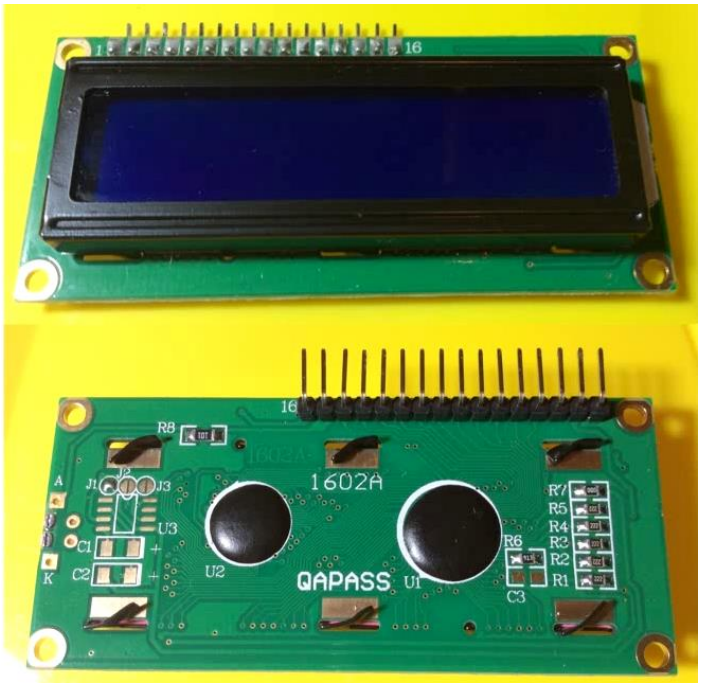
LCD индикатор 16×2

Цель: Изучить работу с LCD индикатором и смоделировать вывод информации на него в среде Proteus.

Сегодня мы начнём изучение **жидкокристаллического индикатора символьного**, который способен выводить определённые символы в две строки по 16 символов в каждую. Изучать мы данный индикатор будем с целью его подключения к микроконтроллеру AVR и управления им.

Выглядит индикатор, с которым мы будем работать вот таким вот образом:



То есть индикатор выполнен в виде модуля, в котором установлен контроллер **HD44780**, предназначенный для управления дисплеем. Также, как мы видим, в данном модуле существуют 16 контактов. Вроде и кажется, что 16 контактов — это много, но на самом деле не так уж и много. За счет контроллера **HD44780**, установленного в модуле, нам не придется подводить по 8 и более контактов к каждому из 32 символов, этим как раз и займётся данный контроллер. Мы лишь будем давать ему определённые команды и посылать определённые данные.

Также следует отметить, что типов таких модулей существует несколько и распиновка контактов может незначительно отличаться. Например, модуль компании Winstar на 1 ножке имеет анод, а наш китайский дисплей, который стоит дешевле, в отличие от Winstar, на данной ножке имеет катод.

Поэтому будьте внимательны и читайте документацию на тот именно модуль, который будет у вас.

Иногда ножки подключения индикаторов нумеруются на плате, как, например, в том модуле, который мы будем сегодня использовать.

Давайте разберёмся, какая ножка для чего служит. Пойдём слева направо:

1 - VSS - это общий провод или "земля"

2 - VDD - питание.

3 - V0 - это ножка, с помощью которой регулируется контрастность дисплея. Как правило берётся переменный резистор на 10 КОм, подключенный крайними ножками на общий провод и на питание, а с центральной ножки данного резистора провод идёт как раз на ножку V0 и посредством регулировки движка резистора мы и регулируем контрастность дисплея в модуле.

4 - RS - это такая хитрая ножка, с помощью которой контроллер дисплея будет "знать", какие именно данные находятся на шине данных. Если мы подадим на данную ножку логический 0, то значит будет команда, если 1 - то это данные.

5 - RW - данная ножка в зависимости от логического состояния на ней говорит контроллеру дисплея, будем мы с него читать или будем мы в него писать данные. Если будет 0 - то мы в контроллер дисплея будем писать, а если 1 - то будем читать данные из контроллера дисплея. Данная функция используется редко. Как правило мы всегда только пишем данные в дисплей. Чтение обычно требуется для того, чтобы определить, что дисплей принял наши данные, либо чтобы определить состояние. Но существуют определённые тайминги, позволяющие нам на слово "верить" контроллеру дисплея, что он наши данные принял и обработал. Также читать мы можем из памяти дисплея, что, в принципе, незачем. Поэтому мы обычно соединяем данный контакт с общим проводом.

6 - E - это так называемая сторнирующая шина, по спадающему фронту (когда 1 меняется в 0) на которой контроллер дисплея понимает, что именно сейчас наступил момент чтения данных на ножках, данных D0 - D7, либо передачи данных из модуля в зависимости также от состояния ножки RW.

Ножки D0 - D7 - это параллельная 8-битная шина данных, через которую и передаются или принимаются данные. Номера 0 - 7 соответствуют одноименным битам в байте данных. Но также есть ещё 4-битный способ передачи данных в контроллер и из контроллера дисплея, когда используются только ножки данных D4 - D7, а ножки D0 - D3 уже не используются. Как правило, такой способ используется в целях экономии ножек порта и именно такой способ мы и будем сегодня использовать, так как мы теряем скорость вдвое, но у нас дисплей символьный и спешить нам некуда. В 4-битном режиме мы передаём или принимаем байт в 2 приёма по половинке, сначала старшую часть байта, затем младшую.

Ножки A и K - это анод и катод для подачи напряжения для питания светодиодной подсветки дисплея. Как правило можно питать от 5 В, но желательно поставить токоограничивающий резистор на 100 Ом и скорее всего тогда подсветка дисплея "проживёт" дольше. Всё это обычно указывается в технической документации на дисплей.

Также данную информацию мы видим в технической документации на дисплей.

