**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1

дисциплина: ЭВМ и периферийные устройства

тема: «Изучение принципов программного управления внешними устройствами на примере вывода информации на цифровой индикатор»

Выполнил: ст. группы ВТ-32

Воскобойников И. С.

Проверил: Шамраев А.А.

Белгород 2021

**Цель работы**: изучить принципы функционирования и возможности программного управления цифровым индикатором, разработать алгоритм и программу для вывода информации на цифровой индикатор.

**Задание**:

Разработать в среде программирования Code Composer программу для микроконтроллера MSP430, которая выполняет вывод информации (фамилию, имя и отчество студента, и группу) на экран цифрового индикатора.

**Теоретические сведения**

**Цифровые порты ввода-вывода**

Устройства MSP430 имеют до 6 портов цифровых входов/выходов от Р1 до Р6. Каждый порт имеет 8 выводов входа/выхода. Каждый вывод индивидуально конфигурируется как вход или выход и каждая линия ввода/вывода может быть индивидуально считана или записана.

Порты Р1 и Р2 имеют возможность вызывать прерывание. Для каждой линии ввода/вывода портов Р1 и Р2 можно индивидуально разрешить прерывания и сконфигурировать их так, чтобы прерывание происходило по фронту или спаду входного сигнала.

**Регистры ввода PxIN**

Каждый бит в каждом регистре PxIN отражает величину входного сигнала на соответствующей ножке ввода/вывода, когда она сконфигурирована на функцию ввода.

Бит = 0: Входной сигнал имеет низкий уровень; Бит = 1: Входной сигнал имеет высокий уровень. **Регистры вывода PxOUT**

Каждый бит в каждом регистре PxOUT содержит значение, которое будет выведено на соответствующую ножку ввода/вывода, сконфигурированную на функцию вывода и имеющую направление на вывод.

Бит = 0: Выходной сигнал имеет низкий уровень;

Бит = 1: Выходной сигнал имеет высокий уровень.

**Регистры направления PxDIR**

Каждый бит в каждом регистре PxDIR позволяет выбрать направление соответствующей ножки ввода/вывода, независимо от выбранной для этой ножки функции. Биты PxDIR для ножек ввода/вывода, выбранные для других функций модуля должны быть установлены так, как это требуется для другой функции.

Бит = 0: Ножка порта переключается на ввод; Бит = 1: Ножка порта переключается на вывод. **Регистры выбора функции PxSEL**

Ножки порта часто мультиплексированы с другими функциями периферийных модулей. Каждый бит PxSEL определяет, как будет использована ножка – в качестве порта ввода/вывода или в качестве функции периферийного модуля.

Бит = 0: Для ножки выбирается функция ввода/вывода

Бит = 1: Для ножки выбирается функция периферийного модуля Установка PxSEL=1 автоматически не определяет направление передачи

информации для ножки. Некоторые функции периферийных модулей требуют конфигурирования битов PxDIR для выбора направления, необходимого для правильной работы этой функции.

**Прерывания Р1 и Р2**

Каждая ножка портов Р1 и Р2 имеет возможность вызова прерывания, конфигурируемую регистрами PxIFG, PxIE и PxIES. Все ножки Р1 – источник одного вектора прерывания, а все выводы Р2 – источник другого одиночного вектора прерывания. Определить источник прерывания – Р1 или Р2 можно путем проверки регистра PxIFG.

**Регистры флагов прерывания Р1IFG, Р2IFG**

Каждый бит PxIFG – это флаг прерывания соответствующей ножки ввода/вывода, устанавливаемый, когда происходит перепад выбранного входного сигнала на ножке. Все флаги прерывания PxIFG запрашивают прерывание, когда установлен соответствующий им бит в PxIE и установлен бит GIE. Каждый флаг PxIFG должен быть сброшен программно. Программное обеспечение также может устанавливать каждый флаг PxIFG, обеспечивая возможность генерации программно-инициированного прерывания.

Бит = 0: Прерывание не ожидается Бит = 1: Прерывание ожидается

Прерывания вызывают только перепады уровней, а не статические уровни. Если любой флаг PxIFG оказывается установленным во время выполнения процедуры обработки прерывания Px или устанавливается после команды RETI выполняемой процедуры обработки прерывания Px, установка флага PxIFGx генерирует другое прерывание. Таким образом, гарантируется, что каждый перепад уровня будет учтен.

Запись в P1OUT, P1DIR, P2OUT или P2DIR может привести к установке соответствующих флагов P1IFG или P2IFG.

Любое событие вызова внешнего прерывания должно иметь длительность, по крайней мере, равную 1,5 MCLK или дольше, чтобы быть гарантировано принятым и вызвать установку соответствующего флага прерывания.

**Регистры выбора фронта прерывания P1IES, P2IES**

Каждый бит PxIES позволяет выбрать, по какому фронту сигнала будет происходить прерывание для соответствующей ножки ввода/вывода.

Бит = 0: Флаг PxIFG устанавливается при изменении уровня сигнала с низкого на высокий;

Бит = 1: Флаг PxIFG устанавливается при изменении уровня сигнала с высокого на низкий.

**Разрешение прерываний P1IE, P2IE**

Каждый бит PxIE разрешает прерывание от соответствующего флага прерываний регистра PxIFG.

Бит = 0: Прерывание запрещено Бит = 1: Прерывание разрешено

**Конфигурирование неиспользуемых выводов порта**

Неиспользуемые ножки ввода/вывода должны быть сконфигурированы на функцию ввода/вывода, в направлении вывода и оставаться неподключенными на печатной плате для уменьшения потребляемой мощности. Значение бита PxOUT может быть любым, поскольку ножка не подключена.

**Регистры цифровых входов/выходов**

Для конфигурирования Р1 и Р2 используются семь регистров. Четыре регистра необходимы для конфигурирования портов Р3-Р6. Регистры цифровых портов ввода-вывода приведены в таблице 4.1.1.

