EE/CS 120B: Introduction to Embedded Systems
University of California, Riverside
Winter 2016

Laboratory Exercise 2

This laboratory exercise requires a parts kit. As components may change from year to year (e.g., due to discontinuation), some parts of the lab may refer to components that are no longer in use. Additionally, there have been a few minor differences between the parts kits offered by the IEEE at UCR and the official parts list maintained by the instructor. This lab is intended to be general enough to handle these differences; however, if there is any concern or confusion, please contact your TA immediately.

Breadboard and Electrostatic Discharge

Review electronics, use of breadboards, and safety when working with electronics.

Before touching any of the sensitive circuit components, please make sure to discharge static electricity off of yourself. To properly discharge static electricity, touch a grounded metal component (i.e. metal on the lab machines). The discharge of static electricity onto a component is known as Electrostatic Discharge (ESD):

http://en.wikipedia.org/wiki/Electrostatic_discharge

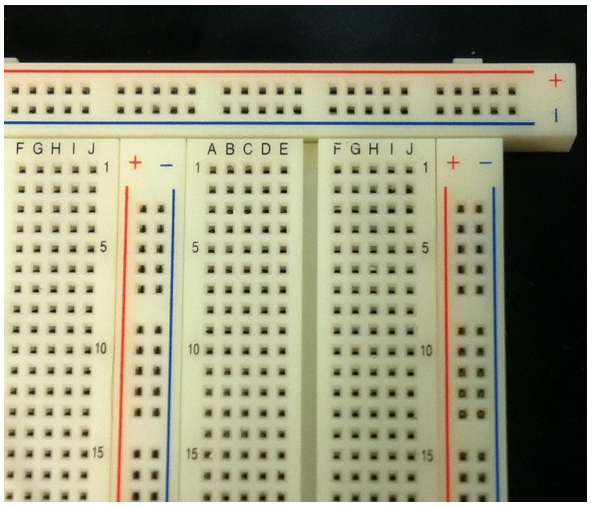
ESD can damage electronic components so take caution when handling sensitive electronic equipment.

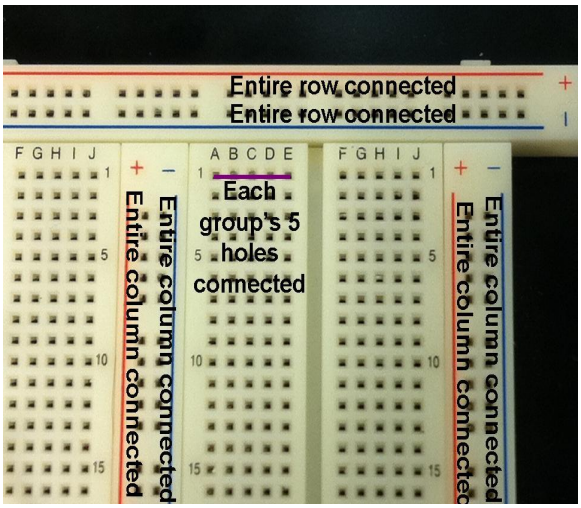
Note: Always use anti-static bags and tubes as needed.

The for the IEEE at UCR provides a detailed overview of the laboratory parts kit including datasheets, video tutorials, etc. (http://ieee.ee.ucr.edu/parts/cs120b/).

In particular, it is a good idea to familiarize yourself with the ATmega1284 datasheet, and pinout diagram. In addition to the IEEE's website, these documents are posted on iLearn.

On a breadboard, the holes along a red line (also called a rail or bus) are connected internally, likewise, the holes along a blue line are also connected internally. Red is for power, blue/black for negative (ground). For the rest of the breadboard, the only internal connections are among the holes in a 5-hole group in a row.

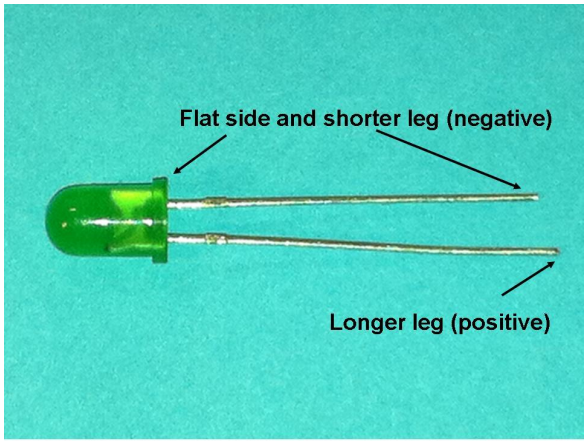




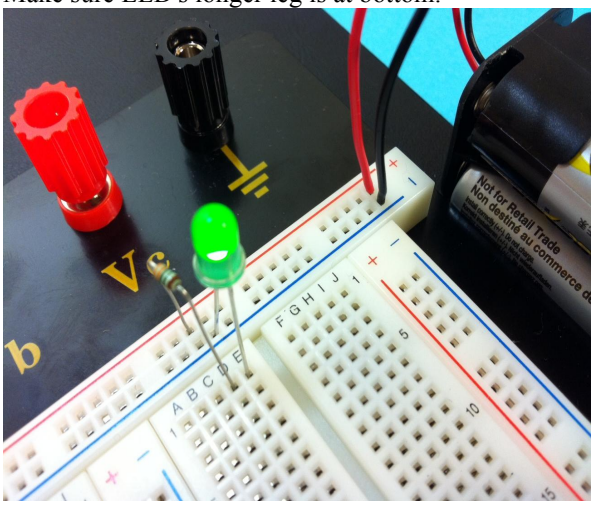
As a first circuit, you might connect a 330Ω resistor to the top power rail and then a 5-hole group, then connect an LED's positive leg to another hole in that same group and the other leg back to the negative rail, as shown (use the specific columns shown).

Note: You may use a graphical resistor calculator or reference a chart if you are unfamiliar with resistors. Resistance is always measured in Ohms (Ω).

http://www.sorion-group.com/Resistor/resistor.htm
http://www.bpesolutions.com/bpemanuals/Resistor.Chart.1.gif



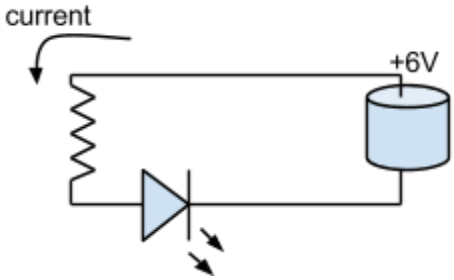
Make sure LED's longer leg is at bottom.

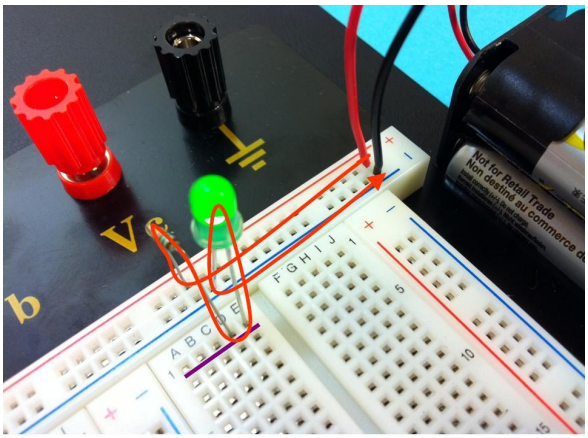


Note: When connecting a battery always connect the positive (red) terminal first and then the negative (black). This is because electrons flow from negative to positive and you reduce the risk of creating a short.

Upon connecting a battery as shown, the LED should light up. Now remove the battery wires.

The schematic for that first circuit is shown below, along with an illustration of how current flows from the battery, through the resistor, through the LED, and back to the battery.





