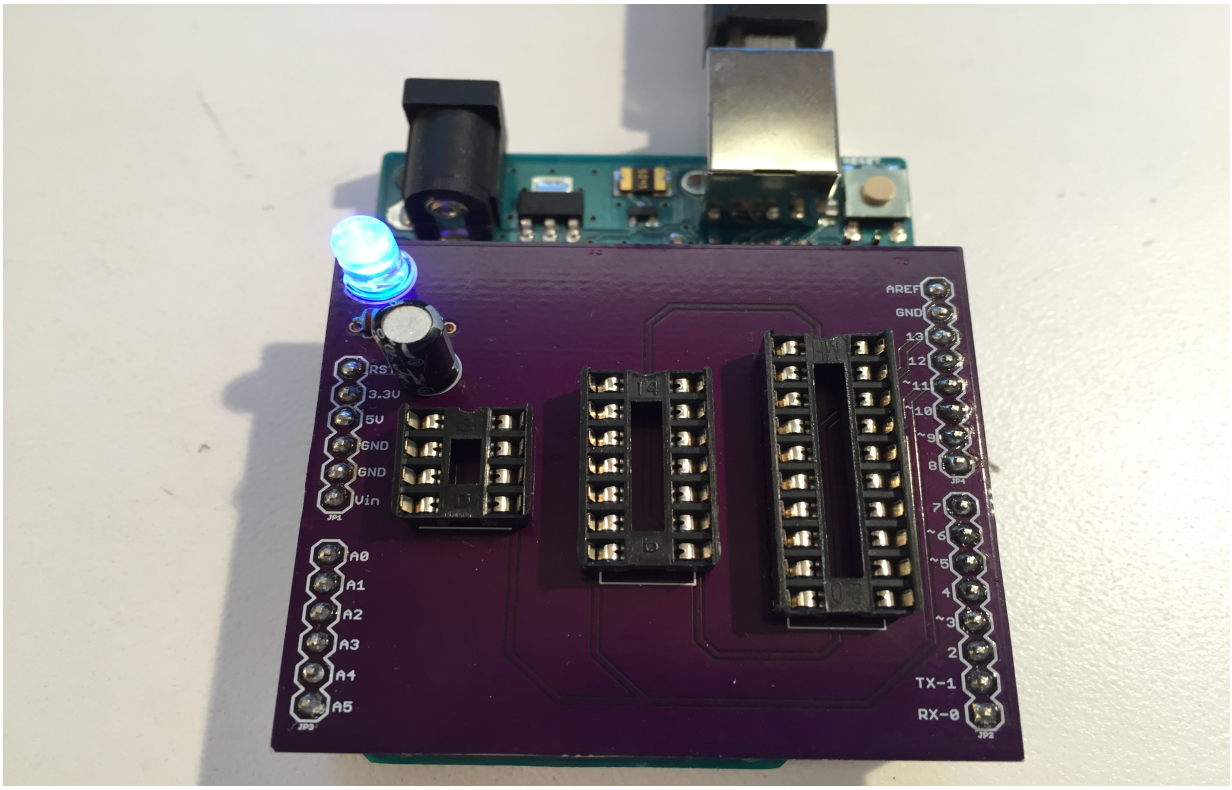
ATTiny Programmer



Project in a Box

ATTiny Programmer

Developed by Colin “MrSwirlyEyes” Keef and Stazia “WigginWannabe” Tronboll

Last Updated: 2017-01-06

Revision 2.0

PLEASE REPORT ANY FEEDBACK/ERRORS/COMMENTS/QUESTIONS/CONCERNS to us at: ProfSwirlyEyes@gmail.com
All inquiries are very much appreciated!

Phuong Truong | Professor Truong Nguyen

Introduction

The objective of this project is to introduce you to system-on-a-chip design. Many projects are prototyped on an Arduino or another microcontroller, and utilize only a subset of the total I/O (input/output) pins and features of the device. This leaves to the microcontroller *stuck* to the project. We now introduce ATTiny microchips. These are of the ATMEL family, similar to the ATMEGA328 that is the heart of many of the Arduino microcontroller variants. ATTiny chips come in various sizes with varying number of I/O pins in a **through-hole** or **SMD** (**Surface Mount Device**) package. Using these chips to replace microcontrollers in existing projects reduces hardware, cost, and frees up your *stuck* microcontroller to be used to prototype your next project. Yet, programming ATTiny chips requires reconstruction of the same circuit to program the chip on a breadboard repeatedly, which takes time and memorization. To solve this problem, this project creates an ATTiny Programmer that will be fitted as an Arduino UNO shield. This device will be able to program three different ATTiny chip sizes, namely the 8-pin, 14-pin, and 20-pin through-hole packages.

Learning Objective

- Understand the Arduino UNO and its various features
- Simple programming in Arduino C/C++
- Using Arduino UNO as ISP to program ATTiny chips
- Understand the benefits of the ATTiny chips and how to use them
- Using ATTiny in a simple program
- PCB design using an ECAD program
- Soldering

Required Materials for Entire Project

Item	Reference	Vendor Link	Quantity	Price	Total	Note
Arduino UNO		Sparkfun	1	24.95	24.95	
Breadboard		Sparkfun	1	4.95	4.95	
ATTiny85	**U1**	Sparkfun	1	2.84	2.84	Can alternatively get an ATTiny45
ATTiny84	**U2**	Sparkfun	1	2.95	2.95	Can alternatively get an ATTiny44
ATTiny4313	**U3**	Digikey	1	2.57	2.57	Can alternatively get an ATTiny2313
8-pin DIP Socket		Sparkfun	1	0.50	0.50	
14-pin DIP Socket		Sparkfun	1	0.50	0.50	
20-pin DIP Socket		Digikey	1	0.29	0.29	
10uF Capacitor	**C1**	Sparkfun	1	0.45	0.45	Electrolytic
LED	**LED1**	Sparkfun	1	0.95	0.95	Color of choice
1K-10K Resistor	**R1**	Sparkfun	1	0.95	0.95	Pack of 20
Male Header Pins		Sparkfun	1	1.50	1.50	Row of 40 pins
ATTiny PCB v3.0		Oshpark	1	20.10	20.10	Pack of 3; Price may vary; 2 extra PCB at end
			Grand Total		63.50	Does not include shipping

Required Tools for Entire Project

- Arduino UNO + USB A (male) to USB B (male) cable
- Computer + Arduino IDE + ATTiny boards
- Computer + PCB CAD program (i.e. EAGLE , KiCAD or Altium's CircuitMaker)
- Internet connection
- Hook-up wire (20-gauge) or male-to-male jumper wire
- Wire cutter
- Soldering Iron + solder
- File (recommended)
- Tape (recommended)
- Needle nose pliers (recommended)
- Digital Multimeter (recommended)
- MrSwirlyEyes' ATTiny Programmer PCB board from GitHub (EAGLE .sch and .brd files) (optional)

Challenge #1: Arduino UNO & Arduino IDE

In this challenge we cover the installation and setup of the Arduino software. In this project we will be using the Arduino IDE to do the work of programming our ATTiny microcontrollers. This challenge explains the Arduino installation process, and the how to install the ATTiny boards into the Arduino IDE. We then cover the setup we will be using to program the ATTiny boards. Lastly, we explain a bit about the hardware's I/O interface of the Arduino UNO which are not clearly labeled like the **Power**, **Analog**, **Digital**, and **SPI** (Serial Peripheral Interface) pins.

Learning Objective

- Understand the Arduino UNO and its various features

Required Materials

- Arduino UNO + USB A (male) to USB B (male) cable

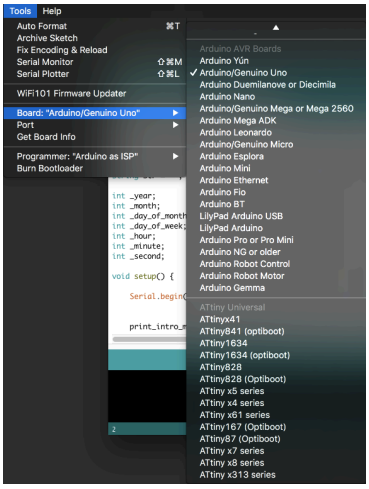
Required Tools

- Arduino UNO + USB A (male) to USB B (male) cable
- Computer + Arduino IDE + ATTiny boards
- Internet connection

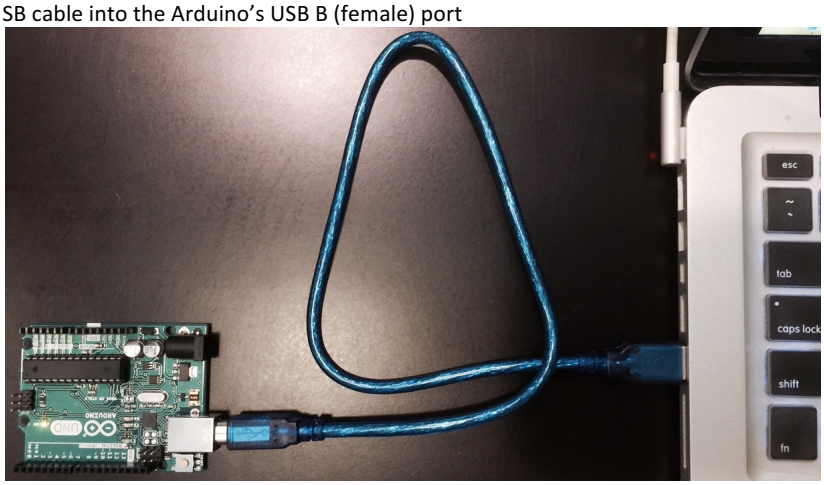
Check Off Requirements

You may either show a TA the following (in person), or email your project TA with the a picture or video (where appropriate), to show proof of completion.