8. Pin assignment

Pin NO.	Symbol	Function		Remark
1	Vss	Power Supply	0V	
2	Vdd		+5V	
3	Vo		For LCD	Variable
4	RS	Register Select (H: Data L: Instruction)		
5	R/W	L: MPU to LCM H: LCM to MPU		
6	E	Enable		
7	DB0	Data Bit 0		
8	DB1	Data Bit 1		
9	DB2	Data Bit 2		
10	DB3	Data Bit 3		
11	DB4	Data Bit 4		
12	DB5	Data Bit 5		
13	DB6	Data Bit 6		
14	DB7	Data Bit 7		
15	A	Anode of LED Unit		
16	K	Cathode of LED Unit		

Теперь немного ознакомимся, каким образом мы будем организовывать процесс общения с дисплеем. То есть каким образом мы будем этим процессом управлять. Ведь контроллер дисплея не "знает" что именно мы от него хотим. Для этого в DataSheet существует вот такая таблица, представляющая собой перечень команд и способы их реализации:

11. DISPLAY CONTROL INSTRUCTION

Instruction	Instruction Code										Description	ExecutionTime(f osc=270kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM set DDRAM address to "00H" from AC	1.52ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed	1.52ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display	38 μ s
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Set display (D) cursor(C) and blinking of cursor(B) on/off	38 μ s
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data	38 μ s
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length of display line (N: 2line/1line)and, display font type F:5X11dots/5X8dots	38 μ s
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	38 μ s
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	38 μ s
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF The contents of address counter of address counter can also be read	0 μ s
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	38 μ s
Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM)	38 μ s

В самой первой колонке данной таблице находятся сами команды. Следующие 2 колонки — это то, в каком состоянии должны в момент команды находиться ножки RS и RW. Дальнейшие 8 колонок показывают нам состояние ножек шины данных, затем идёт колонка с пояснениями к командам, то есть что именно с помощью данной команды мы достигнем. А затем в последней колонке находятся тайминги или временные интервалы, необходимые для того, чтобы та или иная команда или инструкция выполнилась. Причем оговорено, при какой именно частоте генератора это достигается. То есть, в модуле существует генератор, тактирующий работу контроллера дисплея, который настроен на определённую частоту, и данная частота может быть разной.

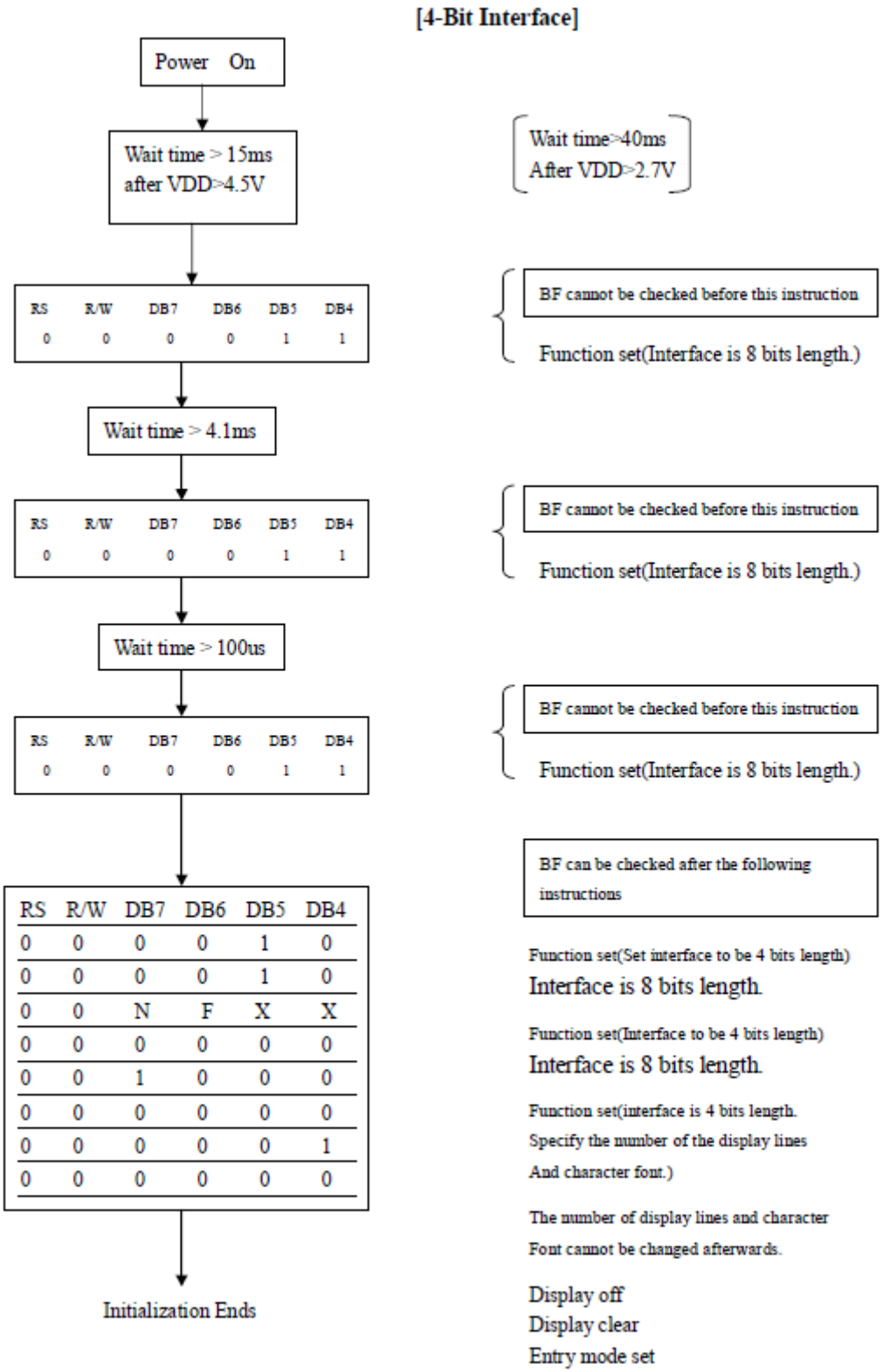
Первая команда **Clear Display** говорит сама за себя. Она очищает дисплей. Вообще за отображение на дисплее у нас отвечает оперативная память DDRAM, также существующая в контроллере дисплея. Вот данную память как раз и очищает данная команда.

