Свежие комментарии

- SmNikolay к записи STM Урок 89. LAN. ENC28J60. TCP WEB Server. Подключаем карту SD
- Narod Stream к записи AVR Урок 3. Пишем код на СИ. Зажигаем светодиод
- strannik2039 к записи AVR Урок 3. Пишем код на СИ. Зажигаем светодиод
- Dmitriy к записи AVR Урок 1. Знакомство с семейством AVR
- Narod Stream к записи STM Урок 9. HAL. Шина I2C. Продолжаем работу с DS3231

Форум. Последние ответы

- Narod Stream в Программирование МК STM32
  1 неделя, 2 дн. назад
- Zandy в Программирование МК STM32
  1 неделя, 3 дн. назад
- Narod Stream в Программирование МК STM32
  3 нед. назад
- Narod Stream в Программирование МК STM32
  3 нед. назад
- fireweb в Программирование МК STM32
  3 нед., 2 дн. назад

Январь 2018

Пн	Вт	Ср	Чт	Пт	Сб	Вс
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

« Дек

Архивы

- Январь 2018
- Декабрь 2017
- Ноябрь 2017
- Октябрь 2017
- Сентябрь 2017
- Август 2017
- Июль 2017

AVR Урок 16. Интерфейс TWI (I2C). Часть 6

Posted on Декабрь 17, 2016 by Narod Stream Опубликовано в I2C, Программирование AVR — 2 комментария ↓

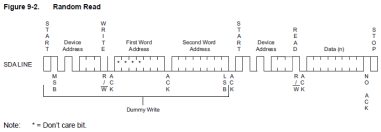
Урок 16
Часть 6

Интерфейс TWI (I2C)

В **предыдущей части** занятия уже продолжили свою работу с отправкой значений в память микросхемы EEPROM по интерфейсу **TWI**. Только проверить мы смогли это, исключительно веря статусам.

А сегодня мы уже это проверим точно, считав данные из микросхемы. Соответственно здесь всё будет не совсем просто. Функция и вообще процедура чтения данных по шине I2C не совсем простая, есть свои нюансы. Но, думаю, используя наш предыдущий опыт, мы обязательно с ней справимся.

Откроем следующую таблицу в технической документации на микросхему



Мета

- Регистрация
- Войти
- RSS записей
- RSS комментариев
- WordPress.org

Заходите на канал Narod Stream

- Июнь 2017
- Май 2017
- Март 2017
- Февраль 2017
- Январь 2017
- Декабрь 2016
- Ноябрь 2016

Это диаграмма считывания из определённой ячейки памяти одного байта.

Она чем-то похожа на процедуру записи, но не совсем это так.

Сначала мы также создаём условие СТАРТ на шине, затем передаём адрес с битом записи, именно записи. Тут вопрос, почему записи. Потому, что мы должны отправить адрес устройства и адрес ячейки памяти. из которой мы потом и заберём байт. Затем подтверждение, затем старшая часть адреса ячейки памяти, подтверждение, младшая часть ячейки памяти, подтверждение.

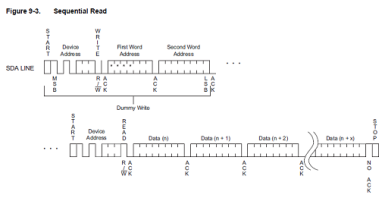
Затем, как ни странно, опять условие СТАРТ. Именно так. Затем уже передаём адрес устройства с битом чтения (1), подтверждение, затем после подтверждения читаем данные из регистра данных, а затем уже подтвержденя не ждём, то есть генерируем бит без подтверждения, то есть шина данных становится в логический 1, а затем условие СТОП. Вот так.

Так что, как видите, здесь не совсем всё так просто.

И это всё при условии, что надо нам считать только один байт. Это удобно, если мы тестируем память на случайное чтение, постоянно читая какой-то случайный бит, поэтому и диаграмма названа Random Read.