The resistor is needed to limit current flow through the LED; the LED datasheet indicates the maximum current that should flow through, typically around 10 mA - 20 mA. It also indicates the typical voltage drop that will occur across the LED; the remaining voltage drop V will occur across the resistor, so the current that will flow can be computed using V = IR (e.g., if the battery outputs 6V and the LED voltage drop is 4V, the remaining 2V will occur across the resistor. If R is 200Ω, then current will be 2V / 200Ω = 10 mA).

Here is an example of resistance calculation for the diagram above.

V_S = 6 volts (V):	direct current voltage source
V_D = 4 volts (V):	voltage required to turn on LED
$V_R = V_S - V_D$ = 2 volts (V):	voltage across the resistor in series with LED
I = 10 milliamps (mA):	max. current flowing through LED for correct operation

$R = V_R / I = 2V / 10mA = 200\Omega$

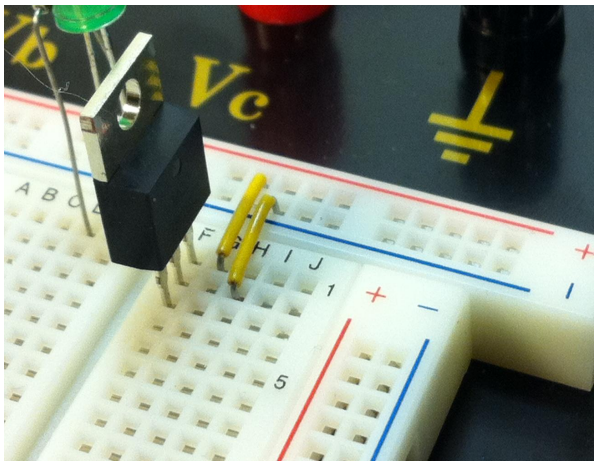
Let's keep the LED and resistor there as an indicator of when the board is powered.

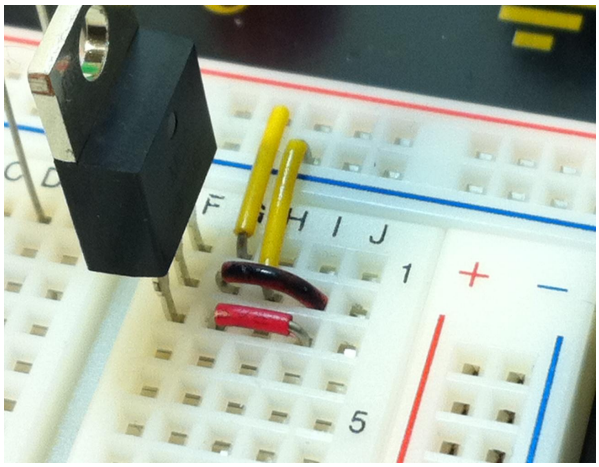
Basic board wiring for power (for non-IEEE Parts Kits)

Note: The IEEE parts kit does not require an external voltage regular. If a voltage regular is not required, you can skip this section.

We'll run power through a voltage regulator that reduces a higher voltage down to 5V, to protect the chips and other devices we'll put on the board. This way, if we accidentally apply too high of input voltage, the regulator will reduce it, or will burn out trying (It is far cheaper to replace a voltage regulator than a microcontroller). The voltage regulator's left pin is power-in, middle is ground, and right pin is power-out (5V, which we'll also call V_{CC}).

Make sure the board is NOT powered (always disconnect power when adding/removing board components). Add the voltage regulator and add connectors from the right pin to the top power rail and from the middle pin to the top ground rail, as shown below on the left.



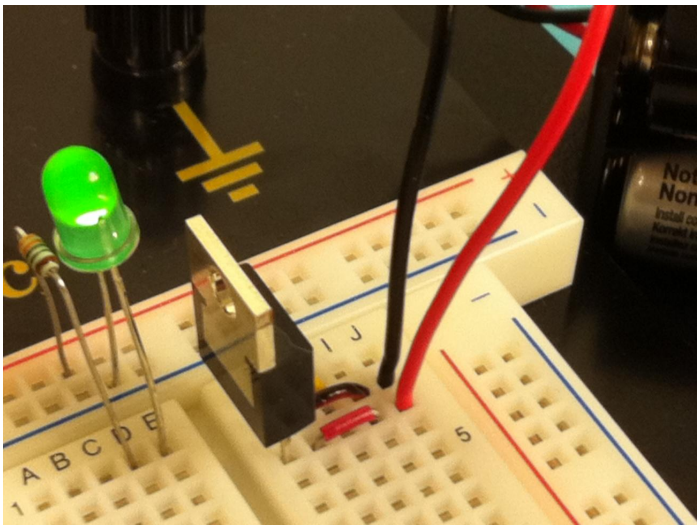


To make clear where the battery pack's black and red wires should connect, add short black and red wires as shown above on the right. With the available wires in our kit, we had to color a wire black (it was previously orange).

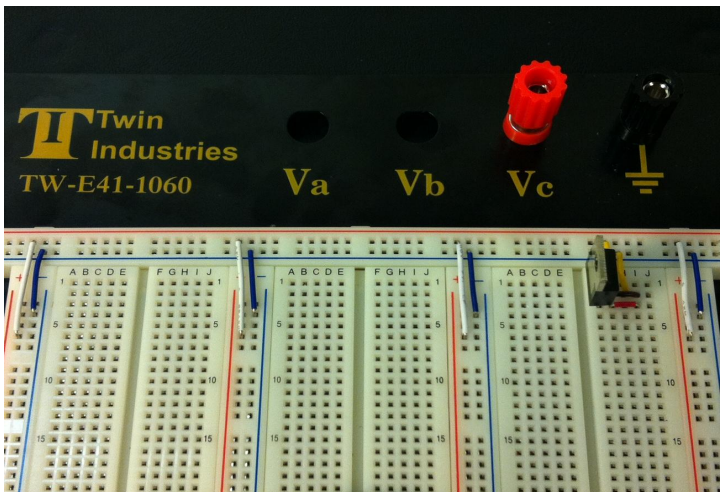
Now connect the battery to the holes next to the short red/black wires, and again the LED should light, as shown below. You might check voltages using a multimeter. Remove the battery wires.

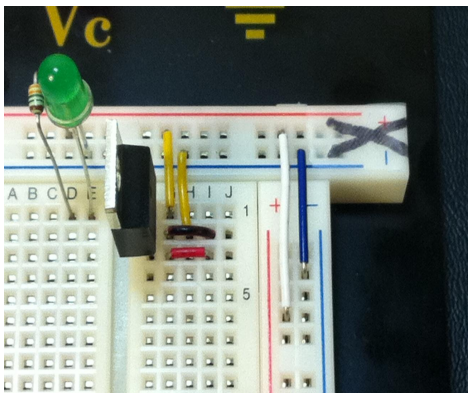
Note: The voltage regulator may get hot during operation, so avoid touching it during operation.

Note: How to Insert/Remove Batteries:
To insert a battery: connect positive (red), then connect negative (black).
To remove a battery: remove negative (black), then remove positive (red).



Connect the horizontal power/ground rails to all vertical power/ground rails as shown below.





On the top rail's right, we've added tape with an "X" to remind ourselves not to plug the battery into the top rail, but rather to plug in next to the voltage regulator.