Остальных команд мы коснёмся несколько позднее.

Теперь идём дальше по технической документации. Дальше идёт объяснение каждой команды и назначение каждого бита, представленного в таблице. Мы можем заметить, что в таблице каждый используемый бит как-то называется.

Дальше идёт объяснение процесса инициализации модуля дисплея. Инициализация любого активного устройства — это неотъемлемая часть программирования. Без первичной инициализации не будет работать ни одно устройство.

Сначала показана инициализация 8-битного режима, а затем 4-битного режима. Нам интересен именно последний способ. Поэтому посмотрим данную страничку.



Мы видим, что всё здесь очень подробно рассказано и показано. Вот эту диаграмму мы и будем использовать, когда будем писать код инициализации дисплея. Опять же требование — 270 кГц частота работы генератора.

Также посмотрим организацию знакомест дисплея в памяти DDRAM. Это нам будет необходимо для написания функции позиционирования.

10. Reflector of Screen and Display RAM

Display position	1-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9	1-10
DDRAM address	00	01	02	03	04	05	06	07	08	09
Display position	1-11	1-12	1-13	1-14	1-15	1-16				
DDRAM address	0A	0B	0C	0D	0E	0F				
Display position	2-1	2-2	2-3	2-4	2-5	2-6	2-7	2-8	2-9	2-10
DDRAM address	40	41	42	43	44	45	46	47	48	49
Display position	2-11	2-12	2-13	2-14	2-15	2-16				
DDRAM address	4A	4B	4C	4D	4E	4F				

Как мы видим, вторая строка находится в области видеопамяти через некоторый пропуск после первой. Во-первых, существуют дисплеи разной размерности, например, дисплей 20x4 на том же контроллере, поэтому и пропуск. Также существует определённая команда, которая передвигает видимую часть памяти, это может быть использовано для подготовки символов в невидимой области, а затем путём передвижения невидимую область мы делаем видимой. Нам это пока не требуется. Если потребуется, то мы обязательно с этим разберёмся, ну либо для какого-то красивого скроллинга дисплея также может это потребоваться.

Подобная документация существует также не только на дисплей, а ещё и на контроллер.

А теперь наконец-то проект.

```
#define F_CPU 8000000UL // Частота 8 МГц
#include <avr/io.h> // Подключение библиотек
#include <avr/interrupt.h>
#include <util/delay.h>
```

Скомпилируем код, чтобы у нас была хотя бы какая-то прошивка.

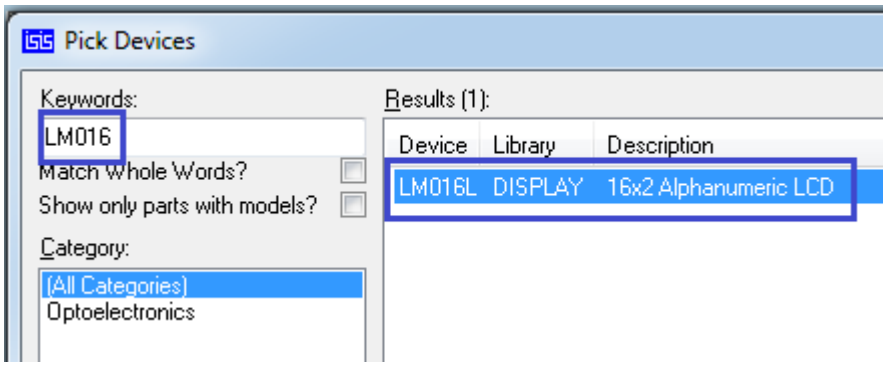
Напишем функцию port_ini(). Под все ножки модуля дисплея мы будем использовать порт D. Так как режим у нас 4-битный, то нам вполне хватит ножек, даже останутся.

```
//————————————————————-
void port_ini(void)
{
PORTD=0x00;
DDRD=0xFF;
}
//————————————————————-
```

