Todd Danko
January 28, 2004

Atmel AVR Tutorial

Introduction

The purpose of this tutorial is to describe step by step, the process of coding, compiling, and uploading programs to Atmel AVR microcontrollers.

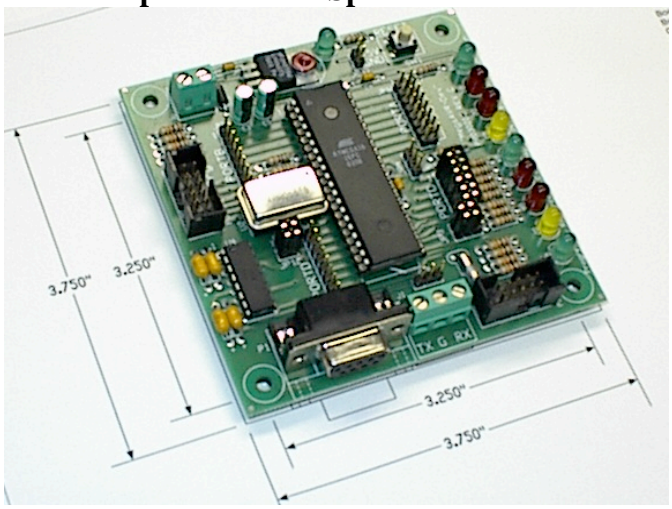
What is needed?

Before beginning, several items are required:

Description	Part Identifier	Source	Price
Development board with ATMega32 microcontroller	The MegaAVR-Dev w/ATMega32	www.prllc.com	$56
Chip Programmer Software and Hardware	The CableAVR ISP Programmer	www.prllc.com	$99
AVR Integrated Development Environment	CodeVisionAVR	www.hpinfotech.ro	Free Evaluation
Serial Cable (RS-232C)	26-117	www.radioshack.com	$14.49
Power Supply*	273-1776	www.radioshack.com	$16.99
Text Book	Embedded C Programming and the Atmel AVR	www.prllc.com	$67.95

* The power supply may be purchased, but any AC to DC converter that outputs 8 to 38 Volts DC, and greater than 1 Amp (1000mA) may be salvaged from unused household appliances (answering machine, modem, etc).

Microcontroller and Development Board Specifications



The ATMega32 AVR microcontroller is a 16 bit RISC processor. When combined with the development board, many features are offered:

- 4, 8 bit Digital I/O Ports (PORTA, PORTB, PORTC, PORTD)
- RS232 communications through:
    - 9-Pin D-Shell
    - Screw terminals
    - Jumper header
- 32K of In-System Programmable FLASH memory
- 1K of EEPROM
- 1K of Internal RAM
- 8, 10 bit, Analog Inputs
    - Either internal or external reference voltage
- 9 I/O controlled LEDs
    - 8 LEDs are jumper selectable.
- 32KHz watch crystal
- Universal clock socket with 6MHz crystal
- 0.1" centered headers
- 10-pin, polarized, ISP and JTAG connectors
- Requires an 8-38VDC power supply

System Overview

The overall process to program an AVR microcontroller is as follows:

1. Define your objective keeping in mind how you will use hardware and software
2. Decide what hardware will be used, and in what configuration
3. Write software that manipulates your hardware appropriately
4. Compile / Assemble software
5. Connect computer to chip
6. Program chip
7. Attach hardware to chip if necessary
8. Run it!

