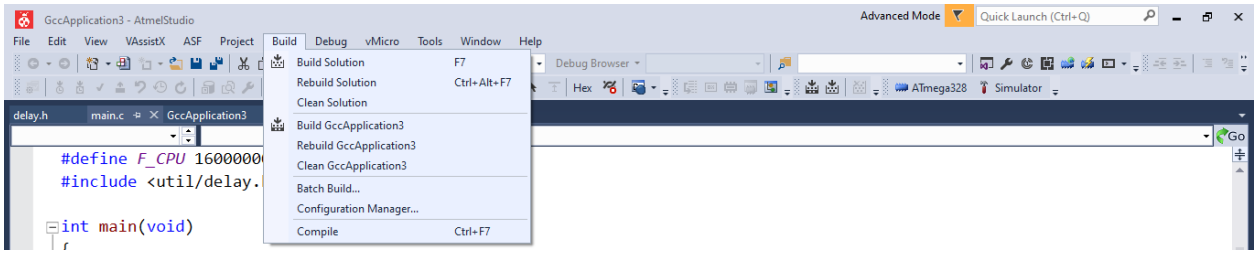


Рисунок 19 – Панель компилятора

Окно View Output содержит сообщения компилятора. В это окно выводится информация о количестве слов кода и данных, о наличии ошибок и другая информация (Рисунок 20). Выводится информация сколько процентов использовано в памяти программ (основной код программы) и памяти данных .

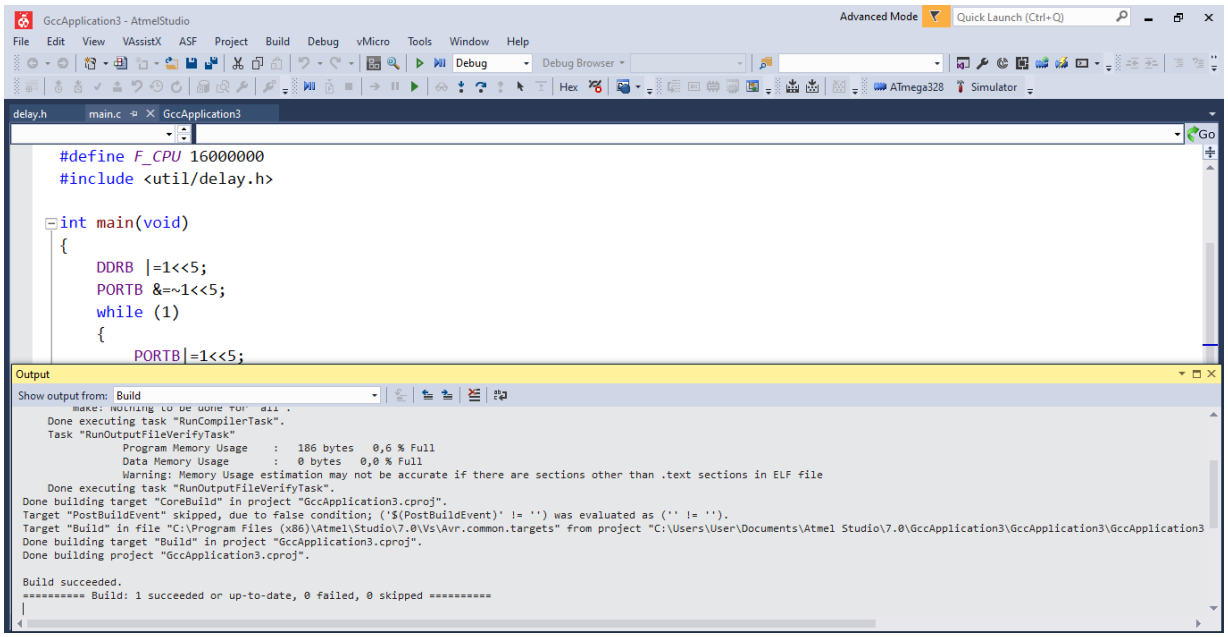


Рисунок 20 – Окно сообщений компилятора

Для локализации ошибок трансляции в случае их наличия можно в окне сообщений установить курсор мыши на сообщение об ошибке и два раза щелкнуть левой кнопкой мыши. При этом в окне редактирования исходного текста программы курсор будет установлен на строку, вызвавшую сообщение об

ошибке, и эта строка будет выделена цветом. В результате трансляции создается выходной файл в указанном формате. Для запуска отладчика необходимо выполнить процедуру Debug, которая вызывается при нажатии на соответствующую кнопку (или F5+Ctrl) на панели управления. Экран Atmel Studio в режиме отладки представлен на рисунке 21.

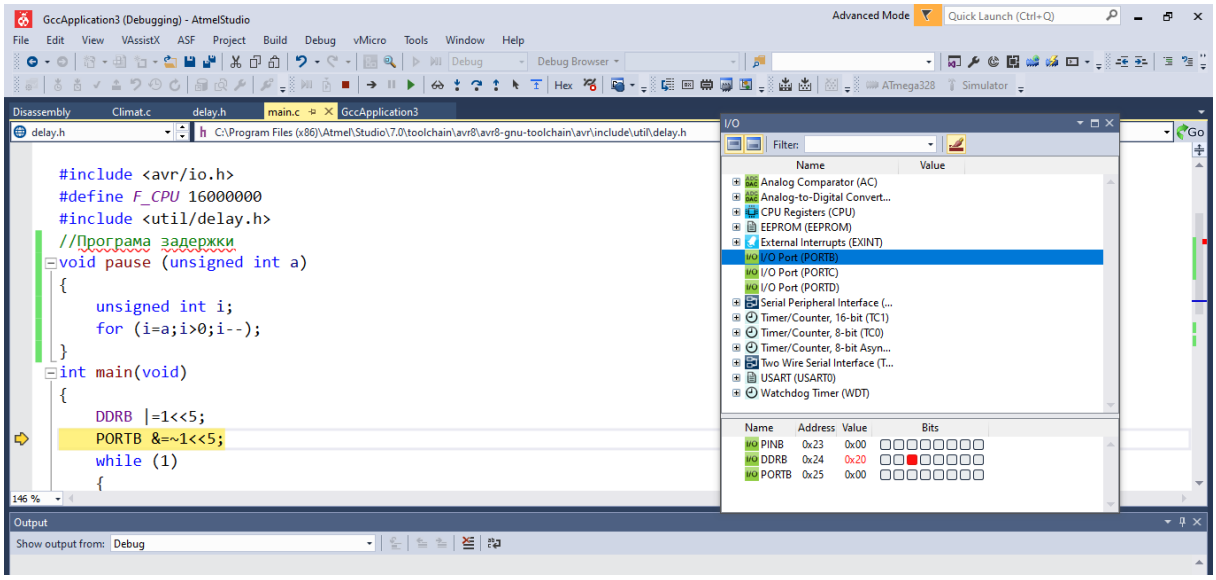


Рисунок 21 – Окно программы в режиме отладки

При выполнении процедуры отладки (или при загрузке объектного файла) автоматически открывается окно исходного текста исполняемой микроконтроллером программы. После выполнения процедуры появляется желтая стрелка, указывающая позицию программного счетчика микроконтроллера (см. рис. 21). Этот указатель всегда находится на строке, которая будет выполнена в следующем цикле. Пользователь может выполнять программу полностью в пошаговом режиме, трассируя блоки функций, или выполняя программу до того места, где стоит курсор. В дополнение можно определять неограниченное число точек останова, каждая из которых может быть включена или выключена. Точки останова сохраняются между сессиями работы. В Atmel Studio для отладки программы предусмотрены две команды пошагового режима: Step Over и Step Into. Разница между ними в том, что команда Step Over не работает в подпрограммах. С помощью команд пошагового режима можно проследить изменения значений в регистрах устройств ввода/вывода, памяти и регистрового файла. К командам шагового режима относятся также Auto Step и Multi Step. Помимо шагового режима, возможна отладка программы с использованием точек останова (Breakpoints). Командой Go запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения точки останова.

Задание для самостоятельной работы

Выполнить разработку и отладку программ в соответствии с индивидуальными заданиями в соответствии с таблицей 2.

Таблица 2- Варианты индивидуальных заданий

Параметры задания	Номер варианта				
	1	2	3	4	5
Порт ввода	PD2	PD3	PD4	PD5	PD6
Порт вывода	PB1	PB2	PB3	PB4	PB5
Условие установки порта вывода в 1	PD2=0	PD3=1	PD4=0	PD5=1	PD6=0
Параметры задания	Номер варианта				
	6	7	8	9	10
Порт ввода	PB1	PB2	PB3	PB4	PB5
Порт вывода	PC0	PC1	PC2	PC3	PC4
Условие установки порта вывода в 1	PB1=0	PB2=1	PB3=0	PB4=1	PB5=0

Порядок выполнения задания

Пример реализации проекта в среде Atmel Studio рассмотрена для варианта исходных данных в котором в качестве порта ввода используется PC1, в качестве порта вывода PB1, а условие установки порта вывода в 1 является равенство нулю PC1.

В соответствии с заданием определяется назначение, соответствующих разрядов портов PC1 на ввод данных и порта PB1 на вывод. Для этого в программе в первых разрядах регистров направления DDR портов C и B устанавливаются в 0 и 1 соответственно:

```
DDRC &=~1<<1;
DDRB |=1<<1;
```

Устанавливаем начальное значение PC1=0:

```
PORTB &=~1<<1;
```

В циклической части алгоритма размещаем условие установки выходного порта в соответствии с заданием.

```
while (1)
{
        b=PORTC;
        if((PORTC&0x02)==0x02) PORTB&=~(1<<1);
        if((b&0x02)==0x00) PORTB|=1<<1;
}
```

Где переменная типа char b обеспечивает временное хранение содержимого порта C. На рисунке 22 представлен полный текст программы.

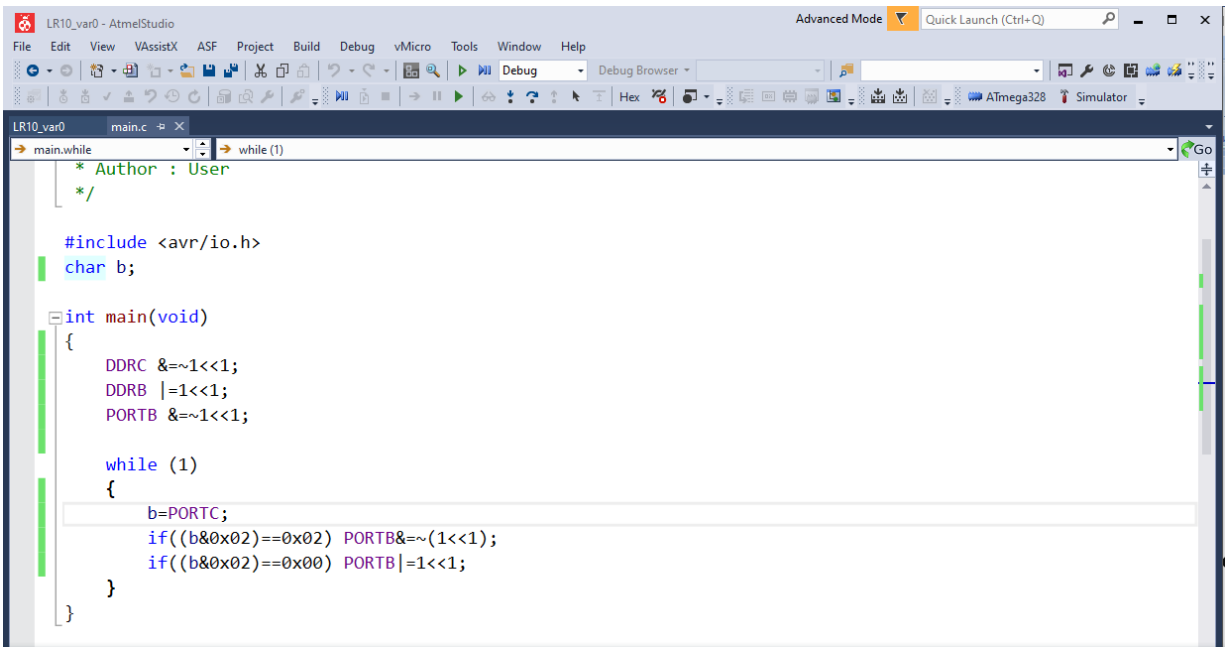


Рисунок 22 – Программа управления состоянием порта

Требования к оформлению отчета

Отчет по лабораторной работе должен содержать:

-титульный лист;

-тему, цель и учебные вопросы;

-исходные данные по заданному варианту;

-результаты отладки программы в режиме симулятора;

-скриншоты экрана с программным кодом и результатами работы представить в отчете.

Сделать выводы о технологиях программирования с использованием интегральных сред программирования Arduino IDE и Atmel Studio.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 3
ПРОГРАММИРОВАНИЕ ТАЙМЕРОВ/СЧЕТЧИКОВ

Цели работы:

– Изучить программирование функций задержки, измерения временных интервалов, генерирования последовательности прямоугольных импульсов заданной частоты и скважности с использованием платформы Arduino Uno и инструментальной среды Arduino IDE

– Изучить программирование встроенных таймеров/счетчиков микроконтроллера ATmega 328 в инструментальной среде Atmel Studio.

– Совершенствовать навыки работы в инструментальных средах Arduino IDE и Atmel Studio.

Учебные вопросы:

1 Создание программ с использованием встроенного таймерам в среде Atmel Studio

2 Программирование временных задержек с использованием библиотеки Arduino

3 Измерение временных интервалов

4 Генерирование последовательности импульсов с заданным периодом и скважностью

1 Создание программ с использованием встроенного таймерам в среде Atmel Studio

В микроконтроллерах ATmega есть два 8-ми и один 16-ти разрядные таймеры/счетчики. В соответствии со своим названием он считает либо временные интервалы (таймер), либо входные события (счетчик). У таймера/счетчика есть регистр TCNT из которого можно прочитать сколько времени прошло с момента запуска таймера. Значение в этом регистре не в минутах или секундах, а в— ‘тиках’ таймера. Чему равен один тик — зависит от тактовой частоты, на которой работает микроконтроллер и от настроек таймера.

Так же у таймера/счетчика есть настраиваемый делитель частоты, который определяет на сколько будет поделена тактовая частота микроконтроллера перед тем как будет подана на таймер/счетчик. Длительность тика таймера является обратной величиной от частоты, полученной в результате деления.

Например:

Тактовая частота = 16000000 Гц (16 МГц)

Делитель = 1024

Частота таймера/счетчика = Тактовая частота/Делитель = 16000000/1024 = 15625 Гц

Длительность периода(тика) = 1/15625 =$64*10^{-6}$ секунды = 64 мкс

Программирования таймеров/счетчиков рассмотрим на примере 16-ти разрядного таймера T1, структурная схема которого представлена на рисунке 1.

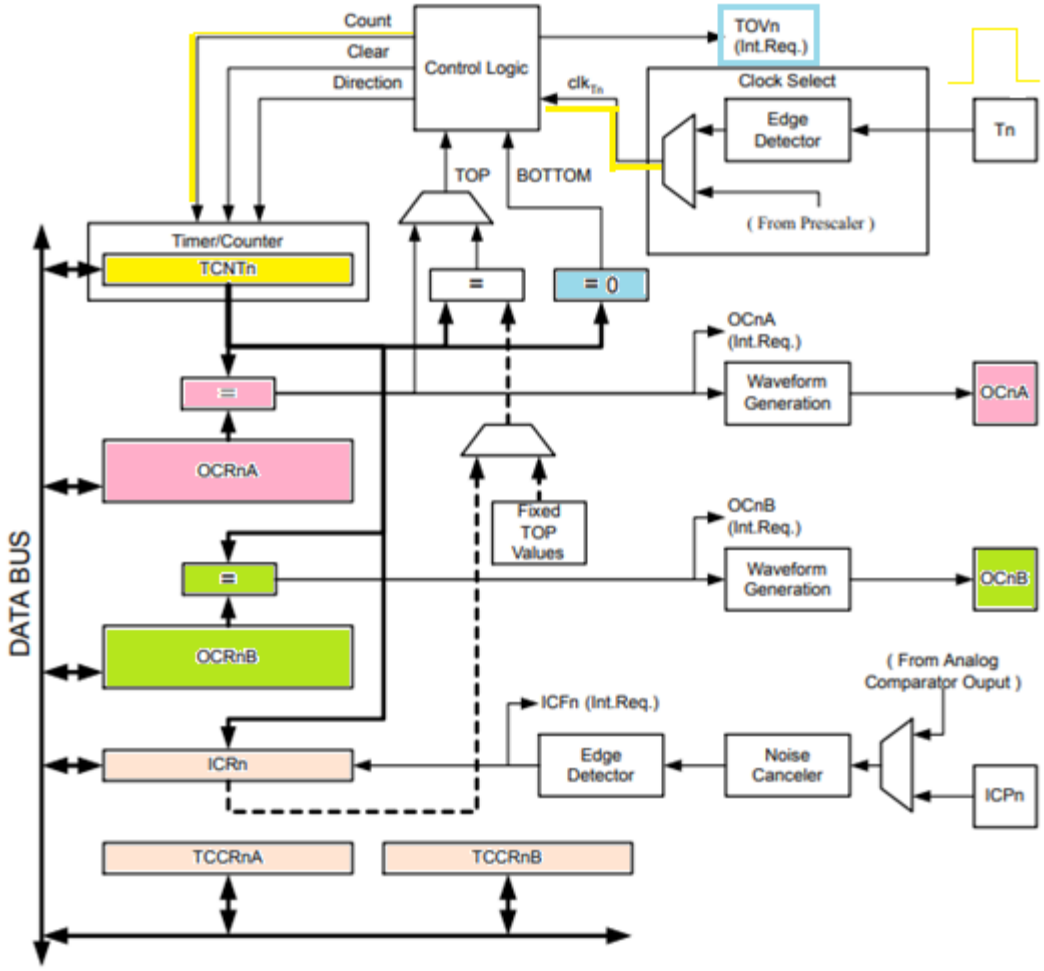


Рисунок 1

Для настройки делителя используются биты CS регистра TCCR.

Например, для таймера 1 в ATmega328 используются регистр TCCR1B и биты CS10, CS11, CS12.

Биты CS12 (2), CS11 (1), CS10 (0) регистра **TCCR1B** устанавливают режим тактирования и предделителя тактовой частоты таймера/счетчика T1:

- 000 - таймер/счетчик T1 остановлен
- 001 - тактовый генератор CLK
- 010 - CLK/8
- 011 - CLK/64
- 100 - CLK/256
- 101 - CLK/1024
- 110 - внешний источник на выводе T1 (11 ножка) по спаду сигнала
- 111 - внешний источник на выводе T1 (11 ножка) по возрастанию сигнала

Пример кода настройки делителя для таймера 1 ATmega328:

```
TCCR1B = (1<<CS12)|(0<<CS11)|(1<<CS10); //clk/1024
```

В ATmega328 таймеры могут быть 8 или 16 разрядными. Разрядность определяет максимальное количество тиков которые может сосчитать таймер. Для 8 разрядного это 256 тиков, для 16 это 65536.

В нашем случае один тик равен 64 мкс, соответственно максимальное значение, которое может измерить 16 битный таймер, это $65536*64*10^{-6}$ = 4.194 секунды, 8-ми битный — 0.01632 секунды.

После того как таймер досчитает до максимального значения, он переполняется, т.е. начинает считать с нуля. Эту ситуацию можно обрабатывать при помощи прерываний. Для этого надо разрешить прерывание по переполнению таймера и выставить бит общего разрешения прерываний.

Пример кода для разрешения прерываний таймера 1 ATmega328:

```
TIMSK |= (1<<TOIE1);  // разрешить прерывание по переполнению таймера счетчика
sei();              // выставить бит общего разрешения прерываний
```

Так же надо определить обработчик данного прерывания — функцию которая будет вызвана при переполнении таймера.

Добавим в эту функцию код изменяющий состояние ножки PB0, к которой подключен светодиод.

Пример кода функции обработчика прерывания для таймера 1 ATmega328:

```
ISR( TIMER1_OVF_vect )
{
 if( PINB & ( 1 << PB0 ) ) {
  PORTB &= ~( 1<<PB0 );
 }
 else {
  PORTB |= ( 1 << PB0 );
 }
}
```

Светодиод будет изменять свое состояние через каждые 4 секунды (частота 0,25 Гц).

Чтобы светодиод менял свое состояние с частотой 2Гц (период 0,5 с), надо чтобы таймер отсчитывал не 65536 тиков, а меньше. Рассчитаем необходимое число тиков:

$0,5\ с/64*10^{-6}=7812$.

Чтобы таймер считал не с нуля, а с некоторого значения, при инициализации и при каждом срабатывании прерывания в регистр TCNT1 будем записывать число (65536 — 7812) = 57724.

Листинг итоговой программы для таймера 1 ATmega328*:*

```
#include <avr/io.h>;
#include <avr/interrupt.h>;

ISR( TIMER1_OVF_vect )
{
 TCNT1 = 57724; //выставляем начальное значение TCNT1
 if( PINB & ( 1 << PB0 ) ) {
  PORTB &= ~( 1 << PB0 );
 }
 else {
  PORTB |= ( 1 << PB0 );
 }
}

int main()
{
 DDRB = ( 1 << PB0 );  // настраиваем PB0 на выход
 TCCR1B = (1<<CS12)|(0<<CS11)|(1<<CS10); // настраиваем делитель
 TIMSK1 |= (1<<TOIE1); // разрешаем прерывание по переполнению
таймера
 TCNT1 = 57724;       // выставляем начальное значение TCNT1
 sei();             // выставляем бит общего разрешения прерываний
 while(1);          // вечный цикл
 return 0;
}
```

Измените частоту моргания светодиода в соответствии с вариантом исходных данных (таблица 1)

Номер варианта	1	2	3	4	5	6	7	8	9	10
Частота, Гц	1	2	3	4	5	6	7	8	9	10

Выполните прошивку программы с помощью программатора USBasp.

2 Программирование временных задержек с использованием библиотеки Arduino

В библиотеке Arduino.h две встроенные функции задержки.

Функция delay(time) “приостанавливает” выполнение кода на time миллисекунд. Тип аргумента unsigned long позволяет приостановить выполнение основной программы на срок от 1 мс до ~50 суток (4 294 967 295 миллисекунд) с разрешением в 1 миллисекунду.

Функция delayMicroseconds(time). Является аналогом функции delay(), но приостанавливает выполнение кода на время time, заданное в микросекундах . Аргумент time имеет тип данных unsigned int и может задержать выполнение основной программы на срок от 4 до 16383 мкс с шагом 4 мкс.

Программирование временных задержек в среде Arduino IDE рассмотрим на примере программы управления светодиодом, закрепленным за 2 выводом разъема платы.

Листинг 1.

```
const int ledPin=2;
void setup() {   pinMode(ledPin, OUTPUT); }
void loop() {
  digitalWrite(ledPin, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);                   // wait for a 0,5 second
  digitalWrite(ledPin, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                   // wait for a 0,5 second
}
```

Подключите светодиод как показано на рисунке 2. Наберите текст программы в текстовом редакторе Arduino IDE. Проверьте программу на наличие ошибок с помощью вкладки Скетч- Проверить/Компилировать. При отсутствии ошибок подключите плату Arduino Uno к персональному компьютеру с помощью кабеля USBA-USBB. Загрузите программу нажатием кнопки в виде горизонтальной стрелки или выбрав Скетч-Загрузка (комбинация клавиш Ctrl/U). При загрузке программы наблюдается моргание светодиодов Rx, Tx. После загрузки произойдет запуск программы и светодиод, соединенный с выводом 2 платы как показано на рисунке 2 начнет менять свое состояние с периодичностью 0,5 с.

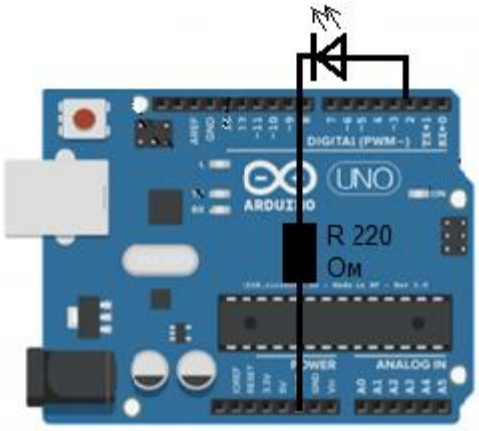


Рисунок 2 – Подключение светодиода

3 Измерение временных интервалов

Инструментом для измерения временных интервалов используются функции, которые считают время со старта микроконтроллера (включения питания). Таких функций в библиотеке Arduino.h две:

millis()

– Возвращает количество миллисекунд, прошедших с запуска. Возвращает переменную типа unsigned long, от 1 до 4 294 967 295 миллисекунд (~50 суток), имеет разрешение 1 миллисекунда, после переполнения сбрасывается в 0. Работает на системном таймере Timer 0

micros()

– Возвращает количество микросекунд, прошедших с запуска. Возвращает переменную типа unsigned long, от 4 до 4 294 967 295 микросекунд (~70 минут), имеет разрешение в 4 микросекунды, после переполнения сбрасывается в 0. Работает на системном таймере Timer 0

Для измерения временных интервалов с использованием встроенных функций millis() и micros() необходимо в программе предусмотреть сохранение времени, накопленного с момента включения микроконтроллера, в специально выделенную переменную в момент начала измеряемого интервала. А в момент окончания измеряемого интервала из текущего значения времени millis() и micros() вычесть значение сохраненного времени старта.

В приведенном ниже примере функция **millis()** обеспечивает моргание светодиода как в предыдущем примере.

Листинг 2.

```
const int ledPin = 2;
unsigned long next_time; // время очередного переключения
                         //   светодиода
int timeout = 500; // половина периода мигания
int led_state = 0; // начальное состояние светодиода - выключен
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, led_state); // гасим светодиод
   next_time = millis() + timeout; // вычисляем время следующего
                                   // переключения
}
void loop() {
  int now_time = millis(); // текущее время
   if( now_time >= next_time ){ // если текущее время превысило
                                // намеченное время, то
     next_time = now_time + timeout; // вычисляем время следующего
                                     // переключения
     led_state = !led_state; // меняем состояние на противоположное
     digitalWrite(ledPin, led_state); // зажигаем или гасим светодиод
}
}
```

Выполните ввод программы, её отладку и компиляцию. При отсутствии ошибок загрузите программу в память микроконтроллера. После завершения загрузки наблюдайте моргание светодиода.

4 Генерирование последовательности импульсов с заданным периодом и скважностью

Для управления цифровыми устройствами необходимо уметь генерировать последовательность прямоугольных импульсов заданной длительности и частоты следования. На рисунке 3 показан пример такой последовательности. Амплитуда импульсов определяется уровнем логической единицы и составляет для микроконтроллера ATmega 328 5В.