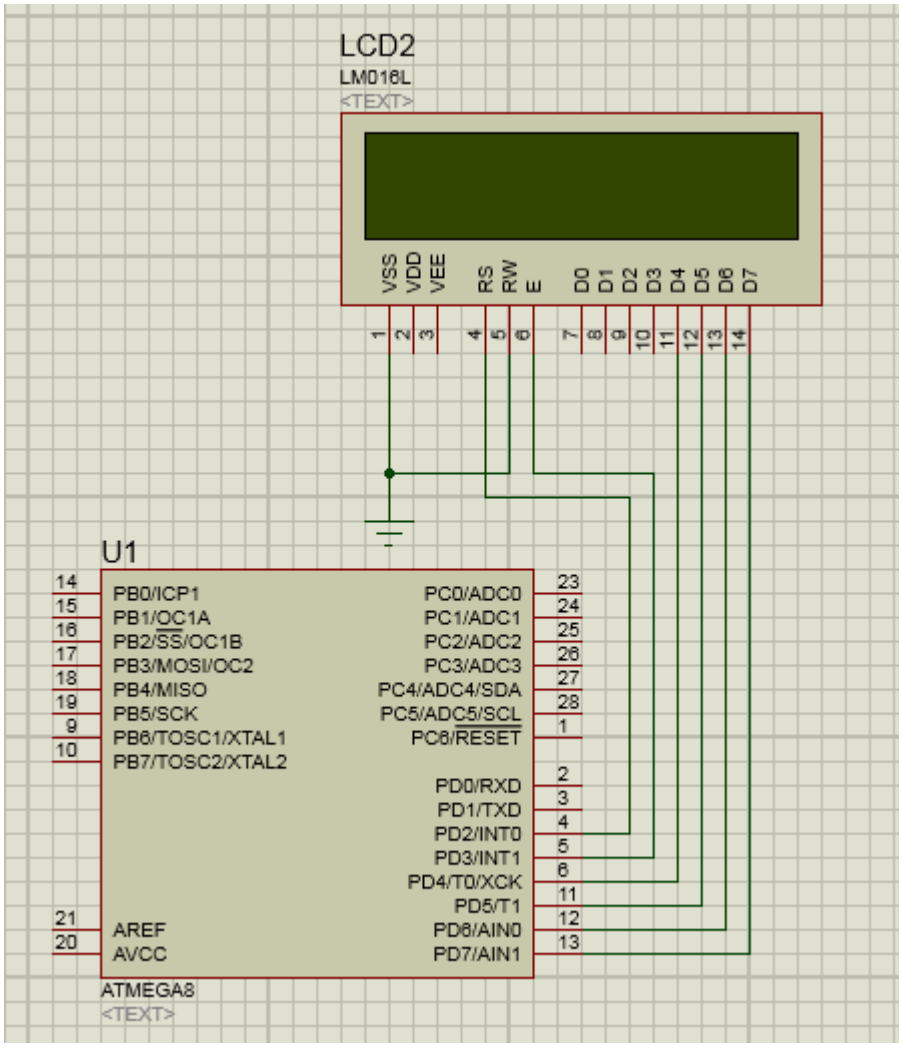
Также давайте данную функцию вызовем в функции main():

```
int main(void)
{
port_ini(); //Инициализируем порты
```

Зайдём теперь в проект в Proteus и найдем там дисплей в библиотеке компонентов:



Подключим его следующим образом:



Если мы запустим проект, то мы увидим всего лишь, что дисплей у нас просто будет светиться и всё.

Вернёмся в проект Atmel Studio и начнём думать, как нам начать писать инициализацию дисплея. По приведённой выше диаграмме из DataSheet мы видим, что мы через определённое время должны три раза дисплею передать двоичное число 11, и тогда контроллер дисплея "поймёт", что от него хотят именно того, чтобы он общался с нами в 4-х битном режиме, и чтобы он в данный режим как-то переключился. Напишем функцию инициализации дисплея. Код данной функции мы разместим перед функцией main():

```
//————————————————————-
void LCD_ini(void)
{
}
//————————————————————-
```

Вызовем данную функцию также в функции main():

```
port_ini(); //Инициализируем порты
LCD_ini(); //Инициализируем дисплей
```

Теперь начнём писать код в теле данной функции. Начнём с того. что судя по диаграмме инициализации из технической документации мы должны подождать сначала не менее 15 миллисекунд.

```
void LCD_ini(void)
{
_delay_ms(15); // Ждем 15 мс
```

А дальше мы должны как-то передавать контроллеру дисплея байты. Но так как мы байты сразу передавать не можем, ибо у нас даже ножки данных половина не подключены, то передавать мы будем полубайты. Так как это не совсем простой процесс, предлагаю завести под это дело ещё одну функцию и назвать её **sendhalfbyte**. В качестве входного параметра мы ей будем передавать unsigned char, так как у нас 4-битных переменных не бывает, мы просто не будем использовать в данной переменной первые 4 бита. Разместим код данной функции над кодом функции LCD_ini:

```
//————————————————————-
void sendhalfbyte(unsigned char c)
{

}
//————————————————————-
```

Сначала мы сдвинем наш входной аргумент влево на 4 бита, так как работаем мы со старшими разрядами шины (4-7). И не просто сдвинем, а этому же аргументу и присвоим, написав после операции сдвига знак **равно.**

```
void sendhalfbyte(unsigned char c)
{
```

```
c<<=4;
```

Давайте ещё напишем несколько макроподстановок для удобства работы с линиями E и RS, то есть их включение и выключение. Зная, на каких ножках порта D данные линии находятся нам будет это сделать очень легко:

```
#include <util/delay.h>
//————————————————————-
#define e1 PORTD|=0b00001000 // установка линии E в 1
#define e0 PORTD&=0b11110111 // установка линии E в 0
#define rs1 PORTD|=0b00000100 // установка линии RS в 1 (данные)
#define rs0 PORTD&=0b11111011 // установка линии RS в 0 (команда)
//————————————————————-
```

Вернёмся в нашу функцию передачи полубайта, включим линию **E**, для того чтобы сказать дисплею, что мы будем передавать команду. Затем мы подождём 50 микросекунд, хотя достаточно судя по таймингам гораздо меньше, затем сотрём информацию на ножках 4-7 порта **D**, остальные ножки не трогаем, затем установим нужные биты на шине данных из переменной **c**, в которой находятся данные биты. Затем мы отключим линию **E**, вот тут как раз и получится отрицательный фронт, подождём ещё 50 микросекунд на всякий случай, хотя написано 38 для записи в DDRAM или в RAM:

```
c<<=4;
e1; //включаем линию E
_delay_us(50);
PORTD&=0b00001111; // стираем информацию на входах DB4-DB7, остальное не трогаем
PORTD|=c;
e0; //выключаем линию E
_delay_us(50);
}
```

Далее передадим в контроллер дисплея двоичное число **11** три раза в функции **LCD_ini**, также применяя соответствующие задержки, взяв их из DataSheet:

```
_delay_ms(15); // Ждем 15 мс
sendhalfbyte(0b00000011);
_delay_ms(4);
sendhalfbyte(0b00000011);
_delay_us(100);
sendhalfbyte(0b00000011);
_delay_ms(1);
```

Дальше передаём двоичное число **10** таким же образом:

```
_delay_ms(1);
sendhalfbyte(0b00000010);
_delay_ms(1);
```

В некоторых DataSheet пишут, что данное число нужно передавать 2 раза, но работает и с 1 передачи. Вы на своих дисплеях можете поэкспериментировать и с 2 передачами числа 10.