Таблица 4.1.1 – Регистры цифровых портов ввода-вывода

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Порт** | **Регистр** | **Обозна- чение** | **Адрес** | **Тип регистра** | **Исходное состояние** |
| P1 | Ввод | P1IN | 020h | Только чтение | – |
| Вывод | P1OUT | 021h | Чтение/запись | Не изменяется |
| Направление | P1DIR | 022h | Чтение/запись | Сброс с PUC |
| Флаг  прерывания | P1IFG | 023h | Чтение/запись | Сброс с PUC |
| Выбор фронта  прерывания | P1IES | 024h | Чтение/запись | Не изменяется |
| Разрешение  прерывания | P1IE | 025h | Чтение/запись | Сброс с PUC |
| Выбор порта | P1SEL | 026h | Чтение/запись | Сброс с PUC |
| P2 | Ввод | P2IN | 028h | Только чтение | - |
| Вывод | P2OUT | 029h | Чтение/запись | Не изменяется |
| Направление | P2DIR | 02Ah | Чтение/запись | Сброс с PUC |
| Флаг  прерывания | P2IFG | 02Bh | Чтение/запись | Сброс с PUC |
| Выбор фронта  прерывания | P2IES | 02Ch | Чтение/запись | Не изменяется |
| Разрешение  прерывания | P2IE | 02Dh | Чтение/запись | Сброс с PUC |
| Выбор порта | P2SEL | 02Eh | Чтение/запись | Сброс с PUC |
| P3 | Ввод | P3IN | 018h | Только чтение | - |
| Вывод | P3OUT | 019h | Чтение/запись | Не изменяется |
| Направление | P3DIR | 01Ah | Чтение/запись | Сброс с PUC |
| Выбор порта | P3SEL | 01Bh | Чтение/запись | Сброс с PUC |
| P4 | Ввод | P4IN | 01Ch | Только чтение | - |
| Вывод | P4OUT | 01Dh | Чтение/запись | Не изменяется |
| Направление | P4DIR | 01Eh | Чтение/запись | Сброс с PUC |
| Выбор порта | P4SEL | 01Fh | Чтение/запись | Сброс с PUC |
| P5 | Ввод | P5IN | 030h | Только чтение | - |
| Вывод | P5OUT | 031h | Чтение/запись | Не изменяется |
| Направление | P5DIR | 032h | Чтение/запись | Сброс с PUC |
| Выбор порта | P5SEL | 033h | Чтение/запись | Сброс с PUC |
| P6 | Ввод | P6IN | 034h | Только чтение | - |
| Вывод | P6OUT | 035h | Чтение/запись | Не изменяется |
| Направление | P6DIR | 036h | Чтение/запись | Сброс с PUC |
| Выбор порта | P6SEL | 037h | Чтение/запись | Сброс с PUC |

# Принципы программного управления выводом символов на экран цифрового индикатора с помощью микроконтроллера MSP430F1611

В лабораторном стенде используется индикатор WH1602A-NGG-CT на базе контроллера HD44780. Данный ЖКИ позволяет отображать 2 строки по 16 символов. Символы отображаются в матрице 5х8 (или 5х10) точек. Между символами имеются интервалы шириной в одну отображаемую точку. Каждому отображаемому на ЖКИ символу соответствует его код в ячейке ОЗУ модуля.

Модуль содержит два вида памяти – кодов отображаемых символов и пользовательского знакогенератора, а также логику для управления ЖК- панелью.

Модуль позволяет:

* работать как по 8-ми, так и по 4-х битной шине данных (задается при инициализации);
* принимать команды с шины данных;
* записывать данные в ОЗУ с шины данных;
* читать данные из ОЗУ на шину данных;
* читать статус состояния на шину данных;
* запоминать до 8-ми изображений символов, задаваемых пользователем;
* выводить мигающий (или не мигающий) курсор двух типов;
* управлять контрастностью и подсветкой;

**Программирование и управление.** Перед началом рассмотрения принципов управления ЖКИ-модулем, обратимся к внутренней структуре контроллера HD44780.

Упрощенная структурная схема контроллера HD44780 приведена на рисунке 4.1.1. Можно сразу выделить основные элементы, с которыми приходится взаимодействовать при программном управлении: регистр данных (DR), регистр команд (IR), видеопамять (DDRAM), ОЗУ знакогенератора (CGRAM), счетчик адреса памяти (АС), флаг занятости контроллера (BF).

Другие элементы не являются объектом прямого взаимодействия с управляющей программой – они участвуют в процессе регенерации изображения на ЖКИ: знакогенератор, формирователь курсора, сдвиговые регистры и драйверы.

Управление контроллером ведется посредством интерфейса управляющей системы. Основными объектами взаимодействия являются регистры DR и IR. Выбор адресуемого регистра производится линией

D/C\_LCD, если

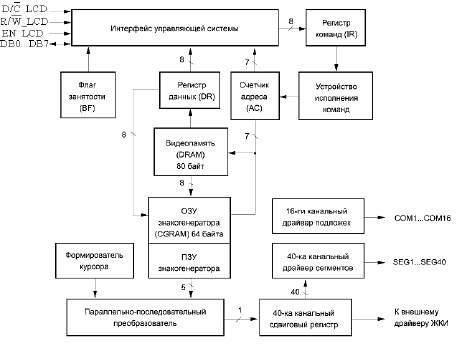
D/C\_LCD= 0 – адресуется регистр команд (IR), еслиD/C\_LCD = 1 – регистр данных (DR).

Рисунок 4.1.1 – Упрощенная структурная схема контроллера HD44780

Данные через регистр DR, в зависимости от текущего режима, могут помещаться (или прочитываться) в видеопамять (DDRAM) или в ОЗУ знакогенератора (CGRAM) по текущему адресу, указываемому счетчиком адреса (АС). Информация, попадающая в регистр IR, интерпретируется устройством выполнения команд как управляющая последовательность. При чтении регистра IR в 7-ми младших разрядах возвращается текущее значение счетчика АС, а в старшем разряде флаг занятости (BF).