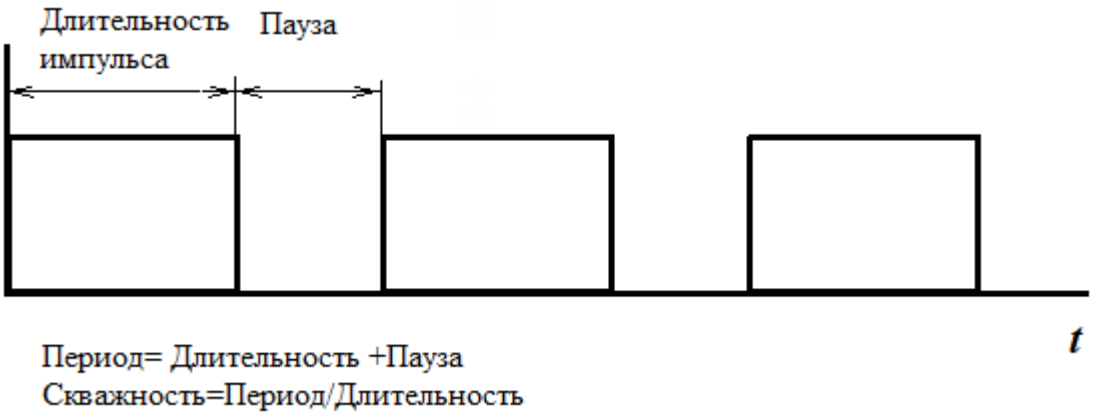


Рисунок 3 – Последовательность прямоугольных импульсов

Реализацию данной последовательности можно выполнить на базе листинга 1 путем задания задержки для установленного в единицу состояния выхода равным *Длительности импульса,* а задержку для паузы формировать задержкой состояния выхода при нулевом значении (LOW) равным величине *Паузы.* Недостатком данного подхода является тот факт, что никаких других действий данная программа уже выполнить не сможет, иначе время длительности и паузы будет меняться случайным образом.

Более предпочтительным является вариант на основе функции `millis()`, при его использовании в промежутке между операторами проверки условия `if( now_time >= next_time_1 )` можно выполнять любые другие действия.

Листинг программы, в которой длительность импульса составляет 500 миллисекунд, а длительность паузы — 150 мс представлен ниже.

Листинг 3

```
const int Pin_Out = 3;
unsigned long next_time_1; // текущее время длительности импульса
unsigned long next_time_2; // текущее время паузы
```

```
int timeout_1 = 500; // длительность импульса
int timeout_2 = 150; // длительность паузы
int out_state = 1; // начальное состояние выхода: включен
void setup() {
   pinMode(Pin_Out, OUTPUT);
   digitalWrite(Pin_Out, out_state); // ставим выход в нулевое состояние
   next_time_1 = millis() + timeout_1; // вычисляем время следующей
                                              // установки выхода в 1
   next_time_2 = millis() + timeout_2; // время выключения паузы
}
void loop() {
   int now_time = millis(); // текущее время
if(out_state==1) {
   if( now_time >= next_time_1 ){ // если текущее время превысило
                                          // намеченное время, то
      out_state = 0;     // меняем состояние на выключено
      digitalWrite(Pin_Out, out_state);
      next_time_2=now_time + timeout_2; // вычисляем время конца паузы
       }
    }
 if(out_state==0) {
   if( now_time >= next_time_2 ){ // если текущее время превысило
                                          // намеченное время завершения паузы, то
      out_state = 1;     // меняем состояние на выключено
      digitalWrite(Pin_Out, out_state);
      next_time_1=now_time + timeout_1; // вычисляем время конца импульса
   }
}
// Здесь могут быть любые другие операторы
}
```

Загружаем программу, включаем питание. Теперь светодиод мигают с разными интервалами включения и выключения

Задание для самостоятельной работы

Выполнить все примеры программ. Внести изменения в листинги программ в соответствии с указанием преподавателя. Скриншоты экрана с программным кодом и результатами работы представить в отчете. Сделать выводы о технологии программирования с использованием Arduino IDE.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №4
ОБРАБОТКА И ФОРМИРОВАНИЕ АНАЛОГОВЫХ СИГНАЛОВ С ИСПОЛЬЗОВАНИЕМ МИКРОКОНТРОЛЛЕРА

Цели занятия:

1. Изучить особенности программирования аналого-цифровых преобразователей и аналоговых компараторов с использованием платформы **Arduino Uno** и инструментальной среды **Arduino IDE**
2. Приобрести навыки работы с аналоговыми сигналами в инструментальных средах **Arduino IDE** и **Atmel Studio.**

Учебные вопросы:

1. Разработка программ с использованием аналого-цифрового преобразователя
2. Настройка аналогового компаратора микроконтроллера Atmega328

1 Разработка программ с использованием аналого-цифрового преобразователя

Программирование АЦП в режиме сценария подробно расписано на сайте http://arduino.ru/Reference/Analogreference

Основные библиотечные функции:

analogRead(pin) -- Функция считывает значение с указанного аналогового входа. Возвращает: int (0 to 1023).

Считывание значение с аналогового входа занимает примерно 100 микросекунд (0.0001 сек), т.е. максимальная частота считывания приблизительно 10,000 раз в секунду.

analogReference(type) -- определяет опорное напряжение относительно которого происходят аналоговые измерения. Type принимает одно из следующих значений:

DEFAULT: стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3.3 В (на платформах с напряжением питания 3.3 В).

INTERNAL: встроенное опорное напряжение 1.1 В на микроконтроллерах ATmega168 и ATmega328, и 2.56 В на ATmega8.

EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF (рекомендуется подключать к выводу AREF через резистор 5 кОм).

Опорное напряжение можно установить, используя функцию analogReference или при помощи битов REFT[1:0] в регистре ADMUX.

Используйте внутреннее опорное напряжение 1.1 В для точных измерений внешних напряжений:

analogReference(INTERNAL1V1); // выбираем внутреннее опорное напряжение 1.1В

Опорное напряжение 1.1 В более стабильно и не зависит от изменения напряжения питания или температуры. Таким образом, можно производить измерения абсолютных значений.

Режимы работы

1) Разовая выборка - это на самом деле то, что Arduino делает при вызове функции analogRead().

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  int val = analogRead(A0);
  Serial.println(val);
  delay(1000);
}
```

Из-за шумов 2 младших бита обычно отбрасывают:

int val = analogRead(A0)>>2;

в итоге получаем значения от 0 до 255, т.е. точность 5/255=19.6мВ.

2) Непрерывная выборка:

Хорошей идеей при непрерывной выборке сигнала является использование прерываний.

Микроконтроллеры ATMega328 и ATMega2560 могут быть переведены в режим непрерывной выборки (free running mode). В этом режиме АЦП запускается автоматически после завершения предыдущей обработки.

Для включения режима непрерывной выборки необходимо установить три регистра: ADMUX, ADCSRA и ADCSRB.

Подключение потенциометра ко входам АЦП. Потенциометры действуют как регулируемые делители напряжения. Они могут быть разных размеров и форм, но всегда имеют три вывода.

Подключаем один крайний вывод потенциометра к земле, а другой к шине 5 В. Потенциометры симметричны, так что не имеет значения, с какой стороны вы подключите шину питания, а с какой землю. Средний вывод соединяем с одним из аналоговых контактов на плате Ардуино как показано на рисунке 1.. При повороте ручки потенциометра аналоговый входной сигнал будет плавно меняться от 0 до 5 В.

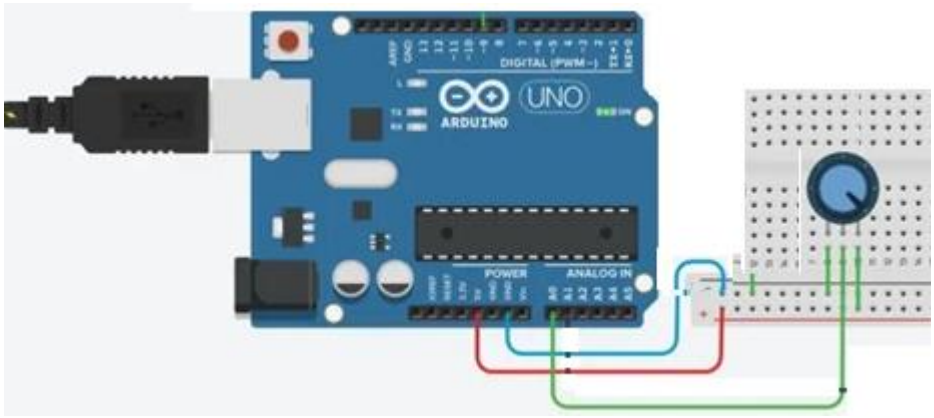


Рисунок 1 – Подключение потенциометра к АЦП

Для того, чтобы считать сигнал от датчика в программу, нам понадобится функция **analogRead()**. Она принимает номер порта в качестве аргумента. А пины, которые можно использовать для аналогового входа, помечены на плате Ардуино как **ANALOG IN**.

Напряжение, поданное на аналоговый вход, обычно от 0 до 5 вольт будет преобразовано в значение от 0 до 1023, с шагом квантования (разрешением) 0.0049 Вольт. Разброс напряжение и шаг может быть изменен функцией **analogReference()**.

Считывание значение с аналогового входа занимает около 100 микросекунд (0.0001 сек), Значит, максимальная частота считывания приблизительно 10000 раз в секунду.

Программу для преобразования аналогового напряжения с выхода потенциометра в двоичный код можно реализовать на базе уже имеющихся в библиотеке Arduino IDE примеров. Например, откроем программу **AnalogInOutSerial** из меню **File — Examples — Analog**. Добавим константу и переменную для подключения потенциометра и получаемого значения от него.

```
const int analogInPin1 = A0;
int sensorValue1 = 0;
float outputValue = 0;
```

В функции **setup()** откроем последовательный порт. А в функции **loop()** добавим считывание сигнала с подключенного потенциометра.

```
void loop() {
// read the analog in value:
sensorValue1 = analogRead(analogInPin1);
```

Когда мы используем данные от аналоговых датчиков и выводим напряжение на пин в зависимости от этих датчиков, необходимо позаботиться о преобразовании чисел из одного диапазона в другой.

Данные с потенциометра будут приходить в виде десятичных значений от 0 до 1023, для преобразования их в диапазон от 0 до 255можно использовать функцию **map()**.

Она пропорционально переносит значение из текущего диапазона значений в новый диапазон. Например,

outputValue = map((sensorValue1, 0, 1023, 0, 255);

В данном случае, значение с подключенного потенциометра формируется АЦП в диапазоне от 0 до 1023. Функция преобразует результат в новый диапазон от 0 до 255.

Так же можно использовать функцию **constrain()**. Она принимает три параметра и выводит значения границ диапазона, если проверяемое значение выходит за границы этого диапазона. Например,

```
sensVal = constrain(0, 5, 150); // 5
sensVal = constrain(200, 5, 150); // 150
sensVal = constrain(95, 5, 150); // 95
// ограничиваем значения sensVal диапазоном от 5 до 150
```

Полный текст программы

```
int analogInPin1 = A0; // Analog input pin
int sensorValue1 = 0;          // value read from the pot
float outputValue = 0;
void setup() {
      // initialize serial communications at 9600 bps:
Serial.begin(9600);
}
void loop() {     // read the analog in value:
sensorValue1 = analogRead(analogInPin1);
      //преобразование результата в напряжение коэффициент 0.0048
      //получен делением 5/1024
outputValue=0.0048*sensorValue1;
      // print the results to the Serial Monitor:
Serial.print("sensor1 = ");
Serial.print(sensorValue1);
Serial.print("\t output = ");
Serial.println(outputValue);
      // wait 0,5 seconds before the next loop for the analog-to-digital
      // converter to settle after the last reading:
delay(500);
}
```

При программировании в среде Atmel Studio настройка регистров производится непосредственно в пользовательской программе. Ниже приведен пример программы измерения напряжения с выводом результата на светодиодный индикатор в виде двоичного кода.

```
/*** Использование АЦП. Светодиодная шкала ***/

#include <avr/io.h>
#include <util/delay.h>

int main (void)
{
DDRD = 0xFF;
PORTD = 0x00;

/*** Настройка АЦП ***/
ADCSRA |= (1 << ADEN) // Включение АЦП
        |(1 << ADPS1)|(1 << ADPS0); // предделитель преобразователя на 8
ADMUX |= (0 << REFS1)|(0 << REFS0) // внешний ИОН
       |(0 << MUX0)|(0 << MUX1)|(0 << MUX2)|(0 << MUX3); // вход PC0

while(1)
{
unsigned int u;
ADCSRA |= (1 << ADSC);                  // Начинаем преобразование
while ((ADCSRA&(1 << ADIF))== 0); // Ждем флага окончания
                                        //  преобразования
 u = (ADCL|ADCH << 8);                  // Считываем  ADC

if (u > 128) // 0.625V
PORTD = 0b00000001;
else
PORTD = 0b00000000;

if (u > 256) // 1.25V
   PORTD = 0b00000011;
if (u > 384) // 1.875V
   PORTD = 0b00000111;
if (u > 512) // 2.5V
   PORTD = 0b00001111;
if (u > 640) // 3.125V
   PORTD = 0b00011111;
if (u > 768) // 3.75V
   PORTD = 0b00111111;
if (u > 896) // 4.375V
   PORTD = 0b01111111;
```

```
if (u > 1020) // 5V
    PORTD = 0b11111111;
_delay_ms(30);
}
}
```

2 Настройка аналогового компаратора микроконтроллера Atmega328

Аналоговый компаратор предназначен для сравнения двух напряжений, поступающих на соответствующие входы. На отладочной плате Arduino Uno в качестве входов аналогового компаратора используются входы AIN0 и AIN1 (D6 и D7). Ко входу AIN0 подключается внешний или внутренний опорный источник напряжения, а на вход AIN1 (инверсный вход) подается напряжения для сравнения с напряжением AIN0 (прямой вход).

Работу компаратора рассмотрим на следующем примере (рисунок 2).

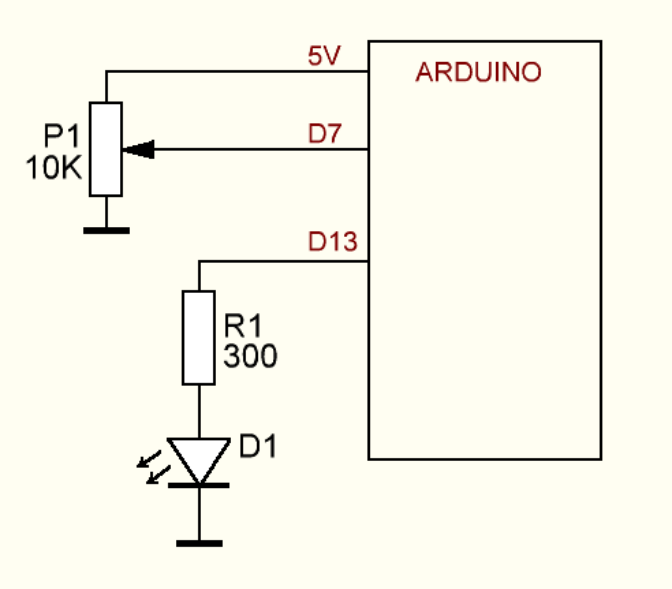


Рисунок 2 – Пример подключения внешнего сигнала

```
void setup() {
  ACSR |= (1 << ACBG);  // подключаем внутренний ИОН ко входу AIN0 D6)
  DDRB |= (1 << 5);     // выход PB5 как выход (D13}
}
void loop() {
  while ((ACSR & 0x20) == 0x20) { // если бит ACO регистра ACSR равен 1, то исполнять код в цикле
    PORTB |= (1 << 5);  // PB5 HIGH
  }
  PORTB &= ~(1 << 5);   // PB5 LOW
}
```

Другой пример программной реализации компаратора

```
// так как компаратор программный нужно задать некоторый предел точности
int delta = 5;
void setup(){
      pinMode(1, OUTPUT);
      pinMode(2, OUTPUT);
      }
      void loop(){
      int u1 = analogRead(A1);
      int u2 = analogRead(A0);
     if (u1-delta > u2)
            {digitalWrite(1, HIGH);
            digitalWrite(2, LOW);
            }
     else if(u2-delta > u1)
            {digitalWrite(1, HIGH);
            digitalWrite(2, LOW);
            }else
            {digitalWrite(1, LOW);
            digitalWrite(2, LOW);
            }
      }
```

Соберите схему с учетом подключения двух потенциометров к аналоговым входам A0 иA1, а также двух светодиодов к цифровым выходам 1 и 2.

Задание для самостоятельной работы

Выполнить все примеры программ. Скриншоты экрана с программным кодом и результатами работы представить в отчете. Сделать выводы о целесообразности разных подходов при обработке аналоговых сигналов.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №5
РАЗРАБОТКА ПРОГРАММ ИЗМЕРЕНИЯ ТЕМПЕРАТУРЫ С ИСПОЛЬЗОВАНИЕМ АНАЛОГОВЫХ И ЦИФРОВЫХ ДАТЧИКОВ

Цели занятия:

Изучить особенности разработки программ с использованием цифровых датчиков температуры. Приобрести навыки работы с последовательными протоколами 1-Ware и I2C в инструментальных средах Arduino IDE и Atmel Studio.

Учебные вопросы:

1. Измерение температуры с использованием датчика DS1820 и протокола обмена 1-Ware
2. Измерение температуры с использованием датчика Milex и протокола I2C

1 Измерение температуры с использованием датчика DS1820 и протокола обмена 1-Ware

DS18B20 цифровой термометр с программируемым разрешением, от 9 до 12–bit, которое может сохраняться в EEPROM памяти прибора. DS18B20 обменивается данными по 1-Wire шине и при этом может быть как единственным устройством на линии так и работать в группе. Все процессы на шине управляются центральным микропроцессором.

Диапазон измерений от –55°C до +125°C и точностью 0.5°C в диапазоне от –10°C до +85°C. В дополнение, DS18B20 может питаться напряжением линии данных (“parasite power”), при отсутствии внешнего источника напряжения.

Каждый DS18B20 имеет уникальный 64-битный последовательный код, который позволяет, общаться с группой датчиков DS18B20 установленных на одной шине. Такой принцип позволяет использовать один микропроцессор, чтобы контролировать множество датчиков DS18B20, распределенных по большому участку. Приложения, которые могут извлечь выгоду из этой особенности, включают системы контроля температуры в зданиях, и оборудовании или машинах, а так же контроль и управление температурными процессами.

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+85.0°C*	0000 0000 1010 1010	00AAh
+25.0°C	0000 0000 0011 0010	0032h
+0.5°C	0000 0000 0000 0001	0001h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1111	FFFFh
-25.0°C	1111 1111 1100 1110	FFCEh
-55.0°C	1111 1111 1001 0010	FF92h

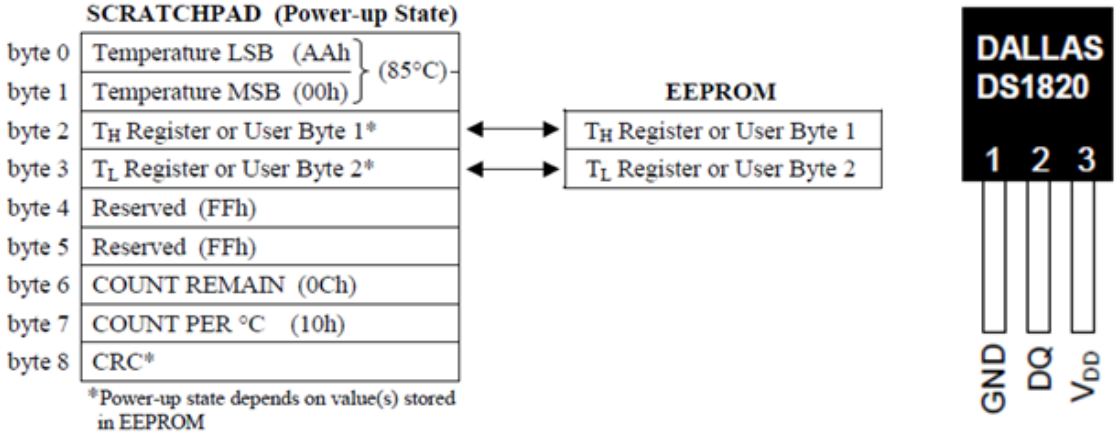


Рисунок 1 - Цифровой температурный датчик DS18B20

DS18B20 – это целое микропроцессорное устройство , которое может хранить значение измерений, сигнализировать о выходе температуры за установленные границы (сами границы мы можем устанавливать и менять), менять точность измерений, способ взаимодействия с контроллером и многое другое. Все это в очень небольшом корпусе, который, к тому же, доступен в водонепроницаемом исполнении.
Микросхема имеет три выхода, из которых для данных используется только один, два остальных – это земля и питание. Число проводов можно сократить до двух, если использовать схему с паразитным питанием и соединить Vdd с землей. К одному проводу с данными можно подключить сразу несколько датчиков DS18B20 и в плате Ардуино будет задействован всего один пин.

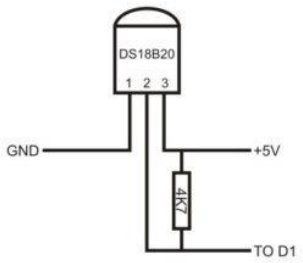


Рисунок 2 – Подключение датчика

- Погрешность измерения не больше 0,5 С (для температур от -10С до +85С), что позволяет точно определить значение температуры. Не требуется дополнительная калибровка.

- Температурный диапазон измерений лежит в пределах от -55 С до +125 С.
- Датчик питается напряжением от 3,3В до 5В.
- Можно программно задать максимальную разрешающую способность до 0,0625С, наибольшее разрешение 12 бит.
- Присутствует функция тревожного сигнала.
- Каждое устройство обладает своим уникальным серийным кодом.
- Не требуются дополнительные внешние элементы.
- Можно подключить сразу до 127 датчиков к одной линии связи.
- Информация передается по протоколу 1-Wire.
- Для присоединения к микроконтроллеру нужны только 3 провода.
- Существует так называемый режим паразитного питания – в нем происходит питание напрямую от линии связи. Для подключения в этом случае нужны только 2 провода. Важно, что в этом режиме не гарантируется корректная работа при температурах выше 100С. Режим паразитного питания удобно обычно применяется для приложений с удаленным температурным датчиком.

1.1 Программирование процесса измерения температуры в среде Arduino Uno

Подключение DS18B20 к Arduino. DS18B20 является цифровым датчиком. Цифровые датчики передают значение измеряемой температуры в виде определенного двоичного кода, который поступает на цифровые или аналоговые пины ардуино и затем декодируется. Коды могут быть самыми разными, ds18b20 работает по протоколу данных 1-Wire. Мы не будем вдаваться в подробности этого цифрового протокола, укажем лишь необходимый минимум для понимания принципов взаимодействия.

Обмен информацией в 1-Wire происходит благодаря следующим операциям:

- Инициализация – определение последовательности сигналов, с которых начинается измерение и другие операции. Ведущее устройство подает импульс сброса, после этого датчик должен подать импульс присутствия, сообщающий о готовности к выполнению операции.
- Запись данных – происходит передача байта данных в датчик.
- Чтение данных – происходит прием байта из датчика.

Для работы с датчиком нам понадобится программное обеспечение:

- Arduino IDE;
- Библиотека OneWire, если используется несколько датчиков на шине, можно использовать библиотеку DallasTemperature. Она будет работать поверх OneWire.

Из оборудования понадобятся:

- Один или несколько датчиков DS18B20;
- Отладочная плата Ардуино уно;
- Коннекторы;

- Резистор на 4,7 кОм (в случае подключения одного датчика пойдет резистор номиналом от 4 до 10К);
- Монтажная плата;
- USB-кабель для подключения к компьютеру.

К плате Arduino UNO датчик подключается просто: GND с термодатчика присоединяется к GND Ардуино, Vdd подключается к 5V, Data – к любому цифровому пину.

Простейшая схема подключения цифрового датчика DS18B20 представлена на рисунке 3.

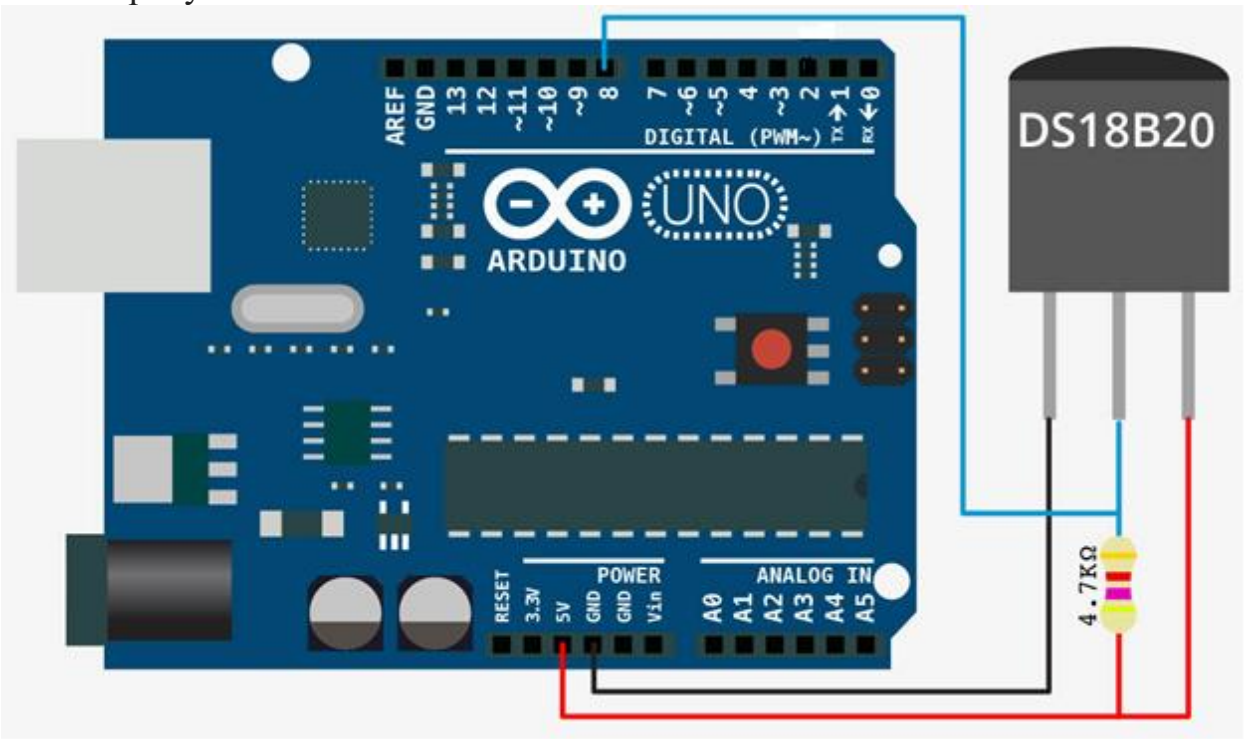


Рисунок 3 – Подключение датчика температуры к плате Arduino Uno

В режиме паразитного питания контакт Vdd с датчика подключается к GND на Ардуино – в этом случае пригодятся только два провода. Работу в паразитном режиме лучше не использовать без необходимости, так как могут ухудшиться быстродействие и стабильность.

Алгоритм получения информации о температуре в скетче состоит из следующих этапов:

- Определение адреса датчика, проверка его подключения.
- На датчик подается команда с требованием прочитать температуру и выложить измеренное значение в регистр. Процедура происходит дольше остальных, на нее необходимо примерно 750 мс.
- Подается команда на чтение информации из регистра и отправка полученного значения в «монитор порта»,
- Если требуется, то производится конвертация в градусы Цельсия/Фаренгейта.

Самый простой скетч для работы с цифровым датчиком выглядит следующим образом. (в скетче мы используем библиотеку OneWire, о которой поговорим подробнее чуть позже).