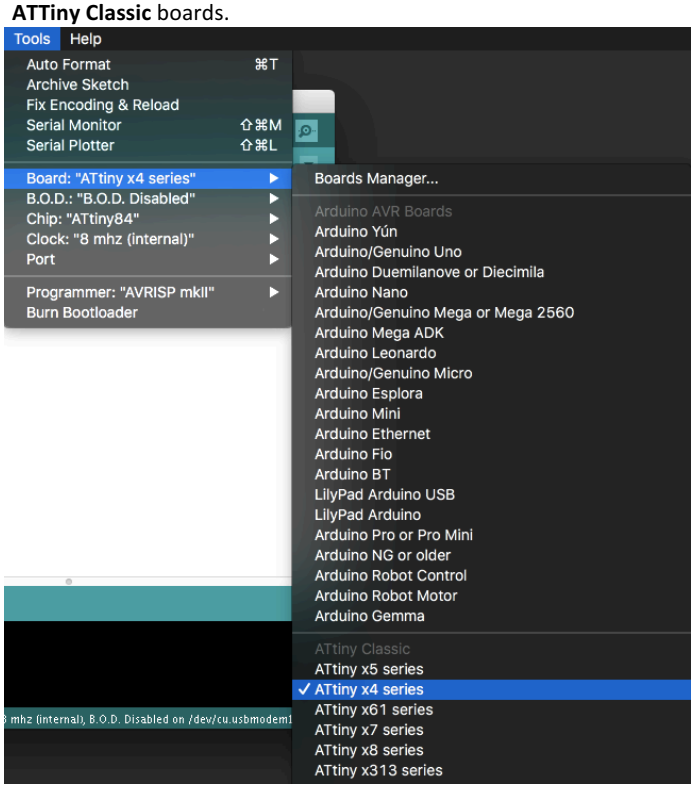
- [ ] Show a TA that Arduino is installed on at least 1 student's laptop
- [ ] Show a TA that ATTiny boards are installed on at least 1 student's laptop
    - i.e. show that the boards exist in the IDE
- [ ] Have a TA check off this box ***if-and-only-if*** the student has confirmed that they have all the necessary parts outlined above in the **Required Materials for Entire Project** section of the **Introduction** (above).
    - Note: This excludes LED's and resistors! You are expected to obtain these parts.



1. Install the Arduino IDE and the ATTiny boards into the environment
    a. Follow the instructions linked here: Installing Arduino IDE & ATTiny boards
2. Now that we have Arduino IDE installed and the board files integrated into the environment, let us explore what some of these new boards are, and what these fields mean. Plug your Arduino UNO into your PC's USB A (female) port and the USB B (male) of the USB cable into the Arduino's USB B (female) port



3. With the Arduino IDE launched, navigate to the **Tools** tab. There are many fields; we will explain some of these, namely:
    a. **Board**
        i. This is where we select the board we are going to program in the Arduino IDE. If you have any any experience with programming Arduinos, namely the Arduino UNO, you probably had it set to **Arduino/Genuino Uno**
        ii. With the installation of the **ATTinyCore** in the Installing **Arduino IDE & ATTiny boards** tutorial, we added several new boards, which are ATTiny variants
        iii. We selected the **ATTiny x4 series** for this explanation
            1. In this project we will use **ATTiny x5 series**, **ATTiny x4 Series**, and **ATTiny x313 series**
        iv. You may notice many fields appear: **B.O.D., Chip, Clock** fields will show up after selecting from the **ATTiny Classic** boards.

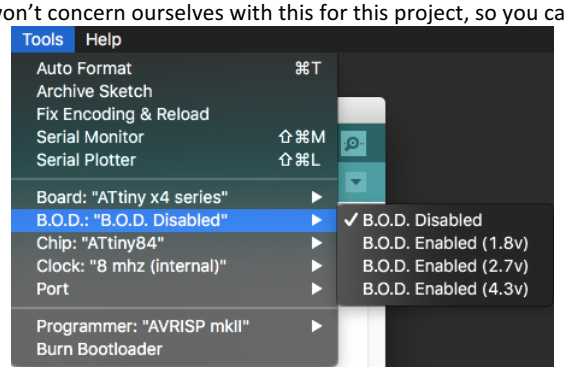


b. **B.O.D.**

i. **B.O.D. or Brown Out Detect** is a feature when the voltage drops across the ATTiny and the board is insufficiently powered, too low to run reliably at the given clock speed. Some consequences include erratic behavior and erasing or overwriting the RAM and EEPROM

ii. By enabling **B.O.D.** at one of the set voltages, we force the ATTiny to turn off until the voltage returns to a level that is stable. This is particular important if you are doing any data-logging, and do not want to corrupt your data. When the voltage returns, the ATTiny will reset and rerun its stored program from the start.

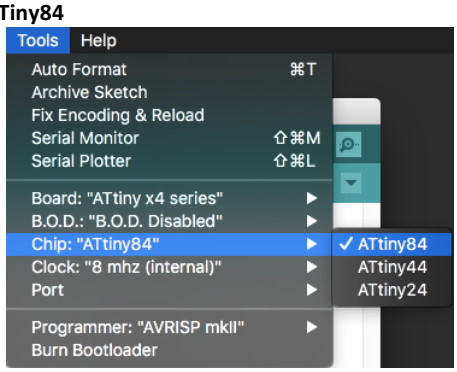
iii. We won't concern ourselves with this for this project, so you can leave it **B.O.D. Disabled**



c. **Chip**

i. Depending on which **Board** we select, the **Chip** will have varying options. Since we selected the **ATTiny x4 series**, we have its corresponding **Chip**s, shown in the image below

ii. Always pick the chip that corresponds to the one you wish to program. In this challenge we will use the **ATTiny84**



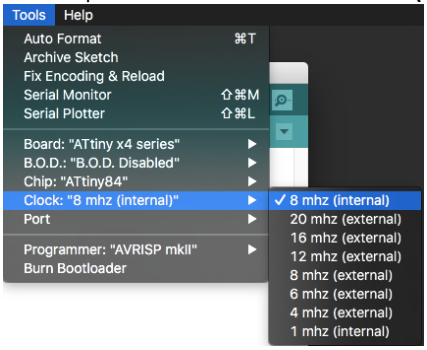
d. **Clock**

i. The **Clock** is simply how fast the ATTiny will run, in hertz. The faster the clock, the more power it will consume

ii. Notice there are options for **(internal)** and **(external)**. The ATTiny alone will support any of the **(internal)** clock speeds. If you wish to run the board faster, then you will need to attach a crystal oscillator to the board

Warning: If you do **(external)** on accident, you will not be able to use the board until you connect the corresponding **(external)** clock

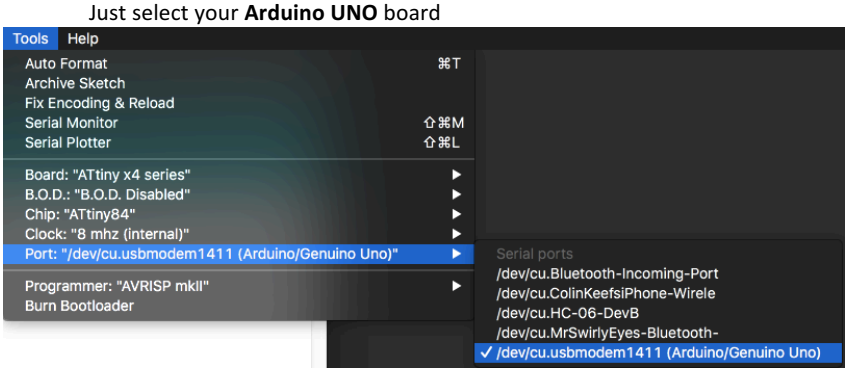
iii. In this project we will use only **(internal)** clock speeds that are supported. We will not worry about power consumption and run the board at **8 mhz (internal)**



e. **Port**

i. The **Port** is where the hardware is connected. It will be the **Arduino UNO** for this project.

ii. **Note:** The text you see in the image below is *variable* across Windows, OS X, and Linux platforms. Just select your **Arduino UNO** board

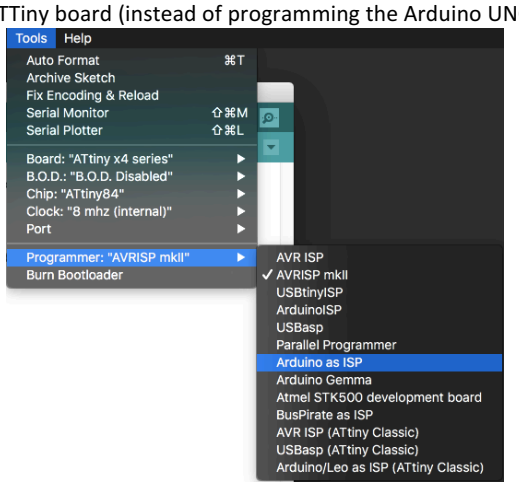


f. **Programmer**

i. The **Programmer** is important. This tells the Arduino IDE how to behave when uploading your program

ii. To program the Arduino UNO we want **Programmer** set to **AVRISP mkII**. This means that the program will be flashed onto the Arduino UNO directly

iii. To program our ATTiny chip variants, we will want to have **Programmer** set to **Arduino as ISP** where **ISP** stands for **In-System Programmer**. This means we will tell the Arduino to pass the code to the ATTiny board (instead of programming the Arduino UNO directly)