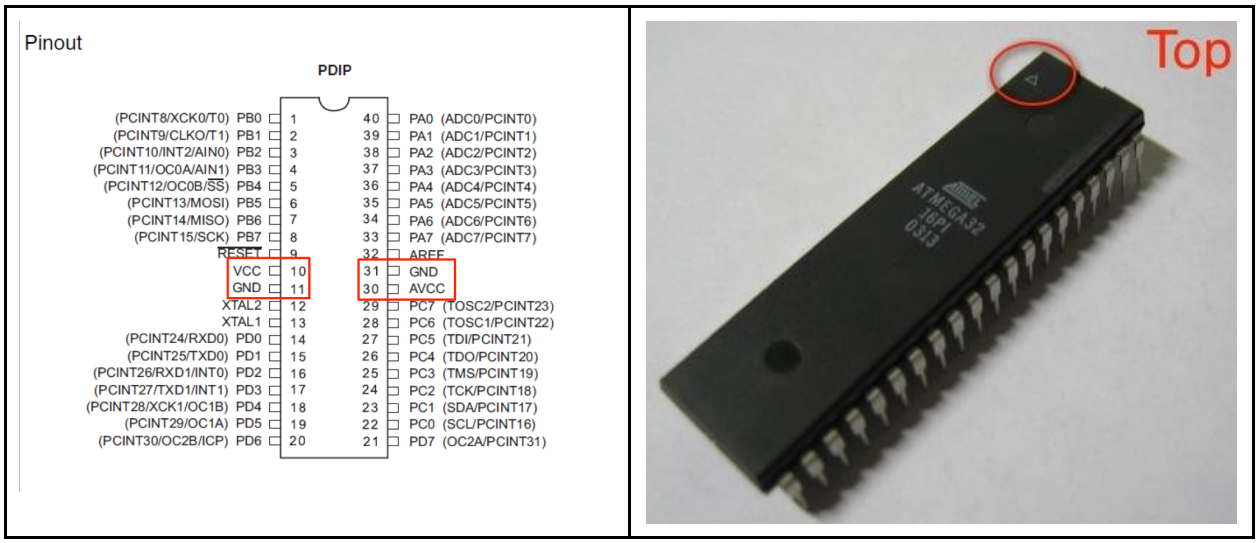
Check that all power/ground rails have power by connecting the battery as earlier and using LEDs or a multimeter on each vertical power/ground pair to ensure they have power. Remove the battery wires.

***Important Note*:** If at any time while the power supply is connected you smell burning, or feel excessive heat coming from a component *disconnect your power source immediately and check your wiring! Do not touch any components, until they cool down.* If this happens you possibly have miswired the board and are burning components. They may still work afterwards but they also may be toast! Be sure to consider this when debugging.

Add the microcontroller

*Before you begin ensure the board is **NOT** powered and you have discharged any static electricity from your person.*

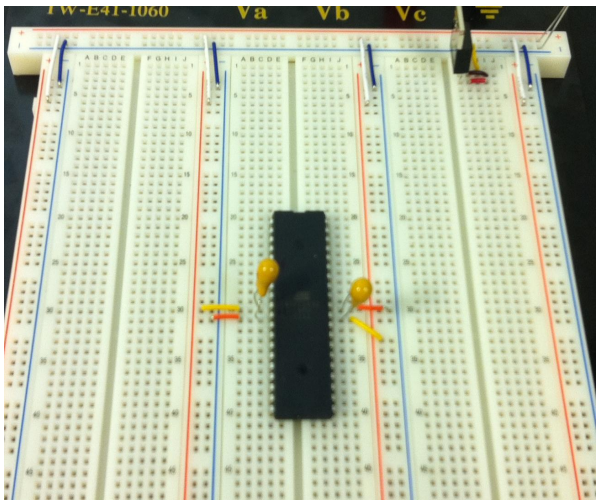
First you will want to determine the orientation of the microcontroller (sometimes abbreviated µC). There is a notch in the top of the microcontroller and pin 1 is marked with a dot or arrow as pictured below.

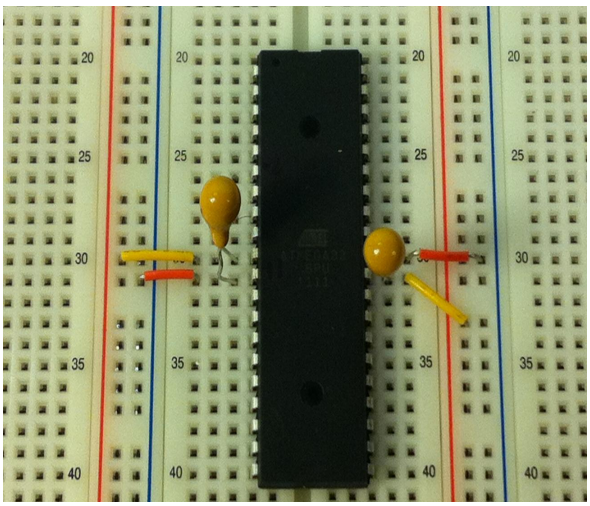


Carefully insert the microcontroller chip onto the breadboard as shown below. Place the top pin in row 21 to simplify determining the chip's pin numbers, such that pin *1* (upper left of chip) is in row 2*1*, pin *2* in 2*2*, pin *3* in 2*3*, etc. The chip's horizontal pin spacing is slightly wider than the board, so place one side partly in, then angle the other side's pins in carefully; once all pins are in holes, gently press downwards until the chip snaps into place. If you have to remove the chip for any reason, use the chip extractor tool. A video demonstrating chip insertion/extraction has been posted on iLearn under 'Assignments.'

Next, we will add wires for both V_{CC} and ground (gnd) to both sides of the microcontroller per the pinout above. Note that V_{CC} and gnd connect to specific pin numbers on each side as shown -- be careful to note that V_{CC} is above ground on the left, but ground is above AV_{CC} (which is V_{CC}; as per the datasheet) on the right.

V_{CC} (red rail) connects to both pin 10 and 30. Ground (blue rail) connects to pin 11 and 31. Add the (optional) capacitors across V_{CC} and ground, to smooth out the 5V being supplied to the microcontroller (reducing spikes or dips). This ceramic capacitor doesn't have a positive or negative side, so don't worry about its orientation (polarity). Confirm your setup with the pictures below.



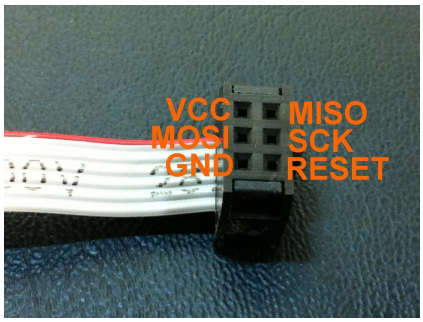


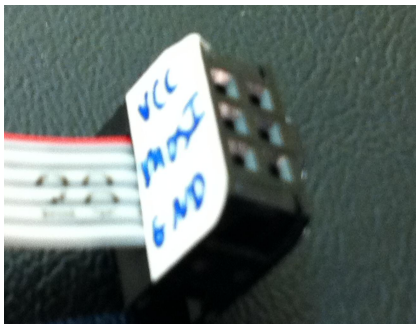
Prepare the AVR In-System Programming (ISP) header for non-IEEE Parts Kits

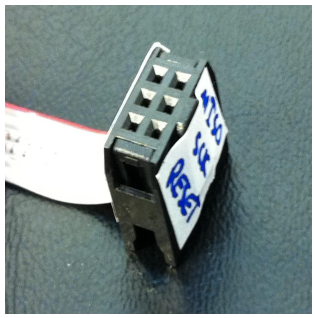
Note: The IEEE parts kit has a small chip so that you don't need to wire the header.

