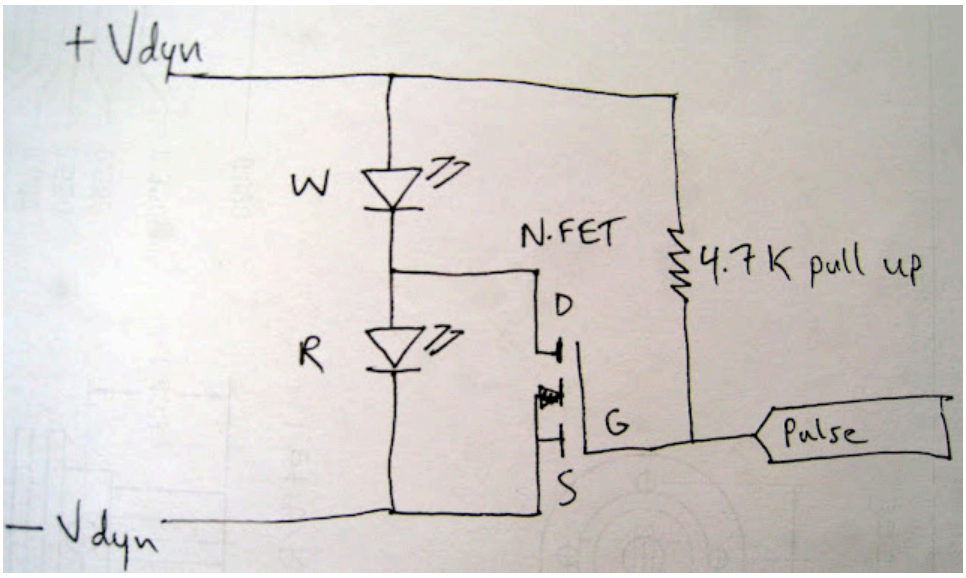
Friday, January 27, 2012

Programming an Attiny10 with AVRISP mkII and AVR Studio 5

Jump to **Attiny10 programming tutorial** if you want to skip my long winded preamble.

I've been tinkering with my **dynamo powered LED flasher circuit** recently. While the standlight works well, the blinking circuit requires that a 5W Zener diode shunt all 500-600 mA of dynamo current during the off cycle of the flash. At a frequency of about 2 Hz and a duty cycle of about 50%, I didn't think this would be a problem, but the damn Zener hits 100°C within a minute and showed no signs of slowing down, suggesting that I was taking it to within an inch of its life. I got some advice about using **a power transistor to shunt the current**, but using a 20W transistor still resulted in the device getting too hot too fast. I don't have room to put a big heat sink on it so I finally gave up on trying to shunt the current and getting rid of the off cycle energy as heat. I'm sure there is an elegant solution out there that allows the energy to be recaptured rather than wasted, but I'm already running the rear red LED close to its maximum current. Instead, I decided that I could do away with a flashing front light and just flash the rear light. This way, the dynamo is never disconnected from the load and the Zener does not need to shunt any current except in the case of a connection failure. A low side N-FET seems to do the trick, shorting out the red LED when the transistor is on:



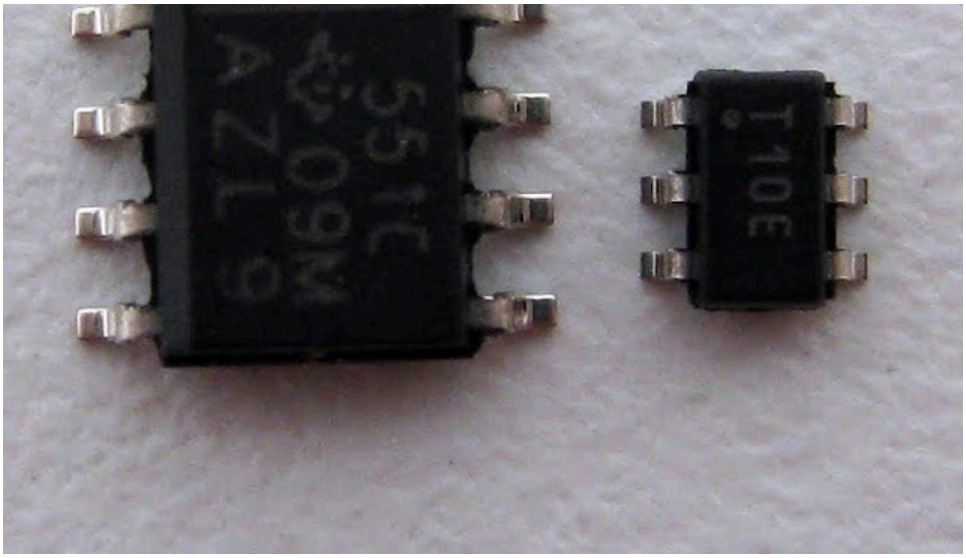
N-MOSFET shorts red LED when ON

Anyway, in the process I also decided that I wanted a flash that was below 50% duty cycle. Ideally around 30% or so to better mimic the strobe pattern of commercial LED flashers. You can **achieve this with a 555 by adding a diode**, but I thought I'd try using a microcontroller to generate the flasher's pulse instead. Why, you might ask, would I want to do something so complicated? Adding a microcontroller to a bike light seems kind of overkill. Well, it turns out I can do it in a smaller package with fewer parts for less money (and learn something along the way). Win!

In my previous design I was using a 555 astable oscillator to generate the pulse. The smallest package size is SOIC8, which is pretty bulky as far as surface mount devices go. The smallest microcontroller I could find was the **Atmel AVR Attiny10** that comes in a breathtakingly small SOT23-6 package.

