<https://habr.com/ru/post/213459/>

Программирование дисплея на контроллере ST7920

* [Assembler](https://habr.com/ru/hub/assembler/)
* Tutorial

Здравствуйте, я хочу рассказать о программировании дисплея на контроллере ST7920 с использованием ATtiny2313 контроллера.  
  


Характеристики дисплея

Приблизительная цена: 15$  
Размер дисплее вместе с платой: 93.0 (Длина) × 70.0(Ширина) × 13.50(Высота) мм  
Размер видимой области: 70x38 мм.  
  
Дисплей имеет 2 режима работы:

* Графический
* Текстовый

В текстовом режиме дисплей имеет 4 строки и 16 знакомест на строку  
В графическом режиме разрешение: 128x64 пикселя.  
  
3 режима подключения:

1. Подключение по 8 битной шине
2. Подключение по 4 битной шине
3. Подключения по SPI (3 битной шине)

Имеются 2 режима работы:

* Нормальный: потребление 450 мкА, 5 В
* Спящий режим: потребление 30 мкА, 5 В

В данном посте я расскажу о:

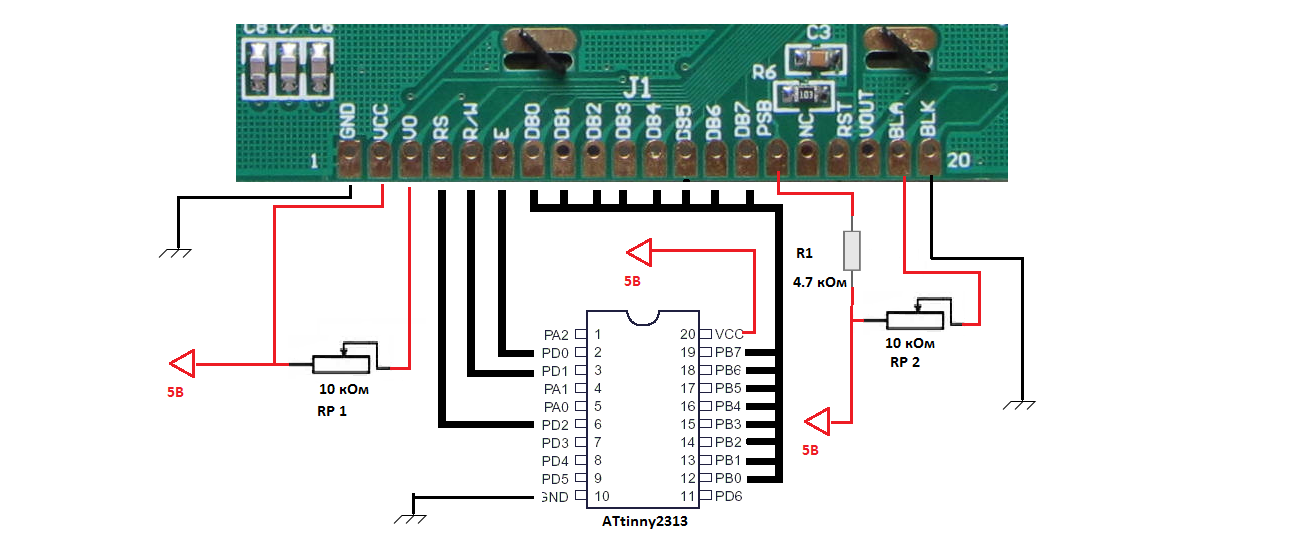
* Работе в текстовом режиме
* Подключения и программирование по 8 битной шине
* Подключения и программирование по SPI

Для того что бы подключить дисплей к контроллеру нам понадобится:

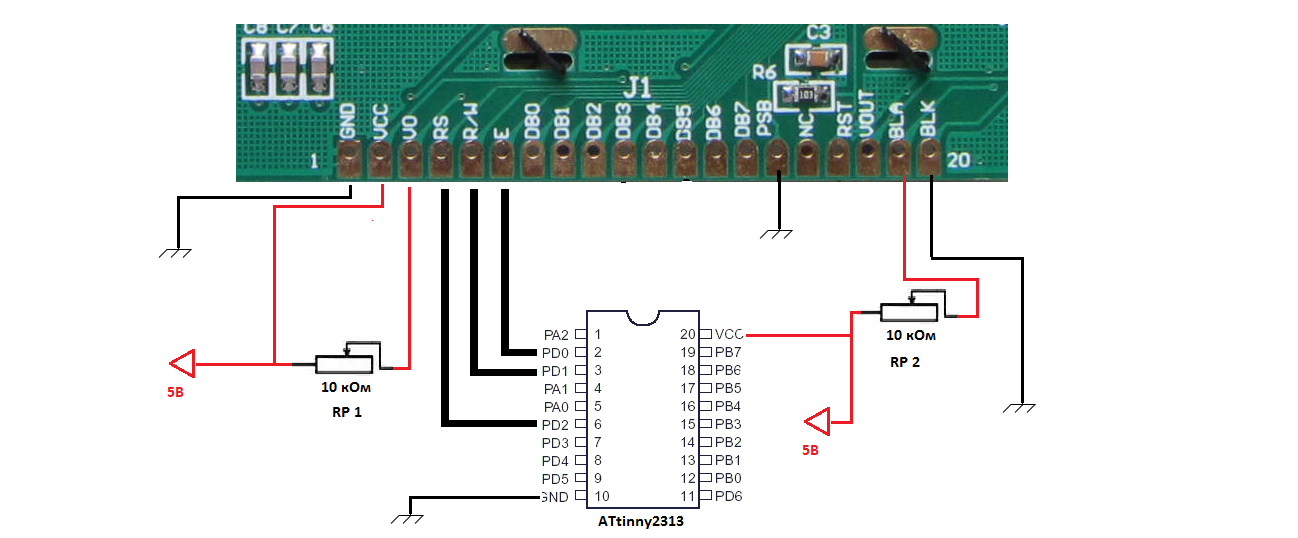
1. Дисплей на контроллере ST7920
2. 2 подстрочных резистора на 10 кОм.
3. Для 8 битного режима резистор на 4.7 кОм (или больше)
4. Контроллер ATtiny2313
5. Источник питание на 5В.

Схема подключения

Подключение по 8 битной шине данных

  
Распиновка контактов:  
GND — Земля  
VCC — +5В  
V0 — Настройка контрастности  
RS — Определяет режим передачи данных (1 — это данные, 0 — это команда)  
RW — Запись или чтения (1 — чтения, 0 — запись)  
E — Строб  
D0-D7 — Шина данных   
PSB — Определяет какой протокол передачи данных будет использоваться ( 1 — 8/4 бит шина, 0 — SPI)  
BLA — Анод подсветки (+)  
BLK — Катод подсветки (-)  
  
На схеме DB0-DB7 и PB0-PB7 не замкнуты, это 8 битная шина данных.  
Реальное соединения таково:  
DB0 — PB0  
DB1 — PB1  
DB2 — PB2  
DB3 — PB3  
DB4 — PB4  
DB5 — PB5  
DB6 — PB6  
DB7 — PB7

Подключение по SPI

  
Распиновка контактов:  
GND — Земля  
VCC — +5В  
V0 — Настройка контрастности  
RS — (CS) Начало/окончание передачи данных (1 — начало, 0 — окончание)  
RW — (SID) Шина данных  
E — (SCLK) Строб  
PSB — Определяет какой протокол передачи данных будет использоваться ( 1 — 8/4 бит шина, 0 — SPI)  
BLA — Анод подсветки (+)  
BLK — Катод подсветки (-)

Подстроечные резисторы

RP1 — Регулятор контрастности  
RP2 — Регулятор яркости

Описание протоколов программирования дисплея

8 битный режим

И так, с начало я расскажу о том как в общих чертах происходит работа с дисплеем.  
Для того что бы работать с дисплеем нам нужно отправлять команды и данные на дисплей.  
К командам относится: Включения/выключение дисплея, отображение курсора, перемещение курсора и т.д. К данным относятся например символы которые вы хотите видеть на дисплее.  
  
Давайте рассмотрим пример того как производится инициализация для 8 битного режима.  
Давайте рассмотрим пример того как производится инициализация:

* Задержка в 50 мкс.
* Отправляем команду установки 8 битного режима.
* Задержка 120 мкс.
* Отправляем команду включения дисплея ( в ней же указывается, включить ли курсор, и мигать ли курсором)
* Задержка в 50 мкс.
* Повторно отправляем функцию установки 8 битного режима
* Задержка 120 мкс.
* Отправляем команду отчистить экран
* Задержка 20 мкс.
* Устанавливаем ENTRY MODE (эта команда говорит о том в какую сторону сдвигать курсор после написания символа, нам соответственно нужно вправо)

Вот и все, после выполнения этих действий, если вы указали в команде включения дисплея отображать курсор, на экране вы увидите курсор.  
  
