Coding Laboratory

A collection of small hacks

Tuesday, 14 October 2008

I2C on an AVR using bit banging

As a exercise I tried to talk to a I2C temperature sensor using bit banging, it was not as easy as I thought so I decided to post the code in case anyone needs to see the solution, if you happen to use my code drop me a line since that will encourage me to post more code :-)

```
// Port for the I2C
#define
#define
#define

// Pins to be used in the bit banging
#define         0
#define         1

#define             ()
        &= ~ (1 <<         );
         |= (1 <<         );
#define             ()
        |= (1 <<         );
         &= ~ (1 <<         );

#define             ()
        &= ~ (1 <<         );
         |= (1 <<         );
#define             ()
        |= (1 <<         );
         &= ~ (1 <<         );

void            (unsigned char  )
{
    if (  > 0)
    {
                ();
    }
    else
    {
                ();
    }

              ();
         (1);

              ();
         (1);

    if (  > 0)
    {
                ();
    }

         (1);
}

unsigned char           ()
{
              ();

               ();
```

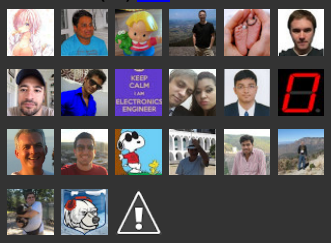
About Me

Raul

View my complete profile

Followers

Followers (74) Next



Follow

Blog Archive

April (1)
February (1)
September (1)
June (1)
October (1)
July (1)
June (1)
February (2)
January (1)
October (1)
September (2)
October (1)
September (2)
September (1)
June (1)
May (1)
March (3)
February (1)
October (2)
April (1)
January (1)
November (1)
July (1)
March (1)
February (1)
October (3)

```
        (1);

    unsigned char   =       ;

            ();
        (1);

    return (  >>       ) & 1;
}

// Inits bitbanging port, must be called before using the functions below
//
void        ()
{
            &= ~ ((1 <<       ) | (1 <<       ));

            ();
           ();

        (1);
}

// Send a START Condition
//
void         ()
{
    // set both to high at the same time
          &= ~ ((1 <<       ) | (1 <<       ));
        (1);

            ();
        (1);

            ();
        (1);
}

// Send a STOP Condition
//
void        ()
{
            ();
        (1);

            ();
        (1);
}

// write a byte to the I2C slave device
//
unsigned char          (unsigned char  )
{
    for (char   = 0;   < 8;  ++)
    {
                (  & 128);

         <<= 1;
    }

    //return I2C_ReadBit();
    return 0;
}


// read a byte from the I2C slave device
//
unsigned char         (unsigned char    )
{
    unsigned char     = 0;

    for (char   = 0;   < 8;  ++)
    {
        <<= 1;
        |=           ();
    }
```

```
    if (    > 0)
    {
                    (0);
    }
    else
    {
                    (1);
    }

         (1);

    return    ;
}
```

Posted by Raul at 13:20

Labels: i2c avr atmel bitbanging

20 comments:

zhengki 20 March 2009 at 14:44

hi raul! i think i'm gonna give your code a try. by the way, what device are you driving and what is your system clock? (for timing references)
do you have external pullup resistors?

thanks!

Reply

Travis Goodspeed 5 June 2009 at 11:13

Damned handy example. Thank you kindly for it.

Reply

Ivan 14 July 2010 at 03:48

Hey, this is exactly what I need, I'll give it a try, thanks a bunch.

Reply

toshu 1 March 2011 at 23:29

code is working fine i checked it with atmega16 on 12MHz...

Reply

Raul 2 March 2011 at 04:56

Awesome Toshu, thanks for sharing!

The Atmega16 is the server?

Reply

Henry 5 September 2011 at 03:54

Thx a lot. This is exactly what I want.

Reply

Marek 16 December 2011 at 14:30

A nice code! Tested using ATtiny25&LCD module&DS18B20 temperature sensor&internal ATtiny sensor!

Thank you a lot!!
Marek (Czech Rep.)

Reply

Marek 16 December 2011 at 14:30

A nice code! Tested using ATtiny25&LCD module&DS18B20 temperature sensor&internal ATtiny sensor!

Thank you a lot!!
Marek (Czech Rep.)

Reply

Boo plus Aroo 2 December 2012 at 01:41

Thanks for the code example. Can you tell me why you are using DDRD when you want to set the port high or low? It seems backwards to me, like you would want PORTD, right?

