Delay using timer

6 posts / 0 new

Author Message

joneewheelock

Level: Hangaround
Joined: Fri. Aug 14, 2015
Posts: 327 View posts

Posted by joneewheelock: Sun. Sep 13, 2015 - 03:01 PM #1

Fivestar widget

Total votes: 0

I wrote function for delay using timer0 in Atmega328P. But sometimes the period in the below example is around 700 msec (as seen on PB1). In most cases it is 996 msec (and not 1000 msec). What is the issue with this program? First of all why I am getting only 996 msec? If it was little more than 1000 msec, I would have thought that it is due to some code in while loop.

Also sometimes why the period is reduced? I am using timer1 for updating RTC (Code is not given below) but even if I comment the code of updating RTC, I have the issue. I enabled interrupt in ISRs so that we do not miss interrupts. But if I do not use timer1, this method of delay works fine.

Is this is the right way of implementing delay using timers or any better way?

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif
```

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

volatile unsigned long delayCounter;

////////////////////////////////////////////////// 1 msec Timer  /
void initializeTimer0ForOneMilliSec (void)
{

        // In case 16 Mhz crystal is connected, each clock pulse =
        // If we use prescaler of 64, each clock pulse = 0.0625*64
        //So we can get 1 msec, we need couner of 1000/4 = 250

        OCR0A = 250-1;            // Set for 1 msec.

        TIMSK0 |= 1<<(OCIE0A);  // Output Compare Interrupt Enable
        TCCR0A |= 1<<(WGM01);  // Mode = CTC: WGM01=1, WGM00=0 in T

        // Timer is activated as soon as the clock source is select
        TCCR0B = (1<<CS01)|(1<<CS00);    // Timer Prescaler Clock/64
}


//This is the ISR for Compare Mode A

ISR(TIMER0_COMPA_vect)
{
        sei();
        delayCounter++;
}


void delaymSecUsingTimer0(unsigned long delaymSec)
{
        delayCounter=0;
        while(delayCounter<delaymSec);
}


// Use Timer0 for getting 1 second interrupts. This can be used fo
void initializeTimer1ForOneSecond (void)
{
        TIMSK1 |= (1<<OCIE1A); // Enable Interrupt Timer/Counter1, C
        TCCR1B |= ((1<<CS12) | (1<<CS10) | (1<<WGM12));     // Clock
        OCR1A = 15625-1;                          // 15625 means 1
}
```

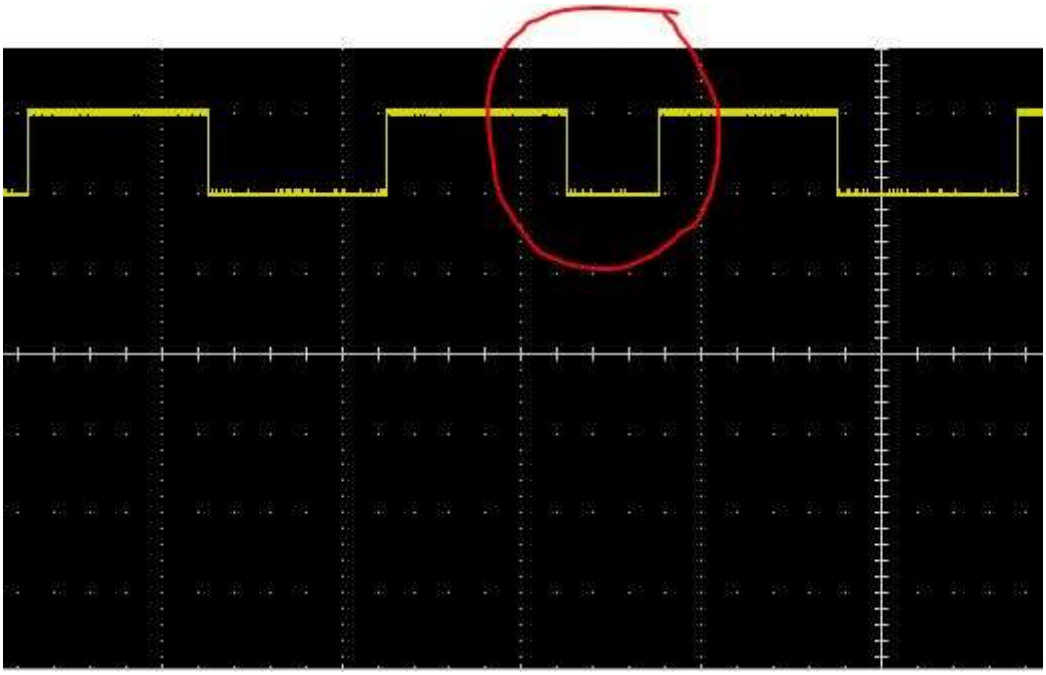
```
ISR(TIMER1_COMPA_vect)
{
        sei();
        //displayDateTime();      // Update the RTC
}

int main(void)
{

        initializeTimer1ForOneSecond();
        initializeTimer0ForOneMilliSec();
        sei();

        DDRB|=1<<1;
        while(1)
        {
                PORTB ^= 1<<1;
                delaymSecUsingTimer0(500);

        }
}
```



Tags:
Tools, Compilers and General Programming

pblase

Level: Hangaround
Joined: Thu. Nov 2, 2006
Posts: 174 **View posts**
Location: Alexandria, VA

Posted by pblase: Sun. Sep 13, 2015 - 09:26 PM **#2**

Total votes: 0

Can't see anything wrong with the basic logic, although I'd do it slightly differently. Put the comparison

```
delayCounter<delaymSec
```

in the ISR with the counter increment and just set a flag.