В принципе, уже контроллер дисплея должен "догадаться", что мы его просим переключиться в 4 битный режим, и следующая команда будет уже с полноправным байтом, переданным поочередно и причем в этой команде уже будет конкретная команда перевода в 4-битный режим. Но для полноправных команд мы напишем другую функцию **sendbyte**, расположив ее код после функции **sendhalfbyte**.

```
//————————————————————-
void sendbyte(unsigned char c, unsigned char mode)
{
}
//————————————————————-
```

В данную функцию мы будем передавать уже два аргумента, один — это данные, а другой это режим, то есть мы здесь будем говорить, данные мы будем передавать или команду. Начнем писать код функции.

Сначала мы с помощью условия узнаем, команда в нашу функцию пришла или данные, и среагируем на это установки в соответствующее состояние шины **RS:**

```
void sendbyte(unsigned char c, unsigned char mode)
{
 if (mode==0) rs0;
 else rs1;
```

Добавим ещё одну переменную:

```
else rs1;
unsigned char hc=0;
```

Сдвинем вправо на 4 пункта наш байт и отправим результат в данную переменную. Тем самым мы в младшую тетраду байта поместим старшую:

```
unsigned char hc=0;
hc=c>>4;
```

Затем передадим сначала её в функцию **sendhalfbyte**, а затем и саму нетронутую переменную **c**. Не важно, что будет в её старшей части, так как функция **sendhalfbyte** работает только с младшей тетрадой.

```
 hc=c>>4;
 sendhalfbyte (hc); sendhalfbyte (c);
}
```

Ну и теперь, применяя вышенаписанную функцию, мы передаём следующий байт, взятый из DataSheet в функции **LCD_ini**

```
_delay_ms(1);
```

Sendbyte (0b00101000, 0); // 4 бит-режим (DL=0) и 2 линии (N=1)

_delay_ms(1);

Здесь мы используем следующую команду

Instruction	Instruction Code										Description	ExecutionTime(fosc=270kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length of display line (N: 2line/1line)and, display font type F:5X11dots/5X8dots	38 μ s

Единица в 5 бите нам говорит о том, что мы используем именно эту команду. Дальше идёт у нас бит DL, говорящий контроллеру дисплея о том, какую размерность передачи данных мы используем:

DL: Interface data length control bit
When DL=High, it means 8-bit bus mode with MPU.
When DL=Low, it means 4-bit bus mode with MPU. When 4-bit bus mode, it needs to transfer 4-bit

И, так как мы включаем режим 4-битной передачи, то в 4 бите у нас будет 0.

Следующий бит — N, который отвечает за количество строк:

N: Display line number control bit
When N=Low, 1-line display mode is set.
When N=High,2-line display mode is set.

У нас 2 строки, поэтому в 3 бит поставим 1.

F — это размер символа

F: Display font type control bit
When F=Low, 5X8 dots format display mode is set .
When F=High, 5X11 dots format display mode.

Здесь также 0.

Передадим следующий байт:

_delay_ms(1);

sendbyte(0b00001100, 0); // включаем изображение на дисплее (D=1), курсоры никакие не включаем (C=0, B=0)

_delay_ms(1);

Посмотрим данную команду в документации:

Instruction	Instruction Code										Description	ExecutionTime(f osc=270kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Set display (D) cursor(C) and blinking of cursor(B) on/off	38 μ s

Соответственно, здесь 3 бит, установленный в единицу, говорит о том, что мы применяем именно эту команду, вторая единичка — это параметр D:

D: Display ON/OFF control bit
When D=High, entire display is turned on.
When D=Low, display is turned off, but display data remains in DDRAM.

Данной единичкой мы включаем дисплей.

Следующий бит — это параметр C:

C: Cursor ON/OFF control bit
When C=High, cursor is turned on.
When C=Low, cursor is disappeared in current display, but I/D register preserves its data.

Здесь мы включаем видимость курсора. Смотреть на курсор нам ни к чему, поэтому 0.

Далее бит B:

B: Cursor Blink ON/OFF control bit
When B=High, cursor blink is on ,which performs alternately between all the high data and display characters at the cursor position. When B=Low, Blink is off.

Этот бит отвечает за мигание дисплея. Также ставим в 0.

Передаём последнюю команду в функции инициализации дисплея:

```
 _delay_ms(1);
 sendbyte(0b00000110, 0); // курсор (хоть он у нас и невидимый) будет двигаться влево
 _delay_ms(1);
}
```

Посмотрим данную команду в документации:

Instruction	Instruction Code										Description	ExecutionTime(f osc=270kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display	38 μ s

Бит 2 говорит о том, что мы используем именно данную команду.

Следующий бит — I/D:

I/D: Increment/decrement of DDRAM address (cursor or blink)
When I/D=High, cursor/blink moves to right and DDRAM address is increased by 1
When I/D=low, cursor/blink moves to left and DDRAM address is decreased by 1.

Данный бит отвечает за движение курсора в ту или иную сторону при вводе данных. Хотя у нас курсор и невидимый, двигаемся мы влево, поэтому поставим 1.

Следующий бит — SH:

SH: Shift of entire display
When DDRAM read (CGRAM read/write) operation or SH="Low", shifting of entire display is not performed. If SH=High, and DDRAM write operation, shift of entire display is performed according to

Данный бит отвечает за смещение дисплея. Так как мы это не применяем, то 0.

На этом инициализация закончена.