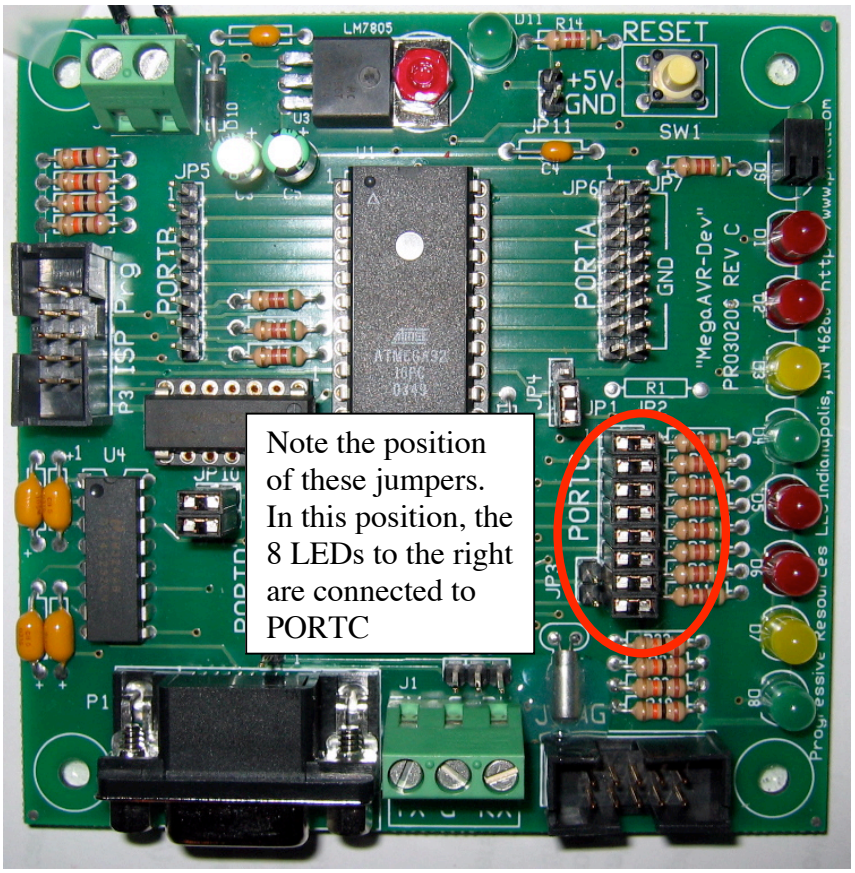
Where to begin…

We will begin by accomplishing three tasks. The first task will be as simple as possible, and each subsequent task will add complexity.

1. Light up an LED on the development board.
2. Reads in a digital word from an eight dipswitch array, and light up corresponding LEDs on an eight LED array.
3. Light up LEDs to count from 0 to 255 in binary, pausing two seconds between each step.

TASK 1:

For the first task, we will use an LED that is attached to the development board. Set the jumpers for PORTC so that they connect PORTC to the array of LEDs (Circled in red)