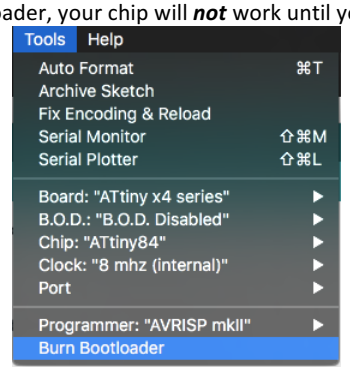


g. **Burn Bootloader**

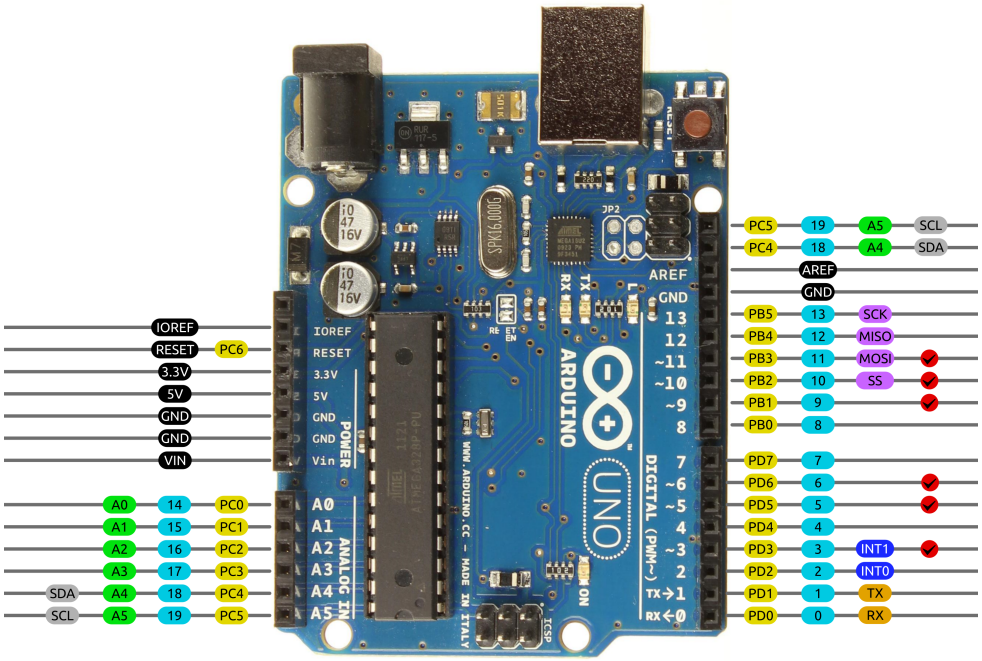
i. The **Burn Bootloader** selection is ***not*** a selectable option. Selecting this will ***immediately*** try to burn the bootloader to your device. This will take the settings configured in the last few steps and burn the fuse bits of the microcontroller

ii. You should always **Burn Bootloader** when first using the chip, you never know what fuse settings came on the board by default

Warning: Do ***not*** accidentally select an **(external)** clock selection and then burn the bootloader, your chip will ***not*** work until you attach the respective external clock!



4. We have gone over our new **ATTiny Classic** boards that we installed into the Arduino IDE
    a. Do not concern yourself with memorizing or completely understanding all these. You are most welcome to do some further reading from the **Resources & References**
5. Now let us turn to the actual hardware of the Arduino UNO. The diagram illustrates the pinout of the Arduino UNO:



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

6. Observe the following group of pins:
    a. **Power –** Rail of pins for power, the most commonly used are: **3.3V**, **5V**, and **GND**
    b. **Analog IN-** Analog inputs into the Arduino for analog data in 10-bit resolution with 2^{10} = 1024 values to work with, not used in this project
    c. **Digital –** Most commonly used pins for binary data (ON/OFF, HIGH/LOW)
        i. The "**~**" symbolizes a **Pulse Width Modulation** pin. These are very powerful, but will not be used in this project
7. **SPI** (Serial Peripheral Interface) is a synchronized data protocol used by microcontrollers (**MCU**s) to communicate with one another over short distances. In an SPI connection, there is always one Master (this will be the Arduino UNO) that will control (or program in our project) peripheral devices, namely the ATTiny
8. The specific **SPI** pins are:
    a. **MISO (Master In Slave Out) –** Slave line used to send data to the Master
    b. **MOSI (Master Out Slave In) –** Master line used to send data to the peripherals (ATTiny)
    c. **SCK (Serial Clock) –** Clock pulses for synchronized data transmitted by the Master
    d. **SS (Slave Select) –** Pin on each device that the Master uses to enable/disable devices

    Note: There are many other pins that are extremely useful and contain powerful features, however we will ***not*** address these here, you are welcome to do some further reading from the **Resources & References**
9. This concludes the introduction to the Arduino and the Arduino IDE, we will now turn to the first of our three ATTiny variants in this project – the ATTiny85
10. Don't forget to fulfill your **Checkoff Requirements**!

Challenge #2: Introduction to ATTiny85

In this challenge, we will go over background for ATTinys in general, most notably about how they will be programmed by the Arduino UNO. In this section, we will mostly concern ourselves with the ATTiny85. Lastly, we will program a very simple program into the ATTiny85 and see the powerful concept of programming ATTiny chips in action.

Learning Objective

- Simple programming in Arduino C/C++
- Using Arduino UNO as ISP to program ATTiny chips
- Understand the benefits of the ATTiny chips and how to use them
- Using ATTiny in a simple program

Required Materials

- Arduino UNO + USB
- Breadboard
- ATTiny85 [**U1**]
- Resistor [**R1**]
- Capacitor [**C1**]
- LED [**LED1**]

Required Tools

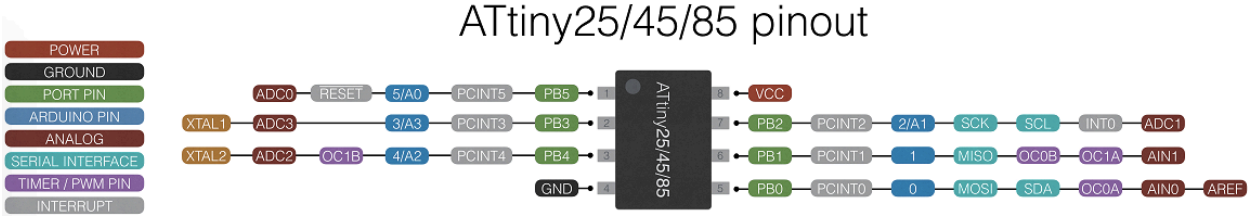
- Arduino UNO + USB A (male) to USB B (male) cable
- Computer + Arduino IDE **installed** + ATTiny boards **installed** (from **Challenge #1**)
- Hook-up wire (20-gauge) or male-to-male jumper wire
- Wire cutter (if using hook-up wire) (optional)

Check Off Requirements

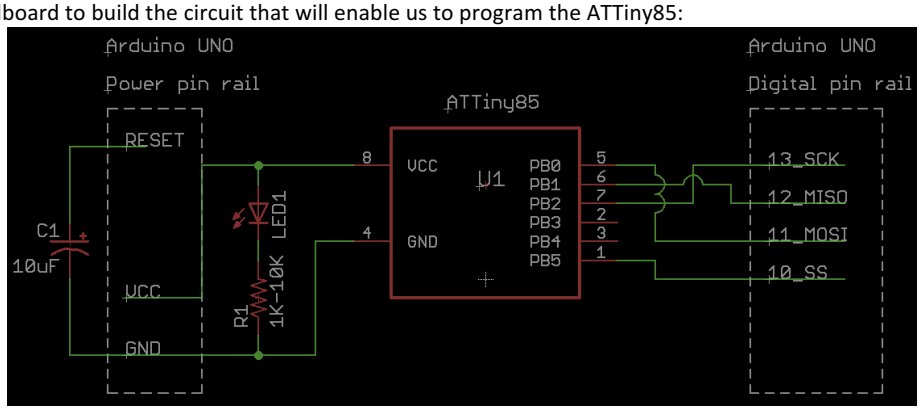
You may either show a TA the following (in person), or email your project TA with the a picture or video (where appropriate), to show proof of completion.