Теперь хотелось бы что-нибудь увидеть на дисплее. Для этого нам нужна будет ещё одна функция, которая будет передавать только данные, а не команду:

```
//————————————————————-
void sendchar (unsigned char c)
{
```

```
sendbyte (c,1);
}
//————————————————————-
```

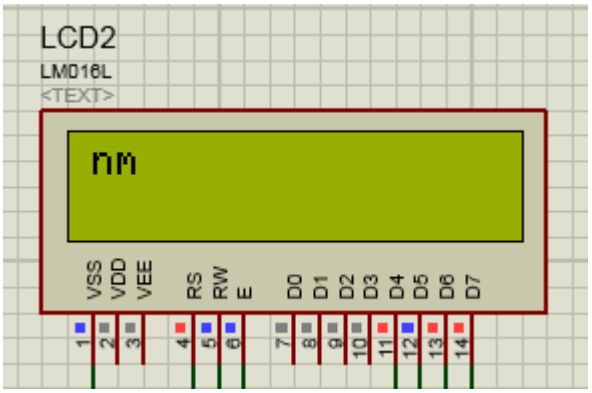
В данной функции мы просто вызываем другую с единичкой, чтобы каждый раз не писать единички. Данную функцию мы напишем после функции **sendbyte**.

Ну и давайте попробуем в дисплей передать пару каких-нибудь символов в функции **main():**

```
LCD_ini(); //Инициализируем дисплей
Sendchar ('n');
sendchar('m');
while (1)
```

Соберём код и запустим проект в Proteus.

Всё работает, и мы видим следующую картину:



Собственно, теперь давайте всё-таки напишем функцию позиционирования, чтобы мы могли выводить символы не только подряд, но и в любое место дисплея. Назовём ее **setpos**. Вообще данная функция будет устанавливать указатель в определенное место DDRAM. Выше была показана таблица организации данной памяти и соответствия её ячеек определённым знакоместам на дисплее. Будем этого и придерживаться при написании кода. Напишем нашу функцию после функции **sendchar:**

```
//————————————————————-
void setpos(unsigned char x, unsigned y)
```

```
{
}
//————————————————————————-
```

Ну, соответственно, у функции будет два входных аргумента — позиция по горизонтали и позиция по вертикали.

Добавим переменную типа **char** и вычислим её значение в зависимости от значений **x** и **y**. Значения входных аргументов мы будем начинать от нуля, а не от единицы. Поэтому чтобы вычислить адрес по **y** в памяти DDRAM, нам достаточно умножить значение **y** на 0x40, так как именно с данного адреса начинается 2 строка, а у нас она будет выглядеть как 1. Затем прибавляем **x,** тем самым получим смещение по горизонтали в данной памяти. И ещё по операции "ИЛИ" вычисленное значение мы сложим с двоичным числом **0b10000000**, так как команда передачи адреса памяти DDRAM выглядит вот так

Instruction table

Instruction	Instruction code										Description	Execution Time (fosc= 270 KHZ
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us

То есть в **6** бите мы передаём единицу для того, чтобы контроллер дисплея "понял" что мы даём именно такую команду — передача адреса памяти DDRAM, чтобы контроллер дисплея установил указатель именно туда, какую позицию мы даём ему в оставшихся семи младших битах — DB6-DB0.

Вот такой мы получим код функции:

```
void setpos(unsigned char x, unsigned y)
{
  char adress;
  adress=(0x40*y+x)|0b10000000;
  sendbyte(adress, 0);
}
```

Ну и давайте теперь вызовем данную функцию в функции **main** для порядка, установив жёстко перед вызовом функции вывода символа указатель памяти на нулевой адрес:

```
LCD_ini(); //Инициализируем дисплей
setpos(0,0);
sendchar('n');
```

Проверим на всякий случай сборку кода и запуск в Proteus. Всё работает, но эффекта данной функции мы не прочувствуем, пока не попробуем вывести ещё какие-нибудь символы куда-то в другое место дисплея. Например, вот так:

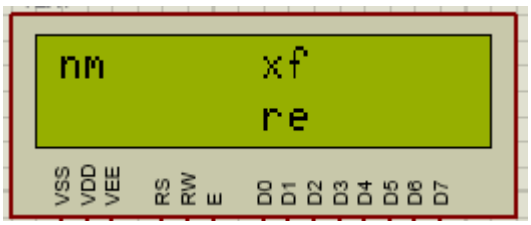
```
sendchar('m');
setpos(8,0);
sendchar('x');
sendchar('f');
```

После сборки и запуска мы должны увидеть вот такой результат:

Также давайте что-нибудь выведем на другую строку дисплея:

```
sendchar('f');
setpos(8,1);
sendchar('r');
sendchar('e');
```

А вот и результат:



Теперь пришло время написать функцию вывода на экран целой строки, так как выводить посимвольно не совсем удобно, хотелось бы работать со строками. Добавим данную функцию прямо перед функцией **main()** и передавать мы ей будем массив символов неопределённой размерности:

```
//————————————————————-
void str_lcd (char str1[])
{

}
//————————————————————-
```

Вызовем данную функцию в **main()**, удалив перед этим весь код посимвольного вывода на дисплей

```
LCD_ini(); //Инициализируем дисплей
setpos(0,0);
str_lcd("Hello World!");
```

Дальше начнём писать тело функции вывода строки. Объявим в теле функции переменную для символа. Переменная у нас будет несколько иного типа. Как правило с таким типом лучше распознаются коды символов. Вы можете, конечно, поэкспериментировать с другими типами:

```
void str_lcd (char str1[])
{
  wchar_t n;
```

Далее мы, соответственно, организуем цикл и будем попеременно перебирать все переданные символы в массиве и выводить их на дисплей. Применим также мы вариант представления нулевого символа **'n'** и именно до него мы и будем перебирать символы:

```
  wchar_t n;
```

```
  for (n=0;str1[n]!='';n++)
    sendchar(str1[n]);
}
```