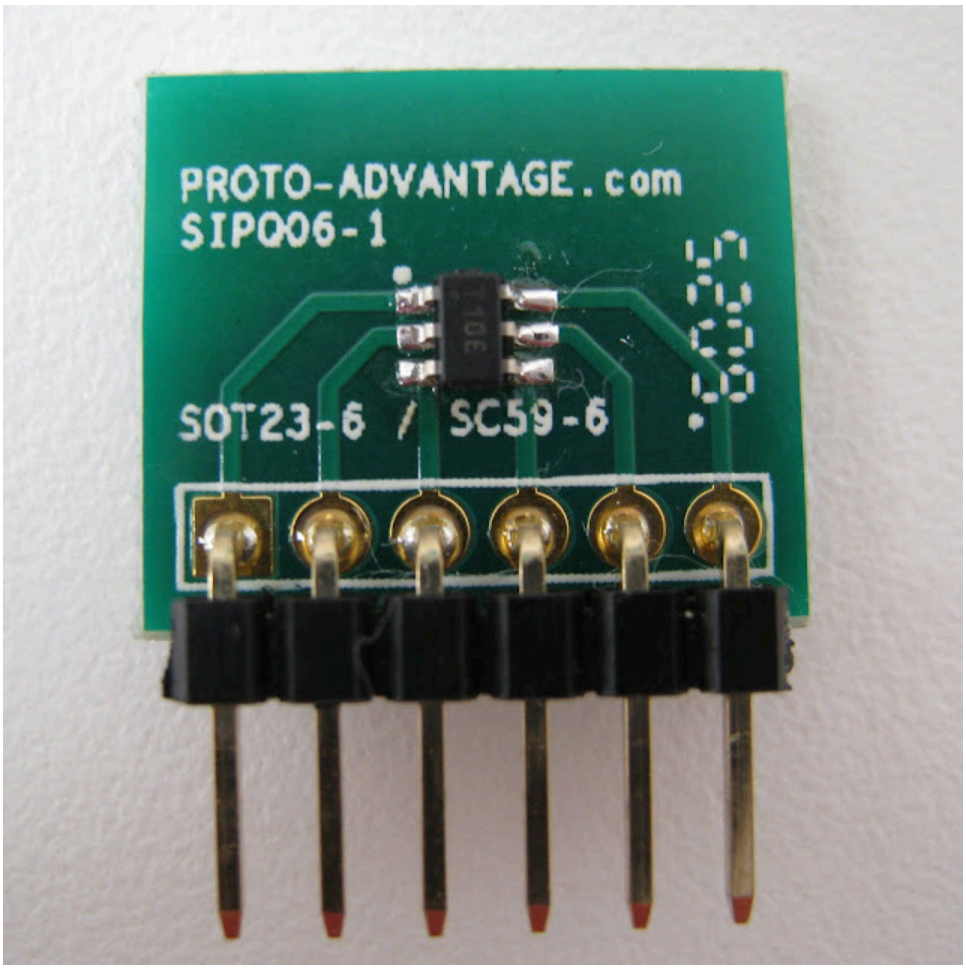
Blog Archive

October (1)
June (4)
May (1)
April (3)
February (5)
January (5)
December (9)
November (7)
October (2)
September (5)
July (1)



TI CMOS SOIC8 555 timer ($1.72) next to SOT23-6 Attiny10 ($0.89)

Surface mount devices are pretty annoying for prototyping, but the advantage of their tiny footprints becomes readily apparent when you need to cram a lot of parts into a small pace. Once I realized how useful SMDs are for creating physically small circuits, I accepted that the prototyping complications were worth it. Getting the Attiny10 onto a breadboard requires a little adapter from the nice people at **Proto-Advantage**. I soldered it using a soldering iron. Not as tricky as it seems.



Attiny10 on SOT23-6 to SIP board

Attiny10 programming tutorial

can compile C for it, leaving assembler (ASM) as the most straightforward option for programming the whopping 1024 bytes of Flash ROM. It took me a few days of fiddling around to get an Attiny10 to put out a pulse to blink my LEDs. I've put together a little tutorial about programming the Attiny10 to fill in the blanks I found in online documentation, both official and in forums and the blogosphere. Perhaps this will make things a little easier on the very rare occasion that another layperson might want to program an Attiny10 using the available Atmel tools.

[rant]
Engineers seem to, rather annoyingly, assume that everyone using their products or protocols is also an engineer and their technical documentation reflects this assumption. They seem to communicate by some Borg-like collective knowledge (otherwise known as an Engineering Degree) that is mostly impenetrable to the rest of us. This is often a barrier to the minority of wannabe engineers who want to occasionally parachute into the engineering world to get something specific done without actually becoming, you know, an *engineer.*
[/rant]

This is a tutorial aimed at programming the Attiny10 in assembler using the Atmel **AVRISP mkII** programmer and **AVR Studio 5**. By extension, it should also work for all members of the Attiny4/5/9/10 family. It really is just about connecting the Attiny10 to the AVRISP and being able to load hex code onto it because even this can be a frustrating challenge for the uninitiated. The **Attiny10 datasheet**, the AVR **Assembler User Guide** and **Instruction Set** are wonderfully indecipherable documents that you need to refer to for actual programming in assembler. There is also this **huge tutorial**.

Adding to the confusion is the fact that there are numerous accounts that AVR Studio 5 is either incompatible with the AVRISP mkII, the Attiny10 or both. As of January 2012, I can say that AVRISP mkII, AVR Studio 5 and Attiny10 all play nicely together on Windows 7.

So, let's start:

Step One: Get what you need

You need an **AVRISP mkII** programmer, **AVR Studio 5** and some **Attiny10s** to get programming. A breadboard and a SOT23-6 to **SIP/DIP** adapter will also come in handy.

There are a bunch of AVR programmers out there. I chose to use the AVR branded one, the AVRISP mkII, which is a little pricer than the **USBTinyISP** but requires no assembly. One gentleman was so averse to using the available Atmel tools that he made **his own programmer and his own IDE**.

AVR Studio 5 is a rather huge and cumbersome piece of software that I'm sure is great for managing complicated multi-thousand line projects written in C. **You may or may not be able to compile C for the Attiny10**, but you can definitely use AVR Studio to write assembler and use the very handy simulator to debug your code before flashing your Attiny.

I'm pretty sure you could also use AVR Studio 4, which is now mature. In fact, many people seem to have delayed upgrading to AVRS5 because it is considered buggy.

