Technoblogy

Arduino and AVR projects

Simple ATtiny USI UART

26th November 2014

This article describes a simple receive-only serial UART for the ATtiny85, using the USI (Universal Serial Interface), the basic serial communication module provided on most ATtiny chips.

I wrote this to receive a 9600 baud serial signal from a GPS module using an ATtiny85 processor. Although there are some published routines to do this, they were all a bit complicated, so I set out to write a minimal routine that would do just what I wanted.

ATtiny85 USI UART

One way to implement serial communication on the ATtiny85 is using software, but this requires tricky programming to get the timing correct, and ties up the processor making it difficult to synchronise reception of the data with other tasks.

The ATtiny85 does provide a hardware USI, which can be used to implement two-wire I2C or three-wire SPI, and this can be used to do part of the job in implementing a UART; Atmel have written an application note describing how to do this [1]. The application note is based on the ATtiny26 which has a more primitive timer/counter than the ones in the ATtiny85, and so requires you to reprogram the timer for each of the 8 bits in the byte being received. In the ATtiny85 the USI shift register is clocked in on a Timer/Counter0 compare match, so the task is slightly simpler; we can set the appropriate count and leave it to receive all 8 bits automatically.

The circuit uses an 8MHz crystal clock for accurate timing. The internal clock is only guaranteed to be accurate to within 10% without special calibration, and that isn't really accurate enough for a UART; after 5 bits we may be half a bit out. If you're prepared to calibrate the internal clock you could dispense with the crystal.

Operation

The hardware/software UART works as follows:

At 9600 baud, with an 8MHz clock, the duration of one bit is 8000000/9600 or 833.3 clock cycles.

First we disable the USI. The input of the USI shift-register is connected to PB0, so we define that as an input, and set up a pin-change interrupt on it:

```
void InitialiseUSI (void) {
  pinMode(DataIn, INPUT);         // Define DI as input
  USICR = 0;                      // Disable USI.
  GIFR = 1<<PCIF;                 // Clear pin change interrupt flag.
  GIMSK |= 1<<PCIE;               // Enable pin change interrupts
  PCMSK |= 1<<PCINT0;             // Enable pin change on pin 0
}
```

The start of a byte causes a pin-change interrupt. In the pin-change interrupt service routine we check that it's a falling edge, and if so we set up Timer/Counter0 in CTC mode. We want to set up a delay of half a bit, to get into the middle of the start bit, which is 416.7 cycles. The closest we can get to that is a prescaler of 8 and a compare match of 52. Finally we clear and enable the output compare interrupt:

```
ISR (PCINT0_vect) {
  if (!(PINB & 1<<PINB0)) {       // Ignore if DI is high
    GIMSK &= ~(1<<PCIE);          // Disable pin change interrupts
    TCCR0A = 2<<WGM00;            // CTC mode
    TCCR0B = 0<<WGM02 | 2<<CS00;  // Set prescaler to /8
    TCNT0 = 0;                    // Count up from 0
    OCR0A = 51;                   // Delay (51+1)*8 cycles
    TIFR |= 1<<OCF0A;             // Clear output compare flag
```

Search

Recent posts

▼ 2021
Low-Power LCD Clock
Measuring Your Own Supply Voltage
100MHz Frequency Meter
Pocket Op Amp Lab PCB
Frequency Divider Using CCL
Pocket Op Amp Lab Cookbook
I2C Detective
Pocket Op Amp Lab
Five LEDs Puzzle Solution
Five LEDs Puzzle PCB
► 2020
► 2019
► 2018
► 2017
► 2016
► 2015
► 2014

Topics

► Games
► Sound & Music
► Watches & Clocks
► GPS
► Power Supplies
► Computers
► Graphics
► Thermometers
► Tools
► Tutorials
► PCB-Based Projects

By processor

AVR ATtiny

► ATtiny10
► ATtiny2313
► ATtiny84
► ATtiny841
► ATtiny85
► ATtiny861
► ATtiny88

AVR ATmega

► ATmega328
► ATmega1284

AVR 0-series and 1-series

► ATtiny3216

```
    TIMSK |= 1<<OCIE0A;           // Enable output compare interrupt
  }
}
```

