

ECED3901 Lab #2: Software Controller

Lab Day: June 1, 2015

Lab Due: June 8, 2015 @ 12:30 PM - Submitted via BBLearn Website (PDF files only), OR printed files in 3901 Mail-Slot at ECED Office

Lab Objective

The goals of this lab are to:

1. Learn about programming the ATmega644P Board from Atmel Studio
2. Setup a printf() and scanf() system for interacting with your robot from your computer
3. Explore a simple processing loop, sending data over printf()

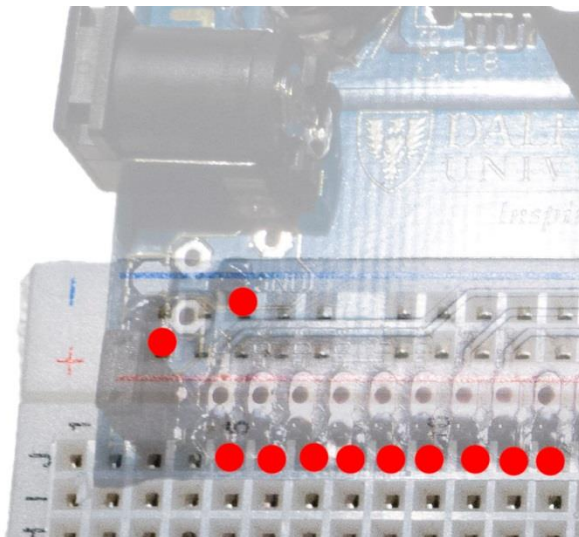
Part 1: Basic Board Setup

This part of the lab assumes you know how to use AVRStudio to build a basic program. Please follow the lab instructions at

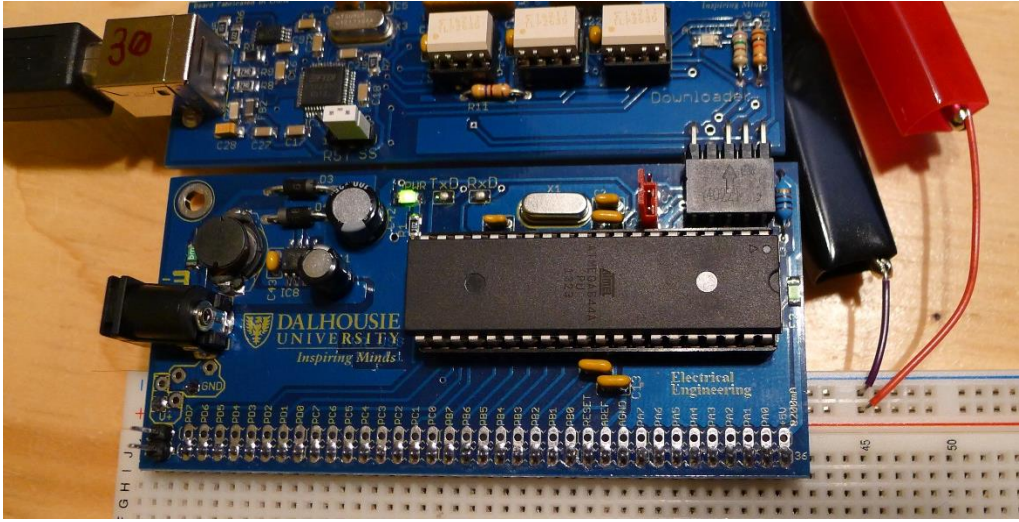
https://github.com/colinoflynn/eced3204/blob/master/labs/lab1_introduction/LAB1.pdf to learn how this works.

Follow Part 1 only (and ignore the Lab Questions too). Two additional notes to help you:

- 1) The following shows how the AVR board plugs into the breadboard:



- 2) You can power the board by applying around 10V onto the breadboard power rails. This might be easier than using the plug-in jack. You should have suitable clips in your parts kit, as an example this shows powering up the board:



Part 2: Printf() Statements

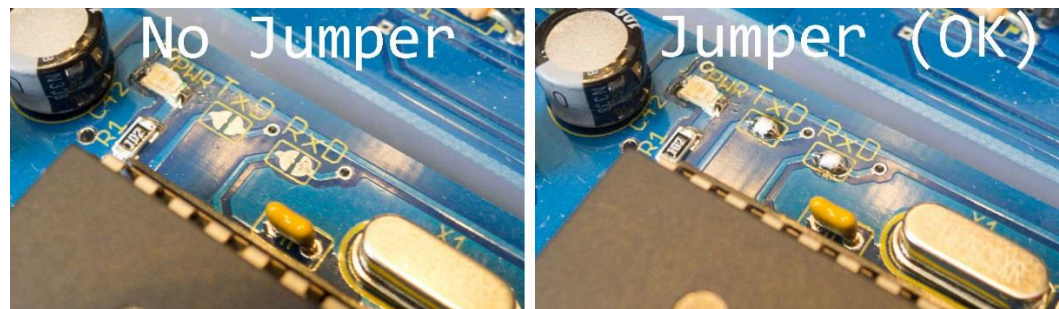
NOTE: This lab is based on my ECED3204 Lab #5. There is an associated video for that lab which is available at <https://www.youtube.com/watch?v=9vljRX5im9o&hd=1> . This video will take you through setup of the serial port, PuTTY terminal, etc. It might be much more convenient than these text-based instructions!

You can also see the full ECED3204 Lab #5 at https://github.com/colinoflynn/eced3204/blob/master/labs/lab5_uart/LAB5.pdf .

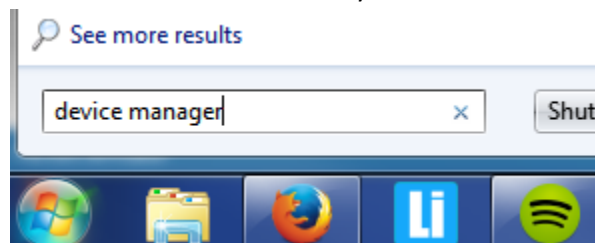
Environment Setup

You will require some setup of your device before being able to work with this lab.

1. Ensure the RXD and TXD jumpers have been bridged with solder. You can look at your board, and if not bridged see the lab technicians to have this fixed:



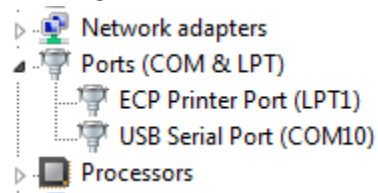
2. The computer you have plugged your 'programmer' board into needs to be configured to use a USB-Serial converter. You can check if this was already done by opening the device manager (NB: the video link above also shows you how to do this which might be easier to follow):



Without the board plugged in, there is no serial ports (you might see a few here too, but we want to look for a change):

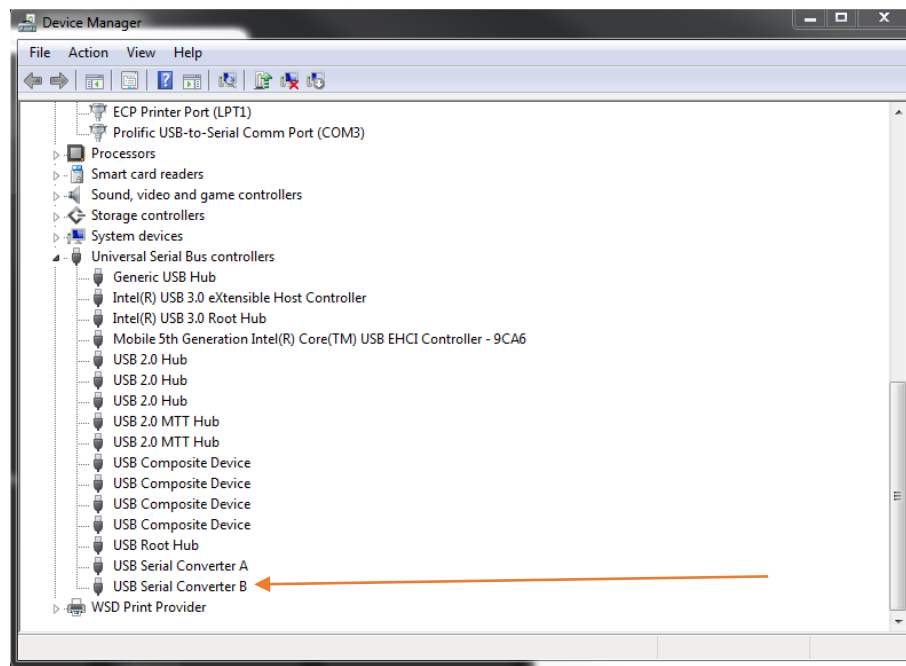


Plugging the 'Programmer Board' into the USB connector gives me a *new* "USB Serial Port":