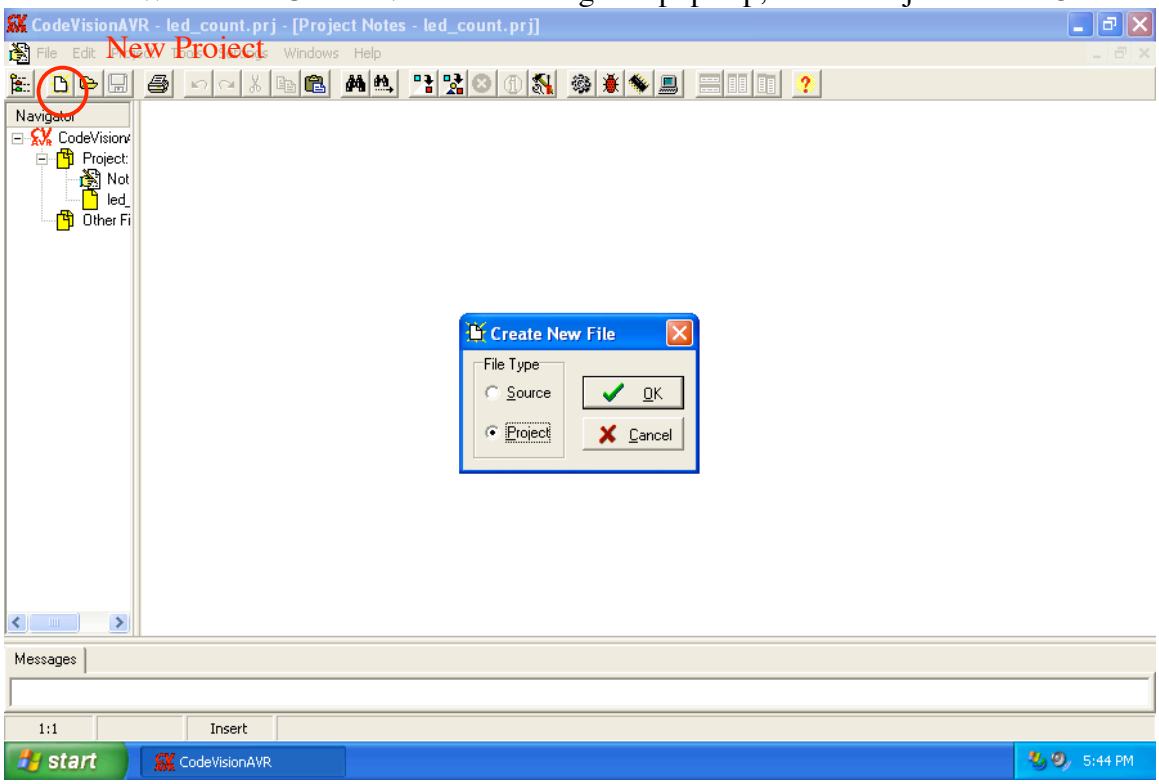
The configuration of the development board is such that +5 volts is connected through a resistor to one leg of each LED, and the ground for each LED is attached to a pin on PORTC. This means that when a pin on PORTC is set high (+5 volts), there is no current flowing through the LED and the LED is off. When a pin on PORTC is set low (ground), current flows through the LED, and the led is on. So, for this setup, turning a pin off, turns the LED on.

So, how do we tell the microcontroller to turn a pin of PORTC off (thus turning a single LED on)? A port can be thought of as an eight bit variable. When you assign a value between 0 and 255 to this eight bit variable, each pin will be set to either on or off (+5 V or Ground). The +5V corresponds to a 1 and the ground corresponds to a 0. So, if we want to make one LED turn on, and the rest stay off, we want to send the number 254 to PORTC. The number 254 in binary is 01111111, so pin0 on PORTC will be ground, and pin1 – pin7 will be +5 V. This means that the LED attached to pin0 will be on, and the rest will be off.

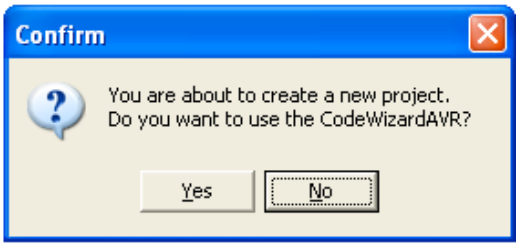
Lets get started:

- Open up CodeVisionAVR on your PC.
- Click on the “New Project” icon to create a new project.

- When the “Create New File” dialog box pops up, click “Project” then “OK.”



- A dialog box titled “Confirm” will pop up asking if you would like to use “CodeWizardAVR.” This is a helpful tool, but it is important to learn to code with out this wizard first. Click “No.”

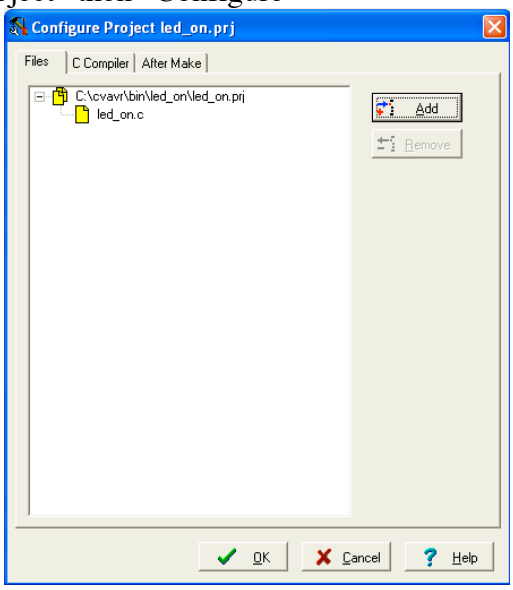


- Save the project to an appropriate file name when prompted
- Click on the “New Project” icon again
- This time, check “Source” and then click “OK”
- Enter your code in the source code window:

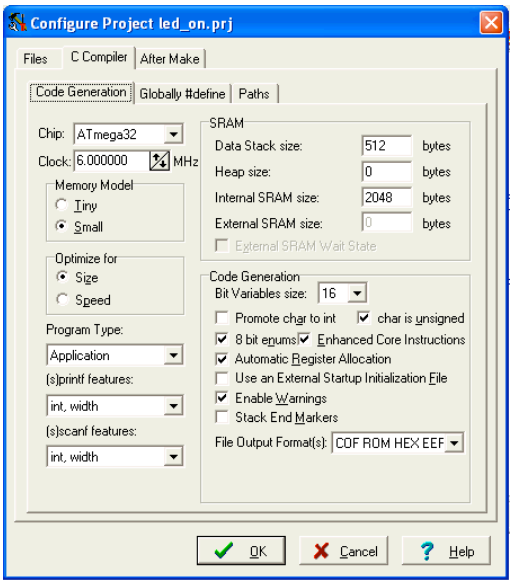
```
/*****************************************
Project : led_on
Comments: This program lights up the LED attached to pin0 or PORTC
*****************************************/
#include <mega32.h>

void main(void)
{
PORTC=0xFF;             //Set all pins of PORTC  to output (p.87 in textbook)
DDRC=0xFF;              //Set initial values for PORTC to high

while (1)
```

```
      {
      PORTC = 254;          //pin0, PORTC = 0 (LED ON), pin1-7=1 (LEDs OFF)
      };
}
```

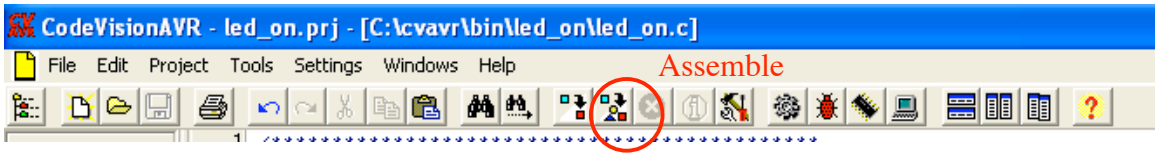
- On the toolbar, click “File” then “Save” to save your source code. Save your source code to an appropriate file name.
- The source code must now be associated with the project:
  - Click “Project” then “Configure”



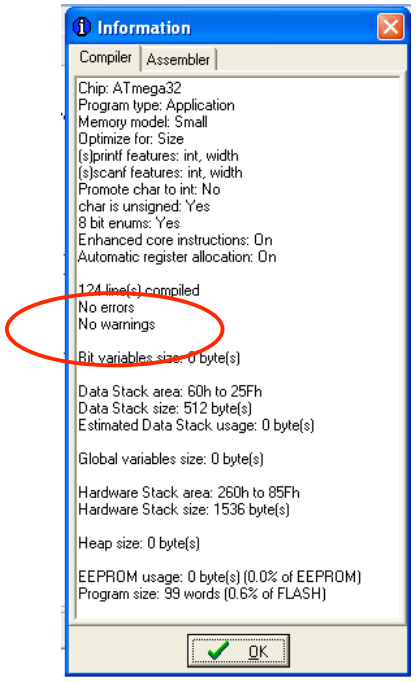
  - Click Add on the “Configure Project” window
  - Select your source code, then click open.
  - Now, click the second tab (C Compiler) of the “Configure Project” window



  - Select the chip type, and clock speed that is appropriate for your chip. Click OK
- Now compile and assemble your code by clicking on the “Assemble” button.



- A dialog box appears. Make sure that the message says "No errors," otherwise, go back to your code and fix the errors.