```
#include <OneWire.h>
/*
* Описание взаимодействия с цифровым датчиком ds18b20
* Подключение ds18b20 к ардуино через пин 8
*/
OneWire ds(8); // Создаем объект OneWire для шины 1-Wire, с помощью которого будет осуществляться работа с датчиком

void setup(){
  Serial.begin(9600);
}

void loop(){
  // Определяем температуру от датчика DS18b20
  byte data[2]; // Место для значения температуры

  ds.reset(); // Начинаем взаимодействие со сброса всех предыдущих команд и параметров
  ds.write(0xCC); // Даем датчику DS18b20 команду пропустить поиск по адресу. В нашем случае только одно устройство
  ds.write(0x44); // Даем датчику DS18b20 команду измерить температуру. Само значение температуры мы еще не получаем - датчик его положит во внутреннюю память

  delay(1000); // Микросхема измеряет температуру, а мы ждем.

  ds.reset(); // Теперь готовимся получить значение измеренной температуры
  ds.write(0xCC);
  ds.write(0xBE); // Просим передать нам значение регистров со значением температуры

  // Получаем и считываем ответ
  data[0] = ds.read(); // Читаем младший байт значения температуры
  data[1] = ds.read(); // А теперь старший

  // Формируем итоговое значение:
  //    - сперва "склеиваем" значение,
  //    - затем умножаем его на коэффициент, соответсвующий разрешающей способности (для 12 бит по умолчанию - это 0,0625)
  float temperature =  ((data[1] << 8) | data[0]) * 0.0625;

  // Выводим полученное значение температуры в монитор порта
  Serial.println(temperature);
```

```
}
```

Библиотека OneWire для работы с DS18B20

Основные команды библиотеки OneWire:

- search(addressArray) – ищет температурный датчик, при нахождении в массив addressArray записывается его код, в ином случае – false.
- reset_search() – производится поиск на первом приборе.
- reset() – выполнение сброса шины перед тем, как связаться с устройством.
- select(addressArray) – выбирается устройство после операции сброса, записывается его ROM код.
- write(byte) – производится запись байта информации на устройство.
- write(byte, 1) – аналогично write(byte), но в режиме паразитного питания.
- read() – чтение байта информации с устройства.
- crc8(dataArray, length) – вычисление CRC кода. dataArray – выбранный массив, length – длина кода.

Важно правильно настроить режим питания в скетче. Для паразитного питания в строке 65 нужно записать ds.write(0x44, 1);. Для внешнего питания в строке 65 должно быть записано ds.write(0x44).

Write позволяет передать команду на термодатчик. Основные команды, подаваемые в виде битов:

- 0x44 – измерить температуру, записать полученное значение в SRAM.
- 0x4E – запись 3 байта в третий, четвертый и пятый байты SRAM.
- 0xBE – последовательное считывание 9 байт SRAM.
- 0x48 – копирование третьего и четвертого байтов SRAM в EEPROM.
- 0xB8 – копирование информации из EEPROM в третий и четвертый байты SRAM.
- 0xB4 – возвращает тип питания (0 – паразитное, 1 – внешнее).

Подключение нескольких датчиков температуры DS18B20 к Ардуино

Все датчики DS18B20 подключаются параллельно, для них всех достаточно одного резистора. При помощи библиотеки OneWire можно одновременно считать все данные со всех датчиков. Если количество подключаемых датчиков более 10, нужно подобрать резистор с сопротивлением не более 1,6 кОм. Также для более точного измерения температуры нужно поставить дополнительный резистор на 100…120 Ом между выходом data на плате Ардуино и data на каждом датчике. Узнать, с какого датчика получено то или иное значение, можно с помощью уникального серийного 64-битного кода, который будет выдан в результате выполнения программы.

Для подключения температурных датчиков в нормальном режиме нужно использовать схему, представленную на рисунке 4.

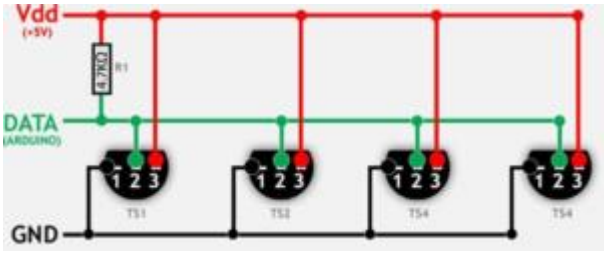


Рисунок 4

В режиме паразитного питания схема выглядит иначе (рисунок 5). Контакт Vdd практически не задействован, питание идет через выход data.

Рисунок 5

1.2 Программирование процесса измерения температуры в среде Atmel Studio

Программа, листинг которой представлен ниже , содержит измерение температуры и отображение температуры на четырех символьном семисегментным индикаторе. Периодичность опроса датчика задается таймером. Для индикатора выбран режим динамической индикации.

Реализация обмена с датчиком температуры по интерфейсу One-Ware с использованием PC0 для ввода данных и PC1 для вывода.

Управление выбором цифры семисегментного индикатора осуществляется через порт B выводы PB1-PB4. Управление сегментами через порт D.

```
/*
 * LessonTerm_LCD.c
*/

#define F_CPU 16000000L
#include <avr/io.h>
#include <math.h>
#include <util/delay.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <stdlib.h>

volatile int display = 0;
int Tcount;
int Time;
uint8_t simv;
```

```
uint8_t count=1;
//-------------------------0-----1-----2-----3-----4-----5-----6-----7-----8------9
unsigned char SEGMENTE[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
unsigned char num_posicii[]={0x10, 0x08, 0x04, 0x02 };
unsigned char segcounter = 0;
unsigned char data[4];
unsigned char dat;

ISR( TIMER1_OVF_vect )
{
   TCNT1 = 65500;; //выставляем начальное значение TCNT1 для периода 0,05 с



   Tcount++;

   dat=0;
   dat=display % 10000 / 1000;
   if(dat==0)data[0]=~SEGMENTE[0]; else data[0]=~SEGMENTE[dat];
   dat=display % 1000 / 100; if(dat==0)data[1]=~SEGMENTE[0]; else data[1]=~SEGMENTE[dat];
   dat=display % 100 / 10; if(dat==0)data[2]=~SEGMENTE[0]; else data[2]=~SEGMENTE[dat];
   dat=display % 10; if(dat==0)data[3]=~SEGMENTE[0]; else data[3]=~SEGMENTE[dat];

   PORTB =num_posicii[segcounter]; //Выбираем  индикатор
   PORTD = data[segcounter];
   segcounter++;
   if(segcounter> 3) segcounter = 0;
}

// Датчик температуры

/* Определяем куда подключен датчик */
#define THERM_PORT PORTC
#define THERM_DDR DDRC
#define THERM_PIN PINC
#define THERM_DQ PC1
#define THERM_DQ_IN PC0
/* Макросы для  изменения режима ввода/вывода */
#define THERM_LOW() THERM_PORT&=~(1<<THERM_DQ)
#define THERM_HIGH() THERM_PORT|=(1<<THERM_DQ)
```

```
// сброс датчика
uint8_t therm_reset(void){
   uint8_t i;
   // опускаем ногу вниз на 480uS
         THERM_LOW();
   _delay_us(500);
         THERM_HIGH();
   // подымаем линию на 60uS
   _delay_us(60);
   // получаем значение на линии в период 480uS
   i=(THERM_PIN & (1<<THERM_DQ_IN));
   _delay_us(420);
   // возвращаем значение (0=OK, 1=датчик не найден)
   return i;
}


// функция отправки бита
void therm_write_bit(uint8_t bit){
   // опускаем на 1uS
   THERM_LOW();
   _delay_us(5);
   // если хотим отправить 1, поднимаем линию (если нет, оставляем как есть)
   if(bit)  THERM_HIGH();
   // ждем 38uS и поднимаем линию
   _delay_us(38);
    THERM_HIGH();
}

// чтение бита
uint8_t therm_read_bit(void){
   uint8_t bit=0;
   // опускаем на 1uS
   THERM_LOW();
   _delay_us(2);
   THERM_HIGH();
   // поднимаем на 11uS
   _delay_us(11);
   // читаем состояние
   if(THERM_PIN&(1<<THERM_DQ_IN)) bit=1;
   // ждем 45 мкс и возвращаем значение
   _delay_us(45);
```

```
    return bit;
}

// читаем байт
uint8_t therm_read_byte(void){
    uint8_t i=8, n=0;
    while(i--){
            // сдвигаем вправо на 1 и сохраняем следующее значение
            n>>=1;
            n|=(therm_read_bit()<<7);
    }
    return n;
}

// отправляем байт
void therm_write_byte(uint8_t byte){
    uint8_t i=8;
    while(i--){
            // отправляем бит и сдвигаем вправо на 1
            therm_write_bit(byte&1);
            byte>>=1;
    }
}


void start_temperature(void){
    therm_reset();
    therm_write_byte(0xcc);
    therm_write_byte(0x44);
}

uint16_t read_temperature(void){
    uint8_t temperature[2];
    int16_t digit;
    therm_reset();
    therm_write_byte(0xcc);
    therm_write_byte(0xbe);
    temperature[0]=therm_read_byte();
    temperature[1]=therm_read_byte();
    therm_reset();
    digit=temperature[0]>>4;
    digit|=(temperature[1]&0xF)<<4;
    _delay_ms(50);
    return(digit);
}
```

```
int main()
{   uint8_t n=0;
   uint8_t Termnew,Term;
   DDRD=0xFF;
   DDRB |= 0x1E;  // настраиваем PB1..PB4 на выход  (управление индикатором)
   DDRB|=(1<<5);

   TCCR1B = (1<<CS12)|(0<<CS11)|(1<<CS10); // настраиваем делитель таймера 1
   TIMSK1 |= (1<<TOIE1); // разрешаем прерывание по переполнению таймера
   TCNT1 = 65500;       // выставляем начальное значение TCNT1

    sei();           // выставляем бит общего разрешения прерываний

   while(1)            // вечный цикл
     {


   // if( Tcount>300 ) {n=n+1;display=n;Tcount=0;}
          if(  Tcount>300  )  {Termnew=read_temperature();  if(abs(Termnew-Term)<5)Term=Termnew;Tcount=0;}
          if(Tcount==0){ start_temperature();}
   _delay_ms(500);
   }
   return 0;
     }
```

Выполните ввод программы в текстовый редактор Atmel Studio. Произведите отладку программы и её компиляцию. Разберите программу на модули работы с датчиком температуры, модуль работы с индикатором и модуль работы с таймером.

2 Измерение температуры с использованием датчика Milex и протокола I2C

Инфракрасные термометры MLX90614 работают на основе явления, называемого излучением черного тела. Все, что находится при температуре выше абсолютного нуля, имеет внутри движущиеся молекулы. Чем выше температура, тем быстрее движутся молекулы. По мере движения молекулы излучают инфракрасное излучение - тип электромагнитного излучения ниже видимого спектра света. По мере того как они нагреваются, они излучают больше

инфракрасного излучения и даже начинают излучать видимый свет. Вот почему нагретый металл может светиться красным или даже белым цветом. Инфракрасные термометры обнаруживают и измеряют это излучение.

Основные характеристики инфракрасного датчика MLX90614:

- Диапазон измеряемой температуры: -40°C ... + 125°C для температуры окружающей среды; 70°C ... + 380°C для температуры объекта;
- Точность измерения - 0,5°C в широком диапазоне температур (0°C ... + 50° C для обоих вариантов);
- Максимальное разрешение измерения температуры 0,02°C;
- Протокол информационного обмена - I2C ;
- Возможность калибровки;
- Расстояние до объекта - 1 см.

Датчик может поставляться на плате или отдельно, контакты для подключения платы и отдельно датчика показаны на рисунке ниже.

Arduino pin:	**MLX90614 pin:**
GND	GND
+3.3v	VCC (+2.8-3.6v)
A5	SCL (I²C Clock)
A4	SDA (I²C Data)

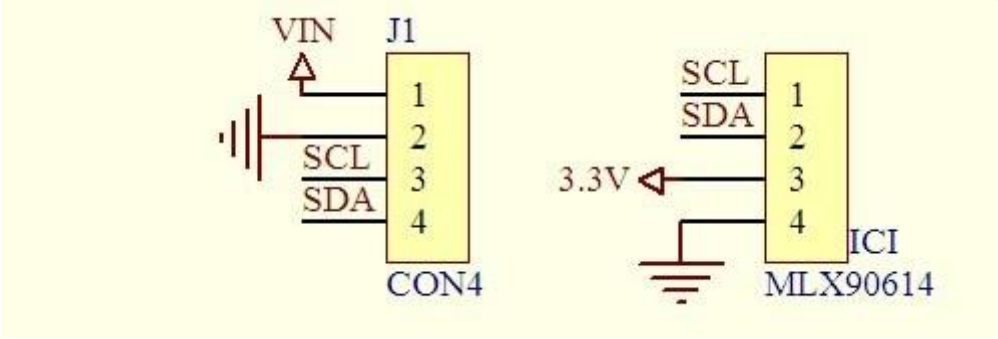


Рисунок – Подключение датчика в виде платы и собственно ИК датчика

Готовый же модуль с обвязкой выглядит вот так

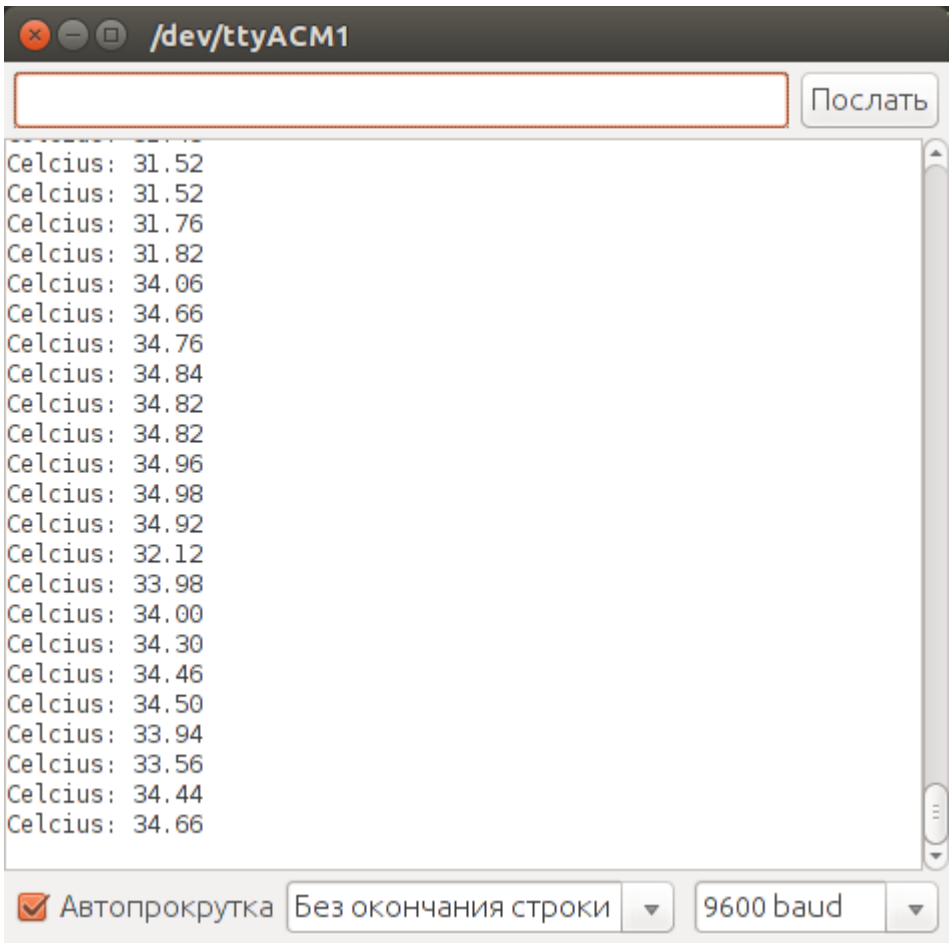
Отдельный датчик можно подключать по следующей схеме

Запишите в Arduino IDE следующий скетч:

```
#include <i2cmaster.h>
void setup(){
  Serial.begin(9600);
  Serial.println("Setup...");

  i2c_init(); //Инициализация шины i2c
  PORTC = (1 << PORTC4) | (1 << PORTC5);//устанавливаем PC4 и
PC5  ( SDA SCL)  в 1
}

void loop(){
  int dev = 0x5A<<1; //0xB4 – адрес датчика
  int data_low = 0;
  int data_high = 0;
  int pec = 0;
  i2c_start_wait(dev+I2C_WRITE);

 i2c_write(0x07);

  // read
  i2c_rep_start(dev+I2C_READ);
  data_low = i2c_readAck(); //Чтение  мл. байта и формирование ack
  data_high = i2c_readAck(); // Чтение  ст. байта и формирование ack
  pec = i2c_readNak();
  i2c_stop();
```

```
//Пересчет температуры в градусы
  double tempFactor = 0.02; // 0.02 degrees per LSB (measurement resolu-
tion of the MLX90614)
  double tempData = 0x0000; // zero out the data
  int frac; // data past the decimal point

  // Удаление старших битов, содержащих информацию об ошибках
  tempData = (double)(((data_high & 0x007F) << 8) + data_low);
  tempData = (tempData * tempFactor)-0.01;

  float celcius = tempData - 273.15;

  Serial.print("Celcius: ");
  Serial.println(celcius);

  delay(1000); // ожидание 1 с и переход на начало цикла
}
```

Записав скетч в Arduino и включив монитор порта, мы можем видеть как меняются показания датчика в зависимости от его направления. На рисунке представлен результат измерения температуры ладони, поднесенной к датчику на 10 см.



В скетче используется библиотека <i2cmaster.h> скачать библиотеку можно с сайта http://arduino-ru.blogspot.com/2008/12/i2cmaster-arduino.html.

В составе стандартной библиотеки Arduino IDE для работы с интерфейсом I2C используется <Wire.h>.

Пример программы с использование м <Wire.h>:

```
#include <Wire.h> // подключаем библиотеку "Wire"
byte val = 0; // значение для передачи потенциометру

void setup() {
  Wire.begin();   // подключаемся к шине I2C как мастер
}

void loop() {
  Wire.beginTransmission(44); // начинаем обмен с устройством с I2C адресом
"44" (0x2C)
  Wire.write(byte(0x00)); // посылаем инструкцию записи в регистр RDAC
  Wire.write(val); // задаём положение 64-позиционного потенциометра
  Wire.endTransmission(); // завершаем I2C передачу

  val++; // инкрементируем val на 1
  if (val == 63) { // по достижении максимума потенциометра
    val = 0; // сбрасываем val
  }
  delay(500);
}
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №6
ИССЛЕДОВАНИЕ ОСОБЕННОСТЕЙ ПРОГРАММИРОВАНИЯ ЖИДКО-КРИСТАЛЛИЧЕСКИХ ИНДИКАТОРОВ

Цели занятия:

- изучить особенности программирования текстовых жидкокристаллических (ЖКИ) экранов с использованием платформы Arduino Uno и инструментальной среды Arduino IDE;

-изучить программирования последовательного интерфейса USART для использования персональной ЭВМ в качестве монитора;

- приобрести навыки разработки программ с использованием средств отображения информации.

Учебные вопросы:

1. Жидкокристаллический двухстройчный дисплей 16x2 MT-16S2
2. Программирование различных задач с отображением результатов работы на ЖКИ

3 Программирование USART для организации связи микроконтроллера с персональным компьютером

1 Жидкокристаллический двухстройчный дисплей MT-16S2

Текстовый экран 16×2 MT-16S2 используется для вывода показаний датчиков, отображения простых меню, подсказок и приветствий в виде двух строк из 16 символов каждая (рисунок 1).

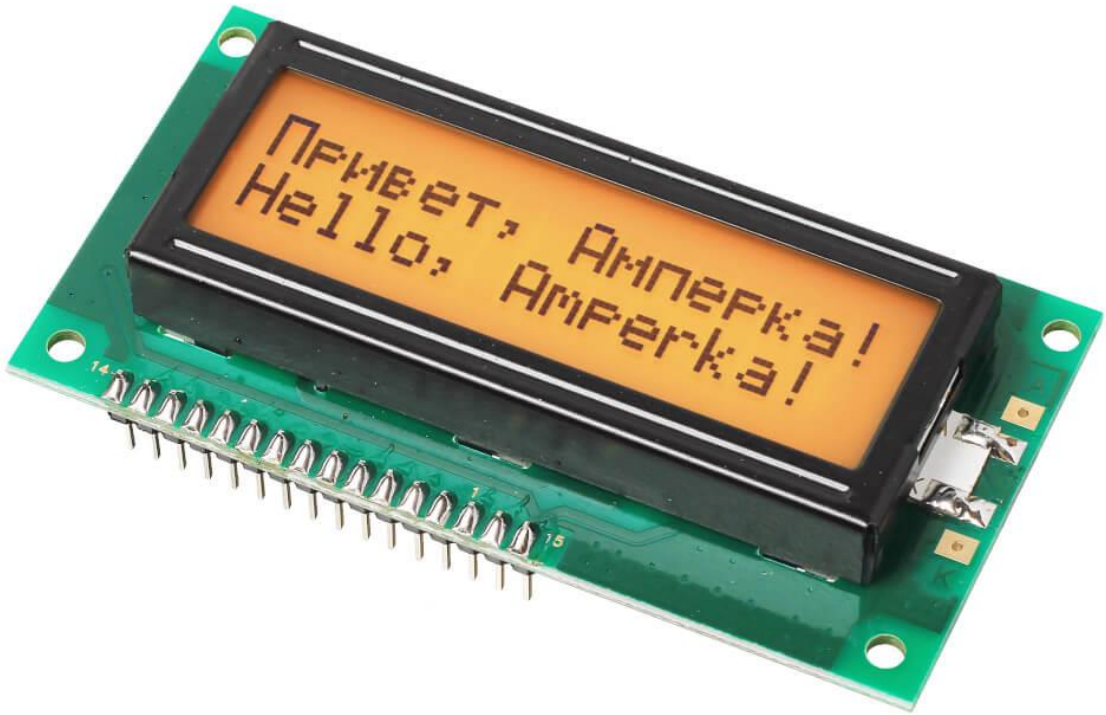


Рисунок 1 – Внешний вид дисплея

В таблице 1 представлено назначение выводов платы индикатора и возможное закрепление выводов платы Arduino Uno при использовании ЖКИ в режиме передачи управляющих кодов по 4 разряда.

Таблица 1 – Назначение выводов индикатора

Вывод	Обозначение		Вывод платы Arduino Uno
1	GND	Общий вывод	GND
2	VCC	Напряжение питания (5 В)	5V
3	VO	Управление контрастностью	GND
4	RS	Выбор регистра (команда или данные)	11
5	R/W	Выбор режима записи или чтения	GND
6	E	Разрешение обращений к индикатору (а также строб данных)	12
7	DB0	Шина данных (8-ми битный режим)(младший бит в 8-ми битном режиме)	—
8	DB1	Шина данных (8-ми битный режим)	—
9	DB2	Шина данных (8-ми битный режим)	—
10	DB3	Шина данных (8-ми битный режим)	—
11	DB4	Шина данных (8-ми и 4-х битные режимы)(младший бит в 4-х битном режиме)	5
12	DB5	Шина данных (8-ми и 4-х битные режимы)	4
13	DB6	Шина данных (8-ми и 4-х битные режимы)	3
14	DB7	Шина данных (8-ми и 4-х битные режимы)	2
15	VCC	Питания подсветки (+)	5V
16	GND	Питания подсветки (–)	GND

На рисунке 2 показано подключение индикатора MT-16S2 к отладочной плате Arduino Uno с использованием макетной платы.