Видеопамять, имеющая общий объем 80 байтов, предназначена для хранения кодов символов, отображаемых на ЖКИ. Видеопамять организована в две строки по 40 символов в каждой. Эта привязка является жесткой и не подлежит изменению. Другими словами, независимо от того, сколько реальных строк будет иметь каждый конкретный ЖКИ-модуль, скажем, 80 х 1 или 20 х 4, адресация видеопамяти всегда производится как к двум строкам по 40

символов.

Модуль WH1602A-NGG-CT содержит ОЗУ размером 32 байт по адресам 00h–0Fh и 40h–4Fh для хранения данных (DDRAM), выводимых на ЖКИ. Адреса отображаемых на ЖКИ символов распределены, как показано в таблице 4.1.2.

Таблица 4.1.2 – Адреса знакомест в памяти модуля WH1602A-NGG-CT

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер  символа | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Строка 1 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| Строка 2 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |

У контроллера HD44780 существует набор внутренних флагов, определяющих режимы работы различных элементов контроллера (таблица 4.1.3). В таблице 4.1.4 приведены значения управляющих флагов непосредственно после подачи на ЖКИ-модуль напряжения питания. Переопределение значений флагов производится специальными командами, записываемыми в регистр IR, при этом комбинации старших битов определяют группу флагов или команду, а младшие содержат собственно флаги.

Таблица 1.1.3 – Флаги, управляющие работой контроллера HD44780

|  |  |
| --- | --- |
| Флаг | Описание |
| 1 | 2 |
| I/D: | режим смещения счетчика адреса АС, 0 - уменьшение, 1 - увеличение. |
| S: | флаг режима сдвига содержимого экрана. 0 - сдвиг экрана не производится,  1 - после записи в DDRAM очередного кода экран сдвигается в направлении,  определяемым флагом I/D: 0 - вправо, 1 - влево. При сдвиге не производится  изменение содержимого DDRAM. изменяются только внутренние указатели  расположения видимого начала строки в DDRAM. |
| S/C: | флаг-команда, производящая вместе с флагом R/L операцию сдвига содержимого экрана (так же, как и в предыдущем случае, без изменений в  DDRAM) или курсора. Определяет объект смещения: 0 - сдвигается курсор,  1 - сдвигается экран. |

Продолжение таблицы 1.1.3

|  |  |
| --- | --- |
| Флаг | Описание |
| 1 | 2 |
| R/L: | флаг-команда, производящая вместе с флагом S/C операцию сдвига экрана  или курсора. Уточняет направление сдвига: 0 - влево, 1 - вправо. |
| D/L: | флаг, определяющий ширину шины данных: 0 - 4 разряда, 1 - 8  разрядов. |
| N: | режим развертки изображения на ЖКИ: 0 - одна строка, 1 - две строки |
| F: | размер матрицы символов: 0 - 5x8 точек, 1 - 5x10 точек. |
| D: | наличие изображения: 0 - выключено, 1 - включено |
| C: | курсор в виде подчерка: 0 - выключен, 1 - включен |
| B: | курсор в виде мерцающего знакоместа: 0 - выключен, 1 - включен. |

Таблица 4.1.4 – Значения управляющих флагов после подачи питания

|  |  |
| --- | --- |
| Флаг | Описание |
| I/D = 1: | режим увеличения счетчика на 1 |
| S = 0: | без сдвига изображения |
| D/L = 1: | 8-ми разрядная шина данных |
| N = 0: | режим развертки одной строки |
| F = 0: | символы с матрицей 5x8 точек |
| D = 0: | отображение выключено |
| C = 0: | курсор в виде подчерка выключен |
| B = 0: | курсор в виде мерцающего знакоместа выключен |

Список управляющих комбинаций битов регистра IR и выполняемые ими команды приведены на рисунке 4.1.2. Так как на момент включения ЖКИ- модуль ничего не отображает (флаг D = 0), то для того, чтобы вывести какой- либо текст необходимо, как минимум, включить отображение, установив флаг D = 1.

Пример последовательности для инициализации ЖКИ-модуля показан на рисунке 4.1.3: $38, $0C, $01, $06 (знак «$» перед числом указывает на шестнадцатеричное основание). $38 устанавливает режим отображения 2-х строк с матрицей 5x8 точек и работу с 8-ми разрядной шиной данных; $0C включает отображение на экране ЖКИ-модуля, без отображения курсоров; $01

– очищает дисплей и помещает курсор в левую позицию, $06 устанавливает режим автоматического перемещения курсора слева направо после вывода каждого символа.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Команда** | **Код команды** | | | | | | | | | | **Описание** | **Время выполнения (270 кГц)** |
| D/ C \_LCD | R/ W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| Clear Display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | В DDRAM записываются нули, и  курсор помещается в самую левую позицию (AC =0) | 1.53 мс |
| Return Home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **-** | Перемещает курсор в самую левую позицию (AC =0)  Содержимое . DDRAM не изменяется | 1.53 мс |
| Entry Mode Set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | SH | Установка направления сдвига курсора (I/D=0/1-влево/вправо) и  разрешение сдвига дисплея (SH=1) при записи в DDRAM | 39 мкс |
| Display ON/OFF  Control | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Включает дисплей (D=1) и выбирает тип курсора (C, B) | 39 мкс |
| Cursor or Display Shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | **-** | **-** | Выполняет сдвиг дисплея или курсора (S/C=0/1 – курсор/дисплей, R/L=0/1 – влево/вправо) | 39 мкс |
| Function Set | 0 | 0 | 0 | 0 | 1 | DL | N | F | **-** | **-** | Устанавливает разрядность интерфейса (DL=0/1 -4/8 бит), число строк (N=0/1- 1/2 строки) и размер матрицы символа (F=0/1 - 5×8 точек  / 5×11 точек) | 39мкс |
| Set CGRAM  Address | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Установка адреса для последующих операций и выбор области CGRAM | 39 мкс |
| Set DDRAM  Address | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Установка адреса для последующих операций и выбор области DDRAM. | 39 мкс |
| Read Busy  Flag and Address | 0 | 1 | BF | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Прочитать флаг занятости и содержимое счетчика адреса | 0 мкс |
| Write Data to RAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Запись данных в активную область  внутреннего ОЗУ(DDRAM/CGRAM). | 43 мкс |
| Read Data from RAM | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Чтение данных из активной области внутреннего ОЗУ  (DDRAM/CGRAM). | 43 мкс |