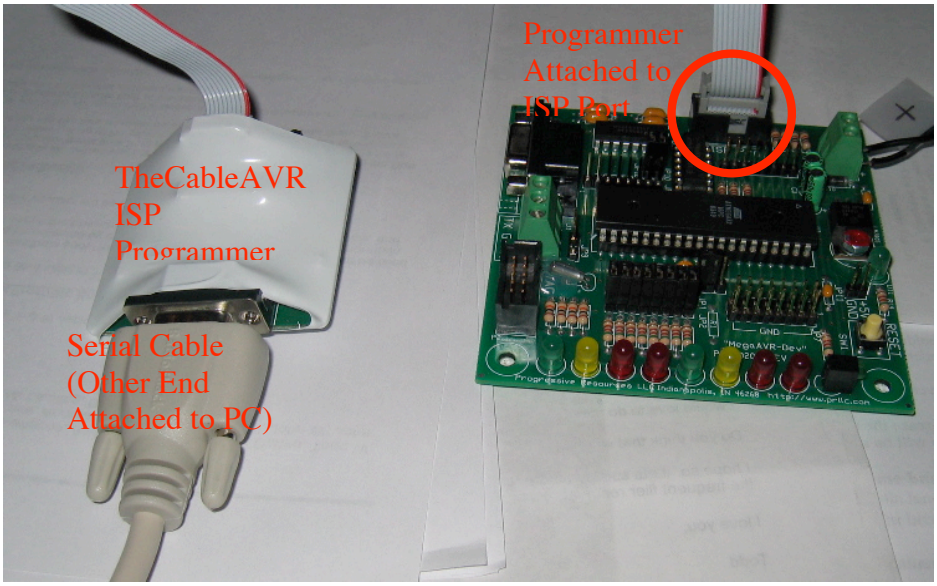
- If there are no errors in the compilation, it is time to program the chip.

Programming the chip

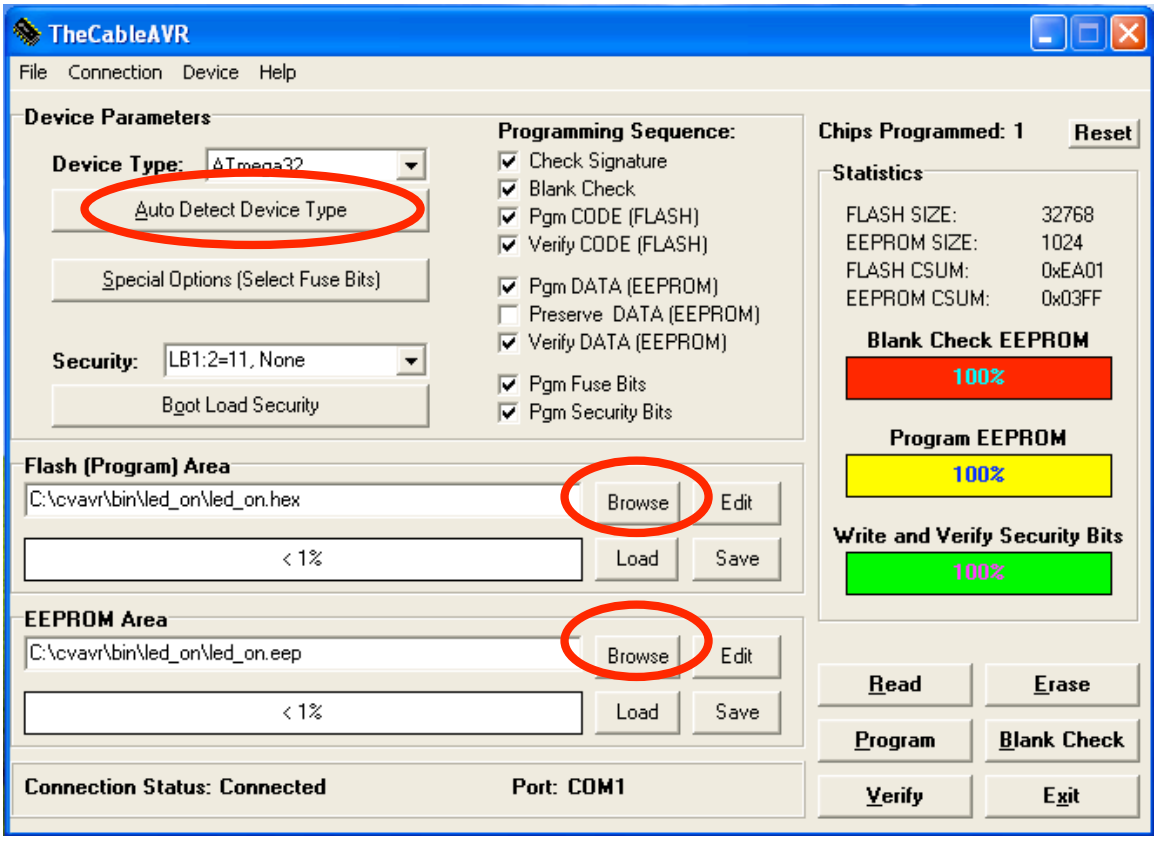
There are several tools available to program your AVR chip. One of the more reliable and easy to use tools is "TheCableAVR ISP Programmer."