- [ ] Show a TA your completed CKT on a breadboard of an ATTiny85 blinking an LED

1. With the Arduino IDE all set up, and some familiarity established regarding how we expect to use the Arduino IDE with the ATTiny boards, in the next three challenges, we will get familiar with each of the ATTiny chip variants we will use in this project
2. Below is an image of the ATTiny25/45/85 pinout. In this project we will use the **ATTiny85**, however the pinouts for the other variants are all identical. (They only differ in the amount of memory, the larger the number, the more memory)

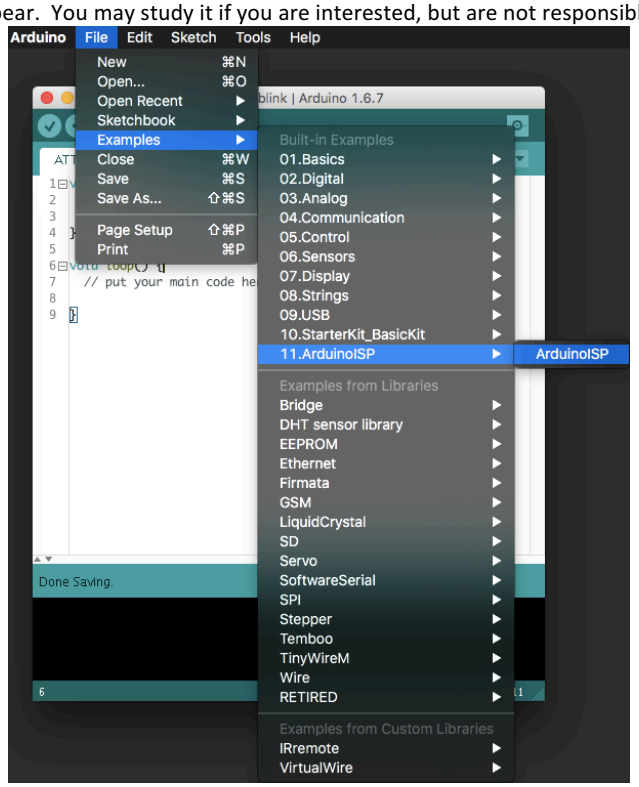


3. Pay close attention to the **Arduino Pin** (blue labels) on the ATTiny25/45/85 pinout. These will be the pins that we will reference when we program the chip in the Arduino IDE
4. The power pins are as follows:
    a. **VCC** should be between 2V to 5V. Its maximum operating voltage is 6V. Typically keep the ATTiny powered between 3V to 5V. This is common to battery cells, and output of the Arduino UNO
    b. **GND** to complete the circuit (CKT) loop
5. The **Serial Interface** pins are what we will use to program the ATTiny chip's via **Serial Peripheral Interface (SPI)**, 3 of these pins are similar from **Challenge #1**, but now we have replaced **SS** with the **RESET** pin:
    a. **MISO (Master In Slave Out) –** Slave line used to send data to the Master
    b. **MOSI (Master Out Slave In) –** Master line used to send data to the peripherals
    c. **SCK (Serial Clock) –** Clock pulses for synchronized data transmitted by the Master
    d. **RESET –** Used by the Master (in our case, the Arduino UNO) to reset the ATTiny
6. We have finally covered all the background information needed to work with an ATTiny and obtained the fundamental knowledge we need about the Arduino UNO from **Challenge #1**, so let us finally build something interesting and start to dig into the ATTiny
7. Use a breadboard to build the circuit that will enable us to program the ATTiny85:

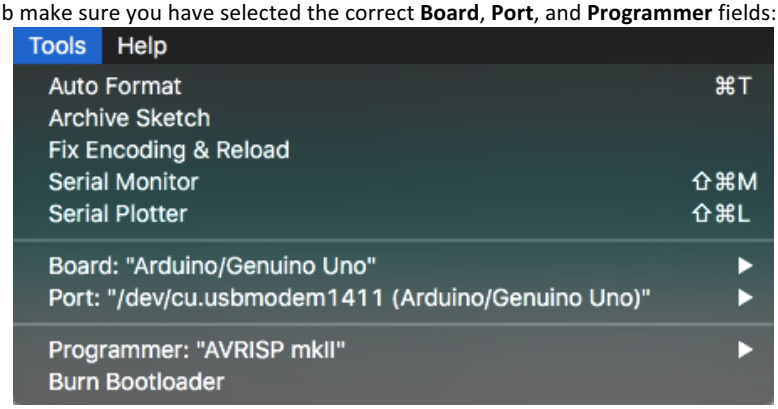


Note: ATTiny schematic pin labels in the image above is very different from the physical board layout

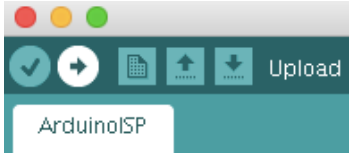
8. We have the circuit that can program the ATTiny85
   Note: Pay attention, this CKT is nearly identical to the CKT to program the other ATTiny variants
9. Launch the Arduino IDE and navigate to open the **ArduinoISP** sketch
   a. **File > Examples > 11.ArduinoISP > ArduinoISP**
   b. A sketch will appear. You may study it if you are interested, but are not responsible for being familiar with it



10. Under the **Tools** tab make sure you have selected the correct **Board**, **Port**, and **Programmer** fields:



11. Program the Arduino UNO with the sketch (**CTRL/CMD + U**) or click on the highlighted arrow

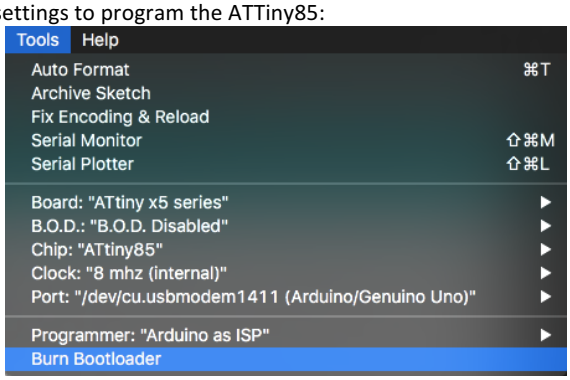


    Note: If you get an error uploading, try removing the 10uF capacitor and trying again. If it is successful this time around, place the capacitor back. This is because the 10uF capacitor is used to avoid an automatic reset of the Arduino when we program an ATTiny
12. If you were successful, it will say **Done Uploading** and maybe "**avrdude done. Thank you**" in the console output
13. What we did was program the Arduino UNO to be an **In-System Programmer,** thus we can now use the Arduino to effectively program the ATTiny (the Arduino is the "middle man" in this arrangement)
14. Close the **ArduinoISP** sketch

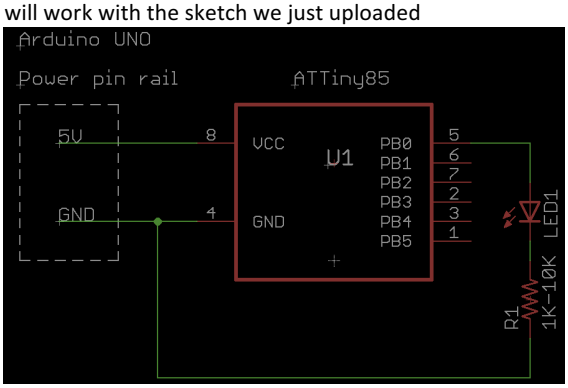
15. Create a new sketch in the Arduino IDE
    a. Save the sketch and name it anything you want, we called it: **ATTiny85_blink**
16. We will now write a program in the Arduino IDE in the **ATTiny85_blink** sketch, which we can directly upload to the ATTiny85:

```
// led1 is connected to pin 5 of ATTiny85
// Note: Pin 5 of ATTiny, but Arduino pin 0 on ATTiny25/45/85 pinout
#define led1 0

void setup() {
  // put your setup code here, to run once:
  // Configure led1 to be OUTPUT (it will emit/output light)
  pinMode(led1, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(led1, HIGH); // Write the led1 pin HIGH (produce 5V at pin 5)
  delay(500);           // Pause the system for 500ms
  digitalWrite(led1, LOW);  // Write the led1 pin LOW (produce 0V at pin 5)
  delay(500);           // Pause the system for 500ms
}
```

    Note: Pin 5 of ATTiny85 chip, but Arduino (programming) pin 0 according to ATTiny25/45/85 pinout (few pages above)
17. Before we program the ATTiny85, recall from **Challenge #**1 having to configure the **Board**, **Chip**, **Clock**, etc. We need to configure the Arduino board settings to program the ATTiny85:



18. Now click **Burn Bootloader**
    a. It may take some time to **Burn Bootloader**
19. Now upload the sketch! (**CTRL/CMD + U**)
20. If it was successful, besides output in the Arduino IDE console, nothing exciting should have happened to the ATTiny85, because we don't have the right CKT for program we uploaded
21. Let us make a simple CKT that will work with the sketch we just uploaded



22. After we construct this CKT, we should see the LED blinking ON and OFF
23. We have successfully programmed our first ATTiny, namely the ATTiny85! Although this was an extremely simple program, please feel free to add more LED's, or other components and make a more interesting program.
24. Don't forget to fulfill your **Checkoff Requirements**!

Challenge #3: Introduction to ATTiny84

This challenge is the same idea as **Challenge #2**, except now we are using the ATTiny84 board. ATTiny84 is nearly the same as an ATTiny85, except that it has more pins and thus more I/O to work with. With more I/O pins, you may perform more functions. It is essentially an upgrade in every way, except that it will take up more area on your board.