Рассмотрим как отправить одну команду на дисплей в 8 битном режиме:

* Устанавливаем низкий уровень E
* Устанавливаем низкий уровень RS
* Устанавливаем низкий уровень RW
* Задержка 1 мкс.
* Устанавливаем высокий уровень E
* Отправляем в порт данных байт команды
* Задержка 1 мкс.
* Устанавливаем низкий уровень E
* Задержка 50 мкс.

Для отправки одного байта данных выполняется абсолютно то же самое, только в начале устанавливается высокий   
уровень RS.  
RS = 0 Команда  
RS = 1 Данные  
  
Вот как отправляется один байт данных:

* Устанавливаем низкий уровень E
* Устанавливаем высокий уровень RS
* Устанавливаем низкий уровень RW
* Задержка 1 мкс.
* Устанавливаем высокий уровень E
* Отправляем в порт данных байт команды
* Задержка 1 мкс.
* Устанавливаем низкий уровень E
* Задержка 50 мкс.

Давайте рассмотрим код отправки команды  
Для начала установим константы что бы было удобнее:

.equ PCom = PORTD ; Управляющий порт к которому подключены RS, RW, E

.equ PW = PORTB ; Порт данных

.equ RS = 2 ; Контакт порта PCom к которому подключен RS

.equ E = 0 ; Контакт порта PCom к которому подключен E

.equ RW = 1 ; Контакт порта PCom к которому подключен RW

.def Data = R18 ; Регистр используется для записи данных в порт

Функция отправки команды:

;Перед вызовом функции в регистр Data нужно установить необходимую команду

LCD12864\_CommandOut: ;Вывод команды на дисплей.

cbi PCom, E ; Устанавливаем низкий уровень E

cbi PCom, RS ; Устанавливаем низкий уровень RS

cbi PCom, RW ; Устанавливаем низкий уровень RW

LCD8\_MACRO\_DELAY 1, 1 ; Задержка 1 мкс.

sbi PCom, E ; Устанавливаем высокий уровень E

out PW, Data ; Отправляем в порт данных байт команды

LCD8\_MACRO\_DELAY 1, 1 ; Задержка 1 мкс.

cbi PCom, E ; Устанавливаем низкий уровень E

LCD8\_MACRO\_DELAY 1, 50 ; Задержка 50 мкс.

Ret

Функция отправки данных:

;Перед вызовом функции в регистр Data нужно установить необходимую команду

LCD12864\_DataOut: ;Вывод данных на дисплей.

sbi PCom, E ; Устанавливаем высокий уровень E

cbi PCom, RS ; Устанавливаем низкий уровень RS

cbi PCom, RW ; Устанавливаем низкий уровень RW

LCD8\_MACRO\_DELAY 1, 1 ; Задержка 1 мкс.

sbi PCom, E ; Устанавливаем высокий уровень E

out PW, Data ; Отправляем в порт данных байт команды

LCD8\_MACRO\_DELAY 1, 1 ; Задержка 1 мкс.

cbi PCom, E ; Устанавливаем низкий уровень E

LCD8\_MACRO\_DELAY 1, 50 ; Задержка 50 мкс.

Ret

В коде был использован макрос LCD8\_MACRO\_DELAY, вот его код

; Пример макроса функции задержки в микросекундах

.MACRO LCD8\_MACRO\_DELAY ; 1 параметр количество задержек, 2 параметр, количество микросекунд в задержки

ldi Temp, @0

ldi Temp1, @1

rcall LCD12864\_Delay

.ENDM

;Пример функции задержки для контроллера на 4 МГц.

;Функция имеет 2 параметра:

;R16 – Количество микросекунд

;R17 – Количество циклов по R16 микросекунд.

LCD12864\_Delay:

push R16 ;Сохраняем младшую задержку в ОЗУ.