Using "The CableAVR ISP Programmer:"

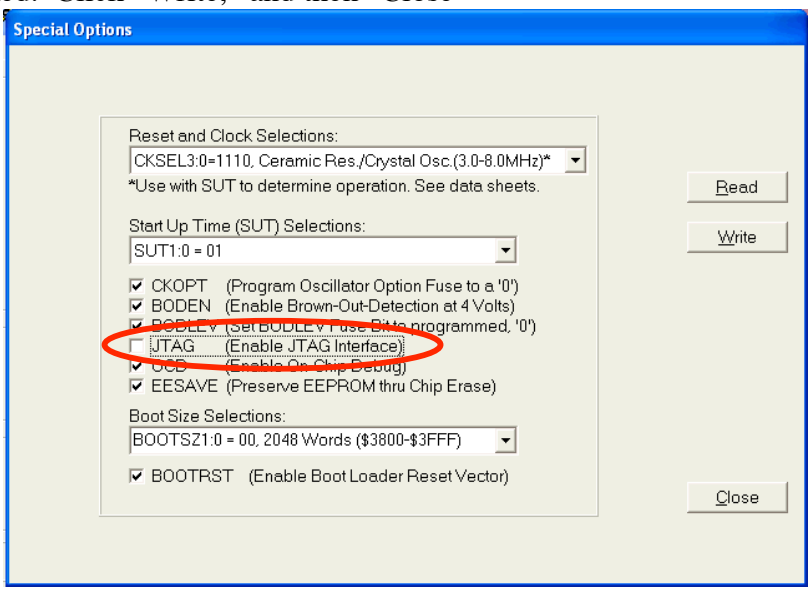
- Compile and assemble your code using CodeVisionAVR. At least two files will be created:
  - Flash program (.hex)
  - EEPROM (.eep)
- Connect the cableAVR tool to an open com port on your PC through a serial cable.
- Connect the cableAVR tool to your development board through the ISP port.



- Power up your development board.
- Open "TheCableAVR" tool.
- Click on "Auto Detect Device Type." This detects the type of chip, and automatically populates the chip specific values.



- Click “Special Options,” and make sure that the “Enable JTAG” box is not checked. Click “Write,” and then “Close"