(Do **NOT** connect USB cable to the PC yet)

Connecting the AVRISP header to the ATmega1284 is slightly awkward; miswiring is a common problem, and thus we'll do a few things now to prevent problems later. First, write the header's six pin hole names on tape applied to the header, as shown below. The AVRISP mkII wire 1 is denoted by the red wire on the ribbon cable and an arrow on the connector [1]

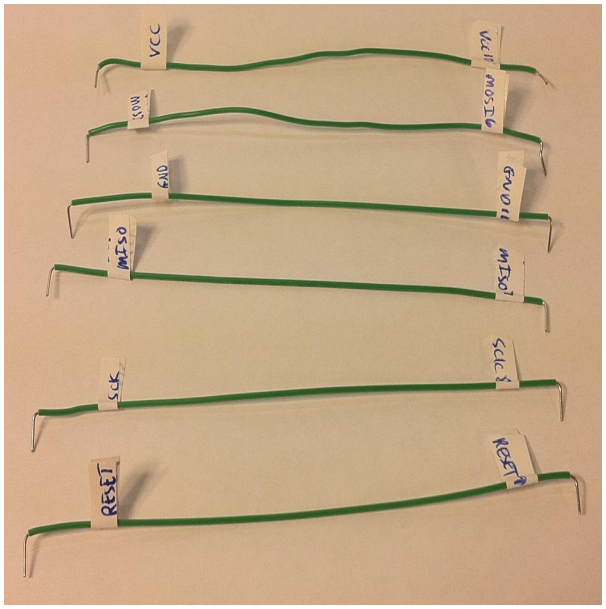






Get six equal-length wires (use the long green wires in your kit) and use tape to label them with the same names on both ends, as shown. On one end, also write the ATmega1284 *pin* number (*not* port number) that matches the name (i.e., GND is 11, V_{CC} is 10, RESET is 9, SCK is 8, MISO is 7, and MOSI is 6).

[1] The AVRISP document's diagram, not shown, shows the header from the top side rather than the bottom side having the holes, which confuses many people.

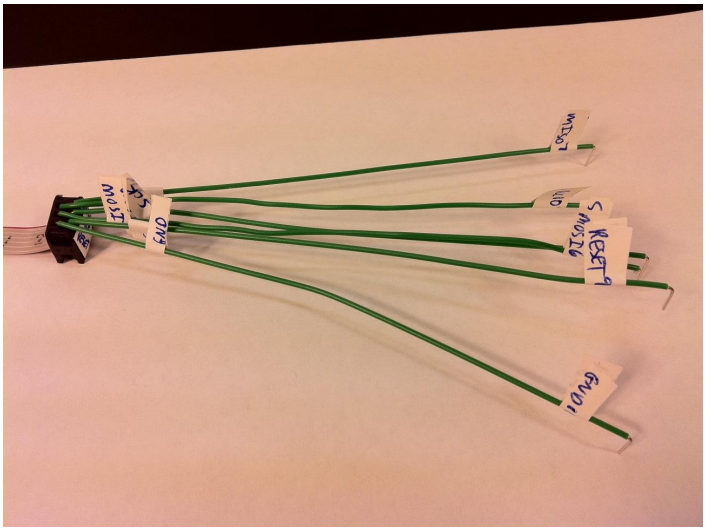


Pinout

PDIP

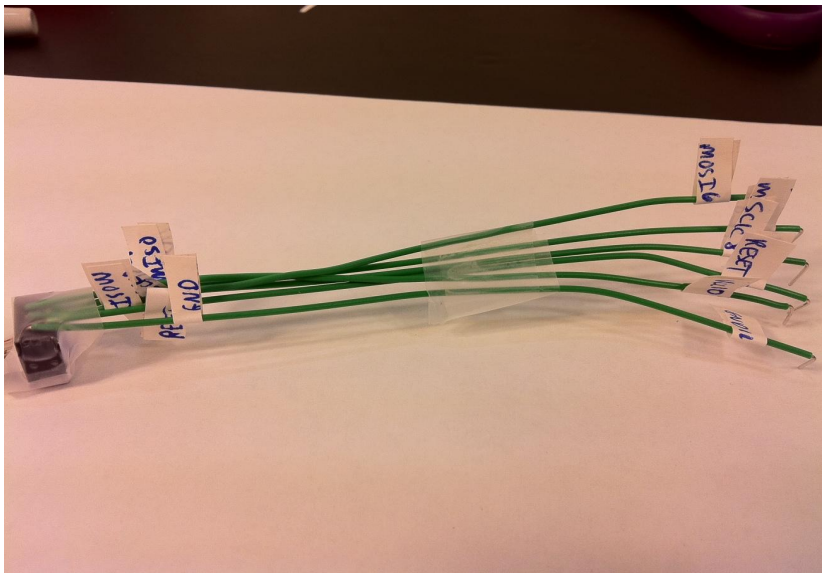
(PCINT8/XCK0/T0) PB0	1	40	PA0 (ADC0/PCINT0)
(PCINT9/CLKO/T1) PB1	2	39	PA1 (ADC1/PCINT1)
(PCINT10/INT2/AIN0) PB2	3	38	PA2 (ADC2/PCINT2)
(PCINT11/OC0A/AIN1) PB3	4	37	PA3 (ADC3/PCINT3)
(PCINT12/OC0B/SS) PB4	5	36	PA4 (ADC4/PCINT4)
(PCINT13/MOSI) PB5	6	35	PA5 (ADC5/PCINT5)
(PCINT14/MISO) PB6	7	34	PA6 (ADC6/PCINT6)
(PCINT15/SCK) PB7	8	33	PA7 (ADC7/PCINT7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2/PCINT23)
XTAL1	13	28	PC6 (TOSC1/PCINT22)
(PCINT24/RXD0) PD0	14	27	PC5 (TDI/PCINT21)
(PCINT25/TXD0) PD1	15	26	PC4 (TDO/PCINT20)
(PCINT26/RXD1/INT0) PD2	16	25	PC3 (TMS/PCINT19)
(PCINT27/TXD1/INT1) PD3	17	24	PC2 (TCK/PCINT18)
(PCINT28/XCK1/OC1B) PD4	18	23	PC1 (SDA/PCINT17)
(PCINT29/OC1A) PD5	19	22	PC0 (SCL/PCINT16)
(PCINT30/OC2B/ICP) PD6	20	21	PD7 (OC2A/PCINT31)

Connect the unnumbered side of each wire to the header, matching each wire name to the hole name:



Tape down the wires onto the header to prevent disconnection. Sort the unconnected side of the wires by pin number (6, 7, 8, 9, 10, 11, with 6 on top), and apply tape to their center to help preserve that order, as shown below. Note that some wires will have to cross.[2]

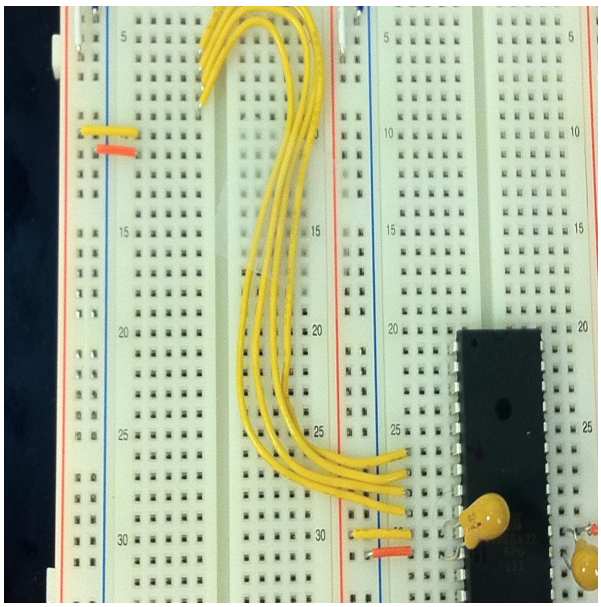
[2] Better connectors can be created later if the student is interested, requiring different connector parts and wires. Ask your TA or Professor.