Learning Objective

- Simple programming in Arduino C/C++
- Using Arduino UNO as ISP to program ATTiny chips
- Understand the benefits of the ATTiny chips and how to use them
- Using ATTiny in a simple program

Required Materials

- Arduino UNO + USB
- Breadboard
- ATTiny84 [**U2**]
- Resistor [**R1**]
- 10uF Capacitor [**C1**]
- LED [**LED1**]

Required Tools

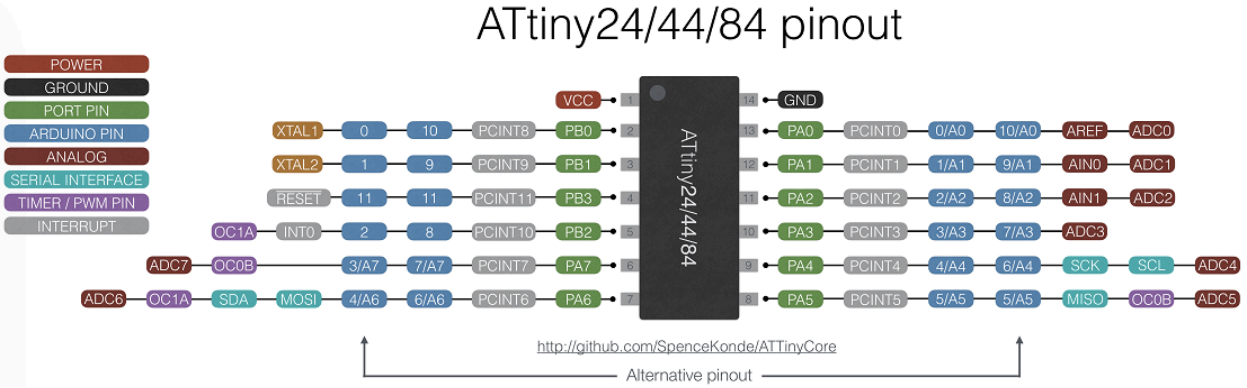
- Arduino UNO + USB A (male) to USB B (male) cable
- Computer + Arduino IDE **installed** + ATTiny boards **installed** (from **Challenge #1**)
- Hook-up wire (20-gauge) or male-to-male jumper wire
- Wire cutter (if using hook-up wire) (optional)

Check Off Requirements

You may either show a TA the following (in person), or email your project TA with the a picture or video (where appropriate), to show proof of completion.

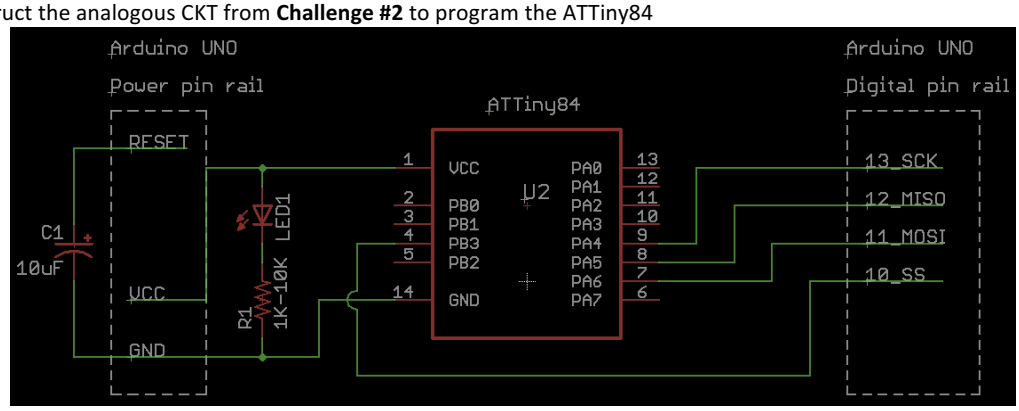
- [ ] Show a TA your completed CKT on a breadboard of an ATTiny84 blinking an LED

1. Similar to **Challenge #2**, we will program the ATTiny84 the exact same way and the same program, except using a different pinout
2. Below is an image of the ATTiny24/44/84 pinout. In this project we will use the **ATTiny84**, however the pinout for the other variants are all similar



Note: We have never used the "alternative pinout" that the image depicts, we use the pinout of the *inside* column

3. Clearly the ATTiny84 variant has more pins, this includes more **digital**, **analog**, and **PWM** pins
    a. Take note of where the **Serial Peripheral Interface** pins are
4. Construct the analogous CKT from **Challenge #2** to program the ATTiny84



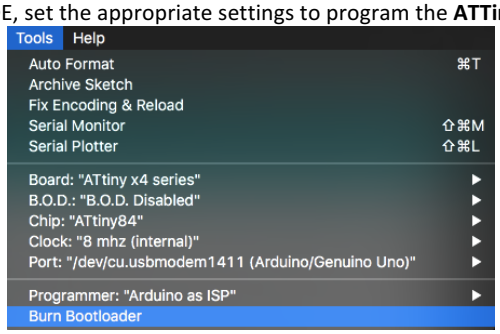
5. If your Arduino UNO still has the **ArduinoISP** sketch from **Challenge #2**, then you do ***not*** have to reprogram it with that same program
    a. If you do ***not*** have the **ArduinoISP** sketch still on your Arduino, go back to **Challenge #2**, and reprogram the Arduino UNO with it
6. With the Arduino UNO as an **ISP**
    a. Similar to **Challenge #**2, create a new sketch in the Arduino IDE
        i. Save the sketch and name it anything you want, we called it: **ATTiny84_blink**

7. We will now write a program in the Arduino IDE in the **ATTiny84_blink** sketch, which we can directly upload to the ATTiny84:

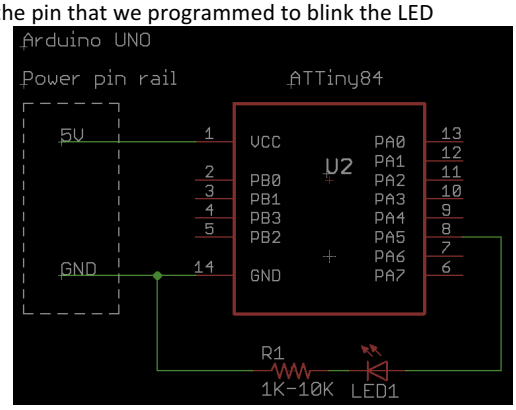
```
// led1 is connected to pin 8 of ATTiny84
// Note: Pin 8 of ATTiny, but Arduino pin 5 on ATTiny24/44/84 pinout
#define led1 5

void setup() {
  // put your setup code here, to run once:
  // Configure led1 to be OUTPUT (it will emit/output light)
        pinMode(led1, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(led1, HIGH); // Write the led1 pin HIGH (produce 5V at pin 8)
  delay(500);           // Pause the system for 500ms
  digitalWrite(led1, LOW);  // Write the led1 pin LOW (produce 0V at pin 8)
  delay(500);           // Pause the system for 500ms
}
```

Note: Pin 8 of ATTiny85 chip, but Arduino (programming) pin 5 according to ATTiny24/44/84 pinout (few pages above)

8. In the **Tools** tab of the Arduino IDE, set the appropriate settings to program the **ATTiny84** chip



9. **Burn Bootloader**
10. Create the appropriate CKT for the pin that we programmed to blink the LED



11. After we construct this CKT, we should see the LED blinking ON and OFF
12. We have successfully programmed the ATTiny84! Although this was an extremely simple program, please feel free to add more LED's, or other components and make a more interesting program.
13. Don't forget to fulfill your **Checkoff Requirements**!

Challenge #4: Introduction to ATTiny4313

This challenge is the same idea as **Challenge #2** and **Challenge #3**, except now we are using the ATTiny4313 board. Again, the ATTiny2313 is an upgrade from the ATTiny chip we used in the last two challenges. It simply has more I/O pins for more functions, again at the cost of even more board space.

Learning Objective

- Simple programming in Arduino C/C++
- Using Arduino UNO as ISP to program ATTiny chips
- Understand the benefits of the ATTiny chips and how to use them
- Using ATTiny in a simple program

Required Materials

- Arduino UNO + USB
- Breadboard
- ATTiny4313 [**U3**]
- Resistor [**R1**]
- Capacitor [**C1**]
- LED [**LED1**]

Required Tools

- Arduino UNO + USB A (male) to USB B (male) cable
- Computer + Arduino IDE **installed** + ATTiny boards **installed** (from **Challenge #1**)
- Hook-up wire (20-gauge) or male-to-male jumper wire
- Wire cutter (if using hook-up wire) (optional)

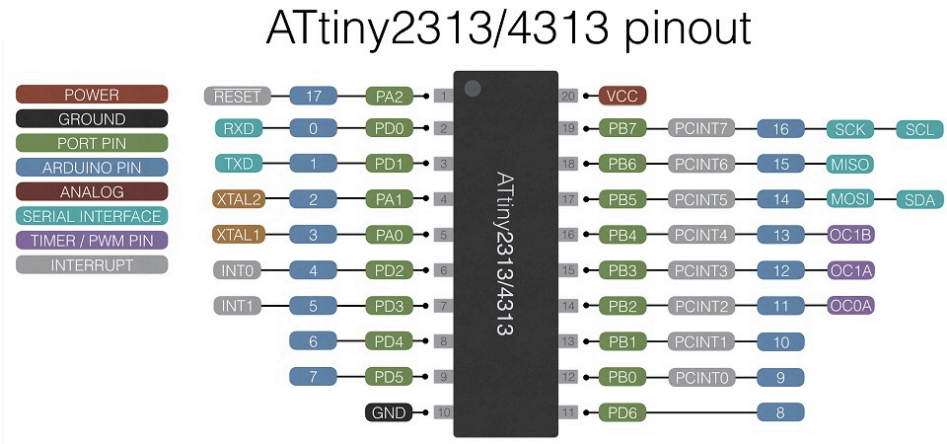
Check Off Requirements

You may either show a TA the following (in person), or email your project TA with the a picture or video (where appropriate), to show proof of completion.