For example, I'd think the only time you need to set the DDR (data direction) would be during the init, and then on it's always the same. Then when you send data you'd use the PORTD command, but instead you use DDRD to send data?

I'm just ramping up to AVRs so I might be confused. Would appreciate any clarification, thanks!

Reply

Sharath Bhargava 12 August 2013 at 01:38

Thanks dude:-)
i am going to refer the same

is it possible to share the datsheet of your microcontroller?

Reply

rr2000 2 January 2014 at 02:25

Hi Raul, thank you for sharing. I want to use your code as a base for a I2C lib for the http://www.smarthomatic.org project. What kind of license do you want to apply to your code? GPL V3 would be great.
Thanks,
rr2000

Reply

rr2000 2 January 2014 at 02:29

Hi Raul, thank you for sharing. I want to use your code as a base for a I2C lib for the http://www.smarthomatic.org project. What kind of license do you want to apply to your code? GPL V3 would be great.
Thanks,
rr2000

Reply

Replies

Raul 3 January 2014 at 03:54

Vi rr2000,

Im happy you find my lib useful, gpl v3 sound good to me if that helps you, feel free to add a reference to this article and give me credits :)
Good luck and all the best!

Reply

Sarge Bowhack 4 January 2014 at 05:45

Some comments:

This code in I2C_WriteBit() seems to be preparing for a stop bit. You don't need it unless the next action is a stop, so it belongs in the beginning of the I2C_Stop() routine, not in I2C_WriteBit().

```
if (c > 0)
{
I2C_DATA_LO();
}
delay(1);
```

(This isn't actually *wrong*, really, but it's an unnecessary step that makes the code a little harder to understand, IMO.)

However, there is something legitimately wrong with this code.
In both I2C_WriteBit() and I2C_ReadBit(), you must wait for clock to be high before moving on. If you read the I2C specification more closely, both the master and the slave(s) can control the clock line. When a slave controls it, it is referred to as "clock stretching" and is used to slow a master that is going too fast to keep up with. If you don't take clock stretching into account, some devices will not function well with this routine.
Therefore, in both functions,

```
I2C_CLOCK_HI();
delay(1);
```

must change to:
I2C_CLOCK_HI();
while ((I2C_PIN & (1 << I2C_CLK)) == 0);
delay(1);

for reliability's sake.

Also, I don't know why "return I2C_ReadBit();" is commented out in I2C_Write(). That is the correct thing to do when writing a byte. Maybe you were running into problems with the ACK routine because clock stretching wasn't taken into account above?
Anyway, my apologies for what might be perceived as criticisms -- I think this is a very handy routine, and should do the trick in a number of cases. Just thought that with a few small tweaks, it might be even better and much more robust.
Thanks for posting it!

Reply

Bart van Deenen 21 April 2014 at 03:38

Thanks for your post and also Sarge Bowhack for his detailed reply with some comments.
I've put a small project for Arduino Ethernet reading a tmp102 sensor on github: here

Reply

Chisanga Mumba 15 December 2015 at 03:39

Nice.

Reply

Chisanga Mumba 15 December 2015 at 03:40

Nice.

Reply

JustAnotherGuy 12 February 2017 at 10:30

Thanks for the post.Have been looking for this.
I just have trouble understanding one part though.
In the I2C_ReadBit() function,won't writing HIGH to the data line before reading the pin damage the receiver pin transistor on the slave? There in no pullup resistor to limit current into the slave.
I would be glad if someone can clarify.

Reply

Unknown 17 February 2017 at 10:54

Worked like a treat

Reply

Leo Heywood 3 July 2017 at 13:37

Thanks for sharing your code. Please let me suggest some changes:

With I2C, neither pin should be "driven" high. The protocol expects the bus to be shorted to ground when it is low and expects the master to release the bus and let the pullup resistor pull it high.
For instance, if the slave wants to stretch the clock, it does this by shorting the clock line to ground and your code is driving it high, FAP!