Характеристики дисплея MT-16S2:

Тип дисплея: текстовый

Цвет: монохромный

Технология: LCD (Liquid Crystal Display)

Индикация: 2 строки по 16 символов

Драйвера матрицы: КБ1013ВГ6

Интерфейс: параллельный 4/8 бит

Тип подсветки: LED

Цвет подсветки (зависит от модели): красный / зелёный / синий / чёрный / янтарный

Цвет символов (зависит от модели): зелёный / чёрный / белый

Напряжение питания: 5 В

Максимальный ток потребления: 1 мА

Потребляемый ток подсветки: 100 мА

Напряжение логических уровней: 3,3–5 В

Габариты: 84×44×13 мм

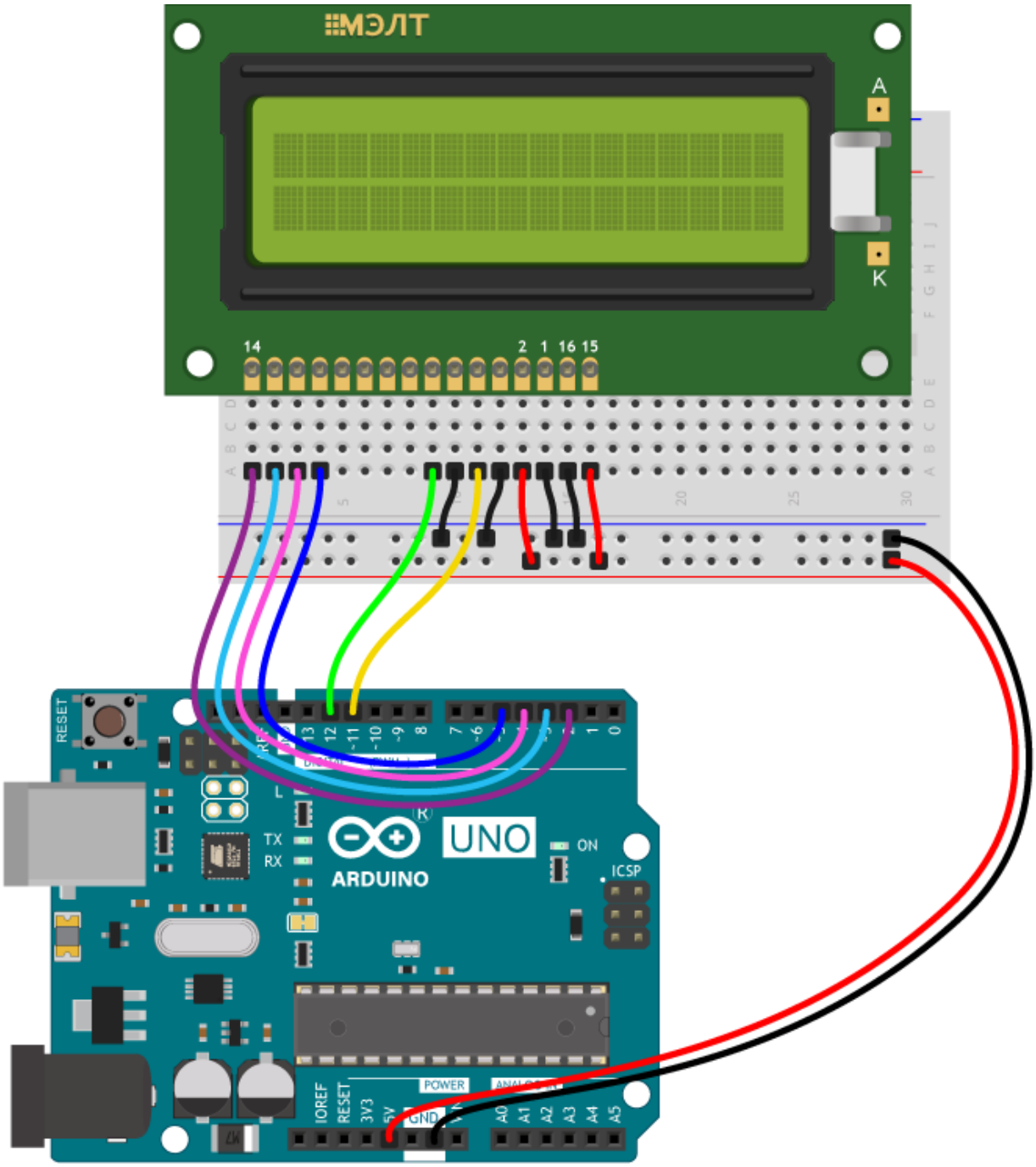


Рисунок 2 – Подключение индикатора к отладочной плате Arduino Uno

Для упрощения работы с LCD-дисплеем используйте встроенную библиотеку Liquid Crystal. В ней вы найдёте примеры кода с подробными комментариями.

2.Программирование различных задач с отображением результатов работы на ЖКИ

Для приобретения навыков работы с ЖКИ выполним несколько типовых задач с выводом неизменяющегося в процессе работы текста (например, заставки в начале работы программы или приветствия), а затем рассмотрим как осуществляется вывод числовых данных.

2.1 Вывод текста

Для вывода текста приветствия, воспользуйтесь следующим листингом 1 программы:

Листинг 1

```
// подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>

// инициализируем объект-экран, передаём использованные
// для подключения контакты на Arduino в порядке:
// RS, E, DB4, DB5, DB6, DB7
LiquidCrystal lcd(11, 12, 5, 4, 3, 2);

void setup() {
  // устанавливаем размер (количество столбцов и строк) экрана
  lcd.begin(16, 2);
  // печатаем первую строку
  lcd.print("Hello world");
  // устанавливаем курсор в колонку 0, строку 1
  // на самом деле это вторая строка, т.к. нумерация начинается с нуля
  lcd.setCursor(0, 1);
  // печатаем вторую строку
  lcd.print("Do It Yourself");
}

void loop() {
}
```

Результаты работы программы показаны на рисунке 3.

Рисунок 3 – Вывод приветствия на экран ЖКИ

Для вывода текста, использующего символы кириллицы можно использовать два способа:

- с помощью встроенной таблицы знакогенератора;
- с помощью библиотеки LiquidCrystalRus.

Таблица знакогенератора. Дисплейный модуль хранит в памяти две страницы знакогенератора (рисунки 4, 5) , которые состоят из различных символов и букв.

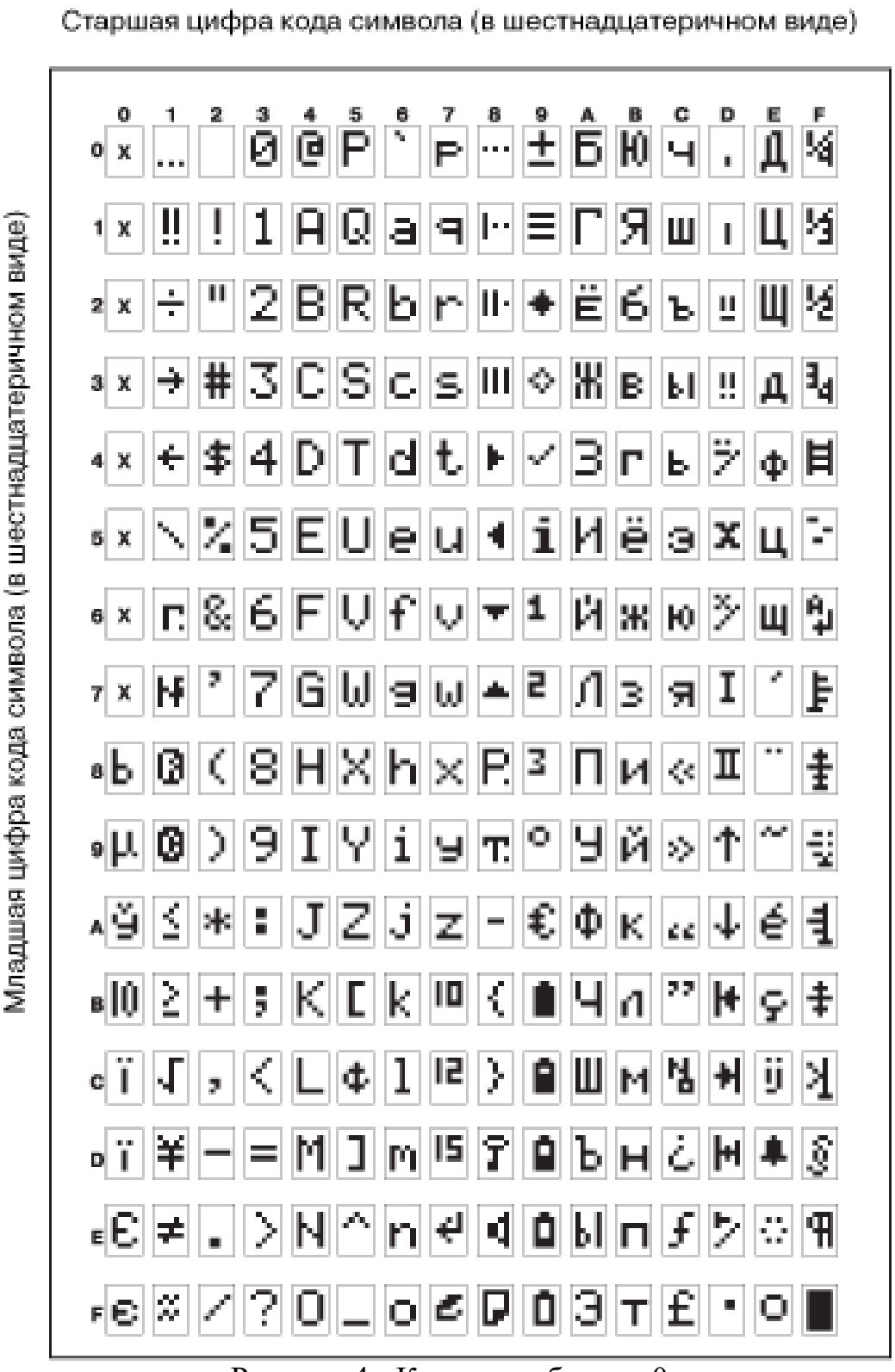


Рисунок 4 - Кодовая таблица «0»

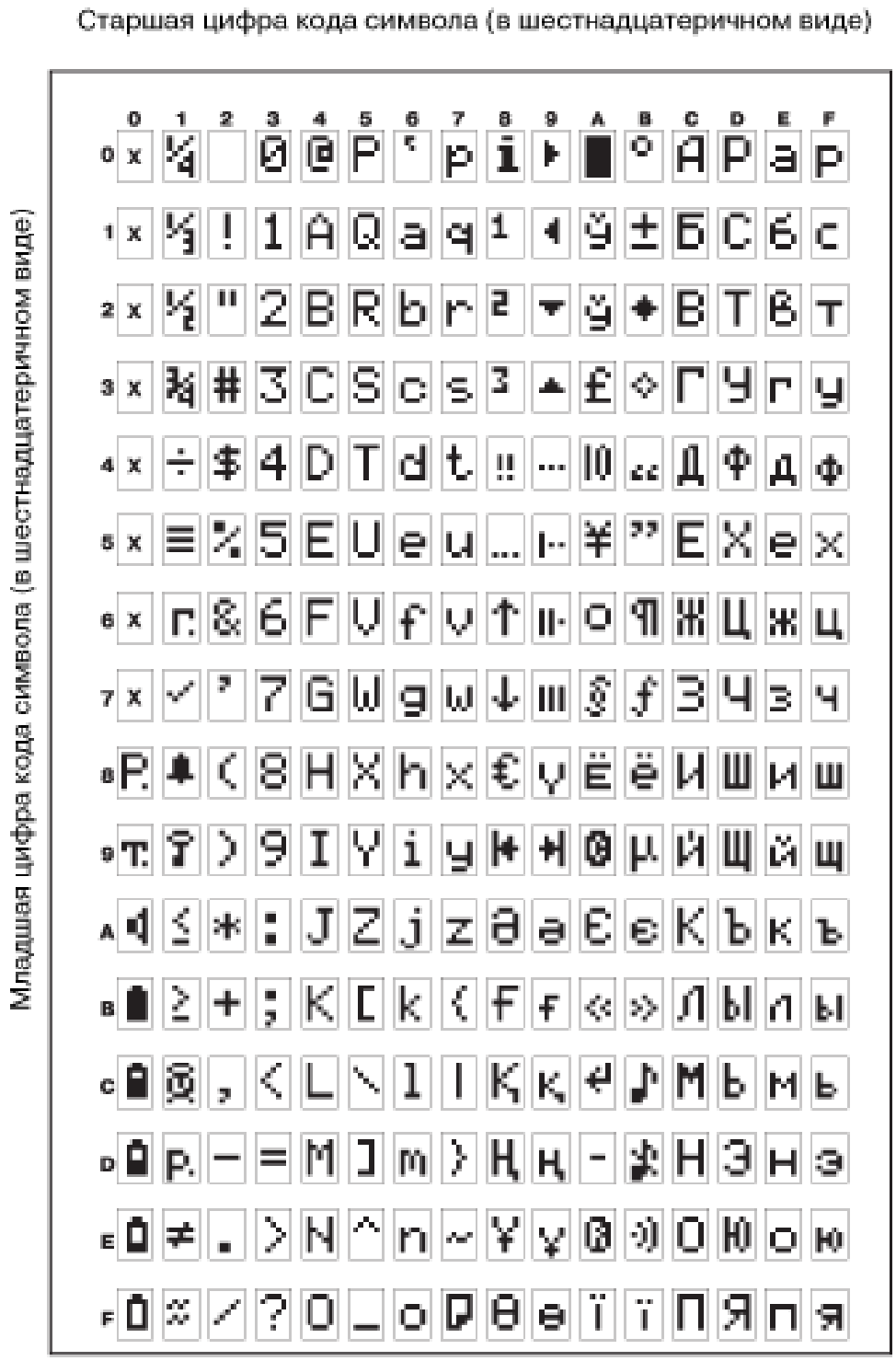


Рисунок 5- Кодовая страница «1»

Для вывода символа на дисплей необходимо передать его номер в шестнадцатеричной системе из таблицы знакогенератора.

Так букве Я соответствует код 0xB1 в шестнадцатеричной системе. Чтобы передать на экран строку «Яndex», необходимо в явном виде с помощью последовательности \x## встроить в строку код символа:

lcd.print("\xB1ndex");

Вы можете смешивать в одной строке обычные символы и явные коды как угодно. Единственный нюанс в том, что после того, как компилятор в строке видит последовательность \x, он считывает за ним все символы, которые могут являться разрядами шестнадцатеричной системы даже если их больше двух. Из-за этого нельзя использовать символы из диапазона 0-9 и A-F следом за двузначным кодом символа, иначе на дисплее отобразится неправильная информация. Чтобы обойти этот момент, можно использовать тот факт, что две записанные рядом строки склеиваются.

Сравните две строки кода для вывода надписи «Яеее»:

```
lcd.print("\xB1eee"); // ошибка
lcd.print("\xB1""eee"); // правильно
```

Используя полученную информацию выведем на дисплей сообщение «Привет, Амперка!»:

Листинг 2:

```
// подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>

// инициализируем объект-экран, передаём использованные
// для подключения контакты на Arduino в порядке:
// RS, E, DB4, DB5, DB6, DB7
LiquidCrystal lcd(11, 12, 5, 4, 3, 2);

void setup() {
  // устанавливаем размер (количество столбцов и строк) экрана
  lcd.begin(16, 2);
  // устанавливаем курсор в колонку 5, строку 0
  // на самом деле это первая строка, т.к. нумерация начинается с нуля
  lcd.setCursor(5, 0);
  // печатаем первую строку
  lcd.print("\xA8""p""\xB8\xB3""e\xBF");
  // устанавливаем курсор в колонку 3, строку 1
  // на самом деле это вторая строка, т.к. нумерация начинается с нуля
  lcd.setCursor(3, 1);
  // печатаем вторую строку
  lcd.print("o\xBF A\xBC\xBE""ep\xBA\xB8");
}

void loop() {
}
```

Результаты работы программы показаны на рисунке 6.

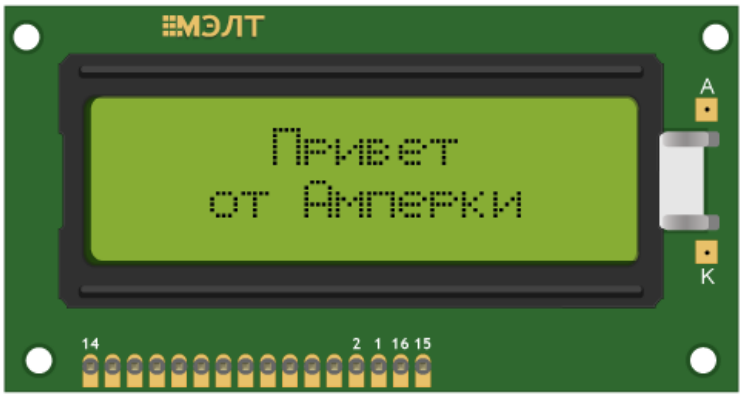


Рисунок 6

Переключение страниц знакогенератора.

Дисплейный модуль хранит в памяти две страницы знакогенератора. По умолчанию установлена нулевая страница. Для переключения между страницами используйте методы:

// переключение с нулевой страницы на первую
command(0x101010);
// переключение с первой страницы на нулевую
command(0x101000);

Дисплей не может одновременно отображать символы разных страниц.

Рассмотрим пример (листинг 3), в котором одна и та же строка будет отображаться по-разному — в зависимости от выбранной страницы (рисунок 7).

Листинг 3:

```
// Подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>

// Инициализируем объект-экран, передаём использованные
// для подключения контакты на Arduino в порядке:
// RS, E, DB4, DB5, DB6, DB7
LiquidCrystal lcd(11, 12, 5, 4, 3, 2);

void setup() {
  // устанавливаем размер (количество столбцов и строк) экрана
  lcd.begin(16, 2);
  // устанавливаем курсор в колонку 5, строку 0
  // на самом деле это первая строка, т.к. нумерация начинается с нуля
  lcd.setCursor(5, 0);
  // печатаем  строку
  lcd.print("\x9b\x9c\x9d\x9e\x9f");
}
void loop() {
  // устанавливаем 0 станицу знакогенератора (стоит по умолчанию)
  lcd.command(0b101000);
  // ждём 1 секунду
  delay(1000);
  // устанавливаем 1 станицу знакогенератора
```

```
  lcd.command(0b101010);
  // ждём 1 секунду
  delay(1000);
}
```

Рисунок 7

Использование библиотеки LiquidCrystalRus

Совсем не обязательно мучатся со знакогенератором, чтобы вывести русский символ. Для решения проблемы скачайте и установите библиотеку LiquidCrystalRus.

Это копия оригинальной библиотеки LiquidCrystal с добавлением русского языка. Добавленный в библиотеку код трансформирует русские символы UTF8 в правильные коды для текстового экрана.

В качестве примера выведем фразу «Привет от Амперки» на дисплей.

```
// подключаем библиотеку LiquidCrystalRus
#include <LiquidCrystalRus.h>

// инициализируем объект-экран, передаём использованные
// для подключения контакты на Arduino в порядке:
// RS, E, DB4, DB5, DB6, DB7
LiquidCrystalRus lcd(11, 12, 5, 4, 3, 2);

void setup() {
  // устанавливаем размер (количество столбцов и строк) экрана
  lcd.begin(16, 2);
  // устанавливаем курсор в колонку 5, строку 0
  // на самом деле это первая строка, т.к. нумерация начинается с нуля
  lcd.setCursor(5, 0);
  // печатаем первую строку
  lcd.print("Привет");
  // устанавливаем курсор в колонку 3, строку 1
  // на самом деле это вторая строка, т.к. нумерация начинается с нуля
  lcd.setCursor(3, 1);
  // печатаем вторую строку
  lcd.print("от Амперки");
}
void loop() {
}
```

Для вывода цифровых данных необходимо выполнить конвертировать двоичный код посимвольно в коды ASCII. Пример листинга подпрограммы конвертирования приведен ниже:

```
void ConvertNum_ASCII(uint32_t Num)
{
int ost,has,q;
uint8_t  mDATA[10]; int c=0;
has=Num;
        for (q=0;q<11;q++) {mDATA[q]=0x30;};
while(has)
{
ost=has %10;
has=has/10;
switch(ost)
{ case 0: mDATA[c]=0x30; break;
  case 1: mDATA[c]=0x31; break;
  case 2: mDATA[c]=0x32; break;
  case 3: mDATA[c]=0x33; break;
  case 4: mDATA[c]=0x34; break;
  case 5: mDATA[c]=0x35; break;
  case 6: mDATA[c]=0x36; break;
  case 7: mDATA[c]=0x37; break;
  case 8: mDATA[c]=0x38; break;
  case 9: mDATA[c]=0x39; break;
}
c++;
}
// UART_putString0 (" = ");
lcd.print("  = ");
for(q=c;q>=0;q--)
  {
//                UART_putChar0 (mDATA[q]);
lcd.print(mDATA[q]);
  }
//         UART_putString0 ("    ");
lcd.print("    ");
};
```

3 Программирование USART для организации связи микроконтроллера с персональным компьютером

Интерфейс USART — последовательный универсальный синхронно-асинхронный приемо-передатчик. Передача данных в USART осуществляется через равные промежутки времени. Этот временной промежуток определяется заданной скоростью USART и указывается в бодах (Для символов, которые могут принимать значения, равные только нулю или единице бод эквивалентен битам в секунду). Существует общепринятый ряд стандартных скоростей:

300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600 бод.

Помимо бит данных USART автоматически вставляет в поток синхронизирующие метки, так называемые стартовый и стоповый биты. При приёме эти лишние биты удаляются. Обычно стартовый и стоповый биты отделяют один байт информации (8 бит), однако встречаются реализации USART, которые позволяют передавать по 5, 6, 7, 8 или 9 бит. Биты, отделённые стартовым и стоповым сигналами, являются минимальной посылкой. USART позволяет вставлять два стоповых бита при передаче для уменьшения вероятности рассинхронизации приёмника и передатчика при плотном трафике. Приёмник игнорирует второй стоповый бит, воспринимая его как короткую паузу на линии.

USART (Universal Synchronous-Asynchronous Receiver-Transmitter)– это последовательная шина с полнодуплексным режимом информационного обмена. Как правило используется для обмена данными между цифровыми устройствами, в том числе между микроконтроллером и персональным компьютером (ПК).

Также USART называют UART (Universal Asynchronous Receiver-Transmitter (универсальный асинхронный приемопередатчик)), как он до какой-то поры и назывался. Просто во второй аббривиатуре отсутствует синхронизация. До появления синхронного UARTа шел обмен только по двум проводам, а потом в данный интерфейс добавили ещё шину синхронизации.

В прямом виде ПК сигналы USART не используют, поэтому для соединения микроконтроллера и ПК используются микросхемы адаптеров, таких как RS232-USART, RS485-USART, USB-USART, CAN-USART или иные.

На рисунке 8 показана функциональная схема USART, организованная в микроконтроллере Atmega328

Как следует из рисунка 8, в USART используются три основных выхода RxD, TxD и XCK.

RxD – это контакт для приема информации.

TxD – контакт для передачи информации.

XCK – контакт синхронизации.

В асинхронном режиме вывод XCK можно не использовать.

Также здесь существует несколько модулей, соответствующий каждый своему выходу, а также ряд управляющих регистров, которые используются для программирования (настройки) USART.

Для реализации интерфейса USART в микроконтроллере Atmega 328 используются многофункциональные выводы порта D (PD0 – RXD, PD1 – TXD) (рисунок 9).

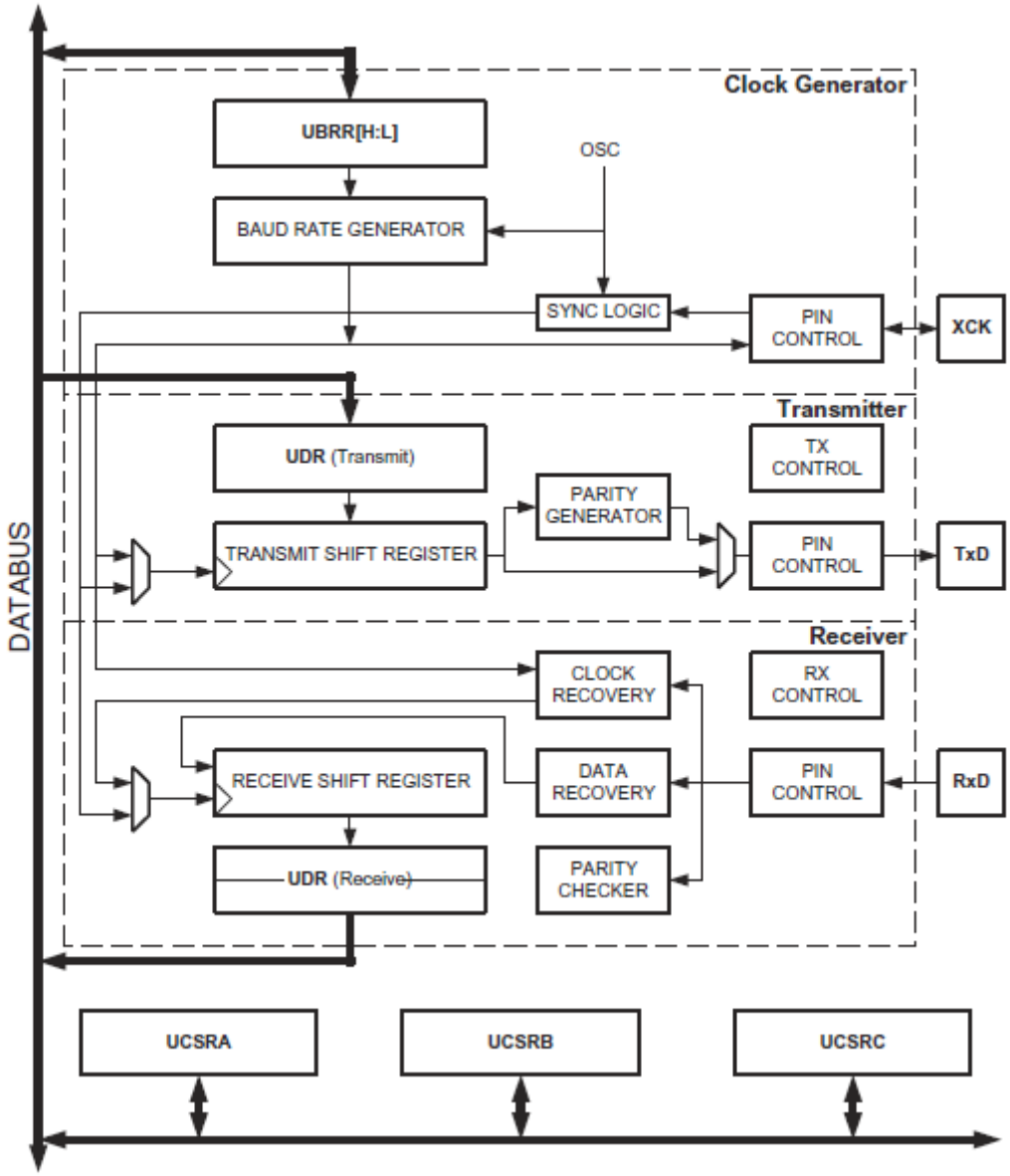


Рисунок 8 – Функциональная схема USART

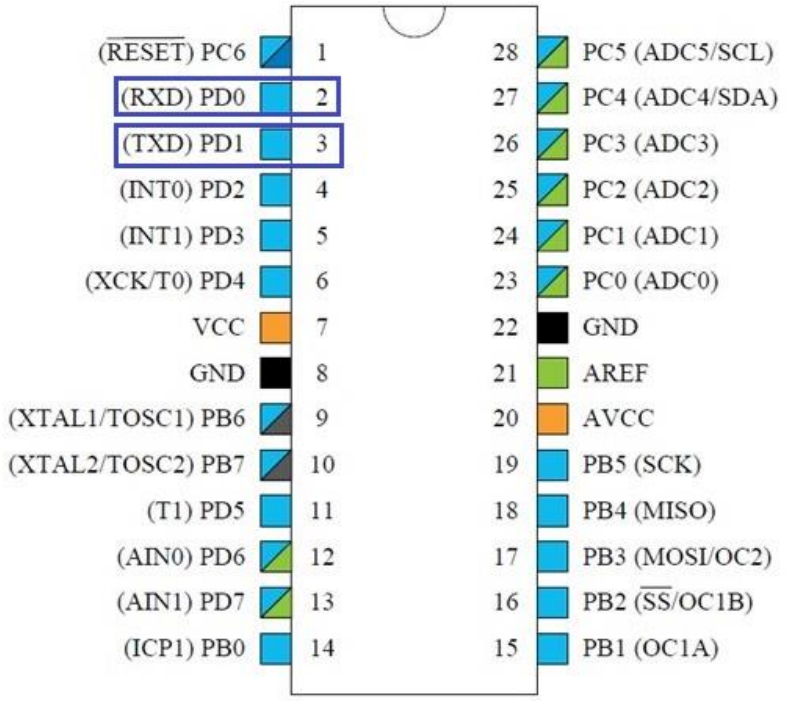


Рисунок 9 – Назначение выводов интегральной микросхемы микроконтроллера Atmeg 328

Два устройства связываются между собой по интерфейсу USART посредством прямого подключения ножки TxD одного устройства к ножке RxD другого и, наоборот, вывод RxD одного устройства мы подключаем к выводу TxD другого. То есть данные с выхода одного устройства попадают на вход другого и наоборот (рисунок 10).