Соберём код и проверим в Proteus работу кода:

Теперь можно попробовать вывести строку ещё и в другое место экрана. Напишем код в **main():**

```
str_lcd("Hello World!");
setpos(2,1);
str_lcd("String 2");
while(1)
```

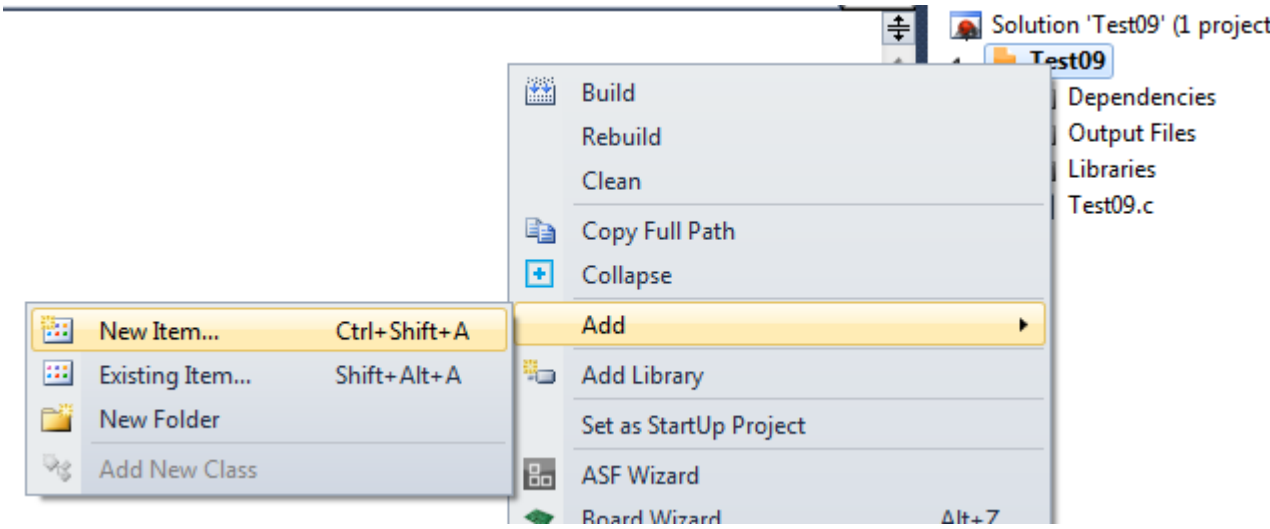
Соберём код и посмотрим результат:

Оформляем функции в отдельный модуль

Дошли мы с кодом до такого состояния, что наш главный и единственный файл с кодом переполнился до того, что в нём теперь тяжело уже вообще что-то найти. Как же мы с этим можем бороться? Бороться с этим мы будем путём оформления кода функций для отдельно взятого устройства или шины, или какой-то технологии в отдельный модуль. Грамотный модуль состоит как правило из заголовочного файла и файла реализации функций. Поэтому давайте для нашего **LCD дисплея** мы так и поступим. Также это всё нужно для того, чтобы подключать данные файлы, если нам потребуется воспользоваться LCD дисплеем. Это будет нашей так называемой библиотекой для дисплея. Конечно, библиотеки обычно пишутся и компилируются в отдельный файл **lib**, но в этом случае обычно нет исходного кода и данные библиотеки не могут быть подправлены. А наша библиотека будет вполне исправимой.

Но прежде, чем создать данную библиотеку, мы создадим главный заголовочный файл и назовём его **main.h**, чтобы убрать в данный файл все подключенные библиотеки, различные глобальные переменные и макроподстановки.

Для этого мы в дереве проектов щёлкаем правой кнопкой по нашему проекту **Test09** и выбираем в контекстном меню субменю **Add**, а в нём уже выбираем пункт **New Item:**



И в открывшемся диалоге выбираем тип файла, который мы будем создавать, "**Include File**" И внизу в имени файла меняем IncFile1 на **main**, затем жмём кнопку **Add.**

Соответственно, у нас создастся файл **main.c** вот с таким содержимым:

```
#ifndef MAIN_H_
#define MAIN_H_

#endif /* MAIN_H_ */
```

В данный файл мы поместим подключения всех заголовочных файлов библиотек и все макроподстановки, а в файле **Test09.c** всё это, конечно, мы удалим:

```
#ifndef MAIN_H_
#define MAIN_H_

#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
//————————————————————-
#define e1 PORTD|=0b00001000 // установка линии E в 1
#define e0 PORTD&=0b11110111 // установка линии E в 0
#define rs1 PORTD|=0b00000100 // установка линии RS в 1 (данные)
#define rs0 PORTD&=0b11111011 // установка линии RS в 0 (команда)
//————————————————————-

#endif /* MAIN_H_ */
```

Но недостаточно данный заголовочный файл подключить в **Solution Explorer**, его также мы должны в файле **Test09.c** подключить в самом начале в код:

```
#include "main.h"
//————————————————————-
```

Соберём проект, ещё раз проверим его работоспособность.

Теперь начнём создавать нашу библиотеку для дисплея.

Для этого мы таким же образом, как и **main.h**, создадим заголовочный файл **lcd.h**

Содержание получится аналогичным, только вместо MAIN везде будет LCD. Подключим данный файл в файле main.h:

```
#include <stdlib.h>
#include "lcd.h"
```

И наоборот, в файл **lcd.h** мы подключим файл **main.h**

```
#ifndef LCD_H_
#define LCD_H_
#include "main.h"
```

Насчет того, что получится какое-то перекрёстное зацикливание, можно не беспокоиться — директивы не дадут такому случиться.

