stm8l software IIC (I2C)

STM8

It is often necessary to connect a sensor to a microcontroller via the I2C protocol. For this, you can use the built-in I2C of the microcontroller or write your own, software. First, you need to familiarize yourself with the theory. The theory is described in great detail here . Having familiarized yourself with the theory, we move on to practice. For the **STM8L152C6T6** microcontroller , we will write a simple example when there is one **master** on the bus . The example will be for **IAR** .

For hardware implementation of I2C in our microcontroller, pins C0 and C1 are used. We use them in our example. C0 will be the SDA data line, C1 will be the SCL clock line. From the theory it is clear that each of these lines can work both for reception and transmission. When the microcontroller starts, all zeros are written in the output register of port C. If we switch, for example, C0 to the output, then it will be pulled to "0". And if we switch it to the input, then "1" will appear on it due to pull-up resistors. Therefore, we will transmit "0" or "1" by switching the pin from the output to the input. We will read the state of the pin, as usual, by setting it to the input. We will write all these modes for SDA and SCL in the corresponding defines:

```
#define SDA_UP     (PC_DDR_bit.DDR0 = 0)   // отпустил SDA в "1"
#define SDA_DOWN   (PC_DDR_bit.DDR0 = 1)   // установил SDA в "0"
#define SDA         PC_IDR_bit.IDR0        // состояние SDA (чтение)

#define SCL_UP     (PC_DDR_bit.DDR1 = 0)   // отпустил SCL в "1"
#define SCL_DOWN   (PC_DDR_bit.DDR1 = 1)   // установил SCL в "0"
#define SCL         PC_IDR_bit.IDR1        // состояние SCL (чтение)
```

If we later want to transfer our I2C to other pins, we will just need to correct these defines. Attention, rake! C0 and C1 will never burn out on the I2C bus because they simply do not have a Push-pull mode. If you use other pins, you must strictly monitor so as not to write "1" to them by mistake. You can read more about the pins here .

To implement the I2C protocol, we will need only 4 commands:
- START; (start of data exchange)
- STOP; (end of data exchange)
- WRITE; (send a byte to the slave)
- READ; (read a byte from the slave)

Since I2C devices are often quite slow, you will have to forcibly limit the data exchange speed. For this, you will need a small time delay Short_Delay. With a core clock of 2 MHz, the delay will be 17 us, with a core clock of 16 MHz, the delay will be 2 us. If necessary, the delay can be easily increased or removed

Live

Comments Publications

penzet → Sprint Layout in OS X 18 → Software for electronics engineer

Vga → EmBitz 6 → Software for electronics engineer

Vga → Rail-to-rail: ideal OU or a clever marketing ploy? 1 → Theory, measurements and calculations

Flint → A little more about 1-wire + UART 56 → Hardware connection to the computer.

penzet → Cross-platform terminal - SerIO 3.x 25 → Software for electronics engineer

Gornist → PIP regulator 2 → Algorithms and software solutions

whoim → CC1101, Treatise on Tracing 89 → Blog im. khomin

OlegG → Clock on Bluetooth LE module 8 → Cypress PSoC

Technicum505SU → Nuances of PWM control of a DC motor by a microcontroller 3 → Theory, measurements and calculations

sunjob → DDS synthesizer AD9833 88 → Blog named after grand1987

DIHALT → W801, LCD screen and fly in the ointment 2 → Detail

nictrace → New Arduino-compatible board. 20 → FPGA

sunjob → Connecting the ARM GCC compiler to CLion 1 → Software for electronics engineers

Vga → CRC32: on STM32 as on PC or on PC as on STM32. 58 → STM32

dmitrij999 → Capturing images from a USB camera using STM32 6 → STM32

Gilaks → Expanding the capabilities of a simple MC up to an ADC on 2 or 1 pin. 8 → Theory, measurements and calculations

sva_omsk → Lithium ECAD - Russian PCB CAD 40 → Software for electronics engineer

x893 → W801 - budget controller with Wi-Fi 4 → Details

podkassetnik → Changing the standard instrument cluster lighting of Logan-like cars 5 → Automotive electronics

Vga → ROPS (Rem Object Pascal Script) - embedded interpreter of the Pascal language. Plugin PSImport_Classes 3 → Algorithms and software solutions

Full broadcast | RSS

altogether.