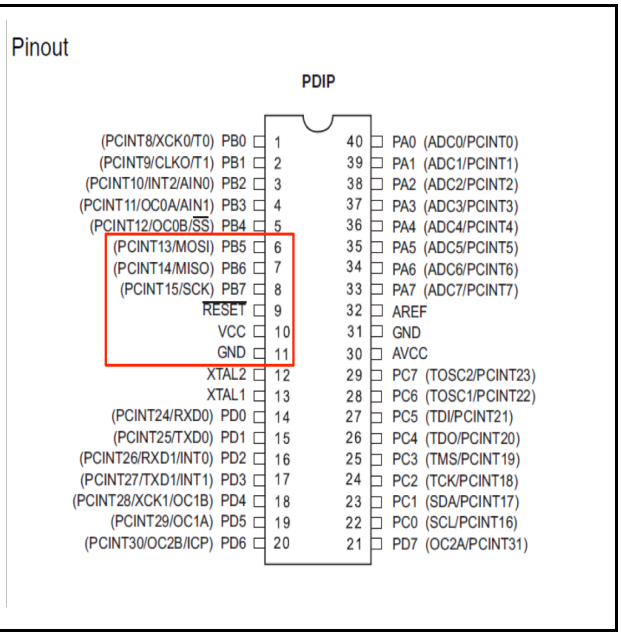


Completed AVRISP header.

Connect the AVRISP header to the microcontroller

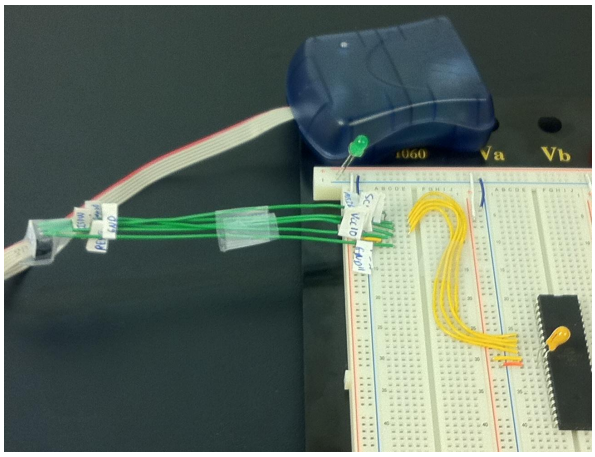
The header's six wires need to connect to the ATmega1284's pins 6-9 and V_{CC}/ground. However, we may be connecting and disconnecting those wires frequently. To avoid such connecting/disconnecting close to the microcontroller, run 4 wires from the ATmega's pins 6-9 to the upper-left of the board, starting with row 6 (to correspond to ATmega's pin 6), and add connections to V_{CC} and ground, as shown. We taped the four-wire group onto the board to keep them neat.

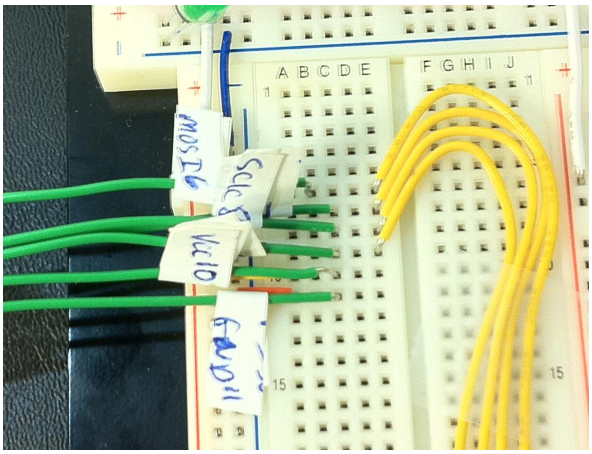




You are now ready to connect the header. Insert each header wire into the proper row -- be careful that the wire numbered 6 is in row 6, the wire numbered 7 is in row 7, etc.

Miswiring is a common error, so take your time.

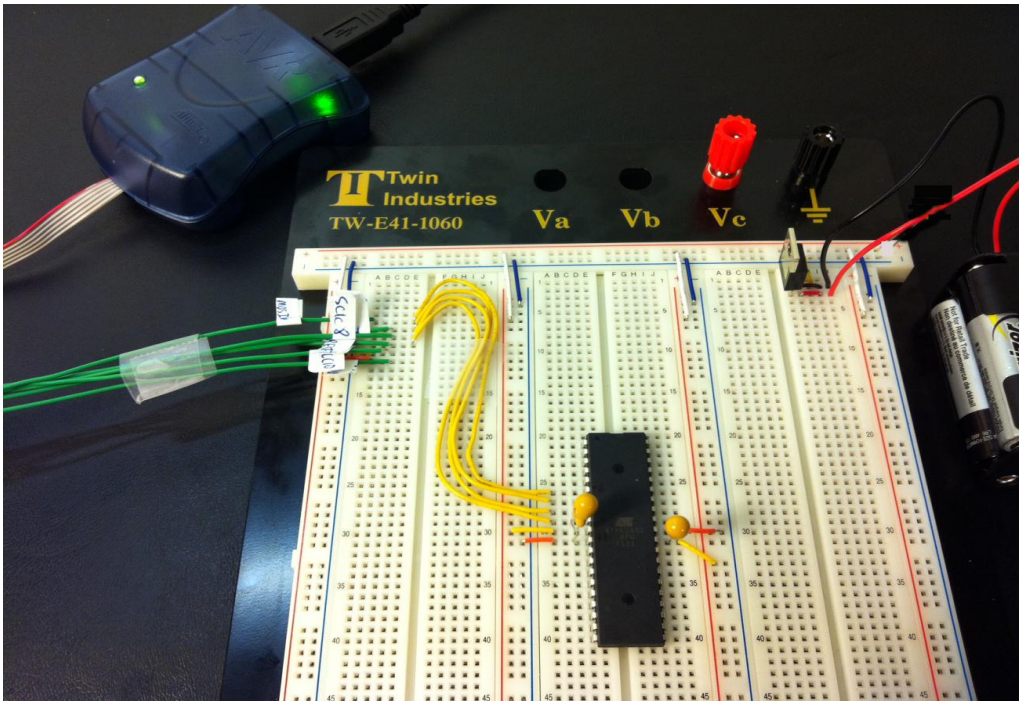




Connect AVRISP to PC

Plug in the USB cable (came with the AVRISP) from the blue AVRISP device to the PC. The PC recognizes a new USB device. **NOTE:** If this is your first time connecting the AVRISP device, your PC may indicate that you need to install the drivers for the device. Follow the on-screen instructions; if you are unsure, you may refer to the 'Installing the AVRISP' section of our AVR FAQ, which is posted on iLearn under 'Assignments.'

The AVR's internal LED (indicating USB connectivity with the PC) should be green. Apply power to the board. The AVRISP's external LED (indicating connectivity with the ATmega1284) should illuminate green.



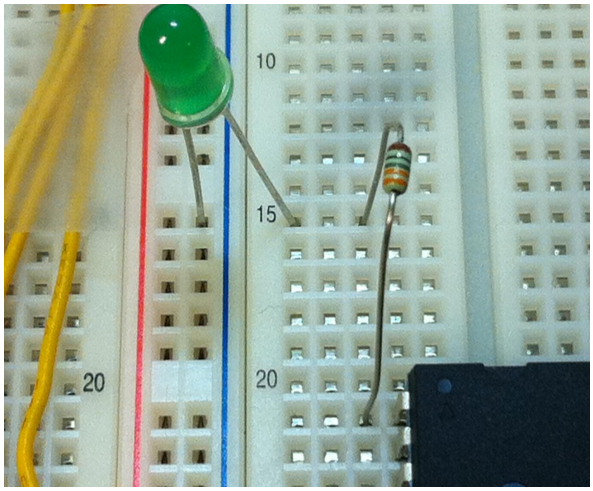
A first program on the microcontroller chip

This program, from an earlier lab, just sets PB3..PB0 to 1. We'll connect an LED to PB0, so the program should turn on PB0's LED.