The compare match interrupt occurs in the middle of the start bit. In the compare match interrupt service routine we reset the compare match to the duration of one bit, 104, enable the USI to start shifting in the data bits on the next compare match, and enable the USI overflow interrupt:

```
ISR (TIMER0_COMPA_vect) {
  TIMSK &= ~(1<<OCIE0A);          // Disable COMPA interrupt
  TCNT0 = 0;                      // Count up from 0
  OCR0A = 103;                    // Shift every (103+1)*8 cycles
  // Enable USI OVF interrupt, and select Timer0 compare match as USI Clock source:
  USICR = 1<<USIOIE | 0<<USIWM0 | 1<<USICS0;
  USISR = 1<<USIOIF | 8;          // Clear USI OVF flag, and set counter
}
```

Note that we set the Wire Mode to 0 with 0<<USIWM0. This ensures that the output of the USI shift register won't affect the data output pin, PB1.

When 8 bits have been shifted in the USI overflow interrupt occurs. The interrupt service routine disables the USI, reads the USI shift register, and enables the pin change interrupt ready for the next byte:

```
ISR (USI_OVF_vect) {
  USICR = 0;                      // Disable USI
  int temp = USIDR;
  Display(ReverseByte(temp));
  GIFR = 1<<PCIF;                 // Clear pin change interrupt flag.
  GIMSK |= 1<<PCIE;               // Enable pin change interrupts again
}
```

The only catch is that the UART sends the bits LSB first, whereas the USI assumes that the MSB is first, so we need to reverse the order of the bits after reception. This can be done by a short software routine ReverseByte():

```
unsigned char ReverseByte (unsigned char x) {
  x = ((x >> 1) & 0x55) | ((x << 1) & 0xaa);
  x = ((x >> 2) & 0x33) | ((x << 2) & 0xcc);
  x = ((x >> 4) & 0x0f) | ((x << 4) & 0xf0);
  return x;
}
```

This works efficiently by first interchanging adjacent single bits, then interchanging adjacent 2-bit fields, then exchanging the two 4-bit fields.

Then I just call Display() to display the byte on a seven-segment display for debugging purposes, but the code to handle the received byte should go here in the final version.

Compiling the program

I compiled the program using the ATtiny core extension to the Arduino IDE [2]. This doesn't include a setting for the ATtiny85 with an 8MHz crystal, so I added the following definition to the boards.txt file:

```
##############################################################################

attiny85at8x.name=ATtiny85 @ 8 MHz  (external crystal; BOD disabled)

attiny85at8x.upload.using=arduino:arduinoisp
attiny85at8x.upload.maximum_size=8192

# Ext. Crystal Osc. 8.0 MHz; Start-up time: 16K CK/14 CK + 65 ms; [CKSEL=1111 SUT=11]
# Brown-out detection disabled; [BODLEVEL=111]
# Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
```

► ATtiny402
► ATtiny414
► ATmega4809

AVR DA/DB-series

► AVR128DA28
► AVR128DA48
► AVR128DB28

ARM

► ATSAMD21

About me

About me
Contact me

Follow @technoblogy

Feeds

RSS feed

```
# Serial program downloading (SPI) enabled; [SPIEN=0]

attiny85at8x.bootloader.low_fuses=0xFF
attiny85at8x.bootloader.high_fuses=0xD7
attiny85at8x.bootloader.extended_fuses=0xFF
attiny85at8x.bootloader.path=empty
attiny85at8x.bootloader.file=empty85at16.hex

attiny85at8x.build.mcu=attiny85
attiny85at8x.build.f_cpu=8000000L
attiny85at8x.build.core=tiny

##############################################################################
```

This adds an **ATtiny85 @ 8MHz (external crystal; BOD disabled)** option to the **Board** submenu. Select this, and choose **Burn Bootloader** to set the fuses appropriately using the Tiny AVR Programmer Board; see ATtiny-Based Beginner's Kit. Then upload the program to the ATtiny85.

Other options

To use this routine with different crystal frequencies, or for different baud rates, you will need to change the prescaler and compare-match values for Timer/Counter0 in the pin-change interrupt routine and USI overflow interrupt routine. With higher baud rates you may need to take into account the delay of handling the interrupt.

Here's the whole ATtiny85 USI UART program: Simple ATtiny USI UART Program.

Updates

10th December 2014: Changed the output compare interrupt routine, **ISR (PCINT0_vect)**, to avoid affecting the Timer/Counter1 interrupt bits. This leaves you free to use the Arduino timing functions **millis()** and **delay()** which use Timer/Counter1.