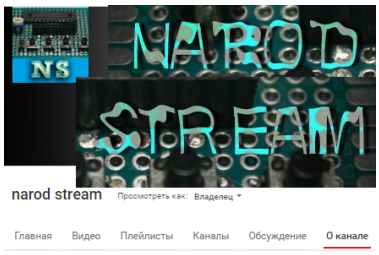
Но в большинстве случаев нам требуется сразу считать несколько байт, расположенных последовательно в ячейках памяти. Пример из жизни: данный тип микросхем очень широко применяется в спутниковых рессиверах и Т2-приёмниках и у нас, например глюканула эта микросхема, что происходит отнюдь не редко. А в памяти данной микросхемы находятся все практически настройки. Мы берем живую микросхему из другого такого же устройства и считываем оттуда прошивку. Вот такая процедура тут уже и пригодится. Затем уже мы эту считанную прошивку либо загружаем в микросхему со сбойными данными, либо в новую, на которую сбойную потом меняем.

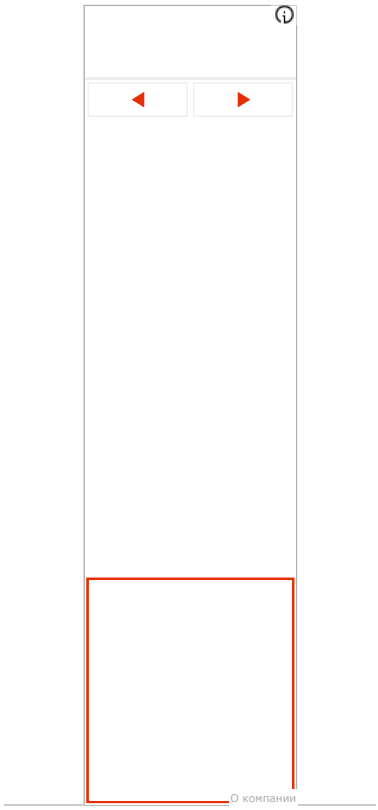
Для такой процедуры существует другая диаграмма в даташите (нажмите на картинку для увеличения размера)



Данная диаграмма нам показывает, как нужно читать байты целыми блоками.

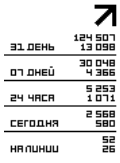
Начало не отличается от предыдущей процедуры, а затем мы начинаем считывать каждый байт, после каждого считанного байта ждём подтверждение, а





Рубрики

- 1-WIRE (3)
- ADC (6)
- DAC (4)
- GPIO (26)
- I2C (19)
- SPI (13)
- USART (8)
- Программирование AVR (131)
- Программирование PIC (7)
- Программирование STM32 (213)
- Тесты устройств и аксессуаров (1)



вот после последнего уже не ждём, то есть генерируем условие NO ASK. Как это проделать, разберёмся по мере написания функций.

Пока пользуемся уже существующими функциями, так как всё начинается с процедуры записи в шину. В функции main() закомментируем весь код процедуры записи серии байтов, и начнем писать процедуру чтения из шины

Скопируем в данную процедуру то, что будет вообще без изменения

```
//Чтение
I2C_StartCondition(); //Отправим условие START
USART_Transmit(TWSR);//читаем статусный регистр
I2C_SendByte(0b10100000);//передаем адрес устройства и бит записи (0)
USART_Transmit(TWSR);//читаем статусный регистр
I2C_SendByte(0);//передаем старшую часть адреса ячейки памяти
USART_Transmit(TWSR);//читаем статусный регистр
I2C_SendByte(0);//передаем младшую часть адреса ячейки памяти
USART_Transmit(TWSR);//читаем статусный регистр
```

После передачи младшего байта адреса памяти мы генерируем условие СТАРТ

```
I2C_SendByte(0);//передаем младшую часть адреса ячейки памяти
USART_Transmit(TWSR);//читаем статусный регистр
I2C_StartCondition(); //Отправим условие START
USART_Transmit(TWSR);//читаем статусный регистр
```

Затем ещё раз передаём адрес устройства, но уже с установленным битом чтения (1)

```
I2C_StartCondition(); //Отправим условие START
USART_Transmit(TWSR);//читаем статусный регистр
I2C_SendByte(0b10100001);//передаем адрес устройства и бит чтения (1)
USART_Transmit(TWSR);//читаем статусный регистр
```

Далее мы должны будем уже читать байты из шины. Только, к сожалению, у нас для этого пока нет функции.

Но не будем этого бояться и напишем данную функцию, так как она будет подобна функции, применяемой для записи, за некоторым отличием. Напишем данную функцию в файле **eepromext.c**.