Рисунок 4.1.2 – Описание команд модуля ЖКИ

Контроллер HD44780 поддерживает как операции записи, так и операции чтения. Чтение регистра DR приводит к загрузке содержимого DDRAM или CGRAM, в зависимости от текущего режима, при этом курсор смещается на одну позицию, как и при записи. Чтение регистра IR возвращает 8 значащих разрядов, причем в 7-ми младших содержится текущее значение счетчика АС (7 разрядов, если адресуется DDRAM, и 6 – если CGRAM), а в старшем – флаг занятости BF. Этот флаг имеет значение 1 когда контроллер занят и 0 – когда свободен. Необходимо учитывать, что большинство операций, выполняемых контроллером, занимают значительное время, около 40 мкс, а время выполнения некоторых доходит до единиц миллисекунд, поэтому цикл

ожидания снятия флага BF должен обязательно присутствовать в программах драйвера ЖКИ-модуля и предшествовать совершению любой операции (естественно, кроме операции проверки флага BF).

Таблица 4.1.5 – Описание сигналов и выводов модуля WH1602A (LCD1)

|  |  |  |  |
| --- | --- | --- | --- |
| Вывод  LCD1 | Название | Подключение к цепи /  [номер вывода] | Примечание |
| 1 | GND | GND | Подключение к шине 0 В |
| 2 | Vcc | +5V | Напряжение питания |
| 3 | Vo | +5V | Операционное напряжение |
| 4 | Data/Command | D/C\_LCD / 32 | Режим: данные/команды |
| 5 | *R* / W | DD7 / 9 | Сигнал чтение/запись |
| 6 | EN | EN\_LCD DB0...DB7 / 33 | Строб данных |
| 7 | DB0 | DD8 / 3 | Шина данных, подключенная к преобразователю уровней DD8 |
| 8 | DB1 | DD8 / 4 |
| 9 | DB2 | DD8 / 5 |
| 10 | DB3 | DD8 / 6 |
| 11 | DB4 | DD8 / 7 |
| 12 | DB5 | DD8 / 8 |
| 13 | DB6 | DD8 / 9 |
| 14 | DB7 | DD8 / 10 |

Таблица 4.1.6 – Описание сигналов и выводов преобразователя

SN74LVC4245A

|  |  |  |  |
| --- | --- | --- | --- |
| Вывод  DD8 | Название | Подключение к цепи /  [номер вывода] | Примечание |
| 1 | 2 | 3 | 4 |
| 1 | VccA(5V) | +5V | Напряжение питания +5В |
| 2 | DIR | DD7 / 9 | Сигнал чтение/запись R/W\_LCD |
| 3 | A1 | LCD1 / 7 | Шина данных, подключенная к модулю LCD |
| 4 | A2 | LCD1 / 8 |
| 5 | A3 | LCD1 / 9 |
| 6 | A4 | LCD1 / 10 |
| 7 | A5 | LCD1 / 11 |
| 8 | A6 | LCD1 / 12 |
| 9 | A7 | LCD1 / 13 |
| 10 | A8 | LCD1 / 14 |
| 11 | GND | GND | Подключение к шине 0 В |
| 12 | GND | GND |
| 13 | GND | GND |
| 14 | B8 | DB7 / 43 | Шина данных, подключенная к  MSP430 |
| 15 | B7 | DB6 / 42 |
| 16 | B6 | DB5 / 41 |
| 17 | B5 | DB4 / 40 |
| 18 | B4 | DB3 / 39 |
| 19 | B3 | DB2 / 38 |
| 20 | B2 | DB1 / 37 |
| 21 | B1 | DB0 / 36 |
| 22 | OE | DD7 / 7 | Сигнал разрешения ввода/вывода  OE\_BF\_LCD |
| 23 | VccB(3,3V) | +3,3 V | Напряжение питания +3,3В |
| 24 | VccB(3,3V) | +3,3 V |