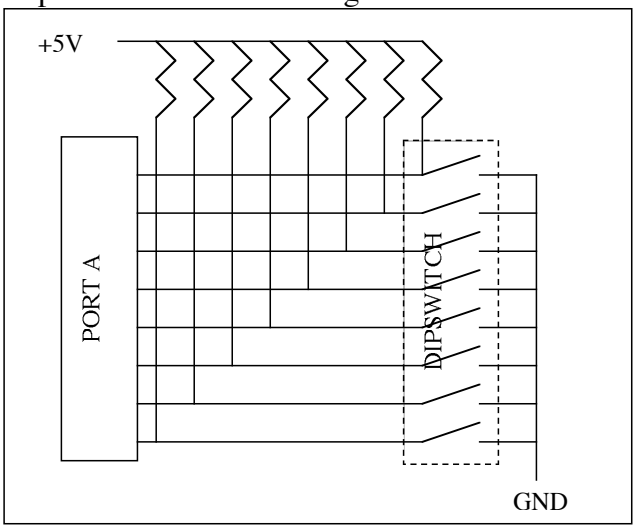


- Click “Browse” under “Flash(Program) Area,” and select your assembled (.hex) file
- Click “Browse” under “EEPROM Area,” and select your EEPROM (.eep) file
- Click on “Program” to program your chip
    - The chip is now programmed

Once the chip is programmed, you will see that the LED attached to pin0 on PORTC is on. The rest of the LEDs on PORTC are off.

TASK 2:
The next task is to read an eight bit dipswitch in from PORTA, and light up corresponding LEDs in an eight LED array.

We must construct a dipswitch circuit following the schematic below:

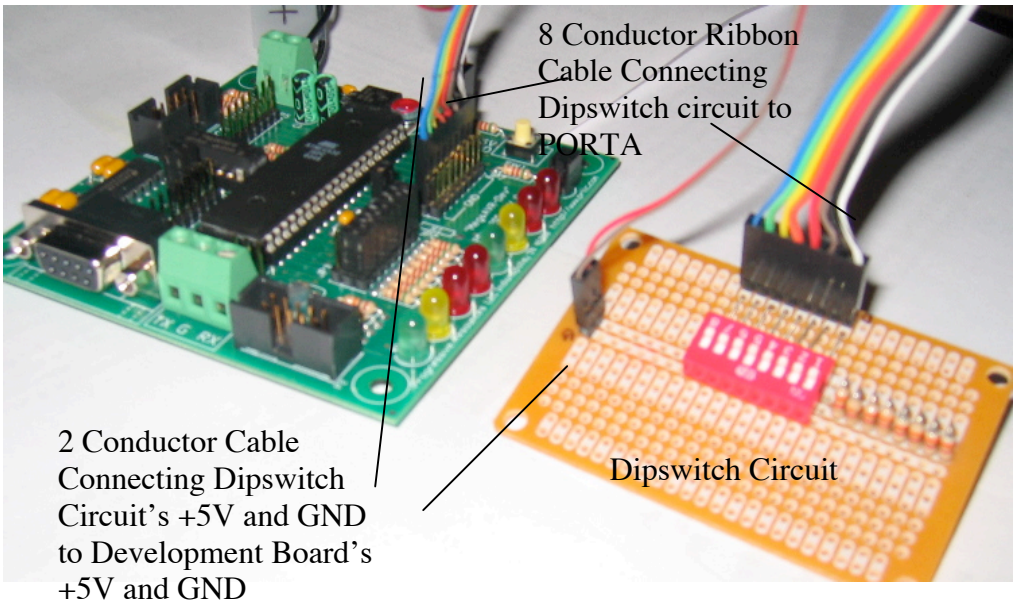


This dipswitch circuit works similar to the LEDs on the development board. When a dipswitch is closed, the corresponding pin on PORTA is brought low. When a dipswitch is open, the corresponding pin on PORTA is brought high. If we write a program that maps the value from PORTA to PORTC, we will have a circuit where closing a switch attached to PORTA lights up the corresponding LED attached to PORTC.

The code to do this follows:

```
/*****************************************
Project : dip_led
Comments: This program reads an eight bit digital word from PORTA, and writes
the digital word out to PORTC.
*****************************************/

#include <mega32.h>

void main(void)
{

PORTA=0x00;                 // Set all pins of PORTA to input
DDRA=0x00;                  // Set initial values for PORTA to low

PORTC=0xFF;                 //Set all pins of PORTC  to output
DDRC=0xFF;                  //Set initial values for PORTC to high

while (1)
      {
      PORTC = PINA;         //Input from PORTA is output to PORTC
      };
}
```

Assemble this code and program your chip just like with the led_on program. Attach your dipswitch circuit to PORTA on your development board and see what happens when you toggle the dipswitches.