ES0:

dec R16 ;- задержка.

cpi R16, 0 ;Закончилась?

brne ES0 ;Нет - еще раз.

pop R16 ;Да? Восстановить задержку.

dec R17 ;Отнять от "количества задержек" разряда.

cpi R17, 0 ;Количество задержек = 0?

brne LCD12864\_Delay

ret

Теперь рассмотрим команды инициализации дисплея в текстовом, 8 битном режиме:

Команда FUNCTION SET: **0 0 1 DL 0 RE 0 0**  
DL:

* Если установлено 1 то устанавливаем 8 бит передачу данных
* Если установлено 0 то устанавливается 4 бита передача данных

RE:

* Если установлено 1 то устанавливается набор расширенных команд
* Если установлено 0 то устанавливается набор базовых команд

Следующая команда это DISPLAY STATUS: **0 0 0 0 1 D C B**  
D:

* Если установлено 1 то дисплей включен
* Если установлено 0 то дисплей выключен

С:

* Если установлено 1 то курсор включен
* Если установлено 0 то курсор выключен

B:

* Если установлено 1 то курсор будет мигать
* Если установлено 0 то курсор не будет мигать

Следующая команда простая CLEAR – отчистка экрана: **0 0 0 0 0 0 0 1**

И последняя команда это ENTRY MODE SET – установка направления движения курсора: **0 0 0 0 0 1 I/D S**

* Если I/D = 1 то курсор сдвигается вправо
* Если I/D = 0 то курсор сдвигается влево

На основе этих четырех команд можно написать функцию инициализации дисплея:

LCD12864\_Init: ;Инициализация дисплея.

LCD8\_MACRO\_DELAY 1, 50 ; Задержка в 50 мкс.

Ldi Data, 0b00110000

rcall LCD12864\_CommandOut ; Отправляем команду установки 8 битного режима.

LCD8\_MACRO\_DELAY 1, 120 ; Задержка в 120 мкс.

Ldi Data, 0b00001111

rcall LCD12864\_CommandOut ; Отправляем команду включения дисплея, включить курсор, мигать курсором

LCD8\_MACRO\_DELAY 1, 50 ; Задержка в 50 мкс.

Ldi Data, 0b00110000

rcall LCD12864\_CommandOut ; Отправляем команду установки 8 битного режима.

LCD8\_MACRO\_DELAY 1, 120 ; Задержка в 120 мкс.

Ldi Data, 0b00000001

rcall LCD12864\_CommandOut ; Отправляем команду отчистить экран

LCD8\_MACRO\_DELAY 1, 20 ; Задержка в 20 мкс.

Ldi Data, 0b00000110

rcall LCD12864\_CommandOut ; установка направления движения курсора вправо

LCD8\_MACRO\_DELAY 1, 50 ; Задержка в 50 мкс.

ret

После выполнения инициализации на экране вы должны увидеть мигающий курсор.

Режим SPI

Теперь о функции приема передачи команды/данных по SPI.  
В этом режиме участвуют 2 линии:

* SID это контакт передачи данных, на дисплее он же RW
* SCLK – это линия строб, на дисплее он же E
* CS – это начала/окончания передачи данных, на дисплее он же RS

В SPI режиме передача одной команды или 1 байта данных происходит при передачи 24 бит  
Протокол передачи данных таков:

* Устанавливаем высокий уровень CS
* Передаем 4 единицы подряд
* Передаем 1 бит RW – чтения или запись
* Передаем 1 бит RS – Команда или данные
* Передаем 0
* Передаем 4 бита старшей половины байта данных
* Передаем 4 нуля
* Передаем 4 бита младшей половины байта данных
* Передаем 4 нуля подряд
* Устанавливаем низкий уровень CS

На этом передача одного байта завершена.  
  
После каждого переданного бита делается строб, то есть:

* Задержка 1 мкс.
* Устанавливаем высокий уровень SCLK
* Задержка 1 мкс.
* Устанавливаем низкий уровень SCLK
* Задержка 1 мкс.