- [ ] Show a TA your completed CKT on a breadboard of an ATTiny4313 blinking an LED

1. We could be extremely redundant and walk you through **Challenge #4** like **Challenge #2** and **Challenge #3**, however we hope by now you understand the idea of where we are going

   Note: We promise this is the last time you will build this CKT on a breadboard, there is a reason to this madness!
2. So for this challenge, we will provide nothing except the image of the ATTiny2313/4313 pinout. In this project we will use the **ATTiny4313**, however the pinout for the other variant is similar



3. Follow the same idea as **Challenge #2** and **Challenge #3** and program the ATTiny4313 with a similar blink sketch
4. The result is having programmed the **ATTiny4313**
5. Thus we have now programmed all three of our ATTiny variants, we have all but one piece before we can develop our goal, the **ATTiny Programmer**
6. Don't forget to fulfill your **Checkoff Requirements**!

Challenge #5: Introduction to PCB Design

In this challenge, we temporarily step away from the Arduino UNO and its IDE, and put down the ATTiny chips and pick up **ECAD (Electronic Computer-Aided Design)** software. Building and designing CKTs by hand is great for prototyping, but (we hope) making the CKT to program the ATTiny chip variants in **Challenge #2**, **#3**, and **#4** made you feel that there is a more effective way to program these chips without continuously reconstructing the (very similar) CKT. It would be even more awesome if we could make something that could program all three of them on ***one board*** – which is the goal of this project, namely the **ATTiny Programmer**! But before we can get to this stage, we need to learn some ECAD. We only provide support for **EAGLE**, and in this project we will teach and demonstrate **EAGLE** only (this is because there is support for **Windows**, **OS X**, and **Linux** platforms). However, you are more than welcome to use alternatives such as KiCAD, Altium's CircuitMaker, or anything else out there.

Learning Objective

- PCB design using an ECAD program

Required Tools

- Computer + PCB CAD program (i.e. EAGLE , KiCAD or Altium's CircuitMaker)
- Internet connection

Check Off Requirements

You may either show a TA the following (in person), or email your project TA with the a picture or video (where appropriate), to show proof of completion.

- [ ] Show a TA that EAGLE or your preferred ECAD tool is installed on at least 1 student's laptop

1. It is pretty difficult to teach one how to use EAGLE or any ECAD program
2. Learning comes from experience, however there are many things we could show you that are nice tricks, and powerful features that are not obvious
3. With that, we will direct you to the designed by students at UCSD from engineering organizations
    a. Instructions for installation are here too: **IEEE/HKN/TBP PCB Design Workshop** **Note:** This is not a completely comprehensive workshop, there is always more one can learn
4. Another great resource is the Sparkfun EAGLE Tutorial
    a. Instructions for installation are here too
    b. Most notably these tutorials from the **Sparkfun EAGLE Tutorial**
        i. Using EAGLE: Schematic
        ii. Using EAGLE: Board Layout
        iii. Making Custom Footprints in EAGLE

    Note: We will not need to make any custom foot prints for this project

    Note: This is not a completely comprehensive workshop, there is always more one can learn
5. After familiarizing with EAGLE, we also want you to install a few libraries into EAGLE that we will use for the **ATTiny Programmer** (and perhaps future projects)
    a. Download the contents of this folder: EAGLE Libraries
    b. Go to the EAGLE folder on your PC
        i. **For OS X**: It is in **Applications > EAGLE-X.X.X > lbr** depending on the version of EAGLE you have
        ii. **For Win**: Depends on where you installed, but the file path is very similar
    c. Unzip and drag-and-drop the contents into the **lbr** folder
6. Now open up EAGLE
7. Open or create any schematic
    a. Navigate **Library > Use…**
        i. In the **ldr** folder, highlight everything (you might be able to use **CTRL/CMD + A**, otherwise just click on the top **.lbr** file, then scroll to the bottom, **hold SHIFT + Click** on the last **.lbr** file. Then click **Open**
8. Now when you click or type **add** in the EAGLE command line, the libraries we just drag-and-dropped inside and then "used" will now be available

    Note: You may need to restart EAGLE
9. We are now equipped with Arduino UNO and Arduino IDE experience, how to program and some understanding of how to use ATTiny and the three variants we are using, and finally we are somewhat familiar with EAGLE
    a. We are finally ready to begin the goal of this project, the **ATTiny Programmer**
10. Don't forget to fulfill your **Checkoff Requirements**!

Challenge #6: The ATTiny Programmer

In this challenge, we will make the **ATTiny Programmer** in EAGLE, then we will show you some powerful tools that are sanity checks for making any board in EAGLE (or other ECAD software). We will show you how to ship it to the fabricator; in this project, we introduce you to OSH Park to send our board to get fabricated. There are other, cheaper alternatives, but boards from OSH Park come relatively fast, and are purple! Then we will solder the **ATTiny Programmer** together, and test it out!

Learning Objective

- PCB design using an ECAD program
- Soldering

Required Materials

- Arduino UNO + USB A (male) to USB B (male) cable
- ATTiny85 [**U1**]
- ATTiny84 [**U2**]
- ATTiny4313 [**U3**]
- 8-pin DIP Socket
- 14-pin DIP Socket
- 20-pin DIP Socket
- 10uF Capacitor [**C1**]
- LED [**LED1**]
- Resistor [**R1**]
- Male header pins

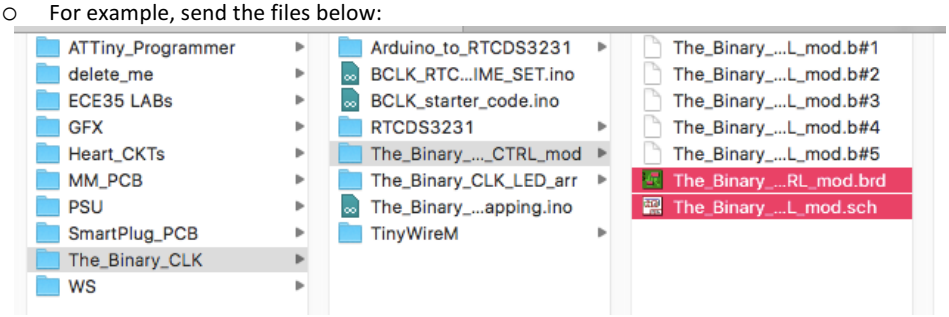
Required Tools

- Arduino UNO + USB A (male) to USB B (male) cable
- Computer + Arduino IDE installed + ATTiny boards installed
- Computer + PCB CAD program (i.e. EAGLE, KiCAD or Altium's CircuitMaker)
- Internet connection
- Wire cutter
- Soldering Iron + solder
- File (recommended)
- Tape (recommended)
- Needle nose pliers (recommended)
- Digital Multimeter (recommended)
- MrSwirlyEye's ATTiny Programmer PCB board from GitHub (EAGLE .sch and .brd files) (optional)

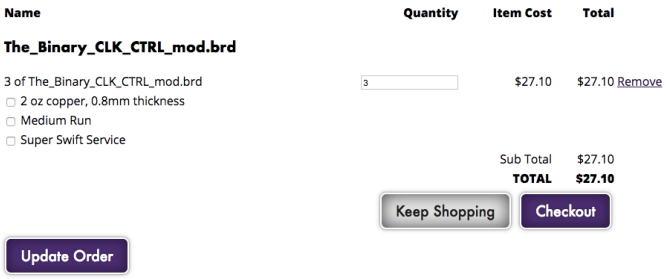
Check Off Requirements

You may either show a TA the following (in person), or email your project TA with the a picture or video (where appropriate), to show proof of completion.

- [ ] Email your project TA your EAGLE .brd and .sch of your version of the ATTiny Programmer (highlighted below)
  - For example, send the files below:



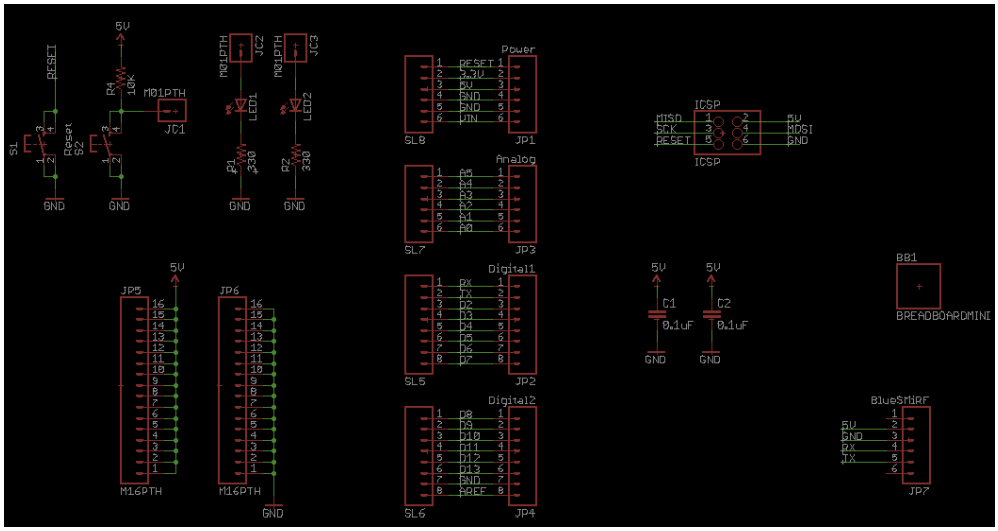
  - You can ignore the .b#N or .s#N files.
- [ ] Email your project TA a screenshot of the cost of your PCB from [OSHPark]
  - For example of the .brd file:



- [ ] Show a TA your completed/soldered ATTiny Programmer PCB board

1. We will now create the **ATTiny Programmer**! Since it is an Arduino UNO shield, we have to make very careful measurements of the Arduino UNO so that the printed PCB board will fit snug onto the Arduino UNO board without a misalignment
    a. Unfortunately, we were not so talented, nor confident we would make perfectly accurate measurements, so we "cheated" a little (or… you could say we were clever)
2. From **Sparkfun**'s website, they provide an EAGLE schematic + board for their **Arduino ProtoShield – Bare PCB**
    a. So, we highly recommend (since we did this ourselves) you download the linked **Arduino ProtoShield – Bare PCB Eagle Files**
3. Open this board in EAGLE
    a. We suggest you put it in your EAGLE project folder
    b. Feel free to rename the file like: **ATTiny_Programmer_v0**
4. Now you can delete almost everything from the schematic, we show before and after images

Before



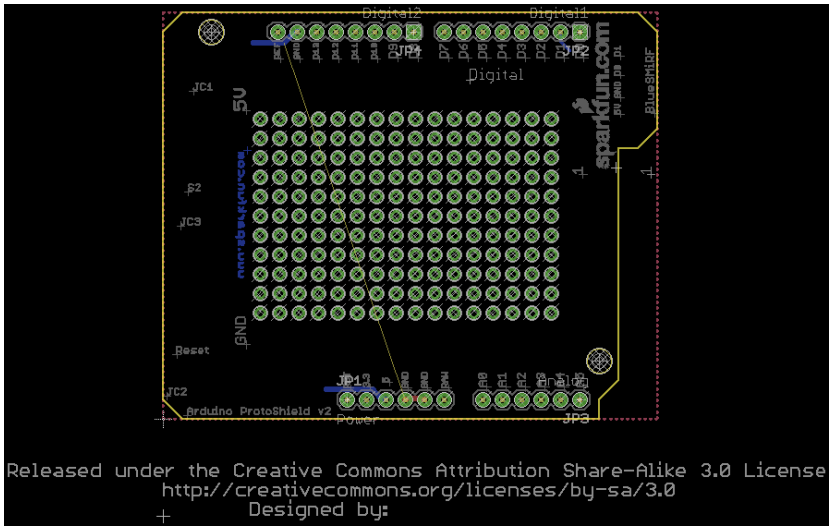
After

Note: Clearly everything is gone (hence why we showed the unnecessary space. Keeping the headers is the ***most important*** part! Because these are in the perfect position on the **.brd** file

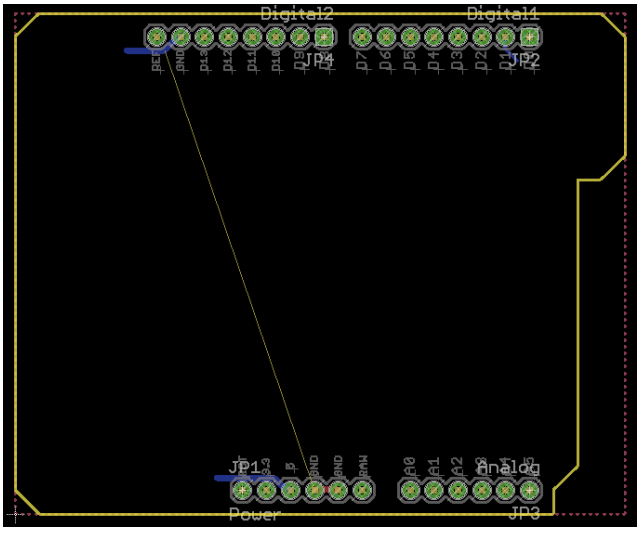
Note: We moved the headers over to the right, you can move them (or leave them) anywhere you want, it just felt natural

5. Similarly, switching to the board, we show before and after images

Before



After

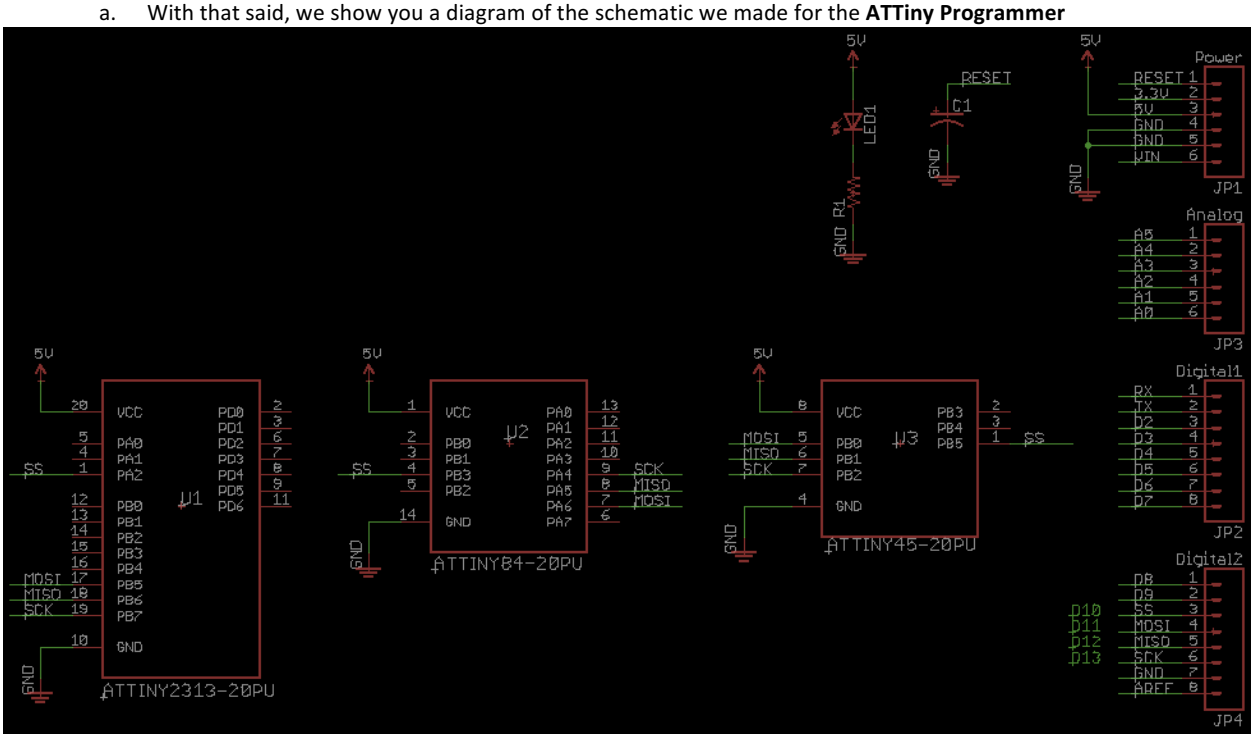


Note: This before and after image is ***AFTER*** you delete stuff from the schematic in step 4! Otherwise it will (obviously) look different with more components

Warning: DO NOT MOVE THE HEADER PINS **JP1**, **JP2**, **JP3**, and **JP4.** Or else when you print your board, it will be misaligned and pretty much useless

6. Next feel free to reshape the board, since there seems to now be a lot of unused space (for now) and that weird tab sticking out. Keep in mind, the size of the board is proportional to the price of the PCB – the smaller, the cheaper

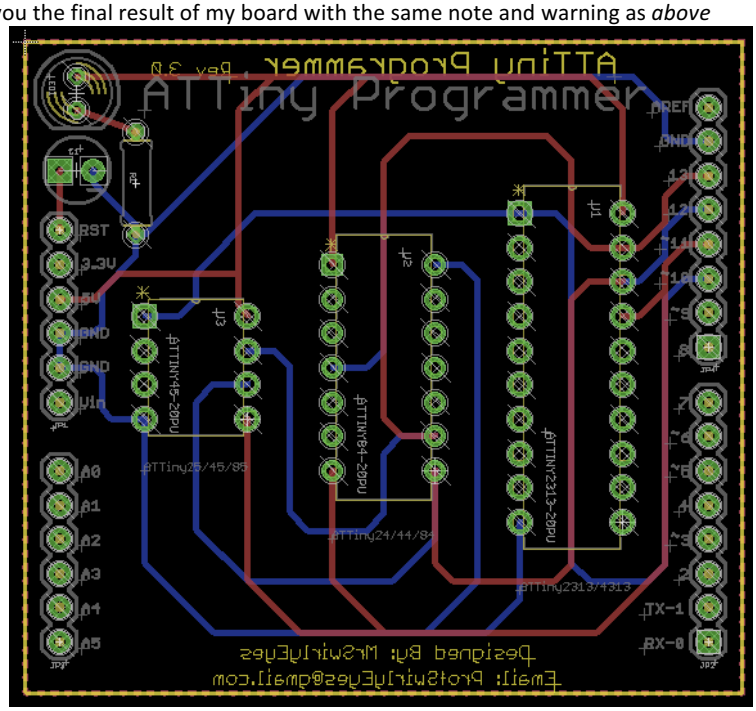
7. After shaping the board to your liking, we will leave the board until after we complete the schematic. You have some freedom for both the schematic or board, however you still must hit some requirements so that your printed board will work
    a. With that said, we show you a diagram of the schematic we made for the **ATTiny Programmer**



Note: We used the footprint (or part) that is the ATTiny2313 and ATTiny45, but it is ***exactly*** the same as the ATTiny4313 and ATTiny85 – respectively

Note: This board has been confirmed to work, however it is far from a perfect board, this is simply our vision of how it all came together

8. Similarly, we show you the final result of my board with the same note and warning as *above*



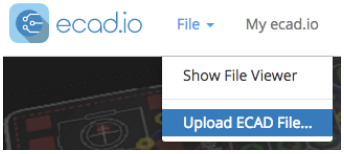
9. We will also provide you all the parts we used in EAGLE so that we can use the same parts (with the same footprints!)