Let's open the theory again and write down all these 4 commands:

START

forms a start condition.

```
void I2C_START()
{
      Short_Delay(1);

      SDA_UP;    // отпустил SDA в "1"
      Short_Delay(1);

      SCL_UP;    // отпустил SCL в "1"
      Short_Delay(1);

      SDA_DOWN; // SDA в "0"
      Short_Delay(1);

      SCL_DOWN; // SCL в "0"
      Short_Delay(1);
}
```

STOP

forms a stop condition.

```
void I2C_STOP()
{
      SCL_DOWN; // SCL в "0"
      Short_Delay(1);

      SDA_DOWN; // SDA в "0"
      Short_Delay(1);

      SCL_UP;     // отпустил SCL в "1"
      Short_Delay(1);

      SDA_UP;     // отпустил SDA в "1"
}
```

WRITE

sends a byte to the slave. Data_out is the byte that will be sent. This can be the slave address or the data itself. At the same time, we control the slave's response ACK.

```
void I2C_WRITE(uint8_t Data_out)
{
      Short_Delay(1);
      for (uint8_t n=0; n<8; n++)
      {
         if(Data_out & 0x80){
         SDA_UP;      // отпустил SDA в "1"
         while(!SDA); // ждём освобождения SDA в "1"
         }
         else {
         SDA_DOWN; // SDA в "0"
         }

         Short_Delay(1);

         SCL_UP;      // отпустил SCL в "1"
         while(!SCL); // ждём освобождения SCL в "1"
         Short_Delay(1);

         SCL_DOWN; // SCL в "0"
```

```
         Short_Delay(1);

         Data_out <<= 1;   //  сдвиг влево
      }

      SDA_UP;     // отпустил SDA в "1"
      Short_Delay(1);
      SCL_UP;      // отпустил SCL в "1"
      while(!SCL); // ждёт освобождения SCL в "1"
      Short_Delay(1);


      if(SDA){   // читает состояние SDA
         Error = 1;  // нет подтверждения ACK
         }
         else {
         Error = 0;  // есть подтверждение ACK
         }

      SCL_DOWN; // SCL в "0"
      Short_Delay(1);
      SDA_DOWN; // SDA в "0"
      Short_Delay(1);
}
```

READ

reads a byte from the slave. The return value Data_in is the byte that is received. Ack_NoAck indicates that this is the last byte and it is time to send a NACK to the slave.

```
uint8_t I2C_READ(uint8_t Ack_NoAck)
{
      uint8_t Data_in = 0;

      Short_Delay(1);

      for (uint8_t n=0; n<8; n++)
      {
        Data_in <<= 1;

        SDA_UP;  // отпустил SDA в "1"
        Short_Delay(1);

        SCL_UP;      // отпустил SCL в "1"
        while(!SCL); // ждёт освобождения SCL в "1"
        Short_Delay(1);

        Data_in |= SDA;   // читает состояние SDA

        SCL_DOWN; // SCL в "0"
        Short_Delay(1);
      }

      if(!Ack_NoAck){  // ACK

        SDA_DOWN; // SDA в "0"
        Short_Delay(1);

        SCL_UP;  // отпустил SCL в "1"
        while(!SCL); // ждёт освобождения SCL в "1"
        Short_Delay(1);

        SCL_DOWN; // SCL в "0"
        Short_Delay(1);

      }
      else {  // NACK ЭТОТ БАЙТ ПРИНИМАЕТСЯ ПОСЛЕДНИМ

        SDA_UP;  // отпустил SDA в "1"
        while(!SDA); // ждёт освобождения SDA в "1"
```

```
        Short_Delay(1);

        SCL_UP;  // отпустил SCL в "1"
        while(!SCL); // ждёт освобождения SCL в "1"
        Short_Delay(1);

        SCL_DOWN; // SCL в "0"
        Short_Delay(1);

        SDA_DOWN; // SDA в "0"
        Short_Delay(1);
      }

return(Data_in);
}
```

Short_Delay

small time delay.

```
void Short_Delay(volatile uint8_t delay_time)
{
  while(delay_time-- > 0);
}
```

I highly recommend connecting an oscilloscope or logic analyzer to the microcontroller and looking at each command separately. To connect external synchronization of the oscilloscope in the example, I programmed the PF0 pin.