The changes necessary to avoid this make the code much simpler:

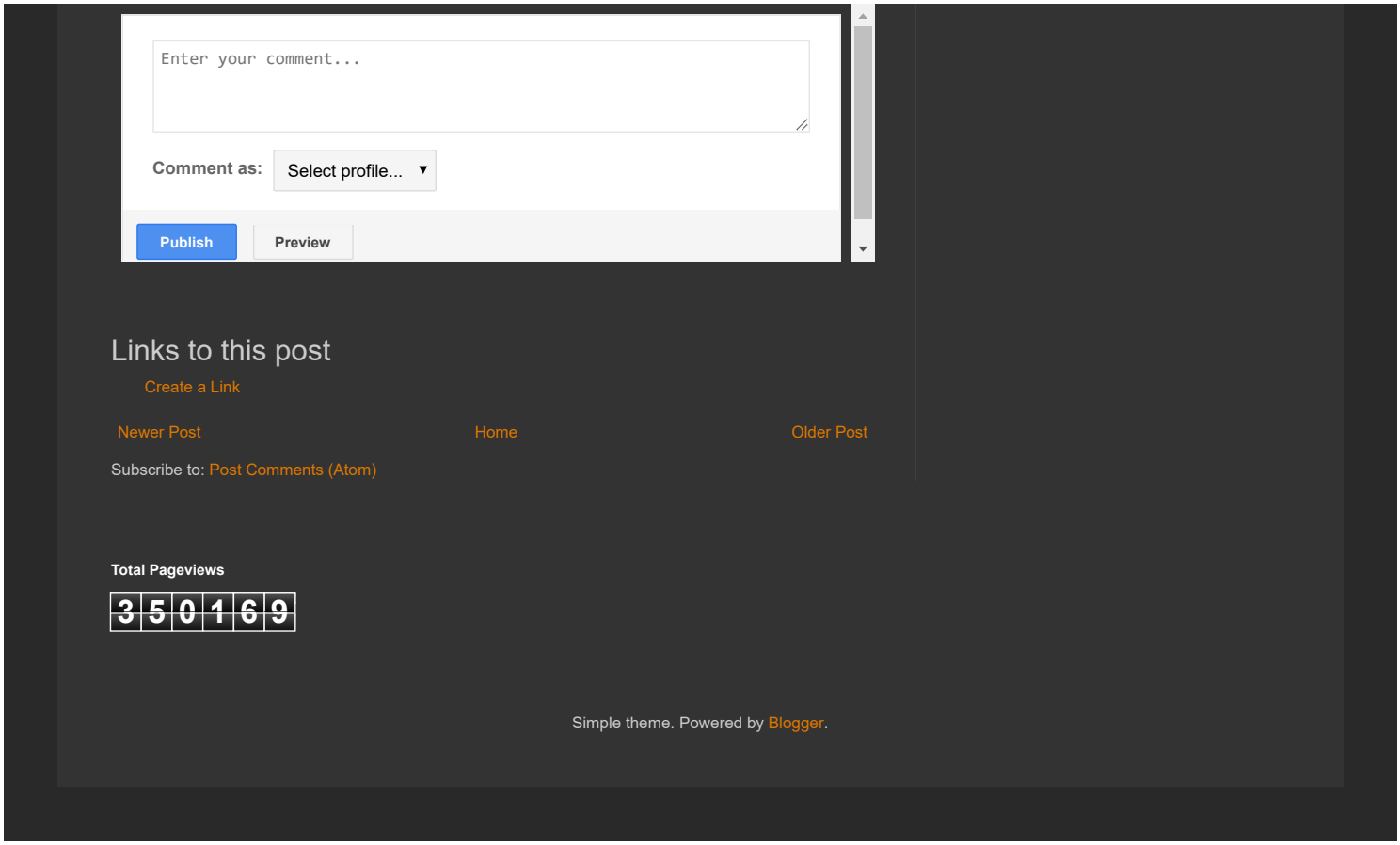
I2C_DDR &= ~((1<<I2C_CLK)|(1<<I2C_DAT)); //bus pulled hi by pullup resistor.
I2C_PORT &= ~((1<<I2C_CLK)|(1<<I2C_DAT)); // lo when output, HI-Z when input. NEVER CHANGE THIS.

The above would be your I2C initialization. Now HI and LO are defined by the data direction:

#define I2C_DATA_HI I2C_DDR&=~(1<<I2C_DATA)
#define I2C_DATA_LO I2C_DDR|=(1<<I2C_DATA), same for I2C_CLK.

Now when the slave holds the clock low, the current is limited by the pullup resistor which is usually 5K to 10 K.

Reply


Enter your comment...
Comment as:
Select profile...
Publish
Preview
Links to this post
Create a Link
Newer Post
Home
Older Post
Subscribe to: Post Comments (Atom)
Total Pageviews
350169
Simple theme. Powered by Blogger.