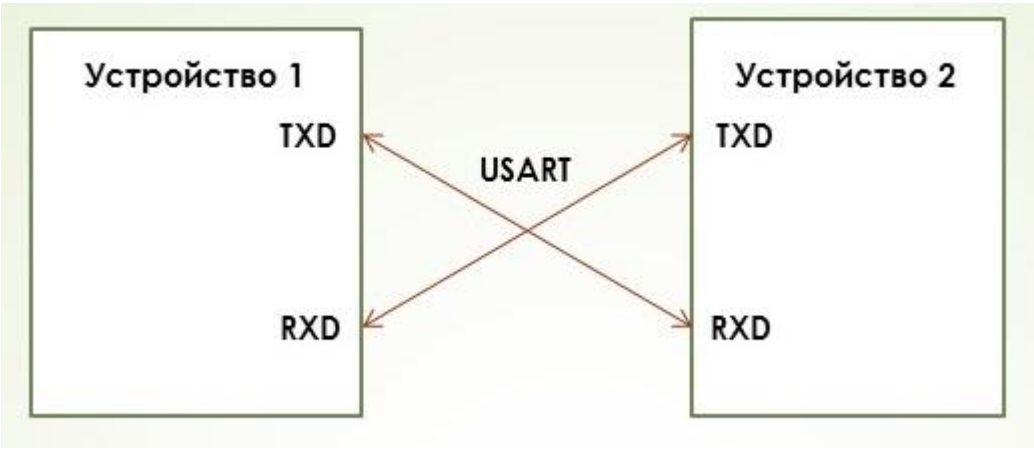


Рисунок 10 – Соединение двух устройств с USART

При работе с шиной USART могут использоваться прерывания по трем событиям:

прием внешних данных в буфер завершен;

регистр данных USART пуст;

передача данных из буферного регистра Tx завершена.

В таблице 2 представлены векторы и условные обозначения данных прерываний.

Таблица 2 – Вектора прерываний от USART

Vector No.	Program Address(2)	Source	Interrupt Definition
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete

Структура (формат) кадра данных, передаваемых по шине USART показана на рисунке 11.

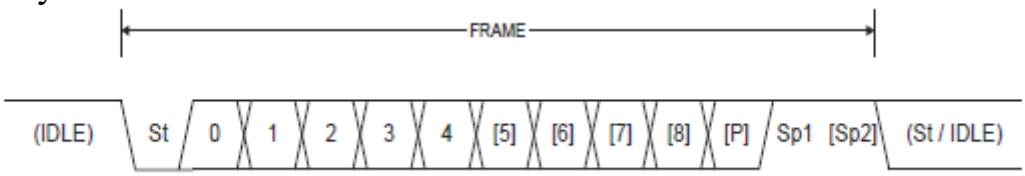


Рисунок 11 – Формат кадра USART

В неактивном режиме, когда шина не используется и данные по определённому выводу не передаются или не принимаются, линия находится в высоком состоянии. В начале кадра идёт обязательный стартовый бит (St). Как видим, он заключается в переходе с высокого состояния на низкое. Длительность одного бита определяется делением одной секунды на скорость, установленную для передачи и приёма данных, которая измеряется в битах в секунду ну или, как ещё называют, в бодах. После того, как время стопового бита истечёт, конроллер будет считать следующие биты информационными. Их может быть от пяти до девяти в зависимости от режима. Вообще, чаще всего используется именно 8, так как удобнее всего данные передавать байтами. Состояние информационного вывода (TXD) во время передачи определённого информационного бита будет диктоваться собственно самим информационным битом. То есть если надо передать единицу – то высокое, если ноль – низкое. Важно заметить также, что передача информационных битов начинается с самого младшего, и затем более старшие.

Следующий бит P – это бит чётности. Если контроль по четности (нечетности) не используется, данный бит в кадре отсутствует.

Затем идут стоповые биты, гласящие о том, что передача посылки закончена. Данных битов может быть один или два также в зависимости от режима. Данные биты передаются с помощью организации высокого логического состояния на ножке. И затем процесс повторяется сначала, то есть передаём следующую посылку.

Порядок расчета параметров для настройки скорости интерфейса представлен в таблице 3. Полученный результат заносится в регистр UBRR.

Например, для платы Arduino Uno с микроконтроллером Atmega 328 частота кварцевого резонатора равна 16 МГц , наиболее распространенная скорость обмена 9600, соответственно, значение регистра UBRR=103.

Таблица 3 – Расчет параметра, определяющего скорость работы интерфейса

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Так как мы синхронный режим не рассматриваем, нас интересуют две верхние строки. Вторая колонка – это расчет скорости в зависимости от частоты тактирования и значения регистра UBRR, а третья – то же самое уравнение, только преобразованное для вычисления необходимого значения UBRR в зависимости от частоты тактирования и нужной скорости шины. Соответственно, скорость шины USART должна быть универсальная и придерживаться определённых стандартов. Но так как в результате деления, мы не можем получить строго определённую скорость, мы же не можем заносить дробные значения в регистр, то существует определённый процент ошибок. Но, как правило, этих ошибок не происходит за счёт как раз стоповых и стартовых битов, в это время идёт поправка.

Теперь различие в таблице первой и второй строки. Существует ещё бит U2X в определённом регистре, который при нулевом значении никак не влияет на скорость обмена, а при значении 1 он скорость удваивает. За счёт этого в формуле в знаменателе будет уже не 16, а 8.

Несмотря на поправку за счет стоповых и стартовых битов, следует стараться выбирать величину значения регистра UBRR и бита U2X так, чтобы процент был меньшим.

Обмен данными по последовательному интерфейсу

Буферные регистры приемника и передатчика располагаются по одному адресу пространства ввода/вывода и обозначаются как регистр данных UDR (Universal Data Register). В этом регистре хранятся младшие 8 разрядов принимаемых и передаваемых данных. При чтении выполняется обращение к буферному регистру UDR приемника, при записи — к буферному регистру передатчика. В модулях USART буфер приемника является двухуровневым (FIFO буфер), изменение состояния которого происходит при любом обращении к регистру UDR. В связи с этим не следует использовать регистр UDR в качестве операндов команд типа «чтение/модификация/запись» (SBI и CBI). Кроме того, следует быть очень аккуратными при использовании команд проверки SBIC и SBIS, поскольку они также изменяют состояние буфера приемника.

Для управления модулями USART используются три регистра: UCSRA, UCSRB и UCSRC.

Описание разрядов регистров UCSRA, UCSRB и UCSRC представлено в таблицах 4-6, соответственно.

Таблица 4- Описание разрядов регистра UCSRA

Разряд	Название	Описание
7	RXC	Флаг завершения приема. Флаг устанавливается в «1» при наличии непрочитанных данных в буфере приемника (регистр данных UDR). Сбрасывается флаг аппаратно после опустошения буфера (в UART — после прочтения регистра данных). Если разряд RXCIE регистра UCSRB установлен, то при установке флага генерируется запрос на прерывание «прием завершен»
6	TXC	Флаг завершения передачи. Флаг устанавливается в «1» после передачи всех разрядов посылки из сдвигового регистра передатчика, при условии, что в регистр данных UDR не было загружено нового значения. Если разряд TXCIE регистра UCSRB (UCSRnB) установлен, то при установке флага генерируется прерывание «передача завершена». Флаг сбрасывается аппаратно при выполнении подпрограммы обработки прерывания или программно, записью в него лог. 1
5	UDRE	Флаг опустошения регистра данных. Данный флаг устанавливается в «1» при пустом буфере передатчика (после пересылки байта из регистра данных UDR в сдвиговый регистр передатчика). Установленный флаг означает, что в регистр данных можно загружать новое значение. Если разряд UDRIE регистра UCR (UCSRB) установлен, генерируется запрос на прерывание «регистр данных пуст». Флаг сбрасывается аппаратно, при записи в регистр данных
4	FE	Флаг ошибки кадрирования. Флаг устанавливается в «1» при обнаружении ошибки кадрирования, т. е. если первый стопбит принятой посылки равен «0». Флаг сбрасывается при приеме стопбита, равного «1»
3	DOR	Флаг переполнения. В USART флаг устанавливается в «1», если в момент обнаружения нового стартбита в сдвиговом регистре приемника находится последнее принятое слово, а буфер приемника полон (два значения). В UART флаг устанавливается в «1», если новый кадр будет помещен в сдвиговый регистр приемника до того, как из регистра данных будет считано предыдущее слово. Флаг сбрасывается при пересылке принятых данных из сдвигового регистра приемника в буфер
2	PE	Флаг ошибки контроля четности. Флаг устанавливается в «1», если в данных, находящихся в буфере приемника, выявлена ошибка контроля четности. При отключенном контроле четности этот разряд постоянно сброшен в «0»
1	U2X	Удвоение скорости обмена. Если этот разряд установлен в «1», коэффициент деления предделителя контроллера скорости передачи уменьшается с 16 до 8, удваивая тем самым скорость асинхронного обмена по последовательному каналу. В USART разряд U2X используется только при асинхронном режиме работы. В синхронном режиме он должен быть сброшен

0	MPCM	Режим мультипроцессорного обмена. Разряд MPCM используется в режиме мультипроцессорного обмена. Если он установлен в «1», ведомый микроконтроллер ожидает приема кадра, содержащего адрес. Кадры, не содержащие адреса устройства, игнорируются

Таблица 5 - Описание разрядов регистра UCSRB

Разряд	Название	Описание
7	RXCIE	Разрешение прерывания по завершению приема. Если данный разряд установлен в «1», то при установке флага RXC регистра UCSRA генерируется прерывание «прием завершен» (если флаг I регистра SREG установлен в «1»)
6	TXCIE	Разрешение прерывания по завершению передачи. Если данный разряд установлен в «1», то при установке флага TXC регистра UCSRA генерируется прерывание «передача завершена» (если флаг I регистра SREG установлен в «1»)
5	UDRIE	Разрешение прерывания при очистке регистра данных UART. Если данный разряд установлен в «1», то при установке флага UDRE в регистра UCSRA генерируется прерывание «регистр данных пуст» (если флаг I регистра SREG установлен в «1»)
4	RXEN	Разрешение приема. При установке этого разряда в «1» разрешается работа приемника USART/UART и переопределяется функционирование вывода RXD (RXDn). При сбросе разряда RXEN работа приемника запрещается, а его буфер сбрасывается. Значения флагов TXC, DOR/OR и FE при этом становятся недействительными
3	TXEN	Разрешение передачи. При установке этого разряда в «1» разрешается работа передатчика UART и переопределяется функционирование вывода TXD. Если разряд сбрасывается в «0» во время передачи, выключение передатчика произойдет только после завершения передачи данных, находящихся в сдвиговом регистре и буфере передатчика
2	UCSZ2	Формат посылок. Этот разряд используется для задания размера слов данных, передаваемых по последовательному каналу. В модулях USART он используется совместно с разрядами UCSZ1:0 регистра UCSRC. В модулях UART, если разряд CHR9 установлен в «1», осуществляется передача и прием 9 разрядных данных, если сброшен — 8 разрядных
1	RXB8	8 й разряд принимаемых данных. При использовании 9 разрядных слов данных этот разряд содержит значение старшего разряда принятого слова. В случае USART содержимое этого разряда должно быть считано до прочтения регистра данных UDR
0	TXB8	8 й разряд передаваемых данных. При использовании 9 разрядных слов данных, содержимое этого разряда является старшим разрядом передаваемого слова. Требуемое значение должно быть занесено в этот разряд до загрузки байта данных в регистр UDR

Таблица 6 - Описание разрядов регистра UCSRC

Разряд	Название	Описание
7	URSEL	Выбор регистра. Этот разряд определяет, в какой из регистров модуля производится запись. Если разряд установлен в «1», обращение производится к регистру UCSRC. Если же разряд сброшен в «0», обращение производится к регистру UBRRH.
6	UMSEL	Режим работы USART. Если разряд сброшен в «0», модуль USART работает в асинхронном режиме. Если разряд установлен в «1», то модуль USART работает в синхронном режиме
5	UPM1	Режим работы схемы контроля и формирования четности. Эти разряды определяют функционирование схем контроля и формирования четности
4	UPM0	
3	USBS	Количество стопбитов. Этот разряд определяет количество стопбитов, посылаемых передатчиком. Если разряд сброшен в «0», передатчик посылает 1 стопбит, если установлен в «1», то 2 стопбита. Для приемника содержимое этого разряда безразлично
2	UCSZ1	Формат посылок. Совместно с разрядом UCSZ2 эти разряды определяют количество разрядов данных в посылках (размер слова)
1	UCSZ0	
0	UCPOL	Полярность тактового сигнала. Значение этого разряда определяет момент выдачи и считывания данных на выводах модуля. Разряд используется только при работе в синхронном режиме. При работе в асинхронном режиме он должен быть сброшен в «0»

Ниже представлен пример тестовой программы для изучения протокола USART, где микроконтроллер Atmega 328 обменивается информацией с терминалом, для наглядности к контроллеру подключен LCD 16X02 дисплей. При нажатии на кнопки 1-3 в терминале высвечиваются соответствующие строки, также если выводить в терминал символы "a" или "b", будет загораться или гаснуть светодиод, подключенный к порту PB0 контроллера. Любые символы, выводимые в терминал будут также высвечиваться на LCD дисплее. Для тестирования в "железе", микроконтроллер подключается к компьютеру через микросхему преобразователя уровней MAX232, для обмена данными используется стандартная программа Hyper Terminal от Microsoft.

Схема соединения микроконтроллера и внешних устройств показана на рисунке 12.

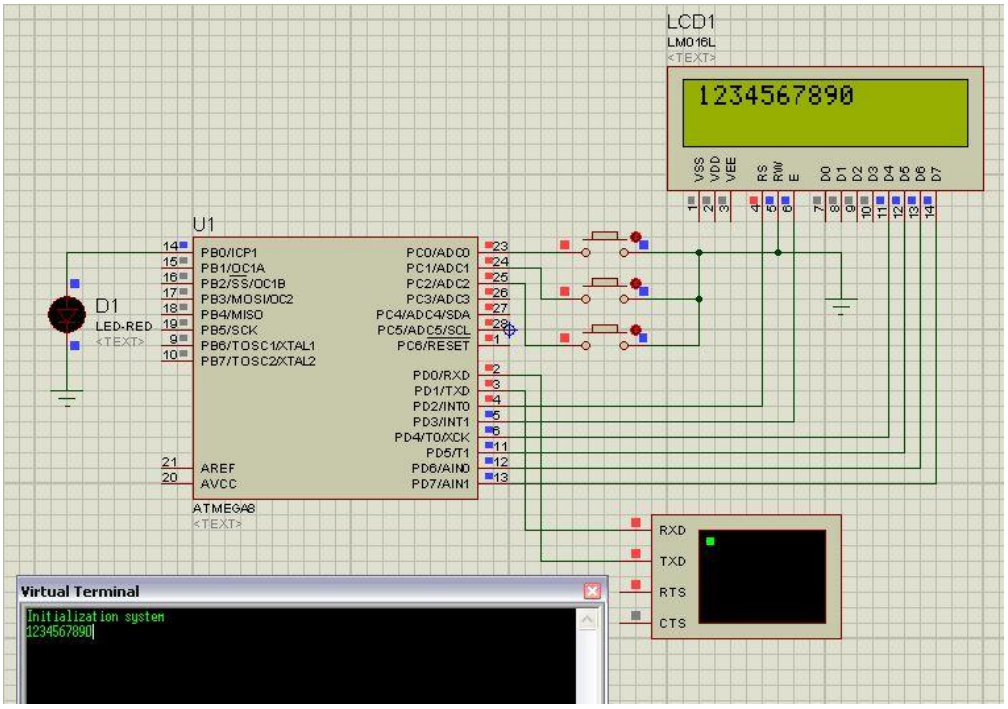


Рисунок 12 – Подключение выводов микроконтроллера

```
/* Пример работы с USART интерфейсом микроконтроллеров AVR */

#include <avr/io.h>#include <avr/interrupt.h>

#define BAUDRATE 9600 // Скорость обмена данными
#define F_CPU 16000000UL // Рабочая частота контроллера

unsigned char NUM = 0;
unsigned char count = 0;
unsigned char byte_receive = 0;
unsigned char i = 1;

// Функция задержки в мкс
void _delay_us(unsigned char time_us)
{ register unsigned char i;

for(i = 0; i < time_us; i++)
{
asm volatile(" PUSH  R0 ");
asm volatile(" POP   R0 ");
}
}

// Функция задержки в мс
void _delay_ms(unsigned int time_ms)
{ register unsigned int i;

for(i = 0; i < time_ms; i++)
{
_delay_us(250);
_delay_us(250);
```

```
_delay_us(250);
_delay_us(250);
}
}

#define RS PD2
#define EN PD3

// Функция передачи команды
void lcd_com(unsigned char p)
{
PORTD &= ~(1 << RS); // RS = 0 (запись команд)
PORTD |= (1 << EN); // EN = 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p & 0xF0); // старший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
PORTD |= (1 << EN); // EN = 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p << 4); // младший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
}

// Функция передачи данных
void lcd_data(unsigned char p)
{
PORTD |= (1 << RS)|(1 << EN); // RS = 1 (запись данных), EN - 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p & 0xF0); // старший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
PORTD |= (1 << EN); // EN = 1 (начало записи команды в LCD)
PORTD &= 0x0F; PORTD |= (p << 4); // младший нибл
_delay_us(100);
PORTD &= ~(1 << EN); // EN = 0 (конец записи команды в LCD)
_delay_us(100);
}

// Функция инициализации LCD
void lcd_init(void)
{
_delay_ms(50); // Ожидание готовности ЖК-модуля

// Конфигурирование четырехразрядного режима
PORTD |= (1 << PD5);
PORTD &= ~(1 << PD4);

// Активизация четырехразрядного режима
PORTD |= (1 << EN);
PORTD &= ~(1 << EN);
```

```
_delay_ms(5);

lcd_com(0x28); // шина 4 бит, LCD - 2 строки
lcd_com(0x08); // полное выключение дисплея
lcd_com(0x01); // очистка дисплея
_delay_us(100);
lcd_com(0x06); // сдвиг курсора вправо
lcd_com(0x0C); // включение дисплея, курсор не видим
}

// Функция вывода строки на LCD
void lcd_string(unsigned char command, char *string)
{
lcd_com(0x0C);
lcd_com(command);
while(*string != '\0')
{
lcd_data(*string);
string++;
}
}

// Функция передачи данных по USART
void uart_send(char data)
{
while(!( UCSRA & (1 << UDRE)));   // Ожидаем когда очистится буфер передачи
UDR = data; // Помещаем данные в буфер, начинаем передачу
}

// Функция передачи строки по USART
void str_uart_send(char *string)
{
while(*string != '\0')
{
uart_send(*string);
string++;
}
}

// Функция приема данных по USART
int uart_receive(void)
{
while(!(UCSRA & (1 << RXC))); // Ожидаем, когда данные будут получены
return UDR; // Читаем данные из буфера и возвращаем их при выходе из подпрограммы
}

// Функция инициализации USART
void uart_init(void)
{
// Параметры соединения: 8 бит данные, 1 стоповый бит, нет контроля четности
// USART Приемник: Включен
// USART Передатчик: Включен
```

```
// USART Режим: Асинхронный
// USART Скорость обмена: 9600

UBRRL = (F_CPU/BAUDRATE/16-1); // Вычисляем скорость обмена данными
UBRRH = (F_CPU/BAUDRATE/16-1) >> 8;

UCSRB |= (1 << RXCIE)| // Разрешаем прерывание по завершению приема данных
(1 << RXEN)|(1 << TXEN); // Включаем приемник и передатчик
UCSRC |= (1 << URSEL)| // Для доступа к регистру UCSRC выставляем бит URSEL
(1 << UCSZ1)|(1 << UCSZ0); // Размер посылки в кадре 8 бит
}

// Прерывание по окончанию приема данных по USART
ISR(USART_RXC_vect)
{
NUM = UDR; // Принимаем символ по USART
byte_receive = 1;
uart_send(NUM); // Посылаем символ по USART

if(NUM == 'a') // Если принят символ "a", включаем светодиод
PORTB |= (1 << PB0);
if(NUM == 'b') // Если принят символ "b", выключаем светодиод
PORTB &= ~(1 << PB0);
}

// Главная функция
int main(void)
{
DDRB |= (1 << PB0); // Светодиод
PORTB = 0x00;

DDRC  = 0x00;
PORTC = 0xFF;

DDRD  = 0b11111110;
PORTD = 0x00;

lcd_init(); // Инициализация LCD
uart_init(); // Инициализация USART

sei(); // Глобально разрешаем прерывания

str_uart_send("Initialization system\r\n"); // Передаем строку по USART
lcd_string(0x80, " AVR USART TEST "); // Выводим строку на LCD
_delay_ms(2500);
lcd_com(0x01); // Очищаем LCD

while(1)
{

if((PINC & (1 << PC0)) == 0) // Если нажата кнопка 1
{
```

```
while((PINC & (1 << PC0)) == 0){} // Ждем отпускания кнопки 1
str_uart_send("Button 1 TEST\r\n"); // Передаем строку по USART
lcd_string(0x80, "Button 1 TEST"); // Выводим строку на LCD
_delay_ms(1000);
lcd_com(0x01); // Очищаем LCD
}

if((PINC & (1 << PC1)) == 0) // Если нажата кнопка 2
{
while((PINC & (1 << PC1)) == 0){} // Ждем отпускания кнопки 2
str_uart_send("Button 2 TEST\r\n"); // Передаем строку по USART
lcd_string(0x80, "Button 2 TEST"); // Выводим строку на LCD
_delay_ms(1000);
lcd_com(0x01); // Очищаем LCD
}

if((PINC & (1 << PC2)) == 0) // Если нажата кнопка 3
{
while((PINC & (1 << PC2)) == 0){} // Ждем отпускания кнопки 3
str_uart_send("Button 3 TEST\r\n"); // Передаем строку по USART
lcd_string(0x80, "Button 3 TEST"); // Выводим строку на LCD
_delay_ms(1000);
lcd_com(0x01); // Очищаем LCD
}

if(byte_receive)
{
byte_receive = 0;
count++;
lcd_data(NUM); // Выводим символ на LCD
if(count > 16) // Если строка заполнена
{
count = 0;
lcd_com(0x01); // Очищаем LCD
}
}
}
}
```

Задание для самостоятельной работы

Выполнить все примеры программ. Внести изменения в листинги программ в соответствии с указанием преподавателя. Скриншоты экрана с программным кодом и результатами работы представить в отчете. Сделать выводы о технологии программирования средств отображения результатов работы программы с использованием ЖКИ и USART.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №7
ПРОГРАММИРОВАНИЕ ПЛК В СРЕДЕ CODESYS

Цель занятия: Приобрести навыки программирования в среде CoDeSys программируемых логических контроллеров

Учебные вопросы

1. Разработка программы управления светофором
2. Отладка программы в режиме эмуляции
3. Визуализация реализуемой программы

1 Разработка программы управления светофором

В данной лабораторной работе предполагается создание программы для простого блока управления движением на перекрестке, имеющем светофоры для двух пересекающихся направлений движения. Понятно, что светофоры должны иметь два противоположных состояния – красный и зеленый. Чтобы избежать несчастных случаев, мы добавим общепринятые переходные стадии: желтый и желто-красный. Последняя стадия должна быть длиннее предыдущей.

При выполнении лабораторной работы будет показано, как управляемые по времени процессы можно представить средствами языков стандарта МЭК 61131-3, как можно комбинировать различные языки в CoDeSys и как пользоваться средствами моделирования CoDeSys.

Порядок выполнения работы.

Запустите CoDeSys и выберите ‘Файл’ ‘*Создать’ ("File" "New").* В окне диалога определим первый **программный компонент (POU – Program Organization Unit)**. По умолчанию он получает наименование PLC_PRG. Не изменяйте его. Тип этого POU, безусловно, должен быть - программа. Каждый проект должен иметь программу с таким именем. В качестве языка программирования данного POU мы выберем язык Continuous Function Chart (CFC). Теперь создайте еще три объекта. Воспользуйтесь командой ‘Проект’ ‘Объект - Добавить’ *("Project" "Object Add")* в системном или в контекстном (нажмите правую кнопку мыши в Организаторе объектов) меню. Создайте: программу на языке Sequential Function Chart (SFC) с именем SEQUENCE, функциональный блок на языке Function Block Diagram (FBD) с именем TRAFFICSIGNAL и еще один аналогичный блок - WAIT, который мы будем описывать на языке Список Инструкции (IL).

В POU TRAFFICSIGNAL мы сопоставим определенные стадии процесса соответствующим цветам. То есть мы удостоверимся, что красный свет зажжен в красной стадии и в желто-красной стадии, желтый свет в желтой и желто-красной стадии и т.д.

В WAIT мы создадим простой таймер, который на вход получает длину стадии в миллисекундах и на выходе выдает состояние ИСТИНА по истечении заданного периода времени.

В SEQUENCE все будет объединено так, чтобы нужные огни зажигались в правильное время и на нужный период времени.

В PLC_PRG вводится входной сигнал включения, разрешающий начало работы светофора и 'цветовые команды' каждой лампы связаны с соответствующими выходами аппаратуры.

Объявления "TRAFFICSIGNAL". В редакторе объявлений определите входную переменную (между ключевыми словами VAR_INPUT и END_VAR) по имени STATUS типа INT. STATUS будет иметь четыре возможных состояния, определяющие соответствующие стадии - зеленая, желтая, желто-красная и красная.

Поскольку наш блок TRAFFICSIGNAL имеет три выхода, нужно определить еще три переменных RED, YELLOW и GREEN. Теперь раздел объявлений блока TRAFFICSIGNAL должен выглядеть как показано на рисунке 1.