1. Prepare the board for the upcoming program by adding an LED to PB0.

    a. Remove power from the board (note that the AVRISP external LED turns red). It's not necessary to unplug the AVRISP.

    b. Add a 330Ω resistor and LED in series between PB0 and ground as shown. Orient the LED properly, with the negative (short) leg plugged into ground. (The resistor limits the current flow, extending the LEDs life and keeping the microcontroller cooler).



2. In AVR Studio, create a new project named "lab_chip" for the ATmega1284 and write the following program (from an earlier lab):

```
#include <avr/io.h>

int main(void)
{
        // Configure port B's 8 pins as outputs
        DDRB = 0xFF; PORTB = 0x00;
        while(1)
        {
                // Write to port B's 8 pins with 00001111
                PORTB = 0x0F;
        }
}
```

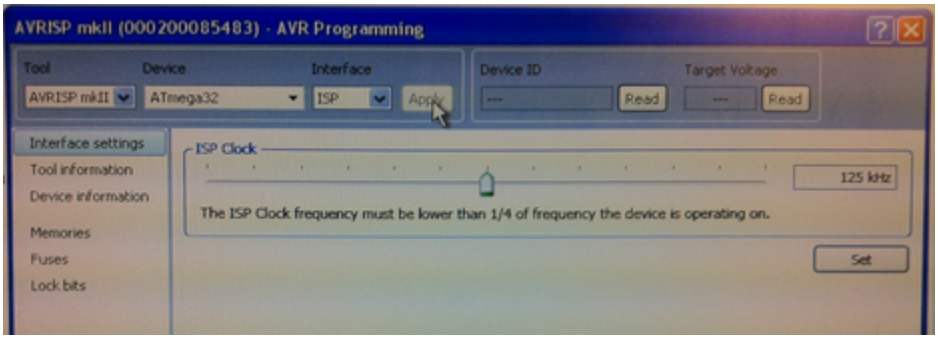
3. Select "Build -> Solution".

4. (Previously, you would next select "Debug -> Continue" to simulate the program. But this time, we're going to download the program onto the ATmega1284 chip, a process called "programming" the chip. In the future, you may want to first run simulation to check or debug your program.).

5. Select "Tools -> AVR Programming". Make sure the USB cable is connected.

NOTE: If this is your first time connecting the AVRISP device, AVR Studio may tell you that you need to update the device's firmware. Follow the on-screen instructions; if you are unsure, refer to the 'Firmware upgrade for the AVRISP section of the AVR FAQ. Do not skip this step.

6. In the AVR Programming popup window, select Tool as "AVRISP mkII", Device as "ATmega1284", and Interface as "ISP". Click "Apply". [3]



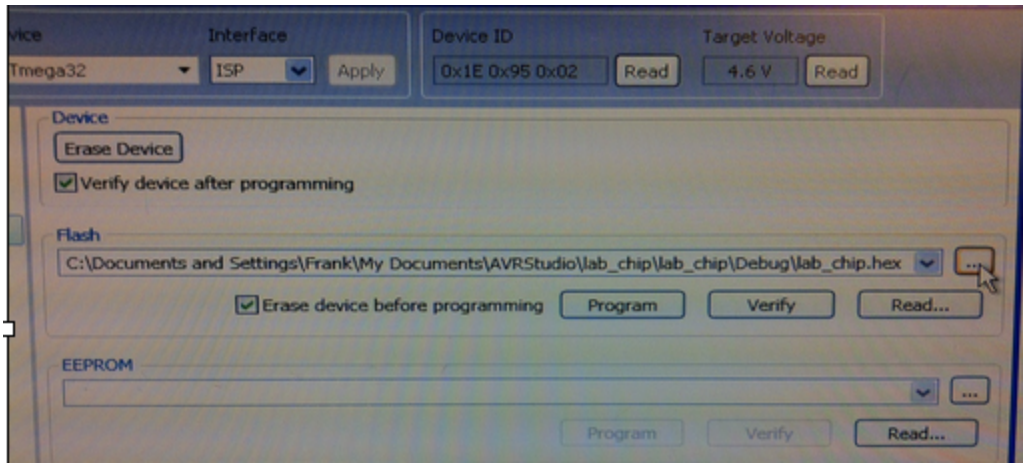
7. To check if the AVRISP is connecting properly to the chip, first add power back to the board. Then click Device ID "Read" and Target Voltage "Read" buttons -- the fields should fill in with values. If instead an error appears:

    a. Make sure you powered the board. The chip must have power to communicate with the AVRISP.

    b. If you still have an error, a likely culprit is incorrect wiring from the AVRISP header to the chip. Double check the wiring carefully.

8. Click on "Memories". To the far right of "Flash" (referring to the chip's flash memory, where the program will be stored), click on the "**...**" to open a file. Find the lab_chip.hex file for the project you built above.

[3] If the mkII doesn't appear as an option (but only the simulator does), make sure your USB cable is plugged in. If it still doesn't appear, try exiting AVR Studio and then running it again.

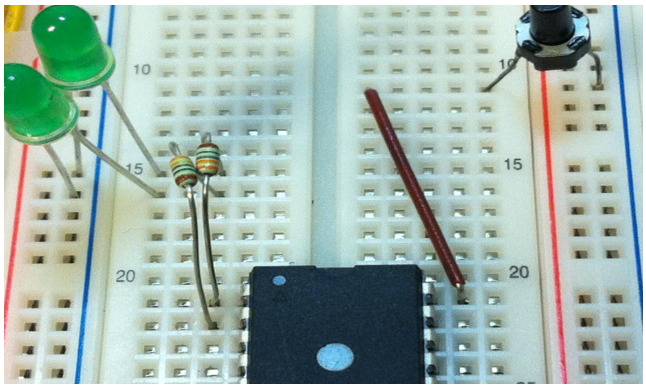


9. Click "Program". The AVRISP external LED should blink orange briefly, and then turn green again. (If not, make sure you still have power to the board). AND, your PB0 LED should illuminate. You can add more LEDs to PB1, PB2, and PB3 and those LEDs should illuminate. An LED connected to PB4 should not illuminate.

A program using a button for input

This program, from an earlier lab, reads an input button and sets two LEDs to 01 (if not pressed) or 10 (if pressed).

1. Prepare the board for the upcoming program. Be sure power is removed. Add an LED to PB1, and add a button connecting to ground on one end and to PA0 on the other. When pressed, the button forms a connection; else there is no connection.



Note: PA0 will be programmed in *pull-up mode*, meaning that when the pin has *no input* the program will read it as 1, and when the pin has a *0 input* (ground) the program will read it as 0. Note how the above button provides PA0 with either no input (when not pressed, so read as 1) or with 0 (when pressed, so read as 0). Therefore, ***pressing*** the button causes PA0 to be ***read as 0***, and releasing as 1.