3rd May 2015: Henry Choi and Edgar Bonet have pointed out that in my original version the COMPA interrupt is called immediately, with the effect that the subsequent data bits are sampled near the beginning of each bit. The solution, incorporated above, is to clear the output compare flag before enabling the output compare interrupt, by adding the line:

```
TIFR |= 1<<OCF0A;
```

in the PCINT0_vect subroutine, and then delay half a bit, into the middle of the start bit, by changing the setting of OCR0A to:

```
OCR0A = 51;                  // Delay (51+1)*8 cycles
```

This ensures that each sample is read in the middle of each bit.

---

1. ^ AVR307: Half Duplex UART Using the USI Module (PDF).
2. ^ ATtiny core for Arduino: arduino-tiny on Google Code.

Next: Minimal GPS Parser [Updated] **Previous:** Timescale Clock

---

23 Comments Technoblogy Disqus' Privacy Policy Login

Recommend 1 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

disqus_IUhdg1pwdO • 7 months ago • edited

Thanks. I'm a bit confused with the way of writing bits to Registers.
2 << WGM00 means value 2 , no shift ? -0b10 , so any reason not using the 1 << WGM01 syntax, which is clearly saying 1 in WGM01 bit ?

• Reply • Share ›

johnsondavies Mod disqus_IUhdg1pwdO • 7 months ago • edited

You're correct that 2<<WGM00 is equivalent to 1<<WGM01. I did it that way because it makes it clearer that it's setting Timer/Counter0 into Mode 2, CTC mode, as defined by all three bits, WGM02 to WGM00.

• Reply • Share ›

Vipin • 6 years ago

Hi,

I am using attiny85 along with HC-05(bluetooth(soft serial)) and oled(two wire protocol). Individually each of these works perfectly but once i combine both oled and bluetooth in single projects it fails. I am using pin 3, 4 (tx, rx) and pin 0, 3 (sda, scl) for communication. Any idea whats going wrong

• Reply • Share ›

Shane Burgess Vipin • 6 years ago

It sounds like you're trying to use pin 3 for both serial transmit and as a clock (scl) for the two-wire interface. This generally doesn't work. Can you try using different pins so they don't overlap? First try individually again, as not every pin will necessarily work for serial communication.

• Reply • Share ›

Vipin Shane Burgess • 5 years ago

No Sir, I am using pin0 and pin2 for I2C comm and pin3 pin4 for Serial Comm.

• Reply • Share ›

johnsondavies Mod Vipin • 5 years ago

My Simple ATtiny USI UART uses the ATtiny85's USI, which is hardwired to use Arduino pin 0 (PB0) for the data in pin, so you need to keep this. Perhaps you can reassign the other pins?

• Reply • Share ›

Shane Burgess • 6 years ago

Hi David, thanks for such great articles and code samples! This has certainly been a deep dive into learning about AVR timers and interrupts for me. I took the time to combine the ATtiny84 and ATtin85 code, and have also added a check for the stop bit (as suggested by Edgar). Additionally, I implemented a buffer that the interrupts write to, allowing the main program loop to poll and read from it in the same way that most Serial libraries work. This does all add some complexity to the program, so it's perhaps too much to cover in a blog post, but I imagine it might be useful to others as well.

Also, I had trouble getting the original ATtiny84 code working alongside other Arduino code, because the Arduino-Tiny core configures millis(), delay(), etc to use Timer0. Since the USI needs to use Timer0, I edited core_build_options.h to set TIMER_TO_USE_FOR_MILLIS == 1 for ATtiny84. With that change, it seems to work. :)

I'm not finished with the code yet, but it's in a functional state and any incomplete parts are

marked with TODO's Feel free to check it out!
https://github.com/acropup/...

• Reply • Share ›

johnsondavies Mod ➚ Shane Burgess • 6 years ago • edited

Thanks for that information - I'm sure it will be useful to other people.

• Reply • Share ›

Edgar Bonet • 6 years ago • edited

Hi!

Thanks for sharing all this, it has been VERY useful for getting me started with the USI in UART mode. However, while porting this to an ATtiny84A and debugging on the scope, I found a few issues I would like to report here:

1) TIM0_COMPA_vect was triggered right after PCINT0_vect.
Fix already suggested by hc: clear the interrupt flag right before setting the interrupt enable:

TIFR0 = 1 << OCF0A;

But this rises the next issue.

