Главная › I2C › AVR Урок 16. Интерфейс TWI (I2C). Часть 3

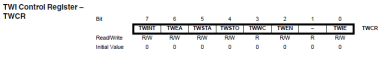
AVR Урок 16. Интерфейс TWI (I2C). Часть 3

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

Урок 16
Часть 3

Интерфейс TWI (I2C)

В **предыдущей части** занятия мы продолжили знакомство с шиной I2C и уже создали проект для того, чтобы занятия проработать на практике. Также мы познакомились с интересой микросхемой EEPROM **AT24C32**, работающей на данной шине.
Продолжаем изучение шины.
Шину мы инициализировали в проекте, выставив ей скрость с помощью регистра TWBR, теперь давайте поближе познакомимся с управляющим регистром — TWCR



Как мы видим, в данном регистре присутствует насколько битов.

TWINT — это бит прерываний. Можно его назвать битом не управляющим, а статусным, так как устанавливается он аппаратно в тот момент, когда определённое задание на шине завершится и будет ожидаться реакция

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

Мета

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

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

программы. А вот сбрасывается данный флаг не аппаратно, а только программно — записью в него логической 1.

TWEA — бит или флаг, разрешающий подтверждение. Если мы его не установим, то мы не будем просить подтверждение от ведомого устройства, а в случае, если контроллер наоборот является ведомым устройством, то с очередной посылкой мы не отправим бит подтверждения в конце какой-то посылки.

TWSTA — бит установки или генерирования условия "Старт".

TWSTO — бит установки или генерирования условия "Стоп".

TWWC — бит ошибочной записи. Устанавливается при попытки записи в адресный буфер, когда флаг TWINT ещё не установился. Ещё называется он флагом коллизий. Данный бит сбросится, когда TWINT будет равен 1.

TWEN — бит, активирующий шину I2C. Если мы его устанавливаем, то шина I2C начинает пытаться выполнять задание в зависимости от условий.

TWIE — бит, который разрешает прерывания.

Теперь статусный регистр **TWSR**

TWI Status Register – TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Пять старших битов регистра содержат код статуса операции, которая выполнялась перед тем, как мы читаем регистр. Как правило мы с помощью маскирования сбрасываем три младших бита в 0, ну конечно не в самом регистре, а в переменной, в которую мы его считали и затем уже исследуем полученный результат. В даташите на контроллер содержится перечень в виде нескольких таблиц различных кодов статуса. Несколько их потому, что есть несколько условий — ведущее или ведомое устройство и чтение или запись происходила.

Ну а два младших бита — **TWPS1** и **TWPS0** — это биты для делителя частоты шины, с которыми мы уже знакомились немного в **1 части занятия**.

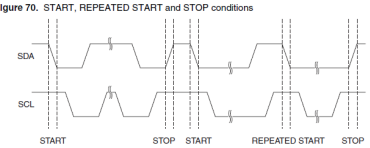
Вот такие вот могут быть варианты комбинаций данных битов и зависимость делителя от этих комбинаций

Table 65. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

Займёмся теперь непосредственно шиной. Как же всё работает:

Figure 70. START, REPEATED START and STOP conditions

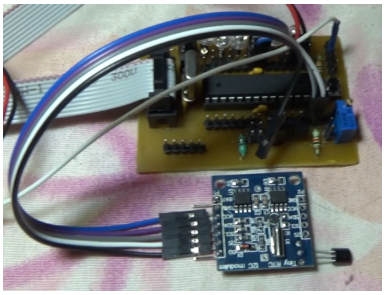


Рубрики

- 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)

Нам для начала любой посылки необходимо сгенерировать условие СТАРТ, чтобы ведущие устройства "проснулись" и начали приём, а потом уже "думали", не их ли адрес к ним пришёл. Условие СТАРТ генерируется путём перехода из высокого логического состояния шины SDA в низкое (отрицательного фронта), а затем черз некоторое время должно то же самое произойти и с шиной SCL. Вот тогда ведущий и поймёт, что по шине началась какая-то передача. А если контроллер у нас ведомый, то мы наоборот должны отследить данный процесс на наших проводах.
Но у нас шина аппаратная и париться на этот счёт нам не нужно, ибо всё контроллер сделает сам.
Соответственно, как мы видим из графика наверху, условие СТОП генерируется наоборот. Сначала положительный фронт на шине SCL, а затем на SDA.

В случае, если контроллер у нас ведомый, то мы, для того, чтобы сгенерировать условие СТАРТ, должны сделать следующее.
Вот таким вот образом у меня подключен модуль с микросхемой



Вернёмся в проект и напишем для генерации условия СТАРТ отдельную функцию в файле twi.c

```
void I2C_StartCondition(void)
{
  TWCR = (1<<TWINT)|(1<<TWSTA)|
(1<<TWEN);
  while(!(TWCR&(1<<TWINT)));//
подождем пока установится TWIN
}
```

Вот такая вот интересная функция. Что же здесь происходит?
А происходит следующее.
Мы сначала устанавливаем определённые биты в регистре управленя, говоря при этом шине о том, что мы посылаем условие СТАРТ (**TWSTA**), а также запускаем шину (**TWEN**). Ну а бит TWINT мы соответственно устанавливаем в единицу. А в ноль он, соответственно установится тогда, когда данное задание закончится. Вот для этого и существует вторая строка, где мы висим в цикле, пока

он, собственно, и не установится. Именно в ноль! А не в единицу. Об этом говорит восклицательный знак в условии.
Давайте сразу и воспользуемся данной функцией, соответственно сначала создав на неё прототип, а затем вызвав в функции main(). Сначала мы, конечно, будем во внешнюю память EEPROM писать, так как читать из неё ещё нечего. Поэтому начнём в main() писать следующий код

```
I2C_Init();

//Чтение
I2C_StartCondition(); //Отправим
условие START
```

Ну и, раз уж у нас есть чем, то давайте считаем статус операции и посмотрим успешно ли всё у нас прошло.
Для этого мы просто отправим значение статусного регистра в шину USART. Младшие три бита мы маскировать не будем, они у нас и так все в нулях.

```
I2C_StartCondition(); //Отправим
условие START
USART_Transmit(TWSR);//читаем
статусный регистр
```

Запустим терминальную программу, нажмем там **Connect**, соберём код и прошьём контроллер.
И вот мы что там видим

Посмотрим в таблице данный статус

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware
0x08	A START condition has been transmitted

То есть условие СТАРТ у нас сгенерировано и отправлено.
Остальные эксперементы с шиной I2C мы будем проделывать в **следующей части**.

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: 575

‹ AVR Урок 16. Интерфейс TWI (I2C). Часть 2

AVR Урок 16. Интерфейс TWI (I2C). Часть 4 ›

Добавить комментарий

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

Комментарий

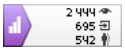
Имя *

E-mail *

Сайт

шесть − = ●

Отправить комментарий



Наверх