What might be happening is that the processor is resetting for some reason. I've found that every so often the watchdog timer is actually running, even if I think it's off, so I always use the watchdog functions. Capture and look at the MCUSR register to find the reason for the last reset.

JohanEkdahl

Level: 10k+ Postman
Joined: Wed. Mar 27, 2002
Posts: 26945 **View posts**
Location: Lund, Sweden

Posted by JohanEkdahl: Mon. Sep 14, 2015 - 06:28 AM **#3**

Total votes: 0

I don't like

1. that you do sei() explicitly in the ISRs. Not needed. The compiler will insert those at the usually most applicable spot - i.e. when the ISR exists.

2. Re the shortened delays: You are not guaranteeing atomic access when testing the variable. Since accessing delayCounter is a multiple instruction sequence, what might happen is this scenario:

To do the comparison, both bytes of the variable needs to be fetched. Lets assume that the complete variable has the value 255 (two bytes 0x00,0xFF) just as this starts. If the low byte is fetched first then it will be fetched as 0xFF. But now the ISR just happens to kick in, incrementing the complete variable, so the bytes in memory are now 0x01,0x00. When the ISR exists the comparison code resumes, and it will now fetch the high byte 0x01 and as far as the comparison goes it will now see a variable value 0x01,0xFF, i.e. decimal 511. It's like the increment went 254, 255, 511.. and your delay becomes too short.

Search this site for "atomic", "atomicity" and the like for a lot more on the subject. Since this is AVR-GCC code the compiler library also has support for helping you set up atomic access. See util/atomic.h and it's documentation.

This might or might not be your problem, but in any case this will come back and bite you sooner or later dealing with multi-byte variable shared with ISRs so you might as well get into the habit right away.

And please do remove those sei()'s. They serve no purpose but to obfuscate the code, and are potentially harmful. There are rare cases where you want to "enable interrupts early" but this is not such a situation.

Top

Log In or **Register** to post comments

skeeve

Posted by skeeve: Mon. Sep 14, 2015 - 06:05 PM

#4

Total votes: 0

996 ms = 1000 ms - 4 ms
4 ms = 1000 * 4 us
4 us = one timer tick
For some reason I do not understand,
you seem to be consistently losing one timer tick per timer period.

756 ms = 500 ms + 256 ms
You seem to have an atomicity issue:
Between 0x0FF and 0x100, the counter becomes
0x1FF while the main line code is testing. Or see edit.

BTW you do not need an ISR.
The only globals you need are the AVR registers.
Do not enable the interrupt.
Test for the interrupt flag.
When true, clear it and decrement delaymSec.

Edit:

The counter could also go from

0x0FF to 0x000 to 0x100.

The main line could fetch the FF from 0FF and the 1 from 100.

Same result.

Moderation in all things. -- ancient proverb

Last Edited: Mon. Sep 14, 2015 - 10:00 PM

Level: Raving Lunatic
Joined: Sun. Oct 29, 2006
Posts: 5534 **View posts**

Top

Log In or **Register** to post comments

joneewheelock

Posted by joneewheelock: Thu. Sep 17, 2015 - 02:09 PM **(Reply to #4)** **#5**

Total votes: 0

Thanks everyone for giving good inputs.

Level: Hangaround
Joined: Fri. Aug 14, 2015
Posts: 327 **View posts**

1. If I do not enable interrupt inside ISR and if I have couple of more instructions inside the ISR, what will happen if other interrupt occur? One second interrupt may be lost right? Or still CPU will manage other interrupt? If 1 sec interrupt is lost, it will be significant. So I thought of enabling interrupt inside ISR using sei.

Example Code where I set one more flag:

```
ISR(TIMER0_COMPA_vect)
{
     static unsigned char counter100mSec;

     sei();    //No need to enable the interrupt.
     PORTB ^= 1<<PB2;
     delayCounter++;
     counter100mSec++;
     if(counter100mSec>99)
     {
          counter100mSec=0;
          hundredmSecFlag=1;
     }
```

2. Yes, I need not use ISR. I could check the Interrupt flag as you suggested, But I am also keeping 100 msec flag inside this ISR as above and I thought it is better.

3. I am really confused with the behavior. In fact I tested by toggling port pin inside the ISR itself (As in the above code) and without doing anything in main. ISR itself is not consistent. Port toggles every 393 Hz, 500 Hz (Correct value) or sometimes even at 996 Hz. May be I need to check the crystal quality!! Not sure of any other hardware issues. So though there may be problem with atomicity (I need to go through that), but in this case the problem seems to be different.

4. I get one 502 mHz (1.99 Sec) pulses if I toggle port pin inside the TIMER1_COMPA_vect() instead of 500 mHz. Is this normal?

Top

skeeve

Posted by skeeve: Thu. Sep 17, 2015 - 03:11 PM

#6

Total votes: 0

Level: Raving Lunatic
Joined: Sun. Oct 29, 2006
Posts: 5534 **View posts**

1. Most other interrupts, including timer interrupts, will pend until handled or cleared in software.
2. Your 100 ms counter will not need to be handled in an interrupt.
3. Your code has at least two problems, one of which is atomicity.

With a .lss file, I could point it out to you.

The main line code is small enough that non-atomicity manifestations are not rare.

4. Your code has at least one problem I do not understand: Your timer periods seem to be one timer tick short.

Suggested strategy:

1. Rewrite code without ISRs.
2. Debug code from 1, possibly with help from avrfreaks.
3. Try to use lessons learned from 1 and 2 to debug ISR-based code.

4a. If successful, report back.
4b. Otherwise ask for more help.

The suggested strategy assumes two goals, getting working code and knowing WTF is going on.

Moderation in all things. -- ancient proverb

Top

Jump To

--Compilers and General Programming