WeCare Components:

WP #001

Making an ATtiny85 AVR work with Arduino V1.0 on Ubuntu

You can use an ATtiny85 (or ATtiny45) with the Arduino software development environment on the Ubuntu operating system. This WeCare Components white paper shows you how.

Note: This procedure is not suitable for use on Microsoft Windows™ machines.

Caveats:

1. You use this procedure at your own risk. WeCare Components will not be held responsible for any consequences.

2. You MUST create a backup copy of any file that this procedure requires you to edit, change or remove. This will allow you to restore your system to its starting state – should that become necessary.

Prerequisites:
You must have a working Arduino development environment installed on your Ubuntu system.

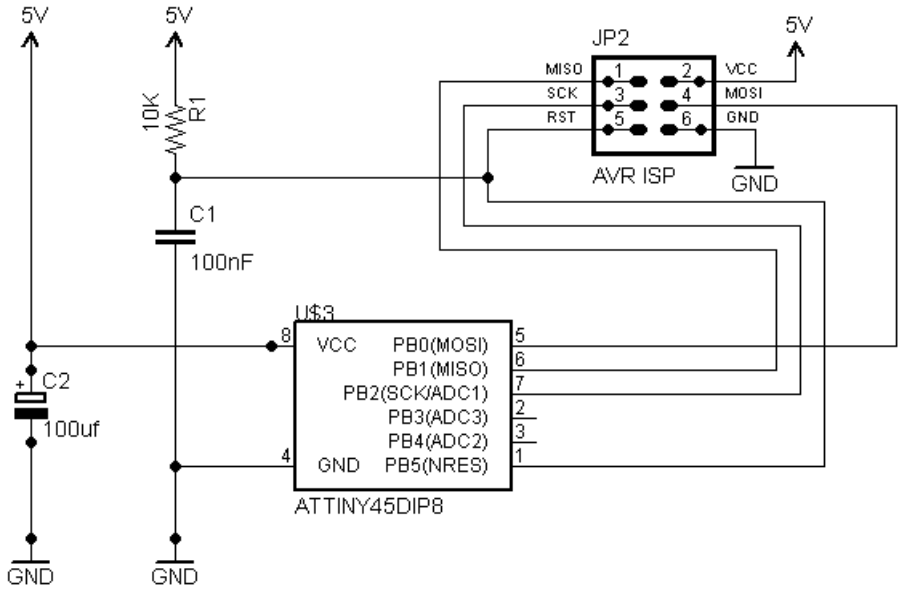
You will need a USB attached AVR ISP programmer installed and working. We recommend the Pololu USB AVR programmer – but there are many other AVR ISP compatible devices.

You'll need a Sparkfun ISP 6-way breakout board which allows the ISP programmer to interface to an AVR installed on a breadboard.

The Procedure:

Connections:

1) Install your ATtiny85 AVR chip on a breadboard (or a solder strip-board if that's your preference) and wire as shown in the diagram below (note the 45 is wired the same as an 85). You'll need the Sparkfun ISP 6-way breakout board and some header pins to make this happen. In many cases you can derive your +5Volt supply from the USB programmer (VCC pin on the breakout board). If you have any problem then you'll need a separate +5V supply for your AVR to drink from.



2) Connect the USB programmer. The programmer presents to Ubuntu as a pair of virtual serial ports.

3) Find out which tty port your USB programmer is auto-configured to. Use the following command.

```
dmesg | grep tty
```

Which should give you an output that looks something like the following:

```
[    0.000000] console [tty0] enabled
[    0.303806] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
[    0.392571] serial8250: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
[    0.482509] 00:08: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
[    0.524803] 00:09: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
```

```
[   19.988367] cdc_acm 1-2:1.0: ttyACM0: USB ACM device
[   19.993007] cdc_acm 1-2:1.2: ttyACM1: USB ACM device
```

If you do have other serial port devices (such as a barcode reader), they will be listed here too, however it should be easy to figure out which is your programmer. If not, try each one until you get useful responses to the first of the commands below.

Use the following command to ensure that the system can interrogate the AVR:

```
avrdude -p t85 -c avrisp2 -P /dev/ttyACM0
```

This command simply causes the programmer to retrieve the chip's signature byte and show it on screen. You should get a response something like:

```
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100% 0.00s
avrdude: Device signature = 0x1e930b
avrdude: safemode: Fuses OK
avrdude done: Thank you.
```

This proves that the lower levels of the stack (avrdude and the operating system) can access the AVR successfully – if you get the message shown above then it's all working at the low level. If not, check your connections and wiring and check you have the right port number before trying again.

Setting the AVR clock Speed

As you probably know, you can use an external crystal to clock AVR chips. However, like most AVRs, the ATtiny85 also has an internal 8MHz master clock – and like them, it ships with the clock set to divide by 8, meaning that, out of the box it will only run at 1MHz. So, while we're here, let's set the AVR to run at its full 8MHz (easily fast enough for most purposes we might use it for).

```
avrdude -p t85 -c avrisp2 -P /dev/ttyACM0 -U lfuse:w:0xE2:m
```

This sets the state of a bit in the lower fuse byte (thus `lfuse`) inside the AVR chip called the CKDIV8 fuse. After using this command to unset the fuse, the AVR will now run at 8MHz.

Even with the ATtiny85 you *can* connect an external crystal with a frequency of up to 20MHz to pins 2 & 3 (with capacitors) if you want to, and then program the clock fuses to use a crystal rather than the internal clock. However, that's beyond our scope here and now. In any case we only have five I/O pins, so sacrificing two of them to use with a crystal is probably not ideal!

You can see how the fuse bytes look and calculate what values to set for any given set of requirements by visiting the following excellent website:

`http://www.engbedded.com/fusecalc/`

So now, we have an AVR ATtiny85 which is running at 8Mhz and talking happily to our desktop Ubuntu system. Now, we need to configure the Arduino development environment to talk to it.