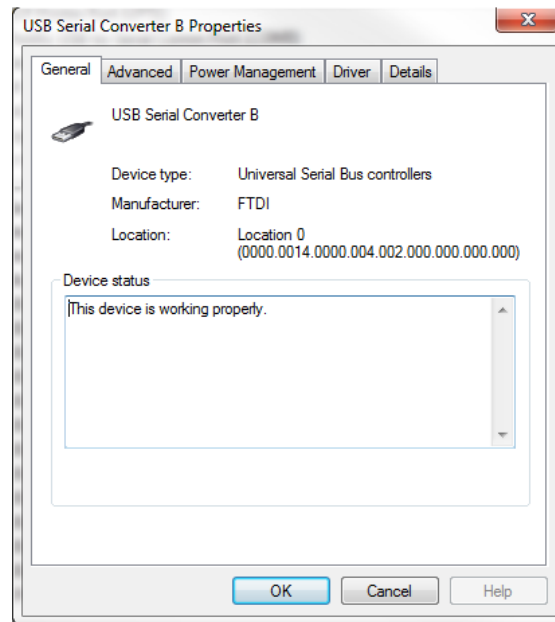


If this does not happen, perform the following steps. If you **DO** get the USB Serial Port, ignore these steps:

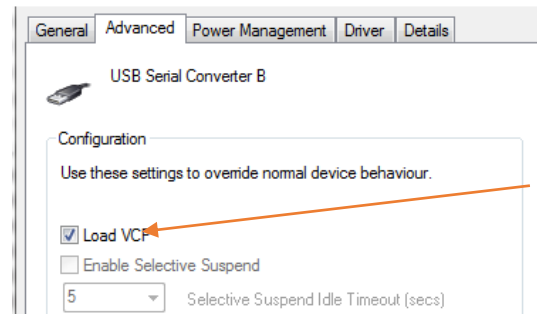
- a) Expand the "Universal Serial Bus Controllers" group, and find the "**USB Serial Converter B**" device (NOT the A device):



- b) Right-click and select 'Properties':



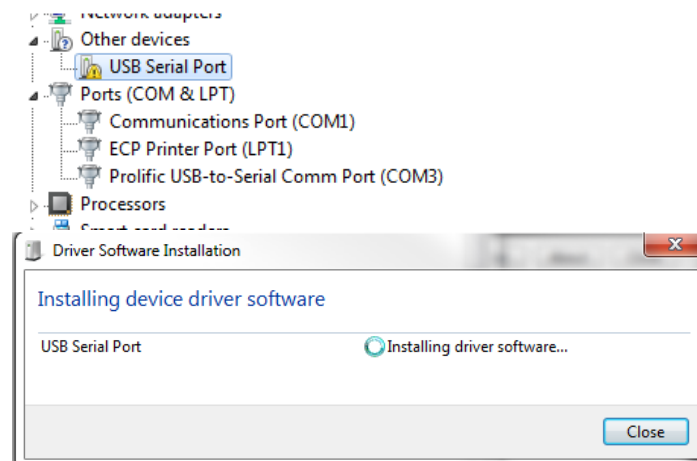
c) Under the 'Advanced' tab, select "Load VCP":



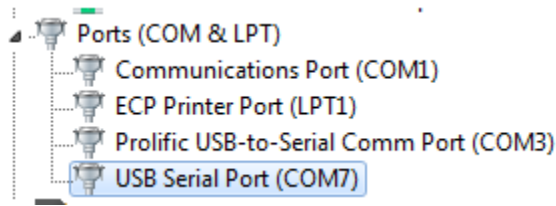
d) Hit "OK".

e) Unplug and replug the USB cable to force the drivers to reload.

f) Wait for device driver installation to finalize:



g) Confirm the USB-Serial is now appearing.



Procedure

1. Make a new Atmel Studio project. Enter the following code and ensure the system builds OK (i.e. it compiles without errors).

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

static int uart_putchar(char c, FILE *stream);
static int uart_getchar(FILE *stream);
FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);
FILE mystdin = FDEV_SETUP_STREAM(NULL, uart_getchar, _FDEV_SETUP_READ);

static int uart_putchar(char c, FILE *stream)
{
    loop_until_bit_is_set(UCSR0A, UDRE0);
    UDR0 = c;
    return 0;
}

static int uart_getchar(FILE *stream)
{
    loop_until_bit_is_set(UCSR0A, RXC0); /* Wait until data exists. */
    return UDR0;
}

void init_uart(void)
{
    UCSRB = (1<<RXEN0) | (1<<TXEN0);
    UBRR0 = 7;
    stdout = &mystdout;
    stdin = &mystdin;
}

int main(void)
{
    init_uart();
    printf_P(PSTR("System Booted, built %s on %s\n"), __TIME__, __DATE__);

    printf_P(PSTR("Hello There. What is your name?\n"));

    char name[32];

    scanf("%s", name);

    printf_P(PSTR("Alright %s. What is a number?\n"), name);

    int number;
```

```

if (scanf("%d", &number) == 1){
    printf_P(PSTR("OK %s, did you pick %d?\n"), name, number);
} else {
    printf_P(PSTR("Look %s - I said a number. Try again.\n"));
}
}