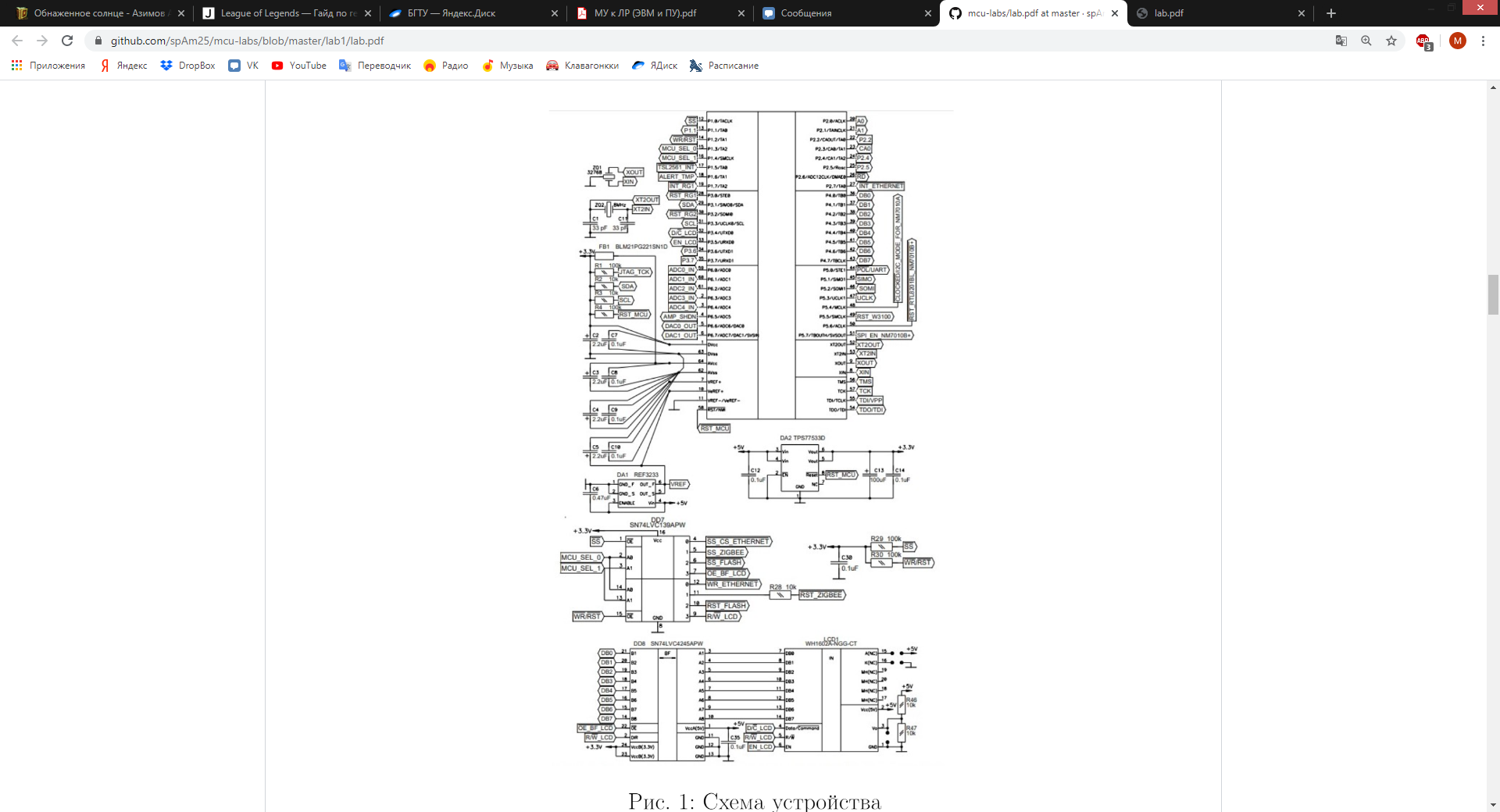
Рисунок 4.1.6 – Принципиальная схема подключения модуля ЖКИ

**Порядок выполнения работы**:

* запустить Code Composer IDE.
* создать пустой проект.
* создать файл ресурса для кода программы и подключить его к проекту.
* выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.
* проверить работоспособность программы и показать результаты работы преподавателю.

**Ход работы:**

Схема стенда



**main.c**

**#include** <msp430.h>

**#include** "stdio.h"

**#include** "system\_define.h"

**#include** "system\_variable.h"

**#include** "function\_prototype.h"

**#include** "main.h"

**#include** "lcd.h"

**#include** "sysfunc.h"

/\*

\* main.c

\*/

**void** **main**(**void**) {

**int** i;

WDTCTL = WDTPW + WDTHOLD;

Init\_System();

Init\_System\_Clock();

LCD\_init();

LCD\_clear();

LCD\_set\_cursor(2);

LCD\_WriteCommand(0x40);

**char** data[] = {

0b00000000,

0b00000100,

0b00001110,

0b00011111,

0b00001110,

0b00000100,

0b00000000,

0b00000000

};

**for** (i=0; i<8; i++)

LCD\_WriteData(data[i]);

LCD\_set\_pos(0,0);

LCD\_WriteData(0);

/\*\*

LCD\_set\_pos(0,0);

LCD\_message("Воскобойников И.");

LCD\_set\_pos(1,0);

LCD\_message("ВТ-32");

\*\*/

}

**lcd.c**

// LCD-display functions

#include "function\_prototype.h"

#include "sysfunc.h"

#include "lcd.h"

//Таблица киррилицы

char LCD\_table[64]={

0x41,0xA0,0x42,0xA1, //0xC0...0xC3 <=> А Б В Г

0xE0,0x45,0xA3,0x33, //0xC4...0xC7 <=> Д Е Ж З

0xA5,0xA6,0x4B,0xA7, //0xC8...0xCB <=> И Й К Л

0x4D,0x48,0x4F,0xA8, //0xCC...0xCF <=> М Н О П

0x50,0x43,0x54,0xA9, //0xD0...0xD4 <=> Р С Т У

0xAA,0x58,0xE1,0xAB, //0xD5...0xD7 <=> Ф Х Ц Ч

0xAC,0xE2,0xAC,0xAE, //0xD8...0xDB <=> Ш Щ Ъ Ы

0x62,0xAF,0xB0,0xB1, //0xDC...0xDF <=> Ь Э Ю Я

0x61,0xB2,0xB3,0xB4, //0xE0...0xE4 <=> а б в г

0xE3,0x65,0xB6,0xB7, //0xE5...0xE7 <=> д е ж з

0xB8,0xA6,0xBA,0xBB, //0xE8...0xEB <=> и й к л

0xBC,0xBD,0x6F,0xBE, //0xEC...0xEF <=> м н о п

0x70,0x63,0xBF,0x79, //0xF0...0xE4 <=> р с т у

0xE4,0xD5,0xE5,0xC0, //0xF5...0xE7 <=> ф х ц ч

0xC1,0xE6,0xC2,0xC3, //0xF8...0xEB <=> ш щ ъ ы

0XC4,0xC5,0xC6,0xC7 //0xFC...0xEF <=> ь э ю я

};

byte LCD\_row, LCD\_col, n;

void LCD\_init()