TASK 3:

The final task is a bit more complex. The object is to count from 0 to 255 in binary through the LED lights on the development board. The counter should operate such that the count is incremented once every two seconds.

We will use the eight LED array on the development board as an output for this project. Keep in mind that setting a pin of PORTC high turns the corresponding LED off, so to make it look like the LEDs are counting from 0 to 255, we will make the digital word sent to PORTC count down from 255 to 0.

How can you time something so long as two seconds when your chip's clock is going at 6MHz? The simple way would be to have a counter that increments once per cycle. Once the count reaches 12,000,000, you can assume that two seconds have passed. Of course, if you do this, you will not really be able to use your processor for anything else. You would also need to count to 12,000,000. The ATMega32 stores variables in 16 bit space making it difficult to store numbers greater than 65,525.

Fortunately, the ATMega32 AVR chip has three timers built into it. The first timer (timer0) is used in this example.

Part of timer0's built in functionality is a user selectable prescaler. This prescaler essentially counts the number of cycles the processor goes through, and allows a signal to be sent every 1024, 512, 256, 128 (etc) cycles. I chose to set the prescaler to 1024. This means that a signal is sent every 6Mhz / 1024 cycles or at a rate of 5859Hz. 1/5859Hz gives us 0.000171 seconds between each signal.

So, what do we do with these signals? Timer0 has a neat feature where the timer internally counts the scaled signals. The counter used is an eight bit counter, so it has a range from 0 to 255. Once 255 is reached, the next count value rolls over to 0. This event is called a "rollover." Every time a rollover occurs, an interrupt signal is sent. When an interrupt signal is sent, a counter counts how many interrupts have occurred. If the timer's internal counter is allowed to count from 0 to 255 (256 steps), there will be a delay of 0.000171 * 256 = 0.0438 seconds between each interrupt. This is not a very useful time frame. Fortunately, we can choose what number to start the timer's internal counter on. In the code, you will notice that the timer's internal counter starts at 110 so, counting from 110 to 255 gives us 146 counts between each interrupt. That is 0.000171 * 146 = 0.025 seconds between each interrupt. This is a pretty convenient value for measuring a period of two seconds.

Now that we have an interrupt being transmitted every 0.025 seconds, we just need to count 80 of these interrupts to know that two seconds have passed. That is exactly what "timecount" does.

For more information on timers, please see page 105 in the textbook.

```
/*****************************************
Project : led_count
Comments: This program counts from 0 to 255 in binary with LEDs.  There is a
two second pause between each count.
*****************************************/

#include <mega32.h>

unsigned int timecount = 0;         // Global timer counter
unsigned int counter = 255;         // Global counter set

// When Timer 0 overflows, conduct this routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0 = 110;      // 146 clock cycles equals 0.025 seconds.  Setting
                  // the clock counter to 110 allows 146 clock cycles
                  // (256-146=110)

   if (++timecount == 80)      // 80 250us ticks in two seconds
   {
      PORTC = --counter;       // Subtract one from counter and send it to
                               // PORTC every two seconds

                               // Note: high is off and low is on for each LED

      timecount = 0;           // Reset time counter
      }
 }

void main(void)
{

PORTC=0xFF;                    // Set all eight bits of PORTC to output
DDRC=0xFF;                     // Set all eight bits of PORTC High

TCCR0=0x05;                    // Clock value: 5.859 kHz (6MHz / 1024)
TCNT0=110;                     // Timer/Counter 0 initialization
TIMSK=0x01;                    // Timer Interrupt initialization

#asm("sei")                    // Global enable interrupts

while (1)
      ;                        // Do Nothing
}                              // End of main
```

Assemble this code and program your chip. The count should begin as soon as programming is complete.

Press the reset button on the development board to restart the count and start a stopwatch to see how long it takes to count to 255. You will find that this type of timer is very accurate.