Рассмотрим функцию передачи команды/данных в режиме SPI, но сперва объявим константы:

.equ PCom = PORTD ; Управляющий порт к которому подключены SID и SCLK

.equ SID = 1 ; RW Шина данных

.equ SCLK = 0; E Строб

.equ CS = 2; RS Начало/конец передачи данных

.def Data = R18 ; Регистр используется для записи данных в порт

А теперь сама функция:

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

Функции отправки команды и данных по последовательному порту

LCD12864\_CommandOut - Отправляет команду

LCD12864\_DataOut - Отправляет данные

Команда или данные должны находится в регистре Data

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

LCD12864\_CommandOut:

ldi r20, 0

rjmp command

LCD12864\_DataOut:

ldi r20, 1

command:

LCD8\_MACRO\_DELAY 1, 1

sbi PCom, CS ; Начинаем передачу данных

sbi PCom, SID ; Устанавливаем SID В 1

; Шлем 4 единицы

rcall strob ; 1

rcall strob ; 1

rcall strob ; 1

rcall strob ; 1

rcall strob ; 1

; Устанавливаем rw на запись

cbi PCom, SID ; rw = 0

rcall strob

; Выберем, команда или данные и отправим ее.

cbi PCom, SID ; rs = 0

cpi r20, 0

breq command1

sbi PCom, SID ; rs = 1

command1:

rcall strob

; Отправляем 0

cbi PCom, SID ; 0

rcall strob

; Началась отправка байта

ldi r20, 8 ; Счетчик бит

for*\_send\_*data:

cpi r20, 0 ; Смотрим не закончились ли биты?

breq stop\_send\_data ; Если закончились то переходим к отправки последних нулей

cpi r20, 4 ; Смотрим, если было отправлено 4 бита то выполняем отправку 4 нулей.

brne no\_strob ; Иначе переходим к отправки следующего бита

; Отправка 4 нуля

cbi PCom, SID

rcall strob

rcall strob

rcall strob

rcall strob

;Отправка следующего бита

no\_strob:

dec r20 ; Уменьшаем счетчик бит

rol Data ; Сдвигаем регистр с данными на 1 влево

brcs send\_bit\_1 ; Если сдвинутый регистр был 1, то флаг C был поднят, а значит переходим на отправку бита 1

; Если флаг С не был поднят, отправляем 0

cbi PCom, SID ; Данные 0 бит

rcall strob

rjmp for\_send\_data

;Отправляем бит 1

send*\_bit\_*1:

sbi PCom, SID ; Данные 1 бит

rcall strob

rjmp for\_send\_data

stop*\_send\_*data:

; Отправка байта закончилась, отправляем 4 нуля

cbi PCom, SID

rcall strob

rcall strob

rcall strob

rcall strob

cbi PCom, SID

cbi PCom, CS ; Передача данных закончена

ret

Функция строб:

strob:

LCD8\_MACRO\_DELAY 1, 1 ; Задержка в 1 мкс

sbi PCom, SCLK ; Устанавливаем высокий уровень SCLK

LCD8\_MACRO\_DELAY 1, 1 ; Задержка в 1 мкс

cbi PCom, SCLK ; Устанавливаем низкий уровень SCLK

LCD8\_MACRO\_DELAY 1, 1 ; Задержка в 1 мкс

ret

.endif

;**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****\*\*\**\*

Текстовый режим

Теперь после того как вы научились инициализировать дисплей вы можете выводить любые символы на экран, например вывести букву A:

ldi Data, 'A'

rcall LCD12864\_DataOut

И на дисплее вы увидите букву A.  
  
И так, теперь о том как устроенно адресное пространство в текстовом режиме:  
Экран делится на **8 столбцов** и **4 строки**, в каждый столбец вы можете записать по **2 обычных символа** или **1 иероглиф**.  
Адресное пространство находится от 0 до 31.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

Как видите первая строчка это адреса от 0 до 7  
Вторая же строчка от 16 до 23  
Третья строчка от 8 до 15  
То есть если вы напишете 16 букв подряд с адреса 0, то они будут в первой строчке,  
но если вы напишите 17 символов, то последний символ будет не на второй строчке, а на третей!

Есть специальная команда установки адреса курсора: **1 AC6 AC5 AC4 AC3 AC2 AC1 AC0**  
С помощью этой команды можно поставить курсор в нужное место вписав за место AC0-AC6 адрес от 0 до 31.

Графический режим

И напоследок, для тех кто хочет использовать графический режим, есть такая статья: [LCD 12864 на контроллере ST7920. Параллельный режим (8 бит)](http://avrprog.blogspot.ru/2013/12/lcd-12864-st7920-8.html)

Библиотека для работы с ST7920

[Ссылка на файл библиотеки](http://yadi.sk/d/15A3GouMJKBjM)