{

wait\_1ms(20);

P3DIR |= (D\_nC\_LCD + EN\_LCD);

Reset\_EN\_LCD();

LCD\_WriteCommand(0x3C);

wait\_1ms(1);

LCD\_WriteCommand(0x3C);

wait\_1ms(1);

LCD\_WriteCommand(0x0C);

LCD\_clear();

LCD\_WriteCommand(0x06);

}

void LCD\_message(const char \* buf)

{

n = 0;

while (buf[n])

{

if ( (LCD\_row < LCD\_MAXROWS-1) && (LCD\_col >= LCD\_MAXCOLS) )

LCD\_set\_pos(++LCD\_row, 0);

if (LCD\_col >= LCD\_MAXCOLS )

LCD\_set\_pos(0,0);

LCD\_WriteData( LCD\_recode(buf[n]) );

LCD\_col++;

n++;

}

}

void LCD\_clear()

{

LCD\_WriteCommand(0x01);

LCD\_row=0;

LCD\_col=0;

}

void LCD\_set\_pos(byte row, byte col)

{

if (row > LCD\_MAXROWS-1)

row = LCD\_MAXROWS-1;

if (col > LCD\_MAXCOLS-1)

col = LCD\_MAXCOLS-1;

LCD\_row = row;

LCD\_col = col;

LCD\_WriteCommand( BIT7 | ((0x40 \* LCD\_row) + LCD\_col) );

}

byte LCD\_get\_row()

{

return LCD\_row;

}

byte LCD\_get\_col()

{

return LCD\_col;

}

void LCD\_set\_cursor(byte cursor)

{

if (cursor > 3)

cursor = 2;

LCD\_WriteCommand(cursor | BIT2 | BIT3);

}

void LCD\_WriteCommand(char byte)

{

LCD\_WriteByte(byte, 0);

}

void LCD\_WriteData(char byte)

{

LCD\_WriteByte(byte, 1);

}

void LCD\_WriteByte(char byte, char D\_nC)

{

DB\_DIR = 0x00;

Set\_MCU\_SEL\_0();

Set\_MCU\_SEL\_1();

Reset\_D\_nC\_LCD();

Set\_nWR\_nRST();

Reset\_nSS();

Set\_EN\_LCD();

Set\_EN\_LCD();

Set\_EN\_LCD();

while (DB\_IN & BIT7);

Reset\_EN\_LCD();

Set\_nSS();

if (D\_nC) Set\_D\_nC\_LCD();

else Reset\_D\_nC\_LCD();

Reset\_nWR\_nRST();

Reset\_nSS();

DB\_DIR = 0xFF;

DB\_OUT = byte;

Set\_EN\_LCD();

Set\_EN\_LCD();

Set\_EN\_LCD();

Reset\_EN\_LCD();

Set\_nSS();

DB\_DIR = 0x00;

Set\_nWR\_nRST();

}

char LCD\_recode(char b)

{

if (b<192) return b;

else return LCD\_table[b-192];

}

**I2C.с**

**#include** "function\_prototype.h"

**#include** "system\_define.h"

**#include** "I2C.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//============================================================================

// Инициализация модуля UART0 для работы в режиме I2C

**void** **Init\_I2C**()

{

P3SEL |= 0x0A; // Выбор альтернативной функции для линий порта P3

// в режиме I2C SDA->P3.1, SCL->P3.3

U0CTL |= I2C + SYNC; // Выбрать режим I2C для USART0

U0CTL &= ~I2CEN; // Выключить модуль I2C

// Конфигурация модуля I2C

I2CTCTL=I2CSSEL\_2; // SMCLK

I2CSCLH = 0x26; // High period of SCL

I2CSCLL = 0x26; // Low period of SCL

U0CTL |= I2CEN; // Включить модуль I2C

// формирование строба сброса I2C-регистров PCA9538 - RST\_RG1->P3.1 и RST\_RG2->P3.2

P3DIR |= 0x05; // переключаем эти ножки порта на вывод,

P3SEL &= ~0x05; // выбираем функцию ввода-вывода для них

P3OUT &= ~0x05; // и формируем строб сброса на 1 мс

wait\_1ms(1);

P3OUT |= 0x05;

}

//============================================================================

//============================================================================

// отправка данных по протоколу I2C

**void** **Send\_I2C**(**unsigned** **char**\* buffer,**unsigned** **int** num, **unsigned** **char** address)

{

**while** (I2CBUSY & I2CDCTL); // проверка готовности модуля I2C

BufTptr=buffer;

I2CSA = address; // установка адреса приемнмка

I2CNDAT =num; // количество передаваемых байт

I2CIE = TXRDYIE+ALIE; // разрешение прерываний по окончанию передачи байта и по потере арбитража

U0CTL |= MST; // режим Master

I2CTCTL |= I2CSTT + I2CSTP + I2CTRX; // инициализировать передачу

**while** ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП

}

//============================================================================

//============================================================================

// прием данных по протоколу I2C

**void** **Receive\_I2C**(**unsigned** **char**\* buffer,**unsigned** **int** num, **unsigned** **char** address)

{

**while** (I2CBUSY & I2CDCTL); // проверка готовности модуля I2C

BufRptr=buffer;

I2CSA=address;

I2CTCTL&=~I2CTRX; // режим приема

I2CNDAT=num;

I2CIE=RXRDYIE; // резрешение прерывания по окончанию приема байта

U0CTL |= MST;

I2CTCTL |= I2CSTT + I2CSTP; // инициализировать прием

**while** ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП

}

//============================================================================

//============================================================================

// отправка байта устройству на шине I2C

**void** **I2C\_SendByte**(**char** data, **char** i2c\_addr)

{

Tx\_Data[0] = data; // отправляемый байт

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr); // вывод по I2C на устройство

}

//============================================================================

//============================================================================

// запись байта в регистр устройства на шине I2C

**void** **I2C\_WriteByte**(**char** reg, **char** data, **char** i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Tx\_Data[1] = data; // записываемые данные