Сначала в данном файле создим глобальную переменную для статуса ошибки, так как вернуть нам надо будет байт, считанный из шины, а возвращаемых аргументов может быть только один, и если что-то не так пойдёт. то мы подключим в главном файле

программы данную переменную и там её считаем. Но, надеюсь, что всё пойдёт нормально, и она нам просто не потребуется

```
#include "eepromext.h"
char err1=0;// сюда вернем ошибку
```

Теперь непосредственно сама функция чтения

```
unsigned char EE_ReadByte(void)
{
  err1=0;
  TWCR = (1<<TWINT)|(1<<TWEN)|
(1<<TWEA);//включим прием данных
  while(!(TWCR & (1<<TWINT)));//
подождем пока установится TWIN
  if ((TWSR & 0xF8) !=
TW_MR_DATA_ASK) err1=1;
  else err1=0;
  return TWDR;
}
```

Главное отличие данной функции от функции чтения состоит в том, что мы теперь не заносим никаких данных в регистр **TWDR**, а, наоборот возвращаем считанное значение из него в качестве возвращаемого аргумента функции.

В управляющем регистре мы практически включаем те же биты, и плюс бит, разрешающий подтверждение от ведомого.

Также затем мы ждём установки флага в 0 по окончанию заполнения регистра TWDR, затем проверяем статус и возвращаем значение регистра данных **TWDR**.

Вот, в принципе и вся функция. Но данная функция нам подходит для чтения не всех байтов. Байт последний, как мы знаем читается по-особому, поэтому для него будет похожая, но несколько другая функция

```
unsigned char EE_ReadLastByte(void)
{
  TWCR = (1<<TWINT)|(1<<TWEN);//
включим прием данных
  while(!(TWCR&(1<<TWINT)));//
подождем пока установится TWIN
  if ((TWSR & 0xF8) !=
TW_MR_DATA_NASK) err1=1;
  else err1=0;
  return TWDR;
}
```

Здесь практически всё один в один, за исключением лишь того, что мы не включаем бит TWEA, разрешающий подтверждение от ведомого, а также статус отслеживаем тоже другой — 0x58.

Ну вот. Соответственно, на все наши функции должны быть прототипы в файле **eepromext.h**

```
int EE_WriteByte(unsigned char c);
```

```
unsigned char EE_ReadByte(void);
unsigned char EE_ReadLastByte(void);
#endif /* EEPROMEXT_H_ */
```

Идём теперь в функцию main() и продолжим код.

Мы также здесь можем использовать цикл, но в нём мы считаем только 31 байт, так как 32-й мы считываем другой функцией

```
I2C_SendByte(0b10101111);//передаем
адрес устройства и бит чтения (1)
USART_Transmit(TWSR);//читаем
статусный регистр
for(i=0;i<=30;i++)
{
  bt[i] = EE_ReadByte(); //прочитаем
байт из микросхемы
  USART_Transmit(TWSR);//читаем
статусный регистр
}
```

Также мы будем сначала смотреть статусы операций, а не сами считанные байты, которые мы, как видим, складываем в наш массив.

Считаем последний байт, используя соответствующую функцию

```
  bt[i] = EE_ReadByte(); //прочитаем
байт из микросхемы
  USART_Transmit(TWSR);//читаем
статусный регистр
}
bt[31] = EE_ReadLastByte(); //
прочитаем байт из микросхемы
USART_Transmit(TWSR);//читаем
статусный регистр
```

Ну и теперь сгенерируем условие СТОП

```
bt[31] = EE_ReadLastByte(); //
прочитаем байт из микросхемы
USART_Transmit(TWSR);//читаем
статусный регистр
I2C_StopCondition(); //Отправим
условие STOP
USART_Transmit(TWSR);//читаем
статусный регистр
```

Давайте пока не будем смотреть принятые байты, а, запустив терминальную програму, запустив в ней соединение, собрав код и прошив контроллер, посмотрим статусы

08	8	00001000
18	24	00011000
28	40	00101000
28	40	00101000
10	16	00010000
40	64	01000000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
50	80	01010000
58	88	01011000
F8	248	11111000

Давайте найдём все новые статусы в таблице. **0x08**, **0x18**, **0x28** и **0xF8** мы уже знаем из прошлой части занятия. Осталось нам разобраться со статусами **0x10**, **0x40**, **0x50** и **0x58**.