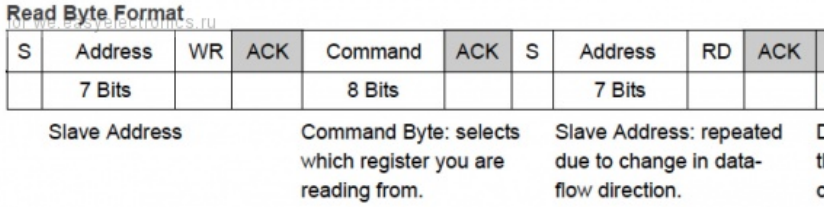
In our example, we use TIM4 to poll our I2C sensor once every 3 seconds. Configuring TIM4.

```
  CLK_PCKENR1_bit.PCKEN12 = 1; //Включаем тактирование таймера 4
  TIM4_PSCR = 0x0F; // Предделитель на  32768 0x0F
  TIM4_ARR = 0xB6;  // Считать до 183-1 -1 вычесть 1 чтобы деление было
  TIM4_IER_bit.UIE = 1;  // Разрешаем прерывание
  TIM4_CR1_bit.URS = 1;  //Прерывание только по переполнению счетчика
  TIM4_EGR_bit.UG = 1;   //Вызываем Update Event, чтобы обновился предде
  TIM4_CR1_bit.CEN = 1;  // Активируем таймер получаем 2 000 000 / 32768
  // таймер срабатывает каждые 3 S
```

The timer will be triggered on overflow and raise the ready flag in the main loop.

```
#pragma vector = TIM4_UIF_vector // Прерывание по переполнению TIM4
__interrupt void TIM4_UIF(void)
{
 TIM4_Start = 1; // поднимаем флаг готовности TIM4
 TIM4_SR1_bit.UIF = 0; // Обнуляем бит выхода из прерывания
}
```

Now let's connect the I2C temperature sensor. Let's take the simplest and cheapest TC74A0. At 3.3V or at 5.0V - any will do. You can download the datasheet for TC74A0 here . The TC74A0 sensor has an address of 72 and by default is ready to transmit the temperature to the master. Open the Read Byte Format protocol:



How to read the temperature from TC74A0? It is stored at address 0x00. Using

the four commands we prepared in advance, we implement what we see in the picture:

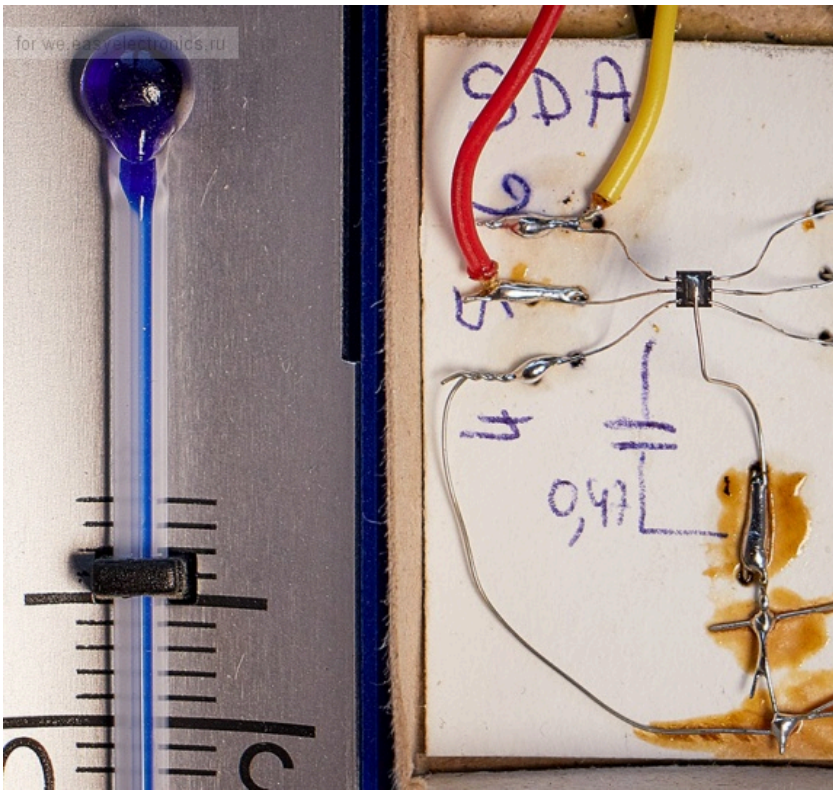
```
while(1)
{

  if(TIM4_Start){  // таймер сработал

      Error = 0;   // флаг ошибки сброшен
      uint8_t Slave_Address = 72;   // 1001 000
      Slave_Address <<= 1;          // 1001 0000
      int16_t temperature;  // температура с датчика

      I2C_START();  // старт
      I2C_WRITE(Slave_Address);  // Адрес для последующей ЗАПИСИ
      I2C_WRITE(0x00);   // БАЙТ НА ОТПРАВКУ 0000 0000
      Short_Delay(5); // чтобы увидеть зазор межну командами
      I2C_START();
      I2C_WRITE(Slave_Address | 1);  // Адрес для последующего ЧТЕНИЯ
      temperature = I2C_READ(1); // ЧИТАЕТ БАЙТ  (1) - последний
      I2C_STOP();

      if(Error == 0){
      printf("t = %d\n",temperature);   // печатает температуру
      }
      else {  // Error = 1 нет подтверждения АСК
      printf("Error = %d\n",Error);   // всякий раз с новой строки
      }

      TIM4_Start = 0;  // обнуляем флаг готовности TIM4
  }
};
```

The obtained temperature value is printed or displayed on your indicator. On the logic analyzer, we get this neat picture:

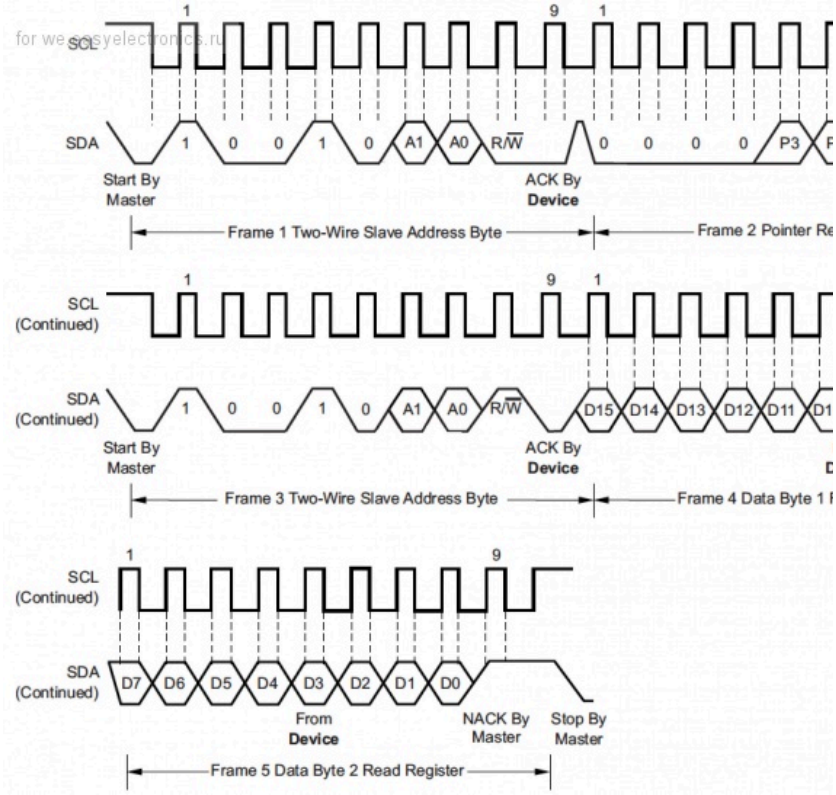
Now let's connect a cooler I2C temperature sensor - TMP116. It has an accuracy of +/- 0.2 degrees, and a temperature range from -40 to +125 degrees! You can download the datasheet for TMP116 here . We connect the TMP116 sensor to the microcontroller, as shown in the photo:

1 leg SCL
2 leg GND
3 leg do not use connect to GND
4 leg address selection. Connect to GND the address will be 72
5 leg + power supply 3.0V Connect to the power supply of the microcontroller.
6 leg SDA
Termo Pad - you can connect anywhere, I connected to GND

Do not forget to connect pull-up resistors to SCL and SDA (I installed 10K). Be sure to install a blocking capacitor for power supply. Mine is soldered to the back of the cardboard. The TMP116 sensor is small, but you can solder it.



Now the TMP116 sensor has an address of 72 and is ready to transmit the temperature to the master by default. In the datasheet we find the Read Word Command Timing Diagramm protocol:



How to read the temperature from TMP116? It is stored at the address 0x00. Using four commands that we have prepared in advance, we implement what we see in the picture:

```
I2C_START();  // старт
I2C_WRITE(Slave_Address);  // Адрес для последующей ЗАПИСИ
I2C_WRITE(0x00);   // БАЙТ НА ОТПРАВКУ 0000 0000
Short_Delay(5); // чтобы увидеть зазор межну командами
I2C_START();  // повторный старт
```

```
    I2C_WRITE(Slave_Address | 1);  // Адрес для последующего ЧТЕНИЯ
    temperature = I2C_READ(0); // ЧИТАЕТ БАЙТ  (0)
    temperature <<= 8;
    temperature |= I2C_READ(1); // ЧИТАЕТ БАЙТ  (1) - последний
    I2C_STOP();

    temperature = (int32_t)temperature * 100 / 128;
```

We insert this code instead of the code of the previous example. The temperature value now takes up 2 bytes. The temperature is printed with an accuracy of hundredths of a degree! If you output this value to the indicator, light the dot in the second digit. On the logic analyzer, we get an equally neat picture:

TMP116 is a fairly complex and sophisticated sensor. But now you can program it the way you want, using the datasheet and a set of 4 simple commands. I hope that now connecting any I2C slave to the STM8L will not be difficult for you. If there is interest, next time we will analyze the hardware I2C. Good luck to everyone! Example in the attachment.

stm8l , software I2C , I2C , IIC , TMP116 , TC74 , temperature sensor , microcontroller

+4 May 29, 2019, 5:22 PM **CreLis** 1

Files in the topic: I2C_TC74A0_PRINT.zip

Comments (6)

RSS Collapse / Expand

Thank you. Interesting, necessary, useful. But I don't really like I2C, it's a bit of a pain. It's easier with uart or spi. But in fact, I have to work with I2C most often. 0

Papandopala
May 29, 2019, 6:15 PM

> *at 2 mhz delay is 17 us*

And I thought only the Chinese on Aliexpress measure the frequency of radio modules in millihertz. For a radio amateur - a shame and disgrace... 0

Lifelover
May 29, 2019, 11:19 PM

Corrected, thanks! 0

CreLis
May 29, 2019, 11:31 PM ↑

> *slave answer ASK*
>
> *time to send to NASK slave*

Actually, they are called ACK and NAK.

Why at the beginning of the START and STOP procedures is the bus set to the state in which it should already be? And even more so - why the delays at this?

I would put the bodies of the SDA*/SCL* macros in brackets. +1

Vga
May 30, 2019, 02:02

0

All comments are relevant. Corrected. Thank you! In the START and STOP procedures there are unnecessary bus translations, I agree. This is so that all commands can be observed on the oscilloscope separately. These bus translations, of course, can be removed and the time of access to the slave can be reduced.

CreLis
May 30, 2019, 7:12 PM

Oh, I just did the same thing for Chinese LoRa modules with the same processor inside.

0

tipok
June 10, 2019, 15:05

Only registered and authorized users can leave comments.