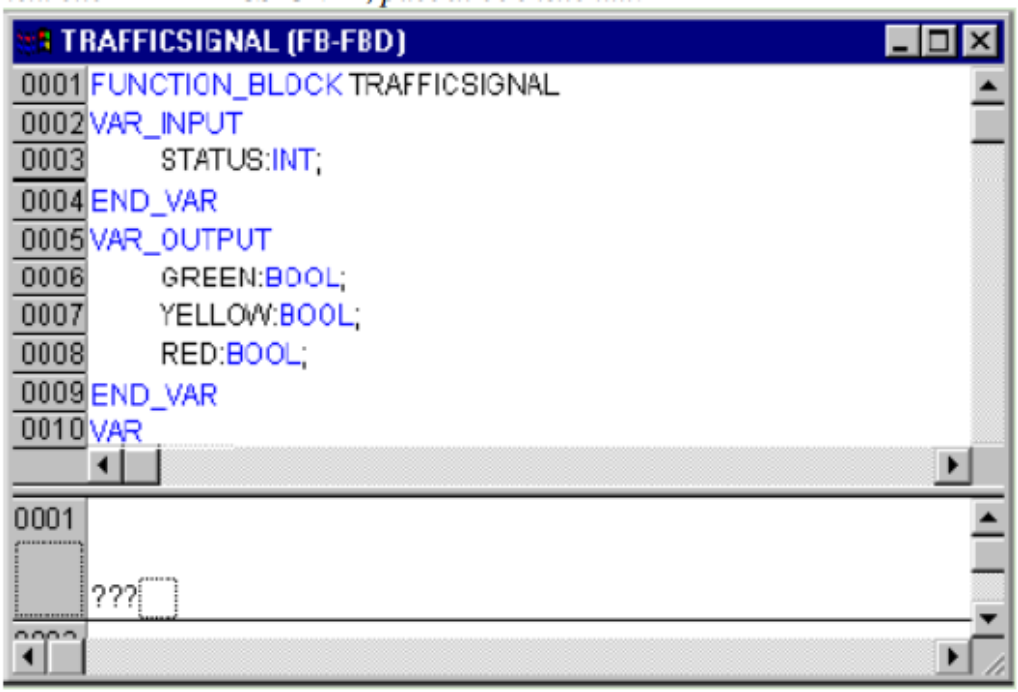
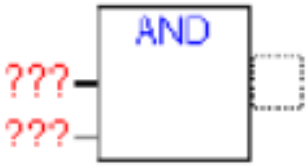
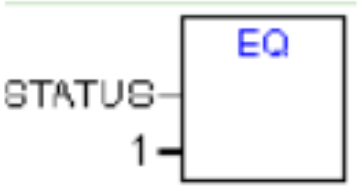


Рисунок 1 - Функциональный блок TRAFFICSIGNAL, раздел объявлений

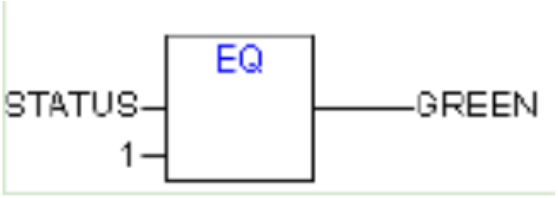
Программирование "TRAFFICSIGNAL". Необходимо описать, что связывает вход STATUS с выходными переменными. Для этого перейдите в раздел кода POU (body). Щелкните на поле слева от первой цепи (серая область с номером 1). Вы теперь выбрали первую цепь. Теперь дайте команду меню 'Вставка' 'Элемент' *("Insert" "Box").* В первой цепи будет вставлен прямоугольник с оператором AND и двумя входами:



Щелкните мышкой на тексте AND и замените его на EQ. Три знака вопроса около верхнего из двух входов замените на имя переменной STATUS. Для нижнего входа вместо трех знаков вопроса нужно поставить 1. В результате Вы должны получить следующий элемент:



Щелкните теперь на месте позади прямоугольника EQ. Теперь выбран выход EQ. Выполните команду меню ‘Вставка’ ‘Присваивание’(*"Insert" "Assign"*). Измените три вопроса ??? на GREEN. Вы теперь создали цепь следующего вида:



STATUS сравнивается с 1, результат присваивается GREEN. Таким образом, GREEN будет включен, когда STATUS равен 1. Для других цветов TRAFFICSIGNAL нам понадобятся еще две цепи. Создаете их командой ‘Вставка’ ‘Цепь (после)’ *("Insert" "Network (after)")*. Законченный POU должен выглядеть как показано на рисунке 2.

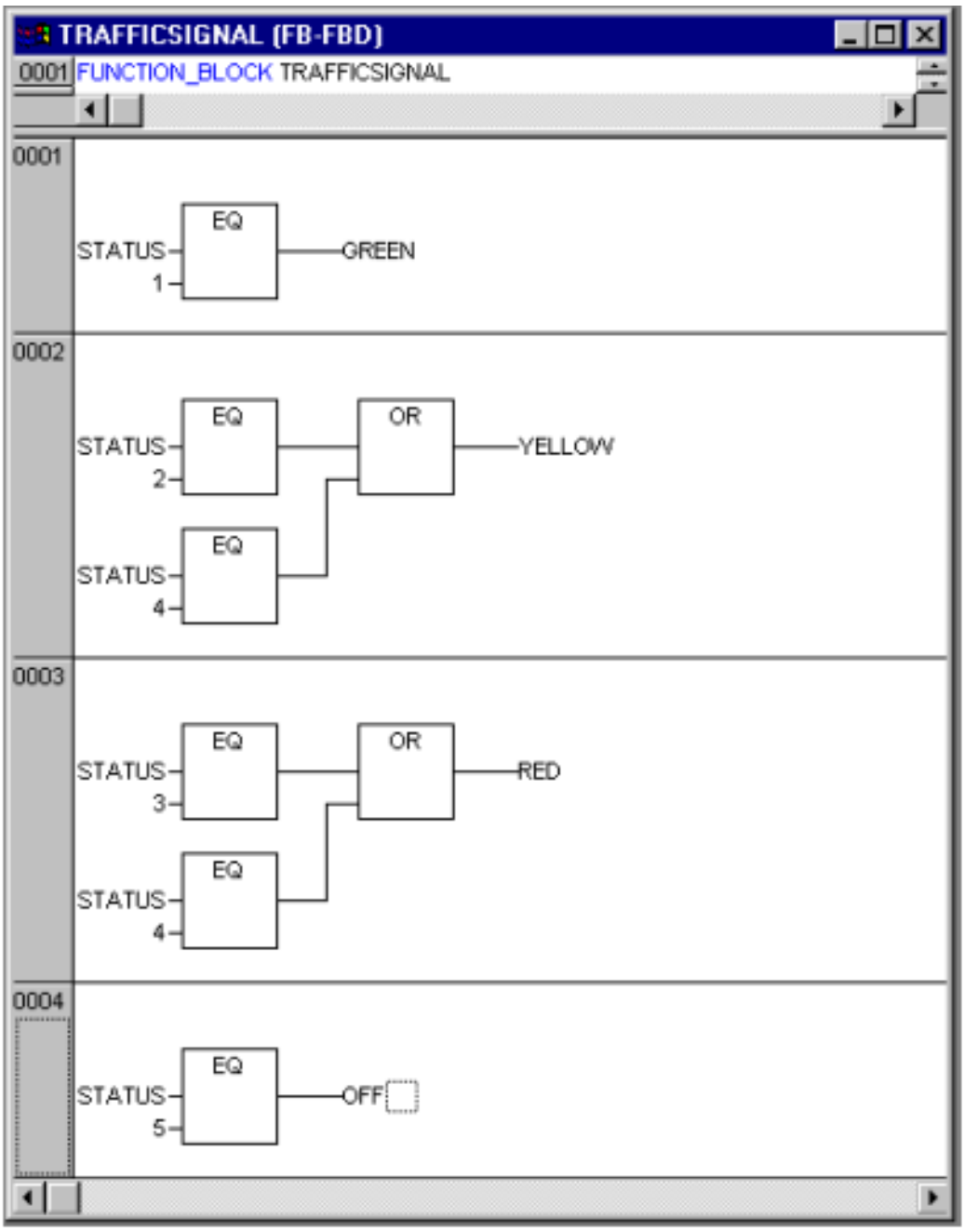


Рисунок 2- Функциональный блок TRAFFICSIGNAL

Чтобы вставить оператор перед входом другого оператора, Вы должны выделить сам вход, а не текст (выделяется прямоугольником). Далее используйте команду ‘Вставка’ ‘Элемент’*("Insert" "Box")*. Теперь наш первый POU закончен. Как и планировалось, TRAFFICSIGNAL будет управлять включением выходов, руководствуясь значением переменной STATUS.

Подключение standard.lib Для создания таймера в POU WAIT нам понадобится POU из стандартной библиотеки. Для этого откройте менеджер библиотек командами ‘Окно’ ‘Менеджер библиотек’ *("Window" "Library Manager"*). Выберете ‘Вставка’ ‘Добавить библиотеку’ *("Insert" "Additional library"*). Должно открыться диалоговое окно выбора файлов. Выберете standard.lib из списка библиотек.

Объявления "WAIT". Теперь давайте вернемся к POU WAIT. Как предполагалось, этот POU будет работать таймером, задающим длительность стадий TRAFFICSIGNAL. Наш POU должен иметь входную переменную TIME типа TIME и генерировать на выходе двоичную (Boolean) переменную, которую мы назовем OK. Данная переменная должна принимать значение TRUE, когда желательный период времени закончен. Предварительно мы устанавливаем эту переменную в значение FALSE в конце строки объявления (но до точки с запятой) ":= FALSE ".

Теперь нам нужен генератор времени POU TP. Он имеет два входа (IN, PT) и два выхода (Q, ET). TP делает следующее: Пока IN установлен в FALSE, ET будет 0 и Q будет FALSE. Как только IN переключится в TRUE, выход ET начнет отсчитывать время в миллисекундах. Когда ET достигнет значения заданного PT, счет будет остановлен. Тем временем выход Q равен TRUE, пока ET меньше PT. Как только ET достигнет значения PT, выход Q снова переключится в FALSE.

Описание всех POU из стандартной библиотеки приведено в рекомендациях по работе в среде CoDeSys.

Чтобы использовать TP в POU WAIT, мы должны создать его локальный экземпляр. Для этого мы объявляем локальную переменную ZAB (отсчитанное время) типа TP (между ключевыми словами VAR, END_VAR). Раздел объявлений WAIT теперь должен выглядеть, как показано на рис. 3

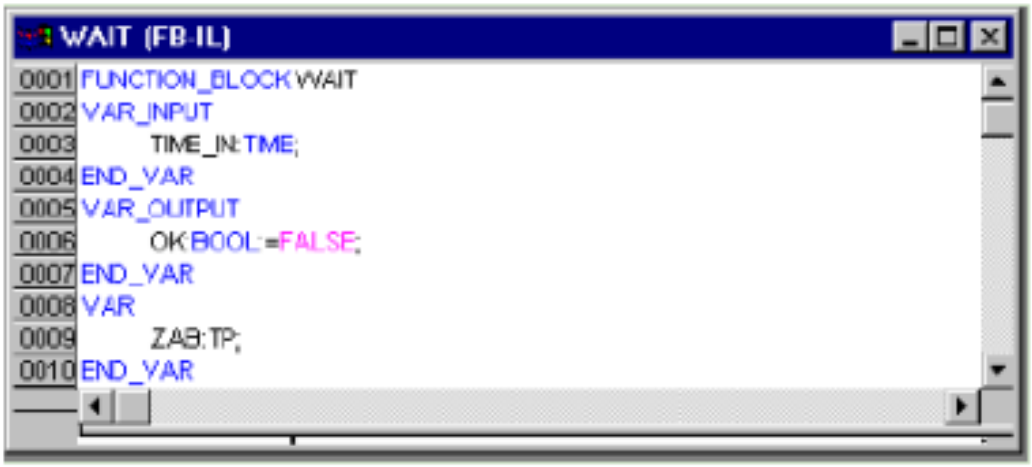


Рисунок 3 – Программирование "WAIT"

Для создания желаемого таймера текст программы должен быть таким как показано на рисунке 4

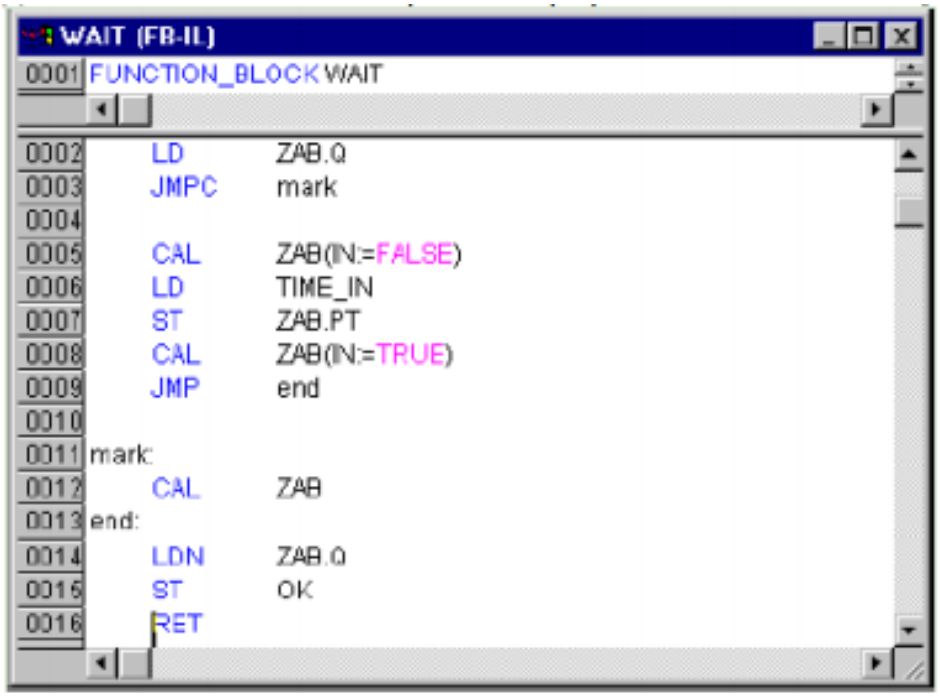


Рисунок 4

Сначала проверяется, установлен ли Q в TRUE (возможно, отсчет уже запущен), в этом случае мы не трогаем установки ZAB, а вызываем функциональный блок ZAB без входных переменных - чтобы проверить, закончен ли период времени. Иначе мы устанавливаем переменную IN ZAB в FALSE и одновременно ET в 0 и Q в FALSE. Таким образом, все переменные установлены в начальное состояние.

Теперь мы устанавливаем необходимое время TIME переменной PT и вызываем ZAB с IN:=TRUE. Функциональный блок ZAB теперь будет работать, пока не достигает значения TIME и не установит Q в FALSE. Инвертированное значение Q будет сохраняться в переменной OK после каждого выполнения WAIT. Как только Q станет FALSE, OK примет значение TRUE.

С таймером покончено. Теперь пришло время объединять наши два блока WAIT и TRAFFICSIGNAL в главной программе PLC_PRG.

"SEQUENCE" версия 1. Сначала объявим необходимые переменные. Это входная переменная START типа BOOL, две выходных переменные TRAFFICSIGNAL1 и TRAFFICSIGNAL2 типа INT и одна переменная DELAY типа WAIT. Программа SEQUENCE теперь выглядит следующим образом (рисунок 5).

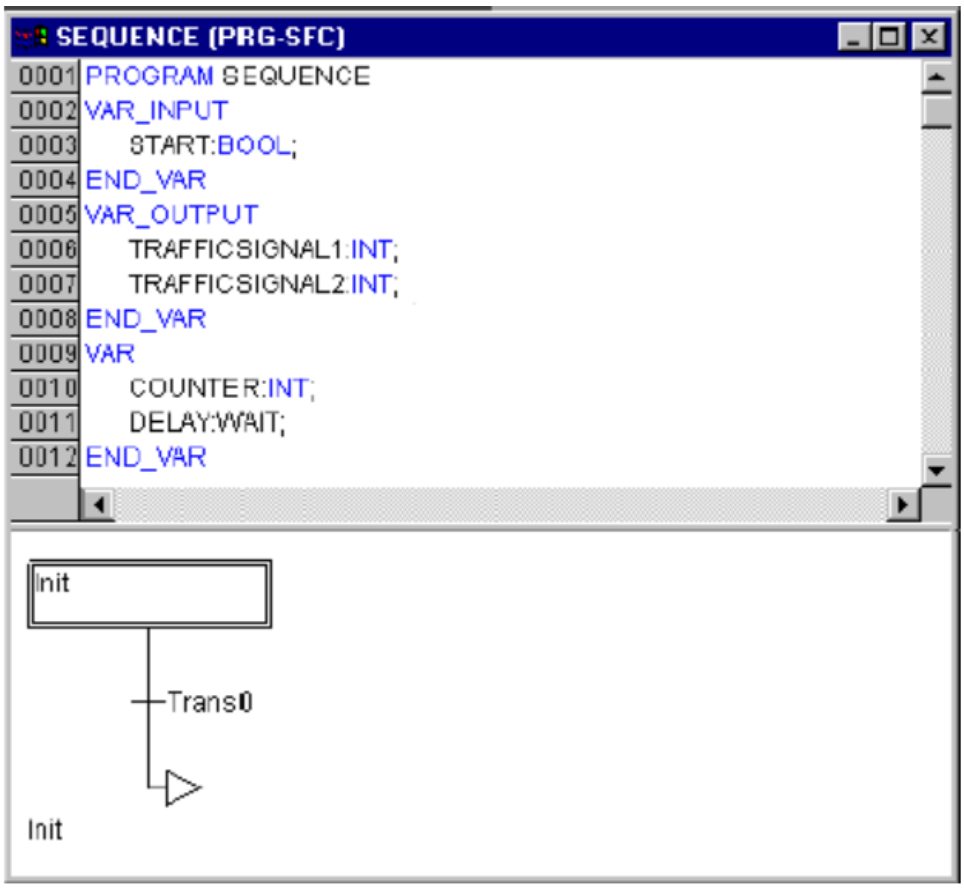


Рисунок 5- Программа SEQUENCE версия 1, раздел объявлений

Создаем SFC диаграмму. Первоначально SFC граф всегда состоит из этапа "Init" перехода "Trans0" и возврата назад к Init, естественно, нам придется несколько дополнить его.

Прежде чем программировать конкретные этапы и переходы, выстроим структуру графа. Сначала нам понадобятся этапы для каждой стадии TRAFFICSIGNAL. Вставьте их, отмечая Trans0 и выбирая команды ‘Вставка’ ‘Шаг-переход (снизу)’(*"Insert" "Step transition (after)"*). Повторите эту процедуру еще три раза.

Для редактирования названия перехода или этапа нужно просто щелкнуть мышкой на нужном тексте. Назовите первый переход после Init "START", а все прочие переходы "DELAY.OK". Первый переход разрешается, когда START устанавливается в TRUE, все же прочие - когда DELAY в OK станет TRUE, т.е. когда заданный период закончится. Этапы (сверху вниз) получают имена Switch1, Green2, Switch2, Green1, ну и Init, конечно, сохранит своё имя. "Switch" должен включать жёлтую фазу, в Green1 TRAFFICSIGNAL1 будет зеленым, в Green2 TRAFFICSIGNAL2 будет зеленым. Наконец, измените адрес возврата Init на Switch1. Если вы всё сделали верно, диаграмма должна выглядеть как показано на рисунке 6.

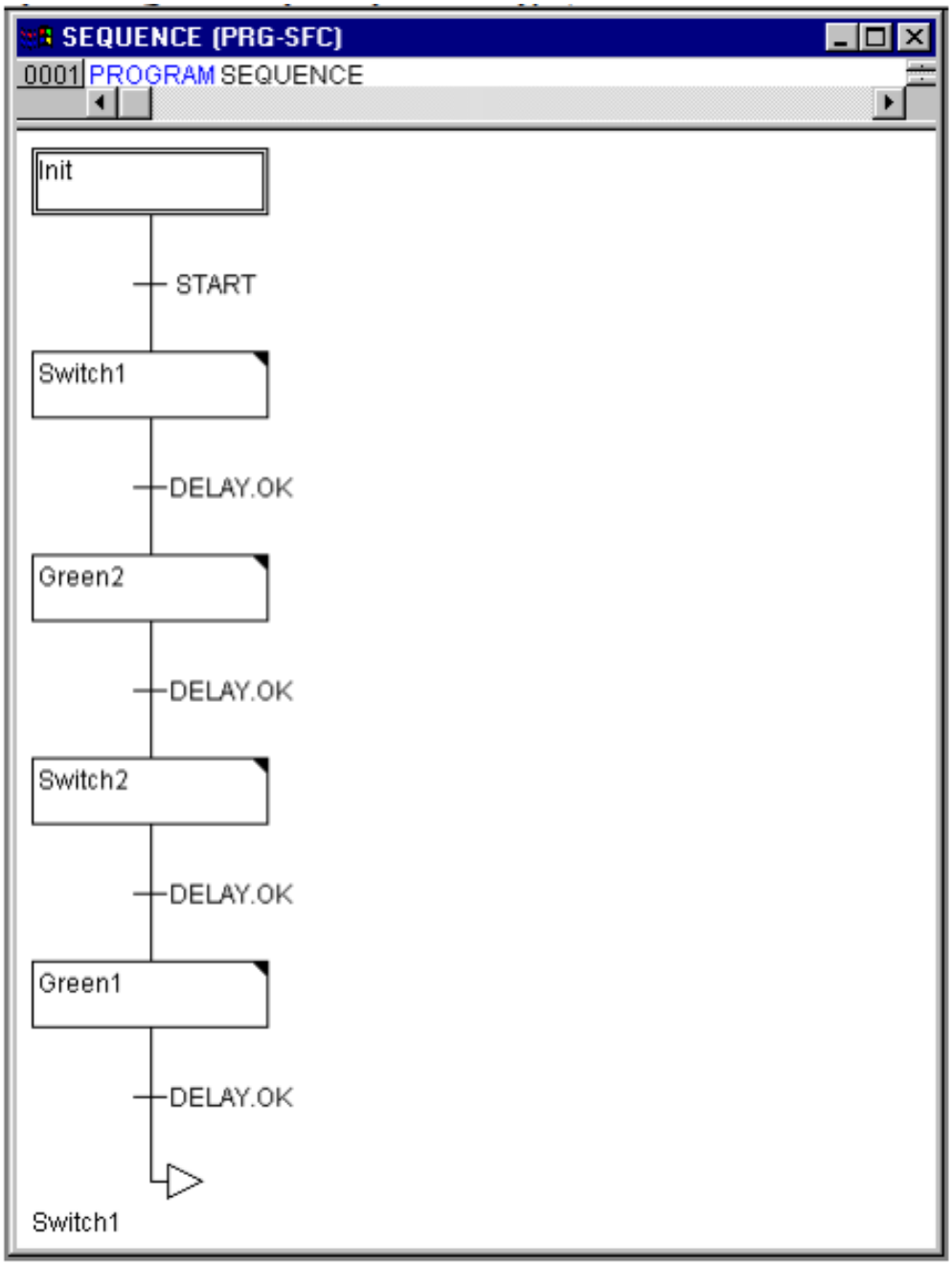


Рисунок 6- Программа SEQUENCE версия 1, раздел инструкций

Теперь мы должны запрограммировать этапы. Если вы сделаете двойной щелчок мышью на изображении этапа, то должен открыться диалог определения нового действия. В нашем случае мы будем использовать язык IL (Список Инструкций).

Программирование этапов и переходов. Во время действия этапа Init проверяем, активен сигнал включения START или нет. Если сигнал не активен, то светофор выключается. Этого можно достичь, если записать в переменные TRAFFICSIGNAL1 и TRAFFICSIGNAL2 число 5 (рисунок 7).

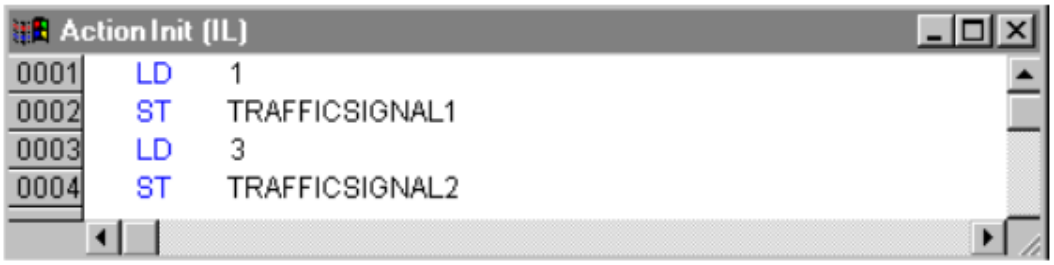
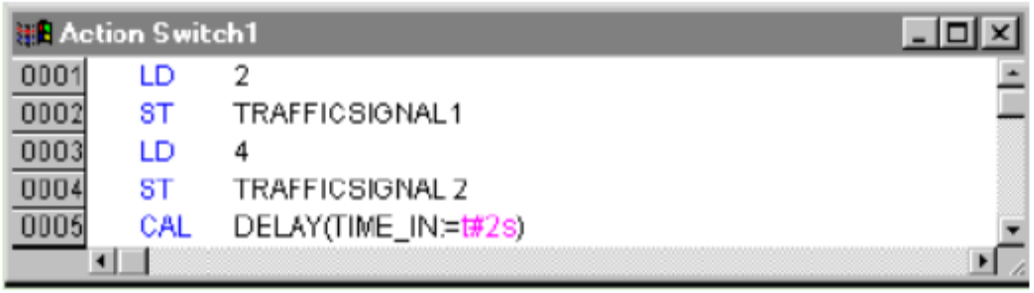


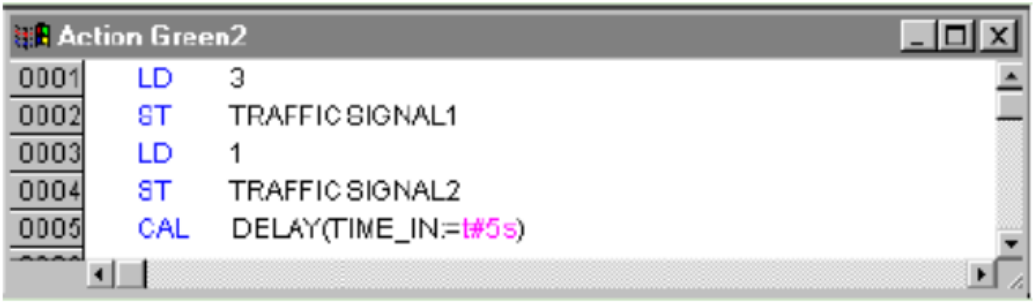
Рисунок 7- Этап Init

В Green1 TRAFFICSIGNAL1 будет зеленым (STATUS:=1), TRAFFICSIGNAL2 будет красным (STATUS:=3), задержка в 5000 миллисекунд. Этап Green1:

Switch1 изменяет состояние TRAFFICSIGNAL1 на 2 (жёлтое) и, соответственно, TRAFFICSIGNAL2 на 4 (жёлто-красное). Кроме того, теперь устанавливается задержка в 2000 миллисекунд. Этап Switch1 выглядит так:

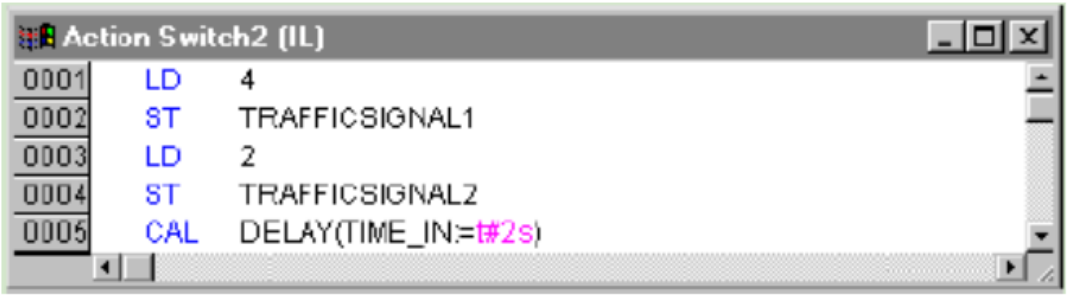


Green2 включает красный в TRAFFICSIGNAL1 (STATUS:=3) и зеленый в TRAFFICSIGNAL2 (STATUS:=1). Задержка устанавливается в 5000 миллисекунд. Этап Green2:



В Switch2 STATUS в TRAFFICSIGNAL1 изменяется на 4 (жёлто-красный), соответственно, TRAFFICSIGNAL2 будет 2 (жёлтый). Задержка теперь должна быть в 2000 миллисекунд.