CKT Element	Ref	Library	Device	Note
ATTiny45	**U1**	Atmel_By_element14_Batch_1-00	ATTINY45-20PU	AKA ATTiny85
ATTiny84	**U2**	Atmel_By_element14_Batch_1-00	ATTINY84-20PU	
ATTiny2313	**U3**	Atmel_By_element14_Batch_1-00	ATTINY2313-20PU	AKA ATTiny4313
LED	**LED1**	led	LED5MM	5mm LED
Resistor	**R1**	resistor → R-US_	R-US_0207/7	
Capacitor	**C1**	SparkFun-Capacitors → CAP_POL	CAP_POLPTH2	
Header pin	**JP2, JP4**	SparkFun-Connectors → M08	M08	Already on board
Header pin	**JP1, JP3**	SparkFun-Connectors → M06	M06	Already on board

Note: If you are wondering "what about the DIP sockets," then do not worry; since we are using the ATTiny elements, they are the same foot print as the DIP sockets! Pretty convenient. (A DIP socket is a component that allows you to remove the ATTiny chip for reprogramming, instead of soldering it directly into the circuit)
Note: The "→" denotes that there is a *subfolder* in the library directory
Note: Sometimes the library will not have these *subfolders*, in which case just find the device

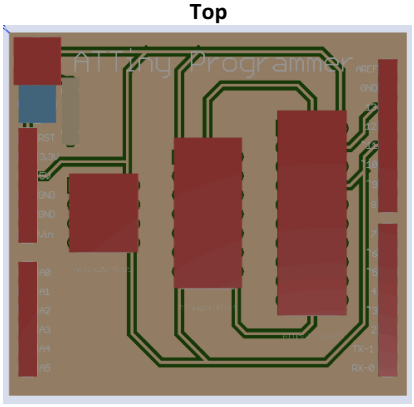
10. Designing the schematic and the board – especially the board – takes lots of time and patience, however we hope you feel that it is a beautiful and artistic aspect
    a. Please do not rush the design, really think about designing it, develop a consistent technique
    Note: This is a ***relatively*** easy project. Develop your skills now, for when you have several more components and hundreds of connections. It will become monstrously more difficult
11. When you are confident you are finished, ***double check everything 7 more times!! Get your friends to check it!***
    a. Run the **ratsnest** command
    b. Run the **drc**
    c. Run the **erc**
    d. Make sure everything is not returning errors, or – in the case of ratsnest – is returning "nothing to do"
        i. If you make a bad print, you waste time and money!! You will not get a refund from any vendor for your own mistakes
    e. Google any errors that you receive if you do not know what they mean or how to fix them
    Note: Clearly we are stressing checking your work! We have made prints in the past that were garbage, and we had to throw away boards that were received from the fab without even soldering a single component
12. Don't forget to fulfill your **Checkoff Requirements**!

13. Again, when you are confident you have checked everything, now we will show you two great tools to gain perspective, to ***again*** double check your work, and to confirm what your board will ***probably*** or is ***supposed*** to look like
    a. Open your browser and go to: **ecad.io**
        i. Create an account
        ii. After registering, navigate on the homepage: **File > Upload ECAD File…**

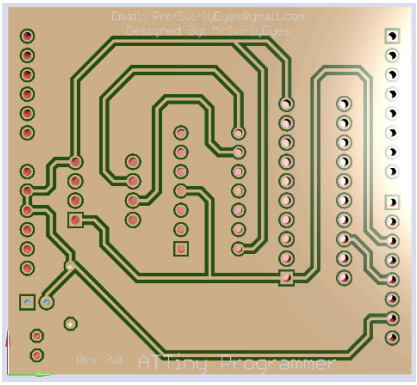


        iii. It will prompt you to upload your ECAD file
        iv. It might take a moment to load, and might give you some error, do not worry about them so much, there is only one reason why we are using this tool – that is, to get a 3D visual perspective of our project
        v. Check out your board, ***again validate your connections!***

Top



Bottom



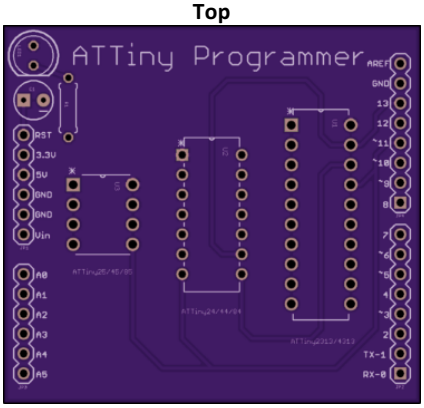
Note: The height of the components are not to scale, we do not care for this tool

Note: The program messed up my Bottom board text printing (won't affect actual print)

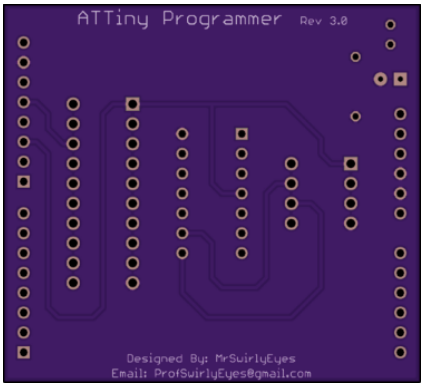
        vi. So why did we show you this tool? Again another perspective and way to double check, also if you want an **STL** format for 3D printing, this is a nice free tool

b. Now, open your browser and go to: **Oshpark.com**
   i. Click on the **Get Started Now** button
   ii. Upload your EAGLE **.brd** file by clicking on the **Select a file on your computer** button and selecting your board
   iii. After a moment, it will ask you for a name and description for your project. You can put just about anything in these fields.
   iv. Click on **Continue**
   v. The next page shows you all the details about your board
   vi. This is your final chance, to… ***again validate your connections!***
   vii. According to OSH Park, they claim your board *ought* to look exactly like the images provided

Top



Bottom



Note: There are other perspectives that OSH Park gives. We did not show them here, but they are also extremely important – they show drill details, silk layer details, and the solder mask (the connections!)

   viii. If you proceed to the **Approve and Order** tab, you can see how much your board will cost (be aware that you get 3 copies for the given price!)

14. Please use these tools before you ship your board to be printed, they are very powerful and insightful
15. After taking advantage of these tools, for the last time we will say it, ***check your work… again!***
    a. If you wish, once you are confident in your work, you can send your board to the fab!
    b. You can alternatively send our board to the fab, you can get our latest version from **MrSwirlyEye's Github for the ATTiny Programmer**
       i. The latest version of the board has been printed, and is confirmed should print perfectly fine without any known error
16. Assuming you are now done, and received your board from the fab, before soldering anything to your board
    a. Check all your connections on your printed board using a **Digital MultiMeter (DMM)**
17. Once you find all the connections are correct, now it is time to solder
    a. If you have never soldered before, please refer here for **How to Solder** for a collection of excellent soldering information
18. If you made it this far, you should have a completely soldered ATTiny Programmer. If you have not tested it out already, repeat **Challenges #2**, **#3**, and **#4** using your new ATTiny Programmer board instead

    Note: It should be obvious that you although you can *place* all 3 ATTiny chip variants on the ATTiny Programmer, you ***cannot*** program all 3 of them at the same time
19. Congratulations! You have completed the **ATTiny Programmer** project!

Conclusion

The goal of this project was to introduce you to ATMEL's ATTiny family of programmable integrated chips, how to use different boards in the Arduino IDE and configure the Arduino UNO as an In-System Programmer, and ECAD PCB design. Although this project alone is not very flashy, it is a tool that makes programming these ATTinys very easy. It will be up to you, the user, to find applications for these ATTinys – which would then make this project feel a bit more appreciated. If you wish to find two projects that we used these in, you can refer to **MrSwirlyEyes' HEART CKT II** and **MrSwirlyEyes' The Binary CLK**. These projects utilize the ATTiny84 and ATTiny4313 – respectively. Another skill we wanted you to gain much experience in is using ECAD software. We believe it is a very powerful skill to have in your skillset and it is pretty satisfying to print your own successful board.

Resources & References

High-Low Tech's guide to understanding ATTiny's and how to program with Arduino as In-System Programmer (ISP):

http://highlowtech.org/?p=1695

Spence Konde's ATTinyCore libraries on GitHub:

https://github.com/SpenceKonde/ATTinyCore

Ladyada's detailed explanation of ATTiny's, most notably the ATTiny2313, about fuses and B.O.D.:

http://www.ladyada.net/learn/avr/fuses.html

Wikipedia on Atmel AVR microcontrollers:

https://en.wikipedia.org/wiki/Atmel_AVR

Atmel's ATTiny25/45/85 Datasheet:

http://www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf

Atmel's ATTiny24/44/84 Datasheet:

http://www.atmel.com/images/doc8006.pdf

Atmel's ATTiny2313/4313 Datasheet:

http://www.atmel.com/images/doc8246.pdf

Arduino UNO Technical Specifications from the official Arduino website:

https://www.arduino.cc/en/Main/ArduinoBoardUno