I'm sure you could also use **avrdude** instead of AVR Studio. It has been **documented** to work with the Attiny10. I have a strong aversion to command line interfaces, so avrdude was out (dead give away that I'm not an engineer...).

Step Two: Connect the programmer

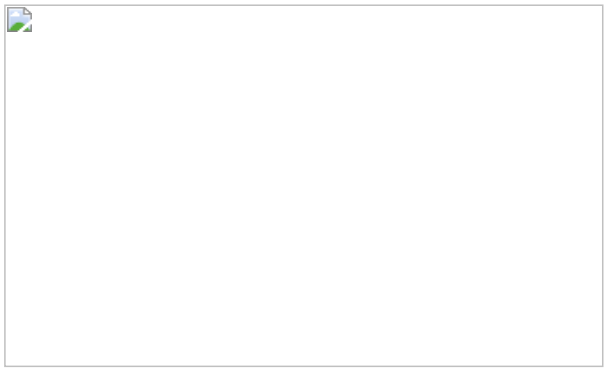
After downloading and installing AVR Studio 5 you should be able to connect the AVRISP mkII to your PC and see a green light glow inside, next to the USB connector. An exterior LED is red to indicate that there is no external power (weirdly, the AVRISP, despite having Vcc and GND connections, cannot be used to power the Attiny during programming).

Confusingly, the Attiny10 and its ilk are not programmed using the same 3 wire MISO/MOSI/SCK **ISP** protocol used by other AVR microcontrollers. Instead, they use the 2 wire Tiny Programming Interface, or **TPI**. What wasn't at all clear to me from the various documentation available was how you connect the AVRISP mkII to an Attiny10. The **AVRISP mkII User Guide** makes no mention of TPI. The mkII is documented as being compatible with the Attiny10, but I had to really hunt around to confirm the pin connections from the programmer to the Attiny10.