Для этого соберём их из таблицы даташита нашего контроллера

Table 67. Status codes for Master Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Applic… To/from TWDR
0x10	A repeated START condition has been transmitted	Load SLA+R or
0x40	SLA+R has been transmitted; ACK has been received	No TWDR action or
0x50	Data byte has been received; ACK has been returned	Read data byte or
0x58	Data byte has been received; NOT ACK has been returned	Read data byte or Read data byte or

0x10 — Повторное условие СТАРТ передано

0x40 — Адрес ведомого устойства плюс бит чтения передан и подтверддён ведущим

0x50 — Байт данных из ведущего был принят, подтверждение возвращено

0x58 — Байт данных из ведущего был принят, возвращено "нет подтверждения".

Как видим, всё у нас правильно, всё соответстсвует техничегской документации. Осталось нам только посмотреть принятые байты

Для этого ещё в одном цикле уже из массива, в который они считаны из шины, отправим их в USART, перед этим закомментировав в момент приёма отправку статусов в данную шину

```
for(i=0;i<=30;i++)
{
  bt[i] = EE_ReadByte(); //прочитаем байт из микросхемы
  //USART_Transmit(TWSR);//читаем статусный регистр
}
```

```
bt[31] = EE_ReadLastByte(); //
прочитаем байт из микросхемы
//USART_Transmit(TWSR);//читаем
статусный регистр
I2C_StopCondition(); //Отправим
условие STOP
USART_Transmit(TWSR);//читаем
статусный регистр
for(i=0;i<=31;i++)
{
  USART_Transmit(bt[i]); //отправим
считанные байты в ПК
}
```

Соберём код, прошьём контроллер и прочитаем результат чтения в терминале

08	8	00001000
18	24	00011000
28	40	00101000
28	40	00101000
10	16	00010000
40	64	01000000
F8	248	11111000
30	48	00110000
31	49	00110001
32	50	00110010
33	51	00110011
34	52	00110100
35	53	00110101
36	54	00110110
37	55	00110111
38	56	00111000
39	57	00111001
3A	58	00111010
3B	59	00111011
3C	60	00111100
00	0	00000000
3E	62	00111110
3F	63	00111111
40	64	01000000
41	65	01000001
42	66	01000010
43	67	01000011
44	68	01000100
45	69	01000101
46	70	01000110
47	71	01000111
48	72	01001000
49	73	01001001
4A	74	01001010
4B	75	01001011
4C	76	01001100
4D	77	01001101
4E	78	01001110
4F	79	01001111

Мы видим здесь все наши байты, то есть процедура чтения также прошла удачно, а также она показала и удачность процедуры записи.

Таким образом. мы с Вами уже как следует изучили специфику шины I2C, организацию её в контроллерах AVR, а также научились её программировать. Также мы познакомились с микросхемой EEPROM, подключили её по данной шине и смогли записать в неё байты и считать.

На **следующем занятии** мы закрепим полученные знания и подключим по шине I2C переходник для модуля символьного дисплея, тем самым также получим возможность подключать дисплей по двум проводам.

Исходный код

Техническая документация на микросхему AT24C32

Программатор и модуль RTC DS1307 с микросхемой памяти можно приобрести здесь:

Программатор (продавец надёжный) **USBASP USBISP 2.0**

Модуль RTC DS1307 с микросхемой памяти

Смотреть ВИДЕОУРОК (нажмите на картинку)

Post Views: 551

‹ AVR Урок 16. Интерфейс TWI (I2C). Часть 5

AVR Урок 16. Интерфейс TWI (I2C). Часть 7 ›

2 комментария на “AVR Урок 16. Интерфейс TWI (I2C). Часть 6”

Павел:
Март 30, 2017 в 4:17 дп

вот в результате чтения не вижу 3D, идут 3B 3C потом 00 и далее 3F… потери?

Ответить

admin:
Март 30, 2017 в 4:05 пп

Не знаю, видимо несоответствие частот какое-то. Посмотрите фьюзы

Ответить

Добавить комментарий

Ваш e-mail не будет опубликован. Обязательные поля помечены *

Комментарий

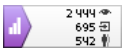
Имя *

E-mail *

Сайт

пять × = **30**

Отправить комментарий



Наверх