2) The code was missing the first bit, and reading the stop bit as a data bit.
Fix: in PCINT0_vect, set OCR0A to 0.5 * bit duration, instead of 1.5. This way TIM0_COMPA_vect runs at the middle of the start bit, and the USI starts shifting one bit later.

3) Suggestion to get rid of TIM0_COMPA_vect: In iInitialiseUSI(), set OCR0A to one bit period. In PCINT0_vect, switch on the USI and set TCNT0 to 256 minus half a bit period. This way the first COMPA event happens 1.5 bit periods after PCINT0_vect.

Regards,

Edgar.

• Reply • Share ›

johnsondavies Mod ➚ Edgar Bonet • 6 years ago

Thanks for your suggestions. I've corrected the code in this article as in your points (1) and (2), and I've created new ATtiny85 and ATtiny84 versions which eliminate the timer compare interrupt service routine as in your point (3):

http://www.technoblogy.com/...

• Reply • Share ›

johnsondavies Mod ➚ Edgar Bonet • 6 years ago • edited

Thanks for your comments and suggestions. In Disqus you have to replace angled brackets with &#60 and &#62 - I've edited your post to fix this. I'll need to think about what you've written and get back to you.

• Reply • Share ›

hc • 7 years ago • edited

I found that the line

```
TIFR |= 1<<OCF0A;
```

always trigger within a few microseconds.

• Reply • Share ›

johnsondavies Mod ➚ hc • 7 years ago

Which ATtiny85 core are you using? I used the arduino-tiny core, which uses Timer/Counter1 for delay() and leaves Timer/Counter0 free for this application. Other cores use Timer/Counter0 for delay(), which could explain why you're getting unexpected interrupts.

• Reply • Share ›

hc ➚ johnsondavies • 7 years ago

I'm using the Arduino-Tiny core as well. I'm not using the delay() function anywhere, but I am using Timer1's overflow interrupt. That shouldn't affect Timer0 though. Also, the COMPA interrupt didn't trigger without any data on the input pin, so it wasn't an extraneous interrupt condition.

When receiving, the 1.5 bit delay did happen intermittently, though. Perhaps it's because the COMPA interrupt disables itself? I have no idea.
The ATtiny just happened to need a manual bit flag I guess.

• Reply • Share ›

johnsondavies Mod ➚ hc • 7 years ago

Let me know if you work out what the problem is. If it's something that could affect other users I should amend the program.

• Reply • Share ›

Neil • 7 years ago

I would like to try this code with a 4800 baud device and a 8Mhz crystal. To use this at a slower baud, the duration of one bit takes 1666.667 clock cycles. One and a half bits takes 2500.0000005 cycles. Could you explain how I would change the prescaler and compare-match values for Timer/Counter0. Any help would be appreciated.

1 • Reply • Share ›

johnsondavies Mod ➚ Neil • 7 years ago • edited

You need to change the prescaler to /64 and then use a compare match of 39, since 39*64=2496 which is as close as you can get. For the delay between bits you need to use a compare match of 26, since 26*64=1664, close to 1666.667. The routines become:

anemometer powered by 12 v. The data produced by the anemometer operates at 4800,8,N,1. I used a 5 volt regulator + Max-232 to covert the RS-232 signal to UART/TTL level, then sent R2Out (pin 9 max 232) to Attiny85 PB0.

I modified ParseGPS from your Tiny GPS Speedometer to handle the parse the $IIVWR sentence from the anemometer and output the result to a 7 segment display.

Here is what worked for my device:

• Reply • Share ›

Neil ➚ Neil • 7 years ago

Pasting my code was messed up by disqus. In a nutshell, the prescaler remained at /8, for function ISR (TIMER0_COMPA_vect) I changed OCR0A = 207, for ISR (PCINT0_vect) I changed OCR0A = 310

• Reply • Share ›

johnsondavies **Mod** ➚ Neil • 7 years ago

Glad you got it working, but I'm a bit puzzled that your values worked because OCR0A is an 8-bit register, so setting it to 310 is effectively the same as setting it to 54.

• Reply • Share ›

Neil ➚ johnsondavies • 7 years ago