Этап Switch2:



Первая версия нашей программы закончена.

2 Отладка программы в режиме эмуляции

Для проверки работы программы в режиме эмуляции, сделайте следующее: Откройте POU PLC_PRG. Каждый проект начинает работу с PLC_PRG. Вставьте в него компонент и замените “AND” на “SEQUENCE”. Входы и выходы оставьте пока свободными.

Выполните компиляцию проекта (‘Проект’ ‘Компилировать’ - *'Project' 'Build'*) и проверте отсутствие ошибок. В окне сообщений вы должны увидеть текст: "0 Errors, 0 Warnings". Теперь включите флажок ‘Онлайн’ ‘Режим эмуляции’ *(‘Online’ 'Simulation mode'*) и дайте команду 'Онлайн' ‘Подключиться’ (‘*Online’ 'Login'*).

Запустите программу 'Онлайн' ‘Старт’ *(‘Online’ 'Run'*). Откройте программу SEQUENCE.

Программа запущена, но не работает, поскольку переменная START должна иметь значение TRUE. Далее это будет делать PLC_PRG, но сейчас вы можете изменить ее вручную. Для этого щелкните дважды мышью по объявлению этой переменной. Ее значение теперь выделено цветом и равно TRUE. Дайте команду записи значений переменных ('Онлайн' ‘Записать значения’ – *‘Online’ 'Write values'*).

Теперь вы можете понаблюдать за работой программы. Активные шаги диаграммы выделяются голубым цветом.

Для продолжения редактирования программы закройте режим онлайн командой 'Онлайн' ‘Отключение’ (‘Online’ 'Logout').

"SEQUENCE" вторая версия. Теперь немного усложним нашу программу. Разумно будет выключать наши светофоры на ночь. Для этого мы создадим в программе счетчик, который после некоторого числа циклов TRAFFICSIGNAL произведет отключение устройства. Для начала нам нужна новая переменная COUNTER типа INT. Объявите её как обычно в разделе объявлений SEQUENCE. Теперь выберете переход после Switch1 и вставьте ещё один этап и переход. Выберете результирующий переход и вставьте альтернативную ветвь вправо. После левого перехода вставьте дополнительный этап и

переход. После нового результирующего перехода вставьте удаленный переход (jump) на Init. Назовите новые части так: верхний из двух новых этапов нужно назвать "Count" и нижний "Off". Переходы будут называться (сверху вниз слева на право) EXIT, TRUE и DELAY.OK. Теперь новые части должны выглядеть как фрагмент, выделенный рамкой (рисунок 7).

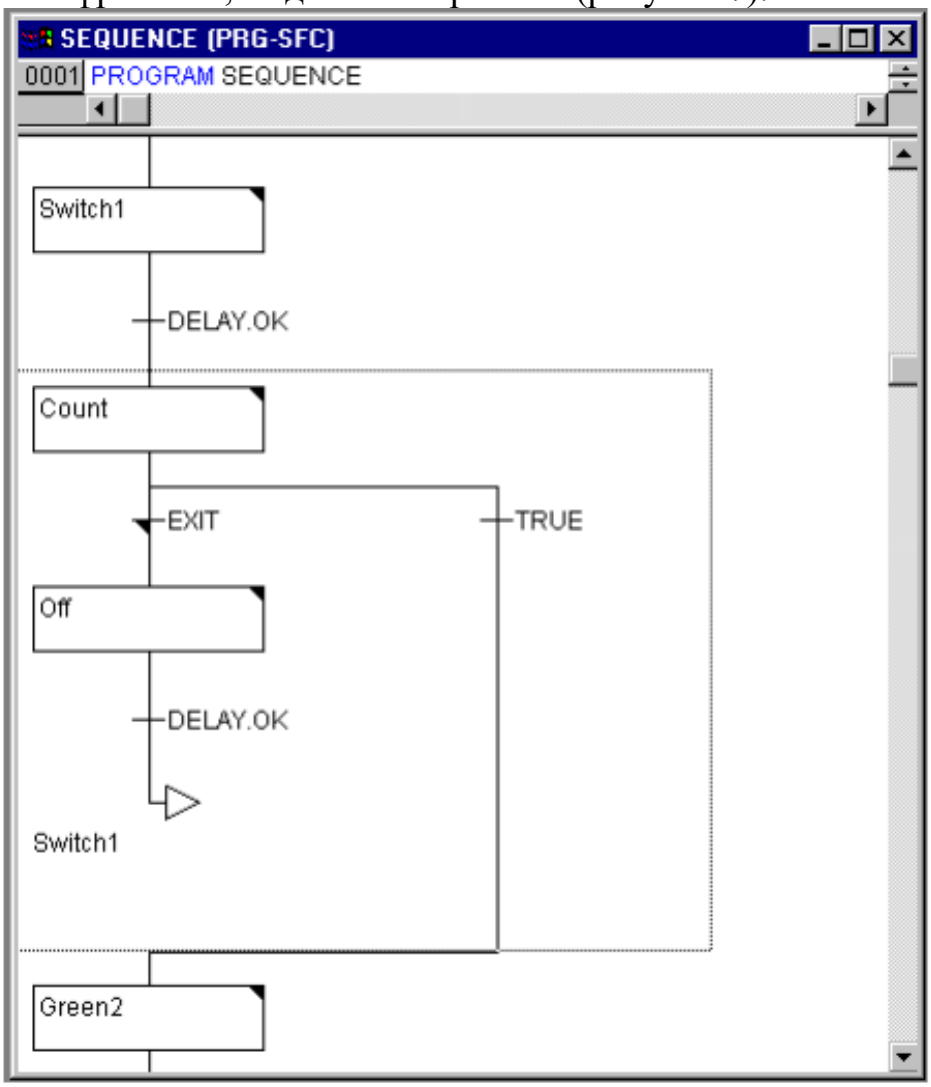
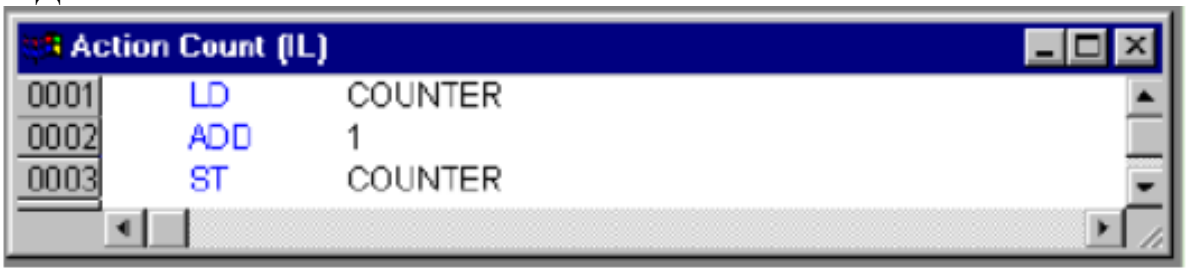


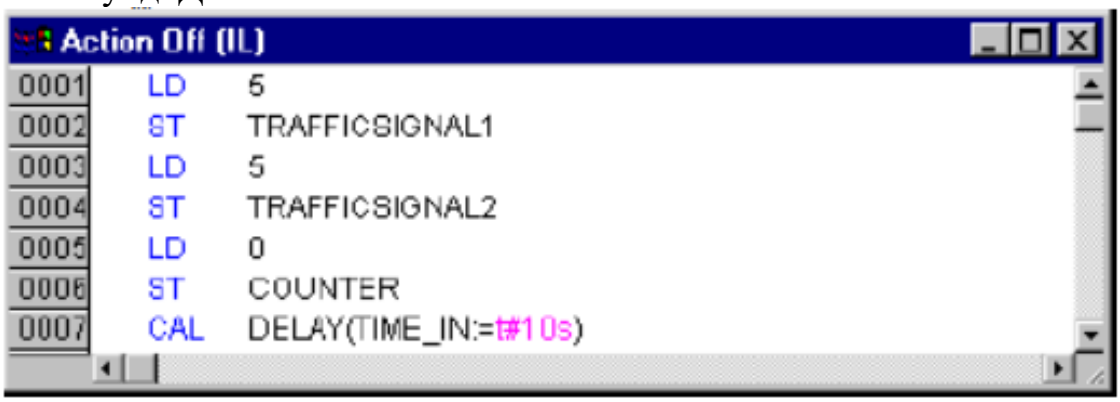
Рисунок 7- Программа "SEQUENCE", раздел инструкций

Теперь два новых этапа и перехода необходимо наполнить содержанием. На этапе Count выполняется только одно действие - COUNTER увеличивается на 1: Действие Count:



На переходе EXIT1 проверяется достижение счетчиком заданного значения, например 7: Переход EXIT:

На этапе Off состояние обоих светофоров устанавливается в 5 (светофор выключен), COUNTER сбрасывается в 0 и устанавливается задержка времени в 10 секунд. Действие Off:



В нашей гипотетической ситуации ночь наступает после семи циклов TRAFFICSIGNAL. Светофоры полностью выключаются до рассвета, и процесс повторяется снова. При желании вы можете еще раз проверить работу программы в эмуляторе, прежде чем продолжить ее усовершенствование.

PLC_PRG . Мы определили два строго коррелированных во времени светофора в блоке SEQUENCE. Теперь полностью закончим программу. Для этого необходимо распределить входные и выходные переменные в блоке PLC_PRG. Мы хотим дать возможность запустить систему выключателем IN и хотим обеспечить переключение всех шести ламп (2 светофора) путем передачи ”команд переключения” на каждом шаге SEQUENCE. Объявим теперь соответствующие Boolean переменные для всех шести выходов и одного входа, затем создадим программу и сопоставим переменные соответствующим IEC адресам.

Следующий шаг - это объявление переменных LIGHT1 и LIGHT2 типа TRAFFICSIGNAL в редакторе объявлений.

Объявление LIGHT1 и LIGHT2:

Для представления шести ламп светофоров нужно 6 переменных типа Boolean. Однако мы не будем объявлять их в разделе объявлений блока PLC_PRG, вместо этого используем глобальные переменные (Global Variables) из ресурсов (Resources). Двоичная входная переменная IN, необходимая для установки переменной START блока SEQUENCE в TRUE, будет определена таким же образом. Выберете вкладку ‘Ресурсы’ (Resources) и откройте список ‘Глобальные переменные’ (Global Variables). Объявление глобальных переменных:

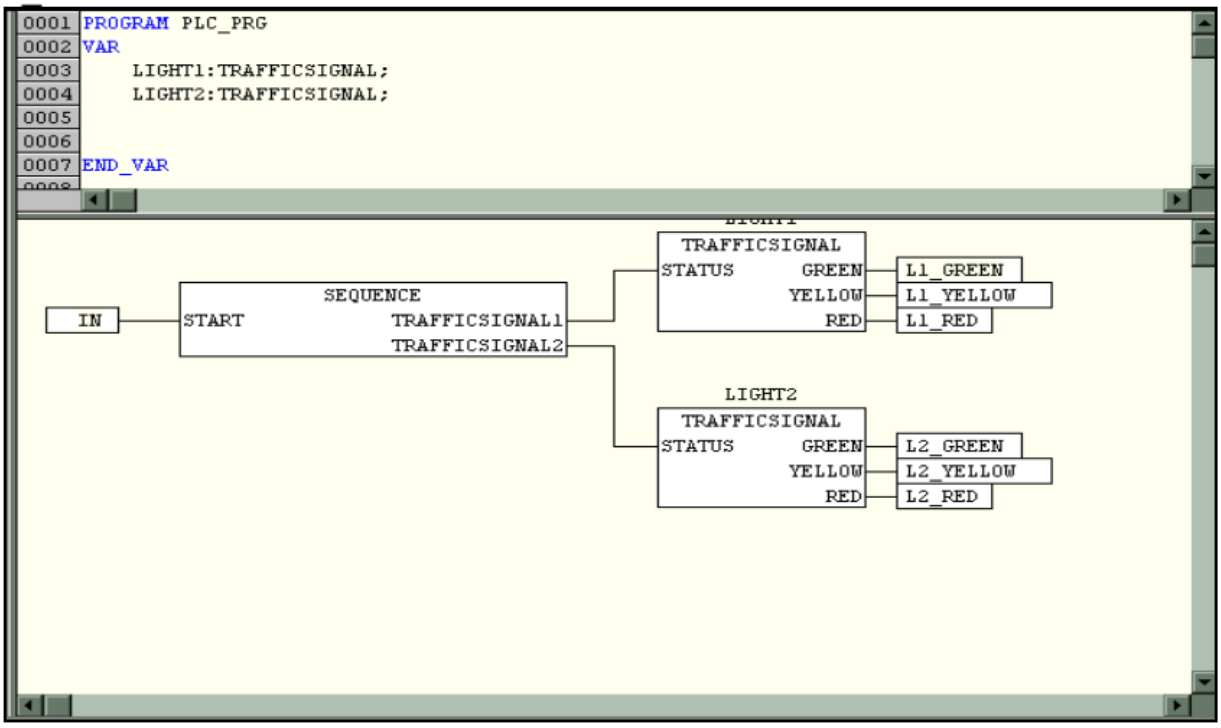
```
0001 VAR_GLOBAL
0002     IN:BOOL;
0003     L1_GREEN:BOOL;
0004     L1_YELLOW:BOOL;
0005     L1_RED:BOOL;
0006     L2_GREEN:BOOL;
0007     L2_YELLOW:BOOL;
0008     L2_RED:BOOL;
0009 END_VAR
0010
```

Закончим PLC_PRG. Для этого мы перейдем в окно редактора. Мы выбрали редактор Непрерывных Функциональных Схем (CFC), и, следовательно, нам доступна соответствующая панель инструментов.

Щелкните правой клавишей мыши в окне редактора и выберите элемент ‘Блок’ (*Box*). Щелкните на тексте AND и напишите ”SEQUENCE”. Элемент автоматически преобразуется в SEQUENCE с уже определенными входными и выходными переменными. Вставьте далее два элемента и назовите их TRAFFICSIGNAL.

TRAFFICSIGNAL - это функциональный блок, и, как обычно, Вы получите три красных знака вопроса, которые нужно заменить уже объявленными локальными переменными LIGHT1 и LIGHT2. Теперь создайте элемент типа Input, который получит название IN и шесть элементов типа Output, которым нужно дать следующие имена: L1_green, L1_yellow, L1_red, L2_green, L2_yellow, L2_red. Все элементы программы теперь на месте, и Вы можете соединять входы и выходы. Для этого щелкните мышью на короткой линии входа/выхода и тяните ее (не отпуская клавишу мыши) к входу/выходу нужного элемента. Наконец Ваша программа должна принять вид, показанный ниже.

PLC_PRG:

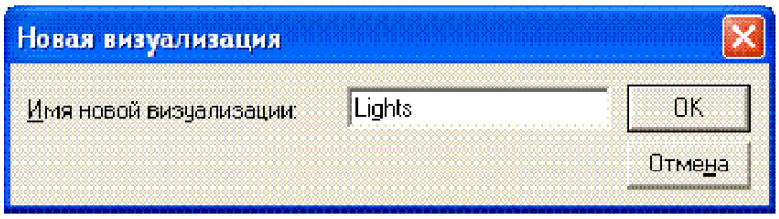


Теперь наша программа полностью готова.

TRAFFICSIGNAL эмуляция. Теперь проверьте окончательно вашу программу в режиме эмуляции. Убедитесь в правильности ее работы, контролируя значения переменных и последовательность выполнения в окнах редакторов CoDeSys.

3 Визуализация реализуемой программы

С помощью визуализации можно быстро и легко оживить переменные проекта. Полное описание визуализации Вы найдете в [1]. Сейчас мы нарисуем два светофора и их выключатель, который позволит нам включать и выключать блок управления светофором. Создание новой визуализации Для того чтобы создать визуализацию, выберите вкладку ‘Визуализации’ *(Visualizations)* в организаторе объектов. Теперь выполните команду ‘Проект’ ‘Объект - Добавить’ (*'Project' 'Object Add'*). Диалог для создания новой визуализации:



Введите любое имя для визуализации, например Lights. Когда Вы нажмете кнопку Ok, откроется окно, в котором вы будете создавать визуализацию.

Вставка элемента в визуализацию. Для создания визуализации светофора выполните следующие действия: · Выберите команду ‘Вставка’ ‘Эллипс’ (*'Insert' 'Ellipse'*) и нарисуйте окружность с диаметром около 2 сантиметров. Для этого щелкните мышью на рабочем поле и, удерживая левую кнопку мыши, растяните появившуюся окружность до требуемого размера. · Дважды щелкните мышью на окружности. Появится диалоговое окно для настройки элемента визуализации. · Выберите категорию ‘Переменные’ (Variables) и в поле ‘Изм. цвета’ (Change color) введите имя переменной .L1_red. Вводить имя переменной удобно с помощью Ассистента Ввода (Input Assistant) (клавиша <F1>) (рисунок 8). Глобальная переменная L1_red будет управлять цветом нарисованной Вами окружности. ·

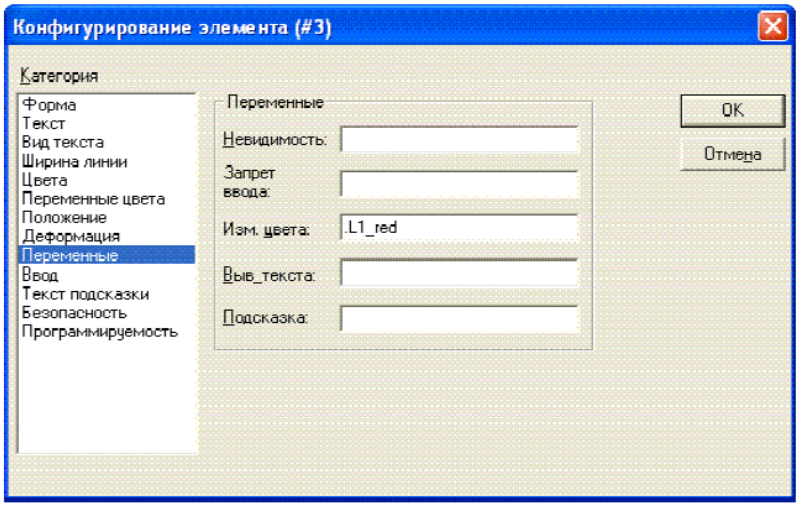


Рисунок 8

Выберите категорию ‘Цвета’ (Colors). В области ‘Цвета’ (Color) нажмите кнопку ‘Заливка’ (Inside) и в появившемся окне выберите любой нейтральный цвет, например, черный. Нажмите кнопку ‘Заливка’ (Inside) в области ’Тревожный цвет’ (Alarm Color) и выберите красный цвет (рисунок 9).

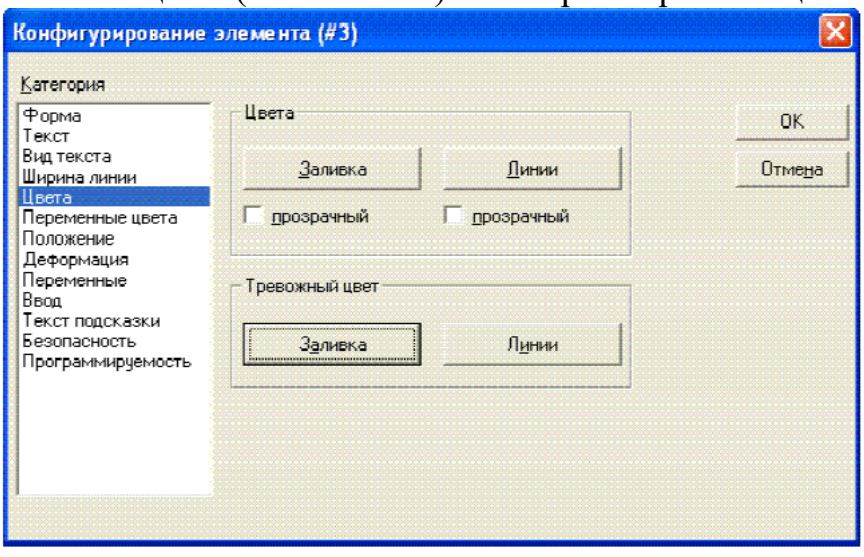


Рисунок 9

Полученная окружность будет черной, когда значение переменной ложно, и красной, когда переменная истинна. Таким образом, мы создали первый фонарь первого светофора.

Остальные цвета светофора. Теперь вызовите команду ‘Правка’ ‘Копировать’ ('Edit' 'Copy') (<Ctrl>+<C>) и дважды выполните команду ‘Правка’ ‘Вставить’ ('Edit' 'Paste') (<Ctrl>+<V>). Вы получите две новых окружности. Перемещать эти окружности можно с помощью мышки. Расположите их так, чтобы они представляли собой вертикальный ряд в левой части окна редактора. Двойной щелчок по окружности приводит к открытию окна для настройки свойств элемента визуализации. В поле ‘Изм. цвета’ (Change Color) диалога ‘Переменные’ (Variables) окон настройки свойств соответствующих окружностей введите следующие переменные:

для средней окружности: .L1_yellow

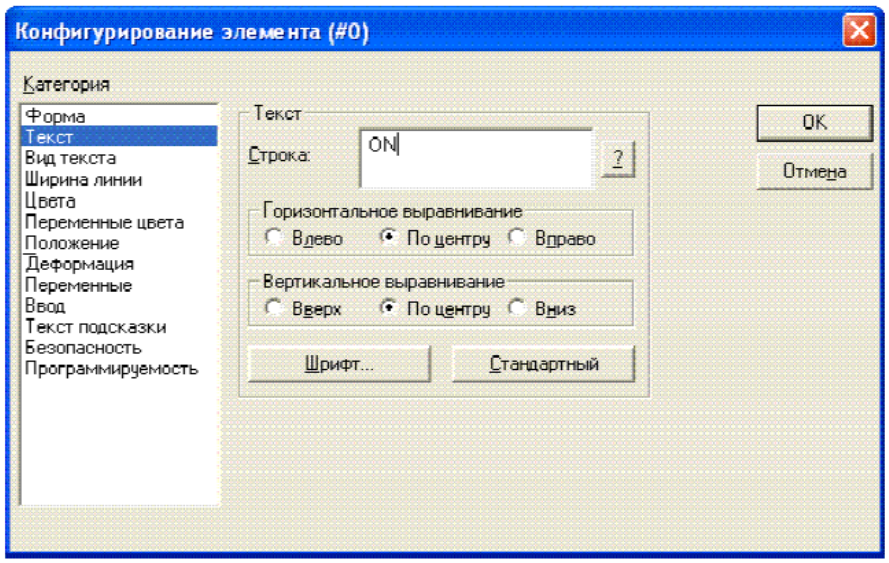
для нижней окружности: .L1_green

В категории ‘Цвета’ (Colors) в области ‘Тревожный цвет’ (Alarm color) установите цвета окружностей (желтый и зеленый).

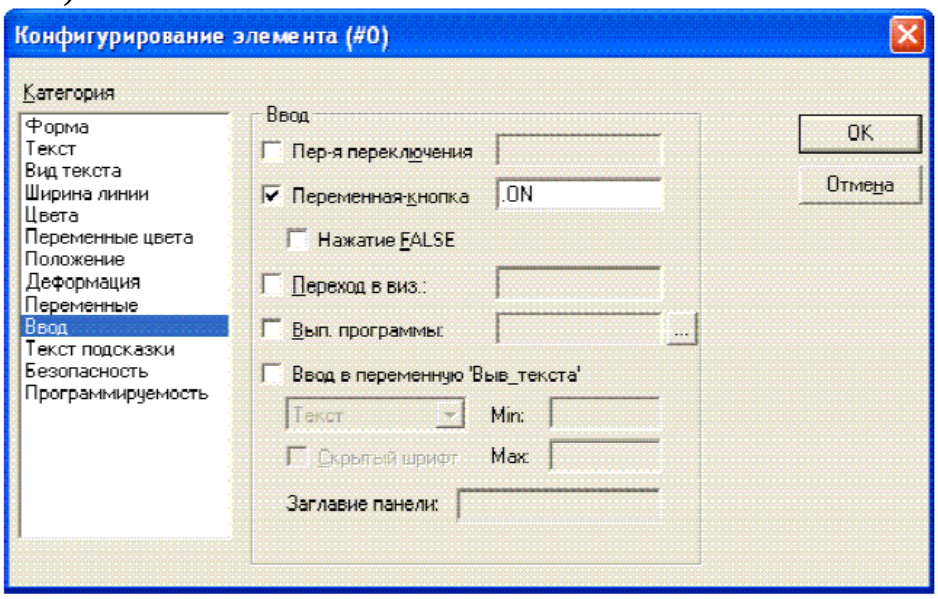
Корпус светофора. Теперь вызовите команду ‘Вставка’ ‘Прямоугольник’ (“Insert” “Rectangle”) и вставьте прямоугольник так, чтобы введенные ранее окружности находились внутри него. Выберите цвет прямоугольника и затем выполните команду ‘Дополнения’ ‘На задний план’ (“Extras” “Send to back”), которая переместит его на задний план. После этого окружности снова будут видны. Активизируйте режим эмуляции, выполнив команду ‘Онлайн’ ‘Режим эмуляции’ – ‘Online’ “Simulation mode”(режим эмуляции активен, если перед пунктом ‘Режим эмуляции’ стоит галочка). Запустите программу путем выполнения команд ‘Онлайн’ ‘Подключиться’ (‘Online’ ‘Login’) и ‘Онлайн’ ‘Старт’ (‘Online’ ‘Run’) и вы увидите, как будут меняться цвета светофора.