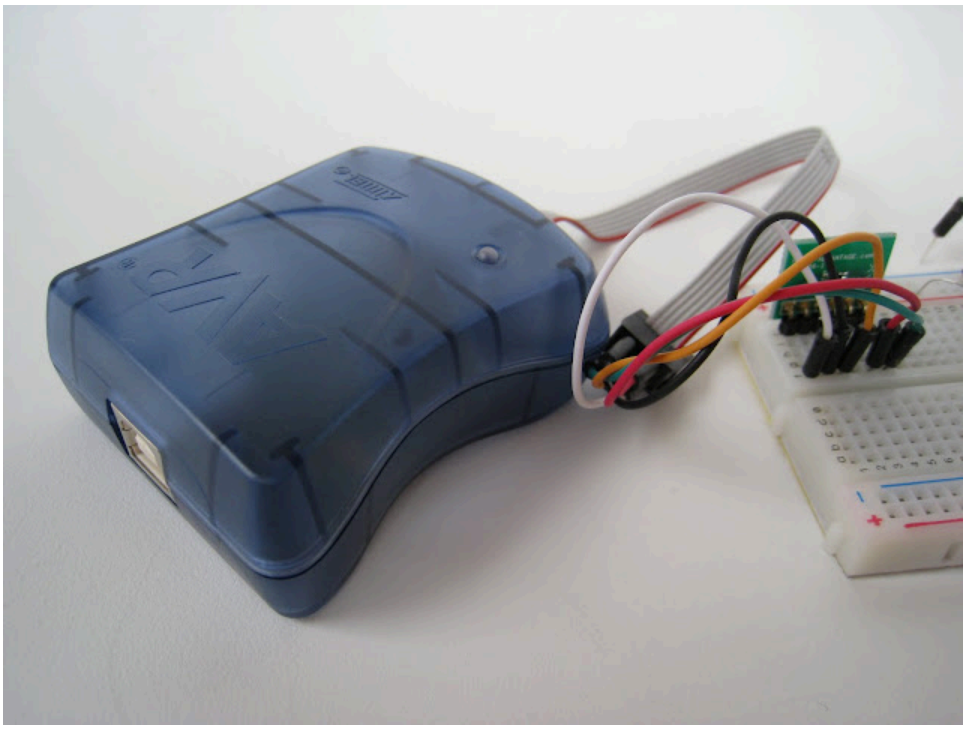
Pin connections from **here**

AVRISP mkII connector	Target pin name	ATTINY10 pin number

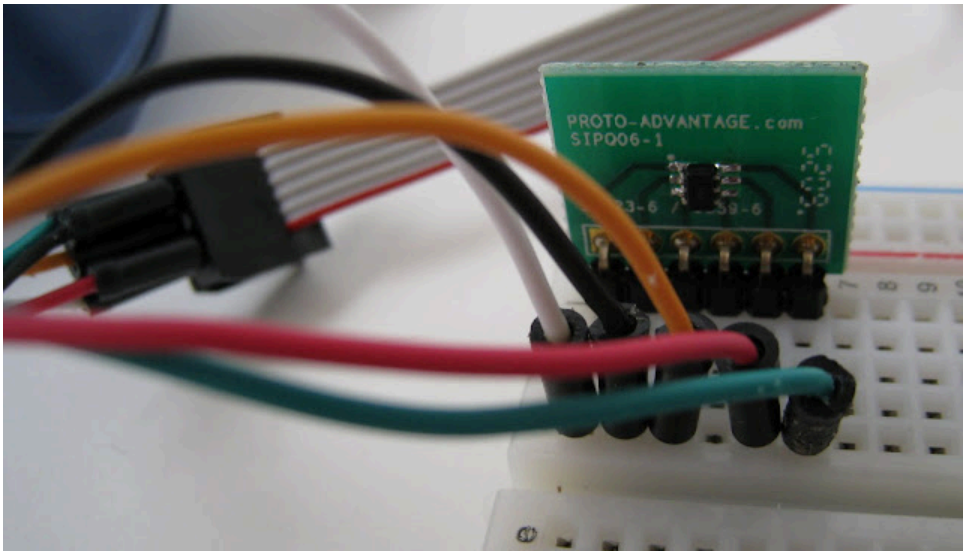
6 GND	**GND**	**2**



The programmer's Pin 1 is opposite to the red mark on the cable and is marked with a hard-to-see arrow/triangle. I hooked it up wrong the first time, so double check that you've identified pin 1 correctly!



AVRISP mkII connected to Attiny10



From left to right: (1)MOSI/TPIDATA, (2)GND, (3)TIPCLK, (5)VCC, (6)RESET

You need to connect an external 1.8-5.5V power source to VCC and GND before you can program the Attiny10. The programmer's exterior LED will go from red to green when it senses the external power. I use a 3.7V lithium ion battery for power.

Step Three: Program the Attiny10

Launch AVR Studio 5 and make a new assembler project: File:New Project... In the dialog select the AVR Assembler Project template. Then select Attiny10 from the Device Selection dialog that pops up. Write your code (I wanted to blink an LED, so I grabbed and modified some ASM I found **here** - if you follow that link note that the Attiny10 has PORTB instead of PORTD). This performs a nested decrement of registers. Two registers for the ON delay and three registers for the OFF delay (ie. longer OFF time than ON time). This is what my code looks like:

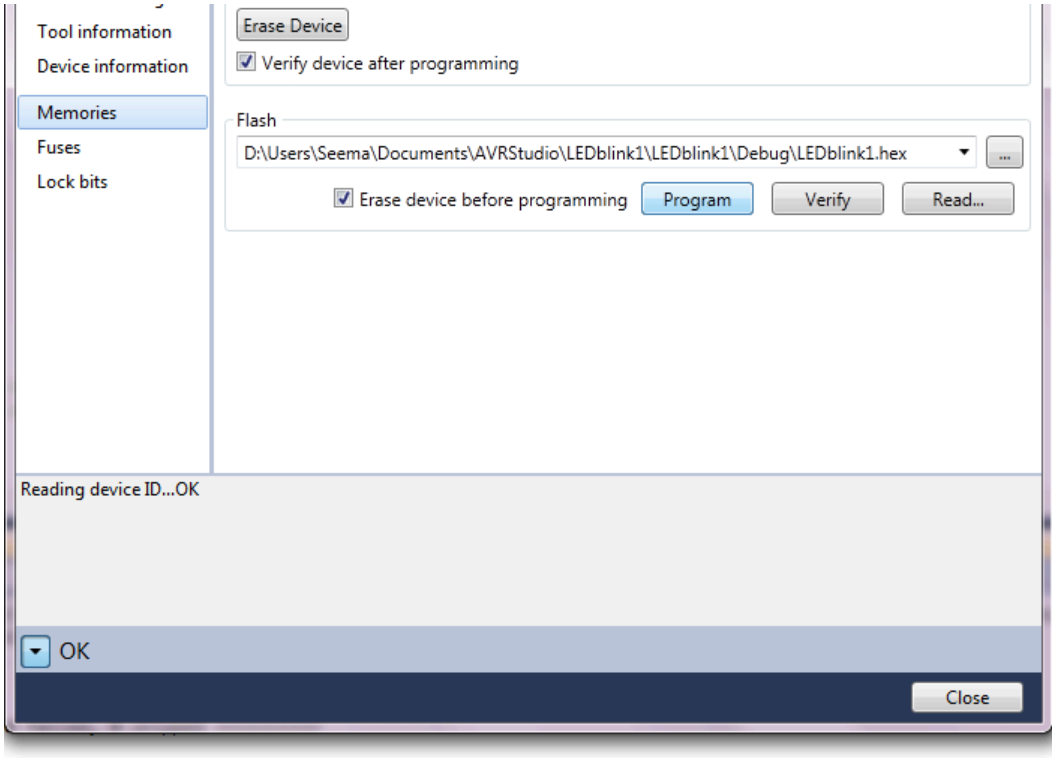
```
 rjmp RESET  ;go and set up PORTB as an output


;name registers (selected >r15 arbitrarily)
 .def  counter1  = r16
 .def  counter2  = r17
 .def  counter3  = r18


;set some variables
 ;time1 and time2 set the value for the final loop in each
delay
 .equ  time1   = 170 ;between 0 and 255
 .equ  time2   = 1
 .equ  led     = 2 ;LED at PB2

RESET: ;set PB2 as an output in the Data Direction
Register for PORTB
 sbi   DDRB, led  ;connect LED to PB2 (Attiny10 pin 4)

flash: ;main loop
```

```
 sbi   PORTB, led ;LED on
 ldi   counter3, time2 ;load counter3 delay
 rcall offDelay
 rjmp  flash  ;return to beginning of loop


onDelay:
 clr   counter1  ;clear counter1


loop1: ;nested loop that decrements counter 1 (255) x
counter2 (time1) times (ie. 255*time1)
 dec   counter1  ;decrement counter1
 brne  loop1     ;branch if not 0
 dec   counter2  ;decrement counter2
 brne  loop1     ;branch if not 0
 ret

offDelay: ;same as onDelay but with a third loop
 clr   counter1
 clr   counter2


loop2: ;decrement counter 1(255) x counter2(255) x
counter3(time2) (ie. 255*255*time2)
 dec   counter1
 brne  loop2
 dec   counter2
 brne  loop2
 dec   counter3
 brne  loop2
 ret
```

This is pretty rudimentary. It basically just kills time by pushing bits around registers. A more elegant solution would be to set up an interrupt triggered by the Attiny10's **16 bit timer** (more on that another time).