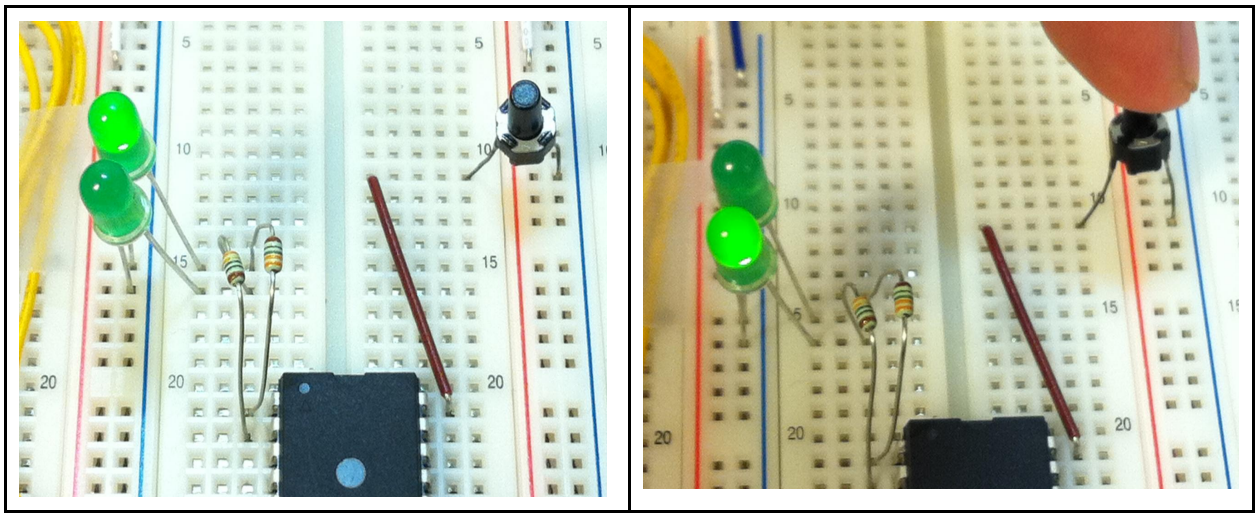
2. Create a new project named "lab_button" with the following program:

```
#include <avr/io.h>
int main(void)
{
	// Configure PORTA as input, initialize to 1s
	// Configure PORTB as outputs, initialize to 0s
	DDRA = 0x00; PORTA = 0xFF;
	DDRB = 0xFF; PORTB = 0x00;

	unsigned char led = 0x00;
	unsigned char button = 0x00;
	while(1)
	{
		// if PA0 is 1, set PB1PB0=01, else =10

		// 1. Read inputs
		// button is connected to A0
		button = ~PINA & 0x01;

		// 2. Perform Computation
		// True if button is pressed
		if (button) {
			// Sets B to bbbbbb01
			// (clear rightmost 2 bits, then set to 01)
			led = (led & 0xFC) | 0x01;
		}
		else {
			// Sets B to bbbbbb10
			// (clear rightmost 2 bits, then set to 10)
			led = (led & 0xFC) | 0x02;
		}

		// 3. Write output
		PORTB = led;
	}
}
```

3. Build the project and program the chip. (Don't forget to power the board).

4. PB0's LED should be on and PB1's LED off. Press the button, and note that PB0's LED turns off and PB1's LED turns on.

Note that PORTA is initialized to 0xFF -- this is essential for pull-up mode. Failing to initialize to 0xFF may result in strange errors from the port's read values being inconsistent.



Mechanical switches (e.g., buttons) are a common input interface to a microcontroller. They are asynchronous (i.e., you as the human user does not share the microcontroller's clock) and they are not electrically clean. Pushing a button once may create multiple electrical transitions between the metal contacts, due to physical bouncing. As a result, a single button push may generate multiple electrical transitions that are read as multiple button pushes. *Debouncing* refers to a variety of hardware and/or software processes that clean the signal to ensure that one electrical transition is read when a button is pushed.

Later on in the class, we will learn techniques for software debouncing; for the purpose of this laboratory exercise, debouncing is an unavoidable problem. Be prepared to deal with it. Note that incorrect application behavior due to debouncing is an external I/O program, and does not necessarily mean that your application is incorrect and needs fixing.

Using PORTC -- Disabling JTAG required

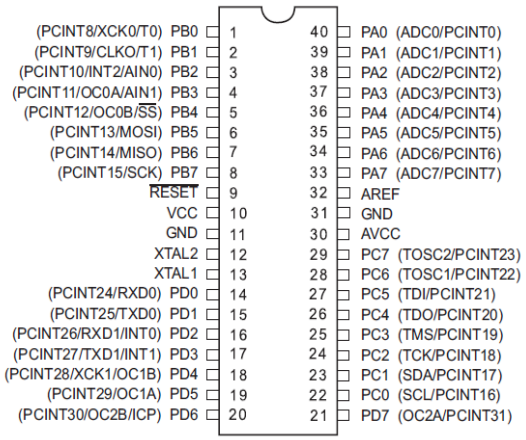
Add 8 LEDs (with resistors) to port C's pins and write a program that sets port C to 0x00 initially and to 0xFF while the PA0 button is pressed. *Observe that, incorrectly, not all the LEDs light.* The reason is that some pins on the ATmega1284 have multiple possible purposes, as indicated in the parentheses of the chip's pinout diagram and as described in the ATmega1284 datasheet:

Table 13-9. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator pin 2) PCINT23 (Pin Change Interrupt 23)
PC6	TOSC1 (Timer Oscillator pin 1) PCINT22 (Pin Change Interrupt 22)
PC5	TDI (JTAG Test Data Input) PCINT21 (Pin Change Interrupt 21)
PC4	TDO (JTAG Test Data Output) PCINT20 (Pin Change Interrupt 20)
PC3	TMS (JTAG Test Mode Select) PCINT19 (Pin Change Interrupt 19)
PC2	TCK (JTAG Test Clock) PCINT18 (Pin Change Interrupt 18)
PC1	SDA (2-wire Serial Bus Data Input/Output Line) PCINT17 (Pin Change Interrupt 17)
PC0	SCL (2-wire Serial Bus Clock Line) PCINT16 (Pin Change Interrupt 16)