Send\_I2C(&Tx\_Data[0], 2, i2c\_addr); // вывод по I2C на устройство

}

//============================================================================

//============================================================================

// чтение байта из регистра устройства на шине I2C

byte **I2C\_ReadByte**(**char** reg, **char** i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr);

Receive\_I2C(&Rx\_Data[0], 1, i2c\_addr); // получаем значение из регистра

**return** Rx\_Data[0];

}

//============================================================================

//============================================================================

// чтение слова (2 байта) из регистра устройства на шине I2C

**int** **I2C\_ReadWord**(**char** reg, **char** i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr);

Receive\_I2C(&Rx\_Data[0], 2, i2c\_addr); // получаем 2 байта значение из регистра

**return** Rx\_Data[0] + (Rx\_Data[1] \* 256);

}

//============================================================================

//============================================================================

//Обработка прерывания от модуля USART0, работающего в режиме I2C

// вектор прерываний для модуля I2C

**#pragma** vector=USART0TX\_VECTOR

**\_\_interrupt** **void** **I2C\_ISR**()

{

**switch**(I2CIV)

{

**case** 0: **break**; // нет прерывания

**case** 2: **break**; // потеря арбитража

**case** 4: **break**; // нет подтверждения

**case** 6: **break**; // прерывание собственного адреса

**case** 8: **break**; // регистр доступен для чтения

**case** 10: // окончание приема байта

\*BufRptr++=I2CDRB;

**break**;

**case** 12: // окончание передачи байта

I2CDRB=\*BufTptr++;

**break**;

**case** 14: **break**; // общий вызов

**case** 16: **break**; // обнаружено условие СТАРТ

**default** : **break**;

}

}

**keys.с**

// Keyboard functions

**#include** "function\_prototype.h"

**#include** "sysfunc.h"

**#include** "keys.h"

byte keycol, keyline, KEYS\_last=0;

**char** table\_keys[12] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '\*', '0', '#'};

// Проверка нажатия клавиши в текущий момент, результат:

// 0 - клавиша не нажата

// ASCII-код клавиши

**char** **KEYS\_scannow**()

{

keyline=0;

// выбираем регистр конфигурации направления (0x03)

// и конфигурируем P4-P7 на вывод - для строба столбцов,

// а P0-P3 на ввод - для опроса строк (1-ввод, 0-вывод)

I2C\_WriteByte(0x03, 0x0F, KEYS\_i2c\_addr);

**for** (keycol=0; keycol<3; keycol++) {

// последовательно подаем сигнал низкого уровня на столбцы (P4-P7)

I2C\_WriteByte(0x01, ~(1<<keycol<<4) & 0xf0, KEYS\_i2c\_addr);

wait\_1ms(1);

// и опрашиваем строки (P0-P3) на наличие нуля

keyline = ~(I2C\_ReadByte(0x00, KEYS\_i2c\_addr)) & 0x0f;

**if** (keyline) **break**;

}

**if** (!keyline) **return** 0; // если не была нажата никакая клавиша - возвращаем 0

**if** (keyline == 4) keyline = 3; // переводим номера разрядов в номер строки

**if** (keyline == 8) keyline = 4;

KEYS\_last = table\_keys[--keyline\*3+keycol]; // получаем код нажатой клавиши из таблицы

**return** KEYS\_last;

}

// Возвращает код последней нажатой клавиши, результат:

// 0 - не нажималась никакая клавиша

// ASCII-код клавиши

**char** **KEYS\_lastkey**()

{

KEYS\_scannow();

**return** KEYS\_last;

}

// Очистка последней нажатой клавиши

**void** **KEYS\_clear**()

{

KEYS\_last = 0;

wait\_1ms(200);

}

// Ожидание нажатия клавиши, результат - ASCII-код нажатой клавиши

**char** **KEYS\_waitkey**()

{

KEYS\_clear(); // очистка последней нажатой клавиши

**while** (!KEYS\_scannow()) // пока не нажата никакая клавиша,

wait\_1ms(1); // сделать паузу

**return** KEYS\_last; // вернуть код нажатой клавиши

}

// пауза с циклическим опросом клавиатуры, прерывается если нажата клавиша

**void** **KEYS\_pause**(byte cnt)

{

byte i;

KEYS\_clear(); // очистка последней нажатой клавиши

**for** (i=0; i<cnt; i++)

**if** (KEYS\_scannow())

**break**;

}

**leds.с**

// LED-indicator functions

**#include** "function\_prototype.h"

**#include** "sysfunc.h"

**#include** "leds.h"

**char** LED\_config=0; // хранится конфигурация светодиодов (вкл/выкл)

**void** **LED\_out**(**char** leds)

{

// регистр конфигурации направления 0x03 конфигурируем на вывод информации (1-ввод, 0-вывод)

I2C\_WriteByte(0x03, 0x00, LED\_i2c\_addr);

I2C\_WriteByte(0x01, leds, LED\_i2c\_addr); // выводим данные в регистр OUTPUT (0x01)

LED\_config=leds; // сохраняем новую конфигурацию

}

// Преобразование номера светодиода в бит, с которым нужно проводить операцию

// 1 = 10000000, 2 = 01000000 ... 8 = 00000001

**char** **LED\_convert**(**char** led)

{

**if**(led<1)

led=1;

**if**(led>8)

led=8;

led=9-led;

**return** (1<<(led-1));

}

// Выключить все светодиоды

**void** **LED\_clear**()

{

LED\_out(0x00);

}

// Инвертировать все светодиоды

**void** **LED\_invert**()

{

LED\_out(LED\_config ^ 0xff);

}

// Включить светодиод с номером от 1 до 8 (слева направо)

**void** **LED\_set**(**char** led)

{

led=LED\_convert(led);