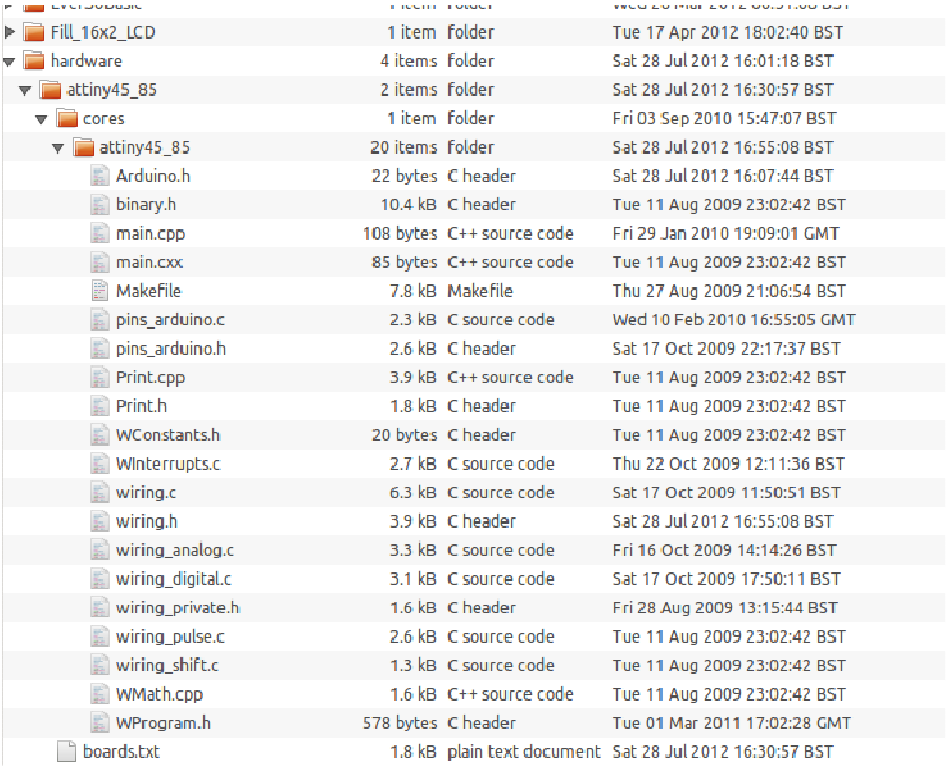
Making Arduino Recognise the ATTiny85:

First you'll need a standard Arduino software install (version 1.0 or later) Please see the Arduino site for details on how to do this.

Then, you need to download the core software required to bridge the ATtiny45 (or ATtiny85) AVR to Arduino. You get this from:

http://hlt.media.mit.edu/wp-content/uploads/2011/06/attiny45_85.zip

In your home folder you will have a sub-folder called "sketchbook" - which is where all your Arduino projects live. In that folder create a new sub folder called "hardware". Extract the contents of the zip file into this folder, so that you have a hierarchy that looks like this:



If you downloaded the same version of the zip file as me you will notice that you don't yet have the "Arduino.h" file shown in the listing above – you need to create it (use gedit

or your favourite text editor – note you must have the upper case A in the filename). Inside that file, just put the single line of text:

```
#include "WProgram.h"
```

Next, you need to edit the file “wiring.h” file and find the line that says.

```
#define round(x)     ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
```

And comment it out by making it look like this.

```
// #define round(x)     ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
```

At the bottom of the screen shot above, there is a file called boards.txt you now need to edit that file and add the following lines to the bottom of it.

```
attiny85avrisp.name=ATtiny85 (8MHz using Pololu USB Prog)
attiny85avrisp.upload.using=avrispv2
attiny85avrisp.upload.maximum_size=8192
attiny85avrisp.build.mcu=attiny85
attiny85avrisp.build.f_cpu=8000000L
attiny85avrisp.build.core=attiny45_85
```

Finally (if you haven’t already got it) you need to add your programmer definition to the following file by executing the following command from a terminal window:

```
sudo gedit /usr/share/arduino/hardware/arduino/programmers.txt
```

(you’ll need to know your sudo password to do this).

If they’re not already there, add the following lines to programmers.txt and then save and close the file.

```
avrispv2.name=AVR ISP v2
avrispv2.communication=serial
avrispv2.protocol=avrispv2
```

Now, you’re ready to crank up the Arduino development environment. When it comes up look at the “tools” menu and select boards. You should have a board called something like

“`ATtiny85 (8MHz using Pololu USB Prog)`“

Select it. On the tools menu, select “programmers” and then select “AVR ISP V2” Then, also from the tools menu, select “serial port” and make sure that the selected port is the same one that we used in the “avrdude” step that we started with.

Now type in an empty sketch

```
void setup()
```

```
{
}

void loop()
{
}
```

Compile it and upload it to the AVR. No errors? You're there! If you do get errors, go over the file modifications we did again and check for mistakes or missing capitalisations and things like that.

ATtiny85 Pin Map

When you have it working the physical pins of the chip map to Arduino like this:

1. Is the reset pin.
2. Is Arduino digital pin 3 or analog input pin 3
3. Is Arduino digital pin 4 (or analog input 2)
4. Is ground
5. Is Arduino digital pin 0 (can do hardware PWM)
6. Is Arduino digital pin 1 (can do hardware PWM)
7. Is Arduino digital pin 2 (or analog input 1)
8. Is VCC and takes between 2.7V to 5.5V

So, as you can see, you have the possibility of 5 inputs. Furthermore you can use the software serial library to implement a serial port. See the Arduino playground for details on the software serial library,