Также все макроподстановки из файла main.h мы заберём в файл lcd,h, а в main.h удалим:

```
#include "main.h"
//————————————————————-
void LCD_ini(void);
void setpos(unsigned char x, unsigned y);
void str_lcd (char str1[]);
void sendchar(unsigned char c);
//————————————————————-
```

А, чтобы забрать все функции по работе с дисплеем из файла Test09.c, мы создадим теперь уже другой файл — **lcd.c**. В нём и будет код реализации всех функций.

Создаётся файл точно таким же образом, только вместо "Include File" мы выбираем тип файла "C File".

Файл **lcd.c** создан. В нём уже не будет никаких директив, единственное, будет авторский комментарий, который мы удалим, чтоб не мешался.

В данном файле мы также подключим заголовочный файл lcd.h:

```
#include "lcd.h"
//———————————————————-
```

Теперь мы в данный файл перенесём полностью все функции, предназначенные для работы с дисплеем, из файла **Test09.c**. В нём останутся только две фунции — **port_ini** и **main()**.

Тем самым мы очень серьёзно разгрузим главный файл приложения, сделав его удобочитаемым.

Но этого нам недостаточно. Ни одна функция теперь не будет "видна" в файле **Test09.c**. Поэтому те функции, которые мы будем использовать в других файлах, мы обязаны объявить, или, как говорят в народе, создать на них прототипы. Делается это обычно в заголовочном файле. Поэтому мы создадим прототипы в заголовочном файле lcd.h. Прототип делается очень легко. Пишется, или обычно копируется заголовок функции со всеми аргументами (всё кроме тела) и в конце ставится точка с запятой. Нам нужны будут функции инициализации дисплея, позиционирования на дисплее и вывода строки на дисплее. Символы мы отдельно пока выводить не будем, поэтому на соответствующую функцию мы прототип не создаём. Вот наши прототипы:

```
#include "main.h"
//———————————————————-
void LCD_ini(void);
void setpos(unsigned char x, unsigned y);
void str_lcd (char str1[]);
//———————————————————-
#define e1 PORTD|=0b00001000 // установка линии E в 1
```

Теперь соберём файл, запустим его в Proteus, и проверим его работоспособность.

Test09.c

```
#include "main.h"
//---------------------------------------
void port_ini(void)
{
	PORTD=0x00;
	DDRD=0xFF;
}
//---------------------------------------
int main(void)
{
	port_ini(); //»нициализируем порты
	LCD_ini();  //»нициализируем дисплей
	setpos(0,0);
	str_lcd("Hello World!");
	setpos(2,1);
	str_lcd("String 2");
	while(1)
	{
	}
}
```

Main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#define F_CPU 8000000UL

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

```
#include <stdio.h>
#include <stdlib.h>

#include "lcd.h"


#endif /* MAIN_H_ */
```

Lcd.h

```
#ifndef LCD_H_
#define LCD_H_

#include "main.h"

//---------------------------------------
void LCD_ini(void);
void setpos(unsigned char x, unsigned y);
void str_lcd (char str1[]);
//---------------------------------------
#define e1    PORTD|=0b00001000 // установка линии E в 1
#define e0    PORTD&=0b11110111 // установка линии E в 0
#define rs1    PORTD|=0b00000100 // установка линии RS в 1 (данные)
#define rs0    PORTD&=0b11111011 // установка линии RS в 0 (команда)
//---------------------------------------

#endif /* LCD_H_ */
```

Lcd.c

```
#include "lcd.h"

//---------------------------------------
```

```
void sendhalfbyte(unsigned char c)
{
	c<<=4;
	e1; //включаем линию E
	_delay_us(50);
	PORTD&=0b00001111; //стираем информацию на входах DB4-DB7, остальное не трогаем
	PORTD|=c;
	e0; //выключаем линию E
	_delay_us(50);
}
//---------------------------------------
void sendbyte(unsigned char c, unsigned char mode)
{
	if (mode==0) rs0;
	else	rs1;
	unsigned char hc=0;
	hc=c>>4;
	sendhalfbyte(hc); sendhalfbyte(c);
}
//---------------------------------------
void sendchar(unsigned char c)
{
	sendbyte(c,1);
}
//---------------------------------------
void setpos(unsigned char x, unsigned y)
{
	char adress;
	adress=(0x40*y+x)|0b10000000;
	sendbyte(adress, 0);
}
//---------------------------------------
```

```
void LCD_ini(void)
{
	_delay_ms(15); //Ждем 15 мс (стр 45)
	sendhalfbyte(0b00000011);
	_delay_ms(4);
	sendhalfbyte(0b00000011);
	_delay_us(100);
	sendhalfbyte(0b00000011);
	_delay_ms(1);
	sendhalfbyte(0b00000010);
	_delay_ms(1);
	sendbyte(0b00101000, 0); //4бит-режим (DL=0) и 2 линии (N=1)
	_delay_ms(1);
	sendbyte(0b00001100, 0); //включаем изображение на дисплее (D=1), курсоры никакие не включаем (C=0, B=0)
	_delay_ms(1);
	sendbyte(0b00000110, 0); //курсор (хоть он у нас и невидимый) будет двигаться влево
	_delay_ms(1);
}
//---------------------------------------
void clearlcd()
{
	sendbyte(0b00000001, 0);
	_delay_us(1500);
}
//---------------------------------------
void str_lcd (char str1[])
{
	wchar_t n;
	for(n=0;str1[n]!='\0';n++)
	sendchar(str1[n]);
}
```

```
//--------------------------------------
```

Задания для выполнения:

1. Изучить работу с символьным дисплеем и законспектировать.
2. Смоделировать представленную программу в среде Proteus.
3. Изменить программу для вывода на экран **ФИО** в первой строке, а **№ группы** – во второй (позиционировать в центре экрана).

Результаты выполнения отправить на e-mail: rasov@rambler.ru с темой LCD_ФИО