Второй светофор. Самый простой способ создать второй светофор – скопировать все элементы первого. Выделите элементы первого светофора и скопируйте их, выполнив команды ‘Правка’ ‘Копировать’ (“Edit” “Copy”) и ‘Правка’ ‘Вставить’ (“Edit” “Paste”). Замените имена переменных, управляющих цветами (например, .L1_red на .L2_red), и второй светофор будет готов.

Переключатель ON. Как описано выше, вставьте прямоугольник, установите его цвет и введите переменную .ON в поле ‘Изм. цвета’ (Change Color) категории ‘Переменные’ (Variables). В поле ‘Строка’ (Content) категории ‘Текст’ (Text) введите имя “ON”.



Для того чтобы переменная ON переключалась при щелчке мышкой на этом элементе, в поле ‘Переменная переключения’ (Toggle variable) категории ‘Ввод’ (Input) введите переменную .ON. Созданный нами переключатель будет включать/выключать светофоры. Отобразить включенное состояние можно цветом, как и для светофора. Впишите переменную в поле ‘Изм. цвета’ (Change Color).



Надписи в визуализации. Под светофорами вставим два прямоугольника. В свойствах элемента в категории ‘Цвета’ (Colors) цвет линии (frame) прямоугольника задайте белым. В поле ‘Строка’ - Contents (категория ‘Текст’ - Text) введите названия светофоров “Light1” “ Light2” (рисунок 10).

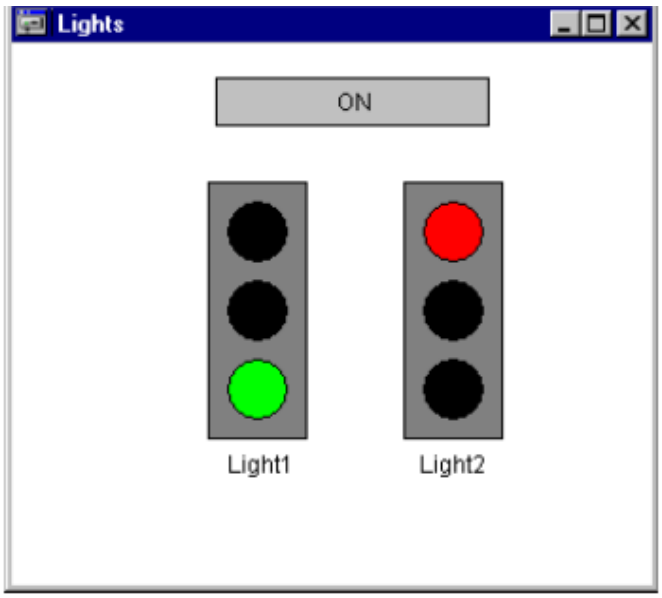


Рисунок 10- Визуализация для проекта Traffic Signal

Задание для самостоятельной работы

Выполнить все примеры программ. Скриншоты экрана с программным кодом и результатами работы представить в отчете. Сделать выводы о целесообразности разных подходов при разработке программного обеспечения контроллеров

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Водовозов, А. М. Микроконтроллеры для систем автоматики [Электронный ресурс] : учебное пособие / А. М. Водовозов. — Электрон. текстовые данные. — М. : Инфра-Инженерия, 2016. — 164 c. — 978-5-9729-0138-8. — Режим доступа: http://www.iprbookshop.ru/51727.html

2.Основы программирования микропроцессорных контроллеров в цифровых системах управления технологическими процессами [Электронный ресурс] : учебное пособие / В. С. Кудряшов, А. В. Иванов, М. В. Алексеев [и др.]. — Электрон. текстовые данные. — Воронеж : Воронежский государственный университет инженерных технологий, 2014. — 144 c. — 978-5-00032-054-9. — Режим доступа: http://www.iprbookshop.ru/47437.html

3.Шегал, А. А. Применение программного комплекса Multisim для проектирования устройств на микроконтроллерах [Электронный ресурс] : лабораторный практикум / А. А. Шегал ; под ред. В. И. Иевлев. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2014. — 116 c. — 978-5-7996-1117-0. — Режим доступа: http://www.iprbookshop.ru/65968.html

4. Алиев, М. Т.
    Микропроцессоры и микропроцессорные системы управления. 8-разрядные процессоры семейства AVR : лабораторный практикум / М.Т. Алиев, Т.С. Буканова ; Поволжский государственный технологический университет. - Йошкар-Ола : ПГТУ, 2016. - 64 с. : схем., табл., ил. - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 978-5-8158-1775-3, экземпляров неограничено

5. Безуглов, Д. А. Цифровые устройства и микропроцессоры : учеб. пособие / Д. А. Безуглов, И. В. Калиенко. - Изд. 2-е. - Ростов н/Д : Феникс, 2008. - 468 с. : ил. - (Высшее образование). - Библиогр.: с. 464-465. - ISBN 978-5-222-13917-2

6.Хартов, В. Я. Микроконтроллеры AVR : практикум для начинающих : учеб. пособие / В. Я. Хартов. - М. : МГТУ, 2007. - 240 с. : ил. - Прил.: с. 231-238. - Библиогр.: с. 239. - ISBN 978-5-7038-3051-2

7.Руководство пользователя по программированию ПЛК в CoDeSys 2.3//https://owen.ru/product/codesys_v2/documentation

8. Баженов, А.В. Цифровые методы реализации пространственно-временной обработки сигналов: Учебное пособие / А.В. Баженов, Н.В. Гривенная, М.Ф. Горяинов, С.В. Малыгин – Ставрополь: ТИС, 2016. – 219 с., ил.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕУЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы
по дисциплине «ВСТРАИВАЕМЫЕ ПРОГРАММИРУЕМЫЕ СИСТЕМЫ»
для студентов направления подготовки
11.04.04 Электроника и наноэлектроника
Направленность (профиль)
Интеллектуальные программируемые системы и устройства

Ставрополь
2026

Содержание

ВВЕДЕНИЕ

Современные стандарты подготовки магистрантов предусматривают значительный объем внеаудиторной работы. Освоение программы курса предполагает получение как теоретических знаний по проблемам социальной психологии, так и формирование навыков практической работы. Поэтому самостоятельная работа в рамках курса ориентирована как на теоретический, так и на практический аспект.

Самостоятельная работа студентов – это выполнение теоретических и практических заданий студентами по усвоению изучаемой дисциплины «Программирование встраиваемых микропроцессорных систем»

Целью самостоятельной работы является закрепление и углубление знаний, полученных студентами на лекциях, подготовке к текущим практическим занятиям, промежуточным формам контроля знаний и к итоговому контролю.

Дидактические цели самостоятельных занятий:

- формирование профессиональных умений;
- формирование умений и навыков самостоятельного умственного труда;
- мотивирование регулярной целенаправленной работы по освоению специальности;
- развитие самостоятельности мышления;
- формирование убежденности, волевых черт характера, способности к самоорганизации;
- овладение технологическим учебным инструментом.

Самостоятельная работа включает те разделы курса, которые не получили достаточного освещения на лекциях по причине ограниченности лекционного времени и большого объема изучаемого материала.

Методическое обеспечение самостоятельной работы по дисциплине состоит из:

1) Определения учебных вопросов, которые студенты должны изучить самостоятельно;

2) Подбора необходимой учебной литературы, обязательной для проработки и изучения;

3) Поиска дополнительной научной литературы, к которой студенты могут обращаться по желанию, если у них возникает интерес в данной теме;

4) Определения контрольных вопросов, позволяющих студентам самостоятельно проверить качество полученных знаний;

5) Организации консультаций преподавателя со студентами для разъяснения вопросов, вызвавших у обучающихся затруднения при самостоятельном освоении учебного материала.

Самостоятельная работа студента – это особым образом организованная деятельность, включающая в свою структуру такие компоненты, как:

- уяснение цели и поставленной учебной задачи;
- четкое и системное планирование самостоятельной работы;
- поиск необходимой учебной и научной информации;
- освоение собственной информации и ее логическая переработка;
- использование методов исследовательской, научно-исследовательской работы для решения поставленных задач;
- выработка собственной позиции по поводу полученной задачи;
- представление, обоснование и защита полученного решения;
- проведение самоанализа и самоконтроля

Виды самостоятельных работ по учебной дисциплине:

- работа с понятийным аппаратом;
- конспектирование первоисточников;
- проектирование цифровых устройств на ПЛИС;
- анализ научных исследований;

– подготовка научного проекта по теме.

Контроль знаний студентов включает формы текущего и итогового контроля. Текущий контроль осуществляется в процессе изучения курса и включает в себя оценку работы студентов на практических занятиях (круглый стол, собеседование, групповые научные проекты, тестирование), а также подготовку домашнего задания.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

Комплексное изучение предлагаемой студентам учебной дисциплины «Социальная психология» предполагает овладение материалами лекций, учебников, программы, творческую работу студентов в ходе проведения практических занятий, а также систематическое выполнение заданий для самостоятельной работы.

В ходе лекций раскрываются основные вопросы в рамках рассматриваемых тем, делаются акценты на наиболее сложные и интересные положения изучаемого материала, которые должны быть приняты студентами во внимание. Материалы лекций являются основой для подготовки студентов к практическим занятиям.

Работа с конспектом лекций

Просмотрите конспект сразу после занятий. Отметьте материал конспекта лекций, который вызывает затруднения для понимания. Попытайтесь найти ответы на затруднительные вопросы, используя предлагаемую литературу. Если самостоятельно не удалось разобраться в материале, сформулируйте вопросы и обратитесь на текущей консультации или на ближайшей лекции за помощью к преподавателю.

Каждую неделю отводите время для повторения пройденного материала, проверяя свои знания, умения и навыки по контрольным вопросам и тестам.

Выполнение лабораторных работ по дисциплине «Программирование встраиваемых микропроцессорных систем»

На первом занятии получите у преподавателя задания по курсу, планы подготовки к лабораторным работам. Обзаведитесь всем необходимым методическим обеспечением.

Задачи лабораторных исследований:

- освоение инструментальных средств разработки встраиваемых микропроцессорных систем,
- изучение языков описания схем,
- ознакомление с современными устройствами на базе микроконтроллеров и ПЛИС,
- разработка и реализация устройств на ПЛК,
- разработка и реализация численных алгоритмов на ПЛК и ПЛИС,
- организация интерфейса между основной ЭВМ и устройством на базе микропроцессорных устройств.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО КОНСПЕКТИРОВАНИЮ ПЕРВОИСТОЧНИКОВ

Изучение и анализ литературных источников является обязательным видом самостоятельной работы студентов. Изучение литературы по избранной теме имеет своей задачей проследить характер постановки и решения определенной проблемы различными авторами, аргументацию их выводов и обобщений, провести анализ и систематизировать полученный материал на основе собственного осмысления с целью выяснения современного состояния вопроса.

Проработка отобранного материала обязательно должна идти с одновременным ведением записей прочитанного и своих замечаний. Запись может иметь как форму конспекта, так и выписок, а также картотеку положений, тезисов, идей, методик, что в дальнейшем облегчит классификацию и систематизацию полученного материала. Такого рода записи являются лучшим способом накопления и первичной обработки материал, одной из обязательных форм организации умственного труда.

Планом удобно пользоваться при подготовке к устному выступлению по выбранной теме. Каждый пункт плана должен раскрывать одну из сторон избранной темы, а весь план должен охватывать ее целиком.

Тезисы предполагают сжатое изложение основных положений текста в форме утверждения или отрицания. Они являются более совершенной формой записей и представляют основу для дискуссии. К тому же их легко запомнить.

Конспектирование – процесс мысленной переработки и письменной фиксации информации, в виде краткого изложения основного содержания, смысла какого-либо текста.

Результат конспектирования – запись, позволяющая конспектирующему немедленно или через некоторый срок с нужной полнотой восстановить полученную информацию. Конспект в переводе с латыни означает «обзор». По существу его и составлять надо как обзор, содержащий основные мысли текста без подробностей и второстепенных деталей. Конспект носит индивидуализированный характер: он рассчитан на самого автора и поэтому может оказаться малопонятным для других.

Для того чтобы осуществлять этот вид работы, в каждом конкретном случае необходимо грамотно решить следующие задачи:

1. Сориентироваться в общей композиции текста (уметь определить вступление, основную часть, заключение).
2. Увидеть логико-смысловую канву сообщения, понять систему изложения автором информации в целом, а также ход развития каждой отдельной мысли.
3. Выявить «ключевые» мысли, т. е. основные смысловые вехи, на которые «нанизано» все содержание текста.
4. Определить детализирующую информацию.
5. Лаконично сформулировать основную информацию, не перенося на письмо все целиком и дословно.

Во всяком научном тексте содержится информация 2-х видов: основная и вспомогательная. Основной является информация, имеющая наиболее существенное значение для раскрытия содержания темы или вопроса. К ней относятся: определения научных понятий, формулировки законов, теоретических принципов и т. д. Назначение вспомогательной информации – помочь читателю лучше усвоить предлагаемый материал. К этому типу информации относятся разного рода комментарии. Основную – записываем как можно полнее, вспомогательную, как правило, опускаем. Содержание конспектирования составляет переработка основной информации в целях ее обобщения и сокращения. Обобщить – значит представить ее в более общей, схематической форме, в виде тезисов, выводов, отдельных заголовков, изложения основных результатов и т. п. Читая, мы интуитивно используем некоторые слова и фразы в качестве опорных. Такие опорные слова и фразы называются ключевыми. Ключевые слова и фразы несут основную смысловую и эмоциональную нагрузку содержания текста.

Выбор ключевых слов – это первый этап смыслового свертывания, смыслового сжатия материала.

Важными требованиями к конспекту являются наглядность и обозримость записей и такое их расположение, которое давало бы возможность уяснить логические связи и иерархию понятий.

По форме *конспекты подразделяются* на формализованные и графические.

1. Формализованные (все записи вносятся в заранее подготовленные таблицы).

Это удобно, во-первых, при конспектировании материалов, когда перечень характеристик описываемых предметов или явлений более или менее постоянен, во-вторых, при подготовке единого конспекта по нескольким источникам. Особенно если есть необходимость сравнения отдельных данных. Разновидностью формализованного конспекта является запись, составленная в форме ответов на заранее подготовленные вопросы, обеспечивающие исчерпывающие характеристики однотипных предметов или явлений.

2. Графические (элементы конспектируемой работы располагаются в таком виде, при котором видна иерархия понятий и взаимосвязь между ними).

По каждой работе может быть не один, а несколько графических конспектов, отображающих книгу в целом и отдельные ее части. Ведение графического конспекта - наиболее совершенный способ изображения внутренней структуры книги, а сам этот процесс помогает усвоению ее содержания.

Можно выделить следующие основные **типы конспектов: плановый, текстуальный, сводный, тематический.**

Плановый – легко получить с помощью предварительно сделанного плана произведения, каждому вопросу плана отвечает определенная часть конспекта:

а) вопросно-ответный (на пункты плана, выраженные в вопросительной форме, конспект дает точные ответы);

б) схематичный плановый конспект (отражает логическую структуру и взаимосвязь отдельных положений).

Текстуальный – это конспект, созданный в основном из цитат.

Сводный конспект – сочетает выписки, цитаты, иногда тезисы; часть его текста может быть снабжена планом.

Тематический – дает более или менее исчерпывающий ответ (в зависимости из числа привлеченных источников и другого материала, например, своих же записей) на поставленный вопрос – тему: обзорный; хронологический.

Роль конспекта – чисто учебная: он помогает зафиксировать основные понятия и положения первичного текста и в нужный момент их воспроизвести, например, при написании реферата или подготовке к экзамену.

Способы конспектирования.

- *Тезисы* – это кратко сформулированные основные мысли, положения изучаемого материала. Тезисы лаконично выражают суть читаемого, дают возможность раскрыть содержание. Приступая к освоению записи в виде тезисов, полезно в самом тексте отмечать места, наиболее четко формулирующие основную мысль, которую автор доказывает (если, конечно, это не библиотечная книга). Часто такой отбор облегчается шрифтовым выделением, сделанным в самом тексте.

- *Линейно-последовательная запись текста.* При конспектировании линейно – последовательным способом целесообразно использование плакатно-оформительских средств, которые включают в себя следующие:

- сдвиг текста конспекта по горизонтали, по вертикали;
- выделение жирным (или другим) шрифтом особо значимых слов;
- использование различных цветов;
- подчеркивание;
- заключение в рамку главной информации.

- С***пособ «вопросов – ответов».*** Он заключается в том, что, поделив страницу тетради пополам вертикальной чертой, конспектирующий в левой части страницы самостоятельно формулирует вопросы или проблемы, затронутые в данном тексте, а в правой части дает ответы на них.

Одна из модификаций способа «вопросов – ответов» - таблица, где место вопроса занимает формулировка проблемы, поднятой автором (лектором), а место ответа – решение данной проблемы. Иногда в таблице могут появиться и дополнительные графы: например, «мое мнение» и т. п.

- С***хема с фрагментами –*** способ конспектирования, позволяющий ярче выявить структуру текста, - при этом фрагменты текста (опорные слова, словосочетания, пояснения всякого рода) в сочетании с графикой помогают созданию рационально - лаконичного конспекта.

- ***Простая схема –*** способ конспектирования, близкий к схеме с фрагментами, объяснений к которой конспектирующий не пишет, но должен уметь давать их устно. Этот способ требует высокой квалификации конспектирующего. В противном случае такой конспект нельзя будет использовать.

- ***Параллельный способ*** конспектирования. Конспект оформляется на двух листах параллельно или один лист делится вертикальной чертой пополам и записи делаются в правой и в левой части листа.

Однако лучше использовать разные способы конспектирования для записи одного и того же материала.

Комбинированный конспект – вершина овладения рациональным конспектированием. При этом умело используются все перечисленные способы, сочетая их в одном конспекте (один из видов конспекта свободно перетекает в другой в зависимости от конспектируемого текста, от желания и умения конспектирующего). Именно при комбинированном конспекте более всего проявляется уровень подготовки и индивидуальность студента.

Принципы составления конспекта прочитанного.

1. Записать все выходные данные источника: автор, название, год и место издания. Если текст взят из периодического издания (газеты или журнала), то записать его название, год, месяц, номер, число, место издания.

2. Выделить поля слева или справа, можно с обеих сторон. Слева на полях отмечаются страницы оригинала, структурные разделы статьи или книги (названия параграфов, подзаголовки и т. п.), формулируются основные проблемы. Справа – способы фиксации прочитанной информации.

Один из видов чтения – углубленное – предполагает глубокое усвоение прочитанного и часто сохранение информации в целях последующего обращения к ней. Эффективность такого чтения повышается, если прочитанное зафиксировано не только в памяти, но и на бумаге. Психологи утверждают, что записанное лучше и полнее усваивается, прочнее откладывается в памяти. Установлено, что если прочитать 1000 слов и затем записать 50, подытоживающих прочитанное, то коэффициент усвоения будет выше, чем, если прочитать 10000 слов, не записав ни одного. Кроме того, при записи прочитанного формируется навык свертывания информации. И, наконец, чередование чтения и записывания уменьшает усталость, повышает работоспособность и производительность умственного труда.

1. Резюмирование.

Резюме – краткий итог прочитанного, содержащий его оценку. Резюме характеризует основные выводы книги, главные итоги.

Выбор языковых средств для построения резюме-выводов подчинен основной задаче свертывания информации: минимум языковых средств – максимум информации. Это обычно одно – три четких, кратких, выразительных предложения, раскрывающих, по мнению автора, самую суть описываемого объекта.

2. Аннотация.

Аннотация – краткая обобщенная характеристика печатной работы (книги, статьи), включающая иногда и его оценку. Это наикратчайшее изложение содержания первичного документа, дающее общее представление о теме.

Основное ее назначение – дать некоторое представление о книге (статье, научной работе) с тем, чтобы рекомендовать ее определенному кругу читателей или воспользоваться своими записями при выполнении работы исследовательского, реферативного характера. Поэтому аннотации не требуется изложения содержания произведения, в ней лишь перечисляются вопросы, которые освещены в первоисточнике (содержание этих вопросов не раскрывается). Аннотация отвечает на вопрос: «О чем говорится в первичном тексте?», дает представление только о главной теме и перечне вопросов, затрагиваемых в тексте первоисточника.

Текст аннотации не стандартизирован. В научной литературе можно встретить различные требования к составлению аннотаций. Например, текст справочной аннотации может включать следующие сведения:

· тип и название аннотируемого документа (монография, диссертация, сборник, статья и т. п.)

· задачи, поставленные автором аннотируемого документа

· метод, которым пользовался автор (эксперимент, сравнительный анализ, компиляция других источников)

· принадлежность автора к определенной научной школе или направлению

· структуру аннотируемого документа

· предмет и тему произведения, основные положения и выводы автора

· характеристику вспомогательных иллюстративных материалов, дополнений, приложений, справочного аппарата, включая указатели и библиографию.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПОДГОТОВКЕ К ЭКЗАМЕНУ

Контроль и оценка знаний студентов является неотъемлемой составной частью учебно-воспитательного процесса в вузе. Экзаменационная сессия – самая ответственная пора в студенческой жизни: она является итогом большой работы студентов в семестре. Экзамен – это метод проверки знаний студентов по части или полному курсу учебной дисциплины, произведенный путем постановки устных или письменных вопросов. Он дает объективную официально фиксируемую оценку успехов студентов за определенный отрезок времени.

Экзамен преследует многогранную цель: во-первых, это – проверка знаний студента, во-вторых, они сами по себе являются важным звеном в овладении наукой, в-третьих, это продолжение учебного процесса; наконец, он имеют большое значение как фактор стимулирования глубокого изучения предмета.

Подготовка к ЭКЗАМЕНУ состоит из двух взаимосвязанных этапов. Первый – систематический труд на протяжении семестра, учебного года, охватывающий все формы учебного процесса: лекции, изучение и конспектирование рекомендованной литературы, активное участие в семинарских (практических) и лабораторных занятиях.

Второй – подготовка непосредственно перед зачетом с оценкой. Она позволяет студентам за сравнительно короткий отрезок времени охватить всю перспективу изученного и лучше понять основные закономерности и явления.

Для непосредственной подготовки к ЭКЗАМЕНУ дополнительного времени не выделяется.

Готовиться надо по строго продуманному графику, последовательно переходя от темы к теме, не пропуская ни одну из них.

Сложные вопросы, недостаточно уясненные в процессе подготовки к зачету, необходимо записать и получить на них разъяснения у преподавателей во время предэкзаменационных консультаций.

Следует заметить, что студенты, которые не ходят на консультации из-за отсутствия, по их мнению, сложных вопросов, поступают опрометчиво. На консультациях очень часто лектор не только отвечает на заданные вопросы, но и по собственной инициативе разъясняет наиболее трудные разделы курса.

Итак, основной задачей подготовки студентов к ЭКЗАМЕНУ является систематизация знаний учебного материала, его творческое осмысливание. Важнейшим учебным пособием на этом этапе работы студента является собственный конспект прослушанных лекций и самостоятельно проработанных тем курса.

В соответствии с положением об экзаменах их, как правило, принимают преподаватели, читающие курс лекций.

Получив билет, студент должен хорошо продумать содержание поставленных вопросов. Значительное число неудачных ответов объясняется неясным пониманием поставленной проблемы. Правильное понимание вопроса обеспечит успех при ответе на него. При подготовке к ответу на билет нужно составить развернутый план по каждому вопросу

Отвечая на вопросы билета, не следует спешить. Надо излагать материал спокойно, не торопясь, владеть собой, следить за построением фраз. Следует избегать подходов издалека, общих рассуждений. Рекомендуется строить ответы четко, последовательно, конкретно, по возможности исчерпывающе. Вместе с тем весьма желательно быстро и правильно иллюстрировать свой ответ примерами.

От экзаменующегося требуется: определение понятий, обоснование выдвинутых положений, свободное оперирование фактическим материалом, дать альтернативные подходы по отдельным проблемам. Логичность, стройность, литературная грамотность изложения являются неотъемлемыми чертами полноценного ответа. Нельзя при ответе допускать ни излишней краткости, переходящей в схематизм, ни многословия. И то, и другое не оправдано. Краткость не дает преподавателю возможности понять, владеет ли студент учебным материалом, а многословие может показать, что студент не умеет акцентировать внимание на главном и говорит слишком расплывчато.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Перечень основной литературы:

1. Сергеев, А. И. Программирование контроллеров систем автоматизации: учебное пособие / А.И. Сергеев, А.М. Черноусова, А.С. Русяев ; Министерство образования и науки Российской Федерации ; Оренбургский Государственный Университет. - Оренбург : ОГУ, 2017. - 126 с. : схем., табл., ил. - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 978-5-7410-1649-7, экземпляров неограничено

2. Основы программирования микропроцессоров Intel для встраиваемых систем : учебное пособие / С.В. Скороход, В.В. Селянкин, С.Н. Дроздов, Д.П. Калачев, Н.Ш. Хусаинов ; Министерство образования и науки РФ ; Южный федеральный университет ; Инженерно-технологическая академия. - Таганрог : Издательство Южного федерального университета, 2016. - 82 с. : табл. - http://biblioclub.ru/. - Библиогр. в кн. - ISBN 978-5-9275-2223-1, экземпляров неограничено

3.Основы программирования микропроцессорных контроллеров в цифровых системах управления технологическими процессами [Электронный ресурс] : учебное пособие / В. С. Кудряшов, А. В. Иванов, М. В. Алексеев [и др.]. — Электрон. текстовые данные. — Воронеж : Воронежский государственный университет инженерных технологий, 2014. — 144 c. — 978-5-00032-054-9. — Режим доступа: http://www.iprbookshop.ru/47437.html

8.1.2. Перечень дополнительной литературы:

1. Мясников, В. И. Программное обеспечение встраиваемых систем : лабораторный практикум / В.И. Мясников ; Поволжский государственный технологический университет. - Йошкар-Ола : ПГТУ, 2018. - 148 с. : табл., ил., схем. - http://biblioclub.ru/. - Библиогр. в кнБиблиогр.: с. - ISBN 978-5-8158-1929-0, экземпляров неограниченно

2.Хартов, В. Я. Микроконтроллеры AVR : практикум для начинающих : учеб. пособие / В. Я. Хартов. - М. : МГТУ, 2007. - 240 с. : ил. - Прил.: с. 231-238. - Библиогр.: с. 239. - ISBN 978-5-7038-3051-2

8.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Баженов, А.В. Методические рекомендации по выполнению лабораторных работ по дисциплине " Программирование встраиваемых микропроцессорных систем", Ставрополь: СКФУ, 2022.- 125 с.

4. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)

1 http://elibrary.rsl.ru
2 http://www.edu.ru
3 http://www.ict.edu.ru
4 http://www.openet.edu.ru
5 www.ncfu.ru/nauchnaya-biblioteka-skfu.html
6 http://elibrary.rsl.ru