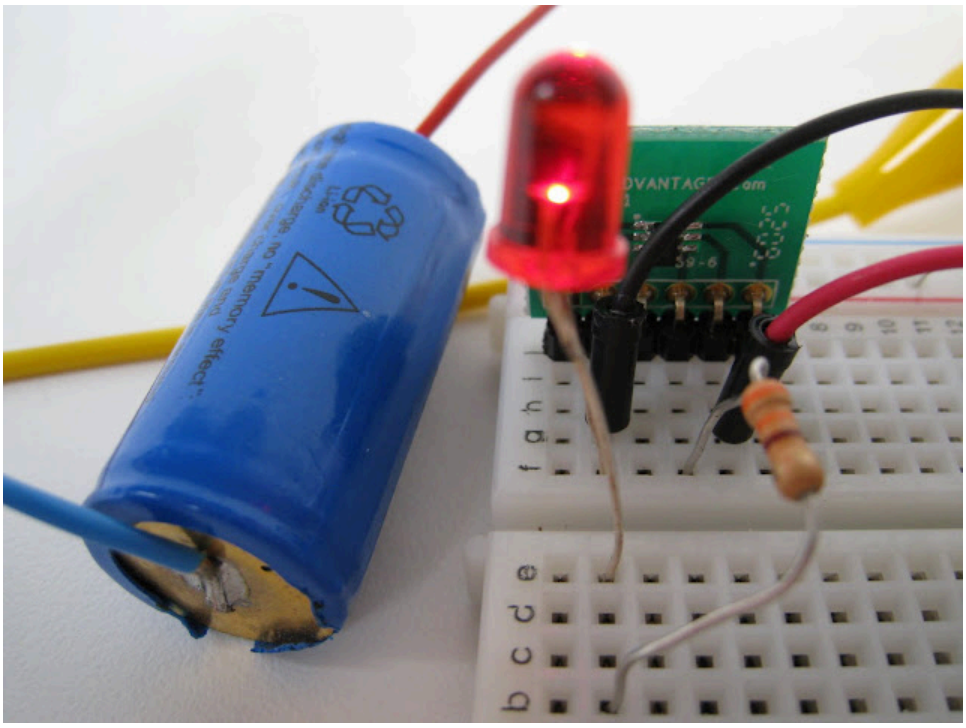
Build your code by selecting Build:Build Solution. This converts the ASM to machine code. If it builds successfully it will generate a .hex file in the project's folder. This is the file you need to load onto the Attiny10. You can simulate your code by going to Debug:Start Debugging and Break and and then hitting F10 to step through the instructions and looking at how the contents of registers and ports change, etc.

When satisfied, you program the chip using Tools:AVR Programming. Select AVRISP mkII in the Tool drop down list and Attiny10 in the Device list. TPI will come up in the Interface list by default. Click 'Apply'. If there is a connection problem, you will discover it here. This is the time to make sure that the Attiny has its own power and is connected to the AVRISP correctly.



Go to Memories and click '...' to select your project's .hex file if the contents of the Flash box aren't pointing to it already. Then click 'Program'. The programmer will load the code and then you can test it out.

I found that the programmer needed to be reset by unplugging it from USB and then reconnecting it before I could get each upload to run properly on the Attiny10. I chose PB2 (pin 4) to blink my LED because it isn't connected to any programmer lines, but the other lines should work even if connected to the programmer, so long as it has USB power. If the programmer is off then the Attiny10 didn't like being connected to it. Once satisfied with your program you can disconnect the programer lines and operate it stand alone:



Attiny10 blinking an LED

These are the two sites I found most useful for learning the ins and outs of Attiny10 programming:

http://irq5.wordpress.com/2010/07/15/programming-the-attiny10/
https://sites.google.com/site/wayneholder/attiny-4-5-9-10-assembly-ide-and-programmer

2 comments:

Unknown said...

Hello,
great project, great tutorial I'm buying a Attiny10 today to start working with it. I'm also thinking on getting a new LED light for my new Kronan. The heatsink is a problem because it is caged inside the frame, I like copper frame idea.
Here is my LED lamp http://bitsnbikes.blogspot.nl/search/label/Solarlight
Maybe we could share some ideas?

June 12, 2012 at 1:35 PM

Anonymous said...

Hello,

Very nice, well done.

As a child of the 50's, I loved my bikes too. BSA Star Rider in blue with Lucas (rim) dynamo and matching front and rear chrome lights.

Still bike now with Ridgeback Element Rapide.

Just getting interested in Atmel and energy monitoring late in life and in using ATtiny. Why use capacitors and resistors etc for small timing jobs when an MCU will do it and better. And, as you say, it keeps the brain active.

Phil_S (Sussex, UK)

November 14, 2013 at 4:26 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)