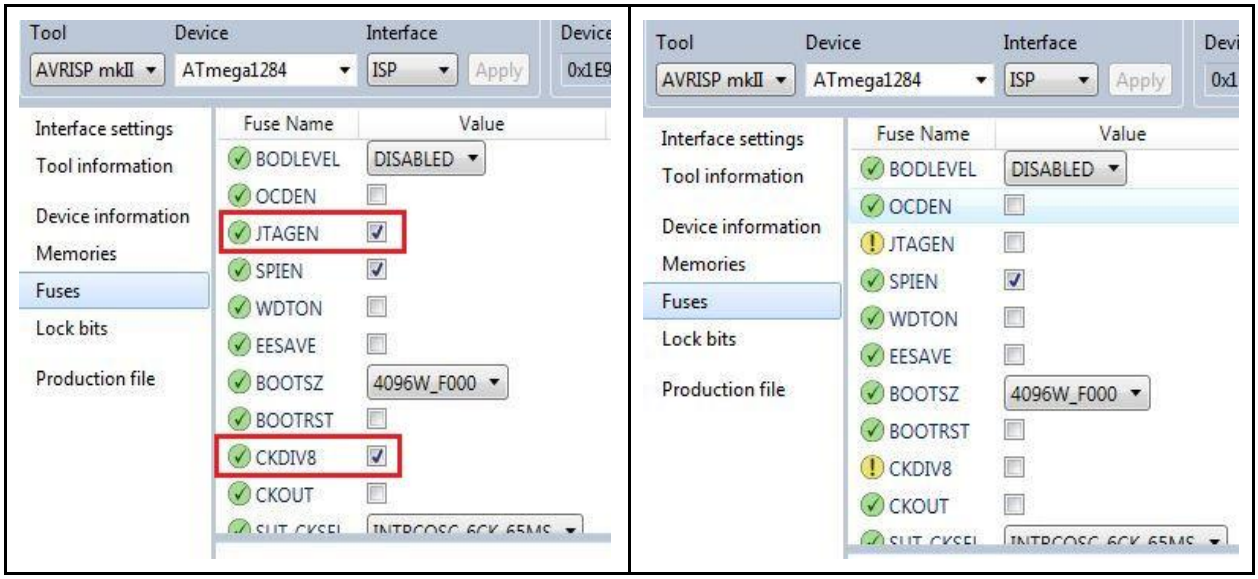
Pinout

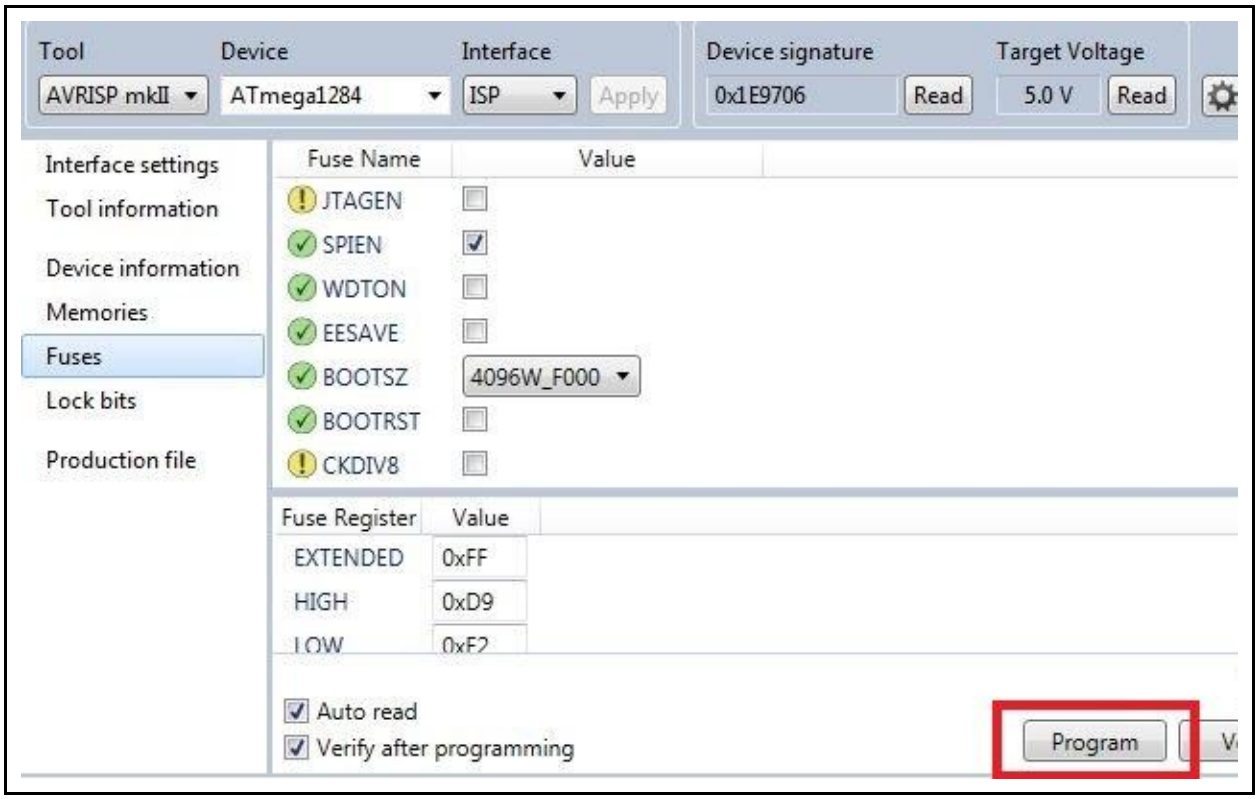
PDIP



"JTAG" is a standard serial interface for advanced chip testing, which we won't be using. Fuses internal to the chip determine which purpose is active for each port. For PORTC, AVR Studio defaults to JTAG being active. If we wish to use port C, when configuring chip programming we must go to the Fuses section, deselect the JTAGEN box, and press Program -- check that the LEDs on PORTC now behave as desired.

Also, when programming the microcontroller, there is an option to divide the clock speed (8Mhz) by 8. In later labs, we will be implementing our own timer and this option will interfere with that. So, make sure to uncheck the “CKDIV8” option in the fuses section to prevent future problems.




Tool
Device
Interface
Device signature
Target Voltage
AVRISP mkII
ATmega1284
ISP
Apply
0x1E9706
Read
5.0 V
Read
Interface settings
Tool information
Device information
Memories
Fuses
Lock bits
Production file
Fuse Name
Value
JTAGEN
SPIEN
WDTON
EESAVE
BOOTSZ
4096W_F000
BOOTRST
CKDIV8
Fuse Register
Value
EXTENDED
0xFF
HIGH
0xD9
LOW
0xE2
Auto read
Verify after programming
Program

Pre-lab

Be sure that you have all needed supplies, ready for use. Read over the entire lab so you'll know what to expect. For credit you need to at least complete through section: "Prepare the AVR In-System Programming (ISP) header", so your header should be ready to plug into the board. You should also attempt to run the basic programs from above as well.

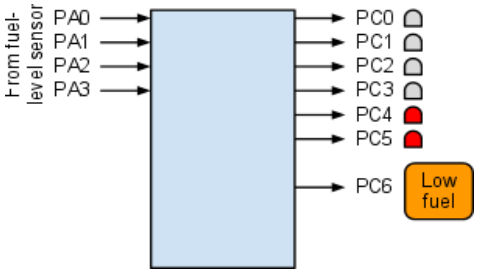
Exercises

Write C programs for the following exercises using AVR Studio for an ATmega1284 following the PES *standard technique*. These are similar to previous exercises you have run in the simulator. For any behavior response caused by a button press, the response should occur almost immediately upon the press, not waiting for the button release (unless otherwise stated). Be sure to count each button press only once, no matter the duration the button is pressed.

When completing these exercises, chances are that a few bugs will be encountered along the way. LEDs are a helpful component for debugging hardware. LEDs can be used to check the input/output of individual pins. For example, If a specific pin is meant to be a certain value in order for your program to proceed, connect an LED to the pin to ensure the pin is the correct value. A helpful hint is to always keep an extra LED connected between ground and a long wire. Whenever you wish to check the value of a desired pin, just connect the other end of the long wire to the desired pin and observe the behavior of the LED.	

It is also recommended that you review the AVR FAQ (on iLearn under 'Assignments') to help you with common debugging issues when working with hardware.

1. (From an earlier lab) A car has a fuel-level sensor that sets PA3..PA0 to a value between 0 (empty) and 15 (full). A series of LEDs connected to PB5..PB0 should light to graphically indicate the fuel level. If the fuel level is 1 or 2, PB5 lights. If the level is 3 or 4, PB5 and PB4 light. 5-6 lights PB5..PB3. 7-9 lights PB5..PB2. 10-12 lights PB5..PB1. 13-15 lights PB5..PB0. Also, PB6 connects to a "Low fuel" icon, which should light if the level is 4 or less. Use buttons on PA3..PA0 and mimic the fuel-level sensor with presses.



Video Demonstration:
http://www.youtube.com/watch?v=_6BjqDXpdVk&feature=youtu.be

2. (From an earlier lab) Buttons are connected to PA0 and PA1. The output for PORTB is initially 0. Pressing PA0 increments PORTB (stopping at 9). Pressing PA1 decrements PORTB (stopping at 0). If both buttons are depressed (even if not initially simultaneously), PORTB resets to 0. If a reset occurs, both buttons should be fully released before additional increments or decrements are allowed to happen. Use LEDs (and resistors) on PORTB. Use a state machine (*not* synchronous) captured in C.

   Note: Make sure that one button press causes only one increment or decrement respectively. Pressing and holding a button should **NOT** continually increment or decrement the counter.

   Note: This state machine (increment/decrement) will be reused several times in subsequent labs.

   Video Demonstration: **http://youtu.be/a0cX_cjr5t0**

Challenge Problem

3. Create your own festive lights display with 6 LEDs connected to port PB5..PB0, lighting in some attractive sequence. Pressing the button on PA0 changes the lights to the next configuration in the sequence. Use a state machine (not synchronous) captured in C.

   Video Demonstration: **http://youtu.be/ceOSKTxOP74**

All students must submit the group's .c source files according to instructions in the lab submission guidelines. Post any questions or problems you encounter to the wiki and discussion boards on iLearn.