LED\_config |= led; // устанавливаем соответствующий разряд

LED\_out(LED\_config); // выводим в регистр

}

// Выключить светодиод с номером от 1 до 8 (слева направо)

**void** **LED\_reset**(**char** led)

{

led=LED\_convert(led);

LED\_config &= ~(led); // сбрасываем соответствующий разряд

LED\_out(LED\_config); // выводим в регистр

}

// Сменить состояние светодиода с номером от 1 до 8 (слева направо)

**void** **LED\_change**(byte led)

{

led=LED\_convert(led);

LED\_config ^= led; // меняем состояние соответствующего разряда (XOR)

LED\_out(LED\_config); // выводим в регистр

}

**void** **LED\_fx1**(**int** n)

{

LED\_clear();

LED\_set(1);

wait\_1ms(n);

LED\_set(3);

wait\_1ms(n);

LED\_set(5);

wait\_1ms(n);

LED\_set(7);

wait\_1ms(n);

LED\_reset(1);

wait\_1ms(n);

LED\_reset(3);

wait\_1ms(n);

LED\_reset(5);

wait\_1ms(n);

LED\_reset(7);

wait\_1ms(n);

LED\_clear();

}

**void** **LED\_fx2**(**int** n)

{

LED\_clear();

LED\_out(0x81);

wait\_1ms(n);

LED\_out(0x42);

wait\_1ms(n\*8);

LED\_out(0x24);

wait\_1ms(n\*5);

LED\_out(0x18);

wait\_1ms(n\*3);

LED\_out(0x24);

wait\_1ms(n\*5);

LED\_out(0x42);

wait\_1ms(n\*8);

LED\_clear();

}

**void** **LED\_fx3**(**int** n)

{

LED\_clear();

LED\_out(0x01);

wait\_1ms(n\*6);

LED\_out(0x02);

wait\_1ms(n\*4);

LED\_out(0x04);

wait\_1ms(n\*2);

LED\_out(0x08);

wait\_1ms(n\*1);

LED\_out(0x10);

wait\_1ms(n\*1);

LED\_out(0x20);

wait\_1ms(n\*2);

LED\_out(0x40);

wait\_1ms(n\*4);

LED\_out(0x80);

wait\_1ms(n\*6);

LED\_out(0x40);

wait\_1ms(n\*4);

LED\_out(0x20);

wait\_1ms(n\*2);

LED\_out(0x10);

wait\_1ms(n\*1);

LED\_out(0x08);

wait\_1ms(n\*1);

LED\_out(0x04);

wait\_1ms(n\*2);

LED\_out(0x02);

wait\_1ms(n\*4);

LED\_clear();

}

**sysfunc.c**

**#include** <msp430.h>

**#include** "sysfunc.h"

// инициализация портов системы

**void** **Init\_System**()

{

P1DIR |= (nSS + nWR\_nRST + MCU\_SEL\_0 + MCU\_SEL\_1); // установка направления портов на вывод

DB\_DIR = 0x00; // шина данных настроена на ввод

}

// инициализация системы тактирования

**void** **Init\_System\_Clock**()

{

**volatile** byte i;

BCSCTL1 &= ~XT2OFF; // включение осцилятора XT2

// MCLK = XT2, SMCLK = XT2

**do** // ожидание запуска кварца

{

IFG1 &= ~OFIFG; // Clear OSCFault flag

**for** (i = 0xFF; i > 0; i--); // Time for flag to set

}

**while** ((IFG1 & OFIFG)); // OSCFault flag still set?

BCSCTL2 |= SELM\_2 | SELS; // установка внешнего модуля тактирования

}

// 2do: сделать точную задержку

**void** **wait\_1ms**(word cnt)

{

**for** (wait\_i = 0; wait\_i < cnt; wait\_i++)

**for** (wait\_j = 0; wait\_j < 1000; wait\_j++);

}

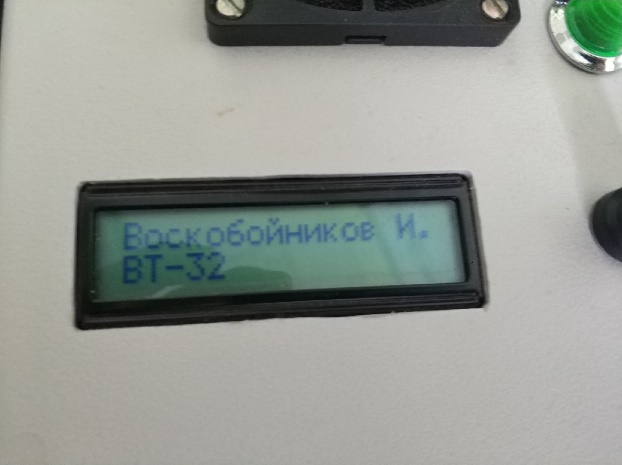
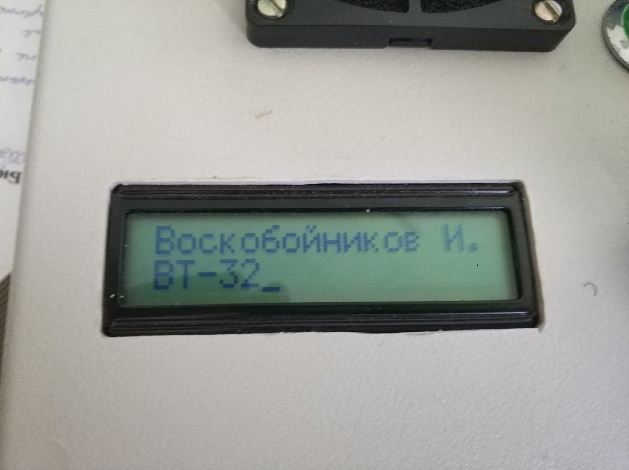
**void** **wait\_1mks**(word cnt)

{

**for** (wait\_i = 0; wait\_i < cnt; wait\_i++);

}

**Фотоотчет**

****