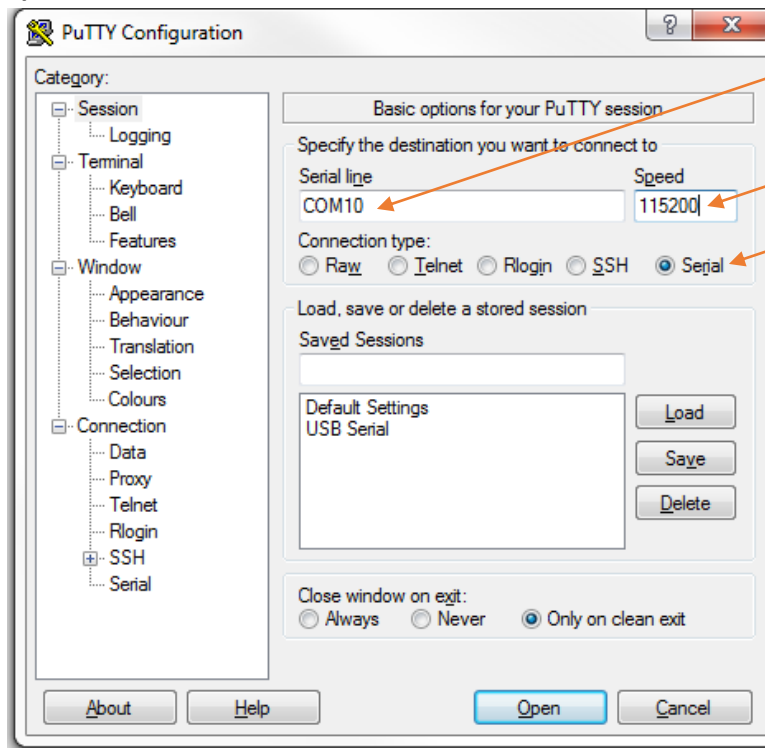
```

2. Before continuing, ensure you have performed the required lab setup from the 'prelab' section. This involves soldering your RXD/TXD jumpers and ensuring the proper serial port driver is loaded.

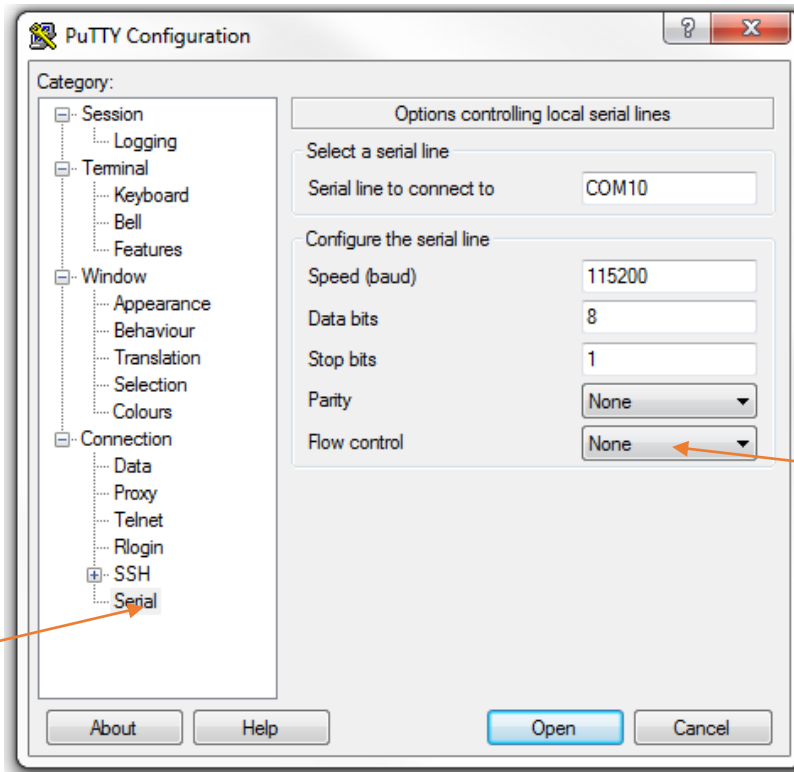
You will also need to know the COM port number. See the device manager, for example I will be using COM10:



3. Open **PuTTY** (type this in the start menu), change the following settings on the main page:
 - a) **Connection Type = Serial**
 - b) **Serial line = COM10 (on my computer, set as required on your device)**
 - c) **Speed = 115200**

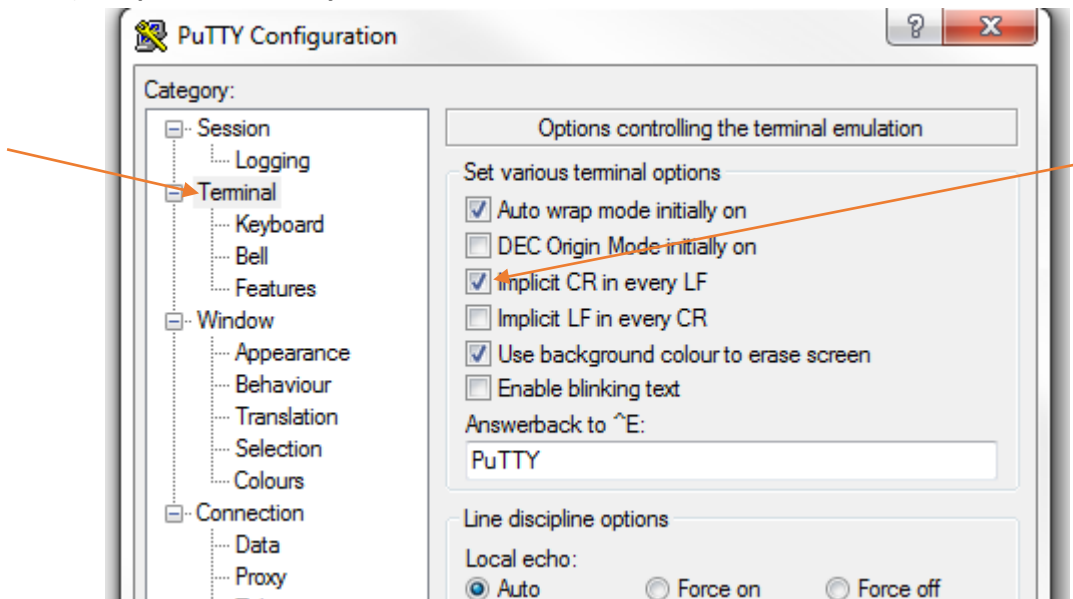


4. Under the Connection→Serial setup, change the following settings:
 - d) **Flow Control: None**



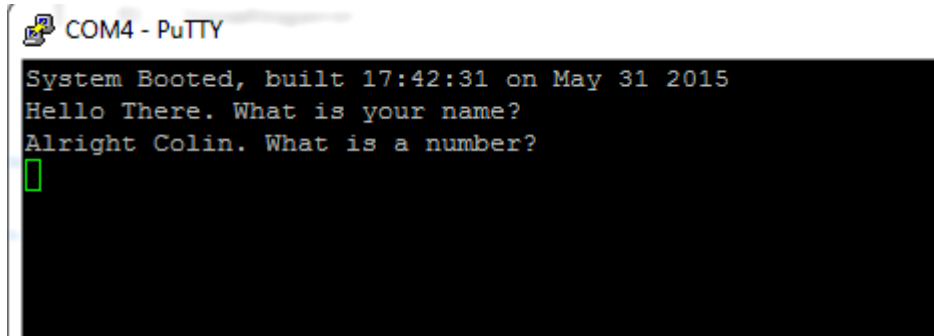
5. Under **Terminal**, change the following settings:

e) Implicit CR in every LF: Checked



6. Hit “Open” to open the serial port. Note you can save the connection as well before hitting Open to avoid you needing to set this all up again each time.

7. Program the AVR device with your code. Check in PuTTY prints a welcome banner, along with allowing you to interact with the program:



```
COM4 - PuTTY
System Booted, built 17:42:31 on May 31 2015
Hello There. What is your name?
Alright Colin. What is a number?
█
```

Note right now it *does not* echo back typed characters. When I am typing my name above nothing is displayed until I hit enter.

Also note it prints the time and date you compiled the code – this can be very useful to ensure you are using the most recent version of your .HEX file!

8. This example code is using the “program memory” macro called PSTR along with special printf_P() calls, which stores data to FLASH memory instead of wasting storage in SRAM.

As a test, change the main routine to instead use regular printf() statements as below:

```
int main(void)
{
    init_uart();
    printf("System Booted, built %s on %s\n", __TIME__, __DATE__);

    printf("Hello There. What is your name?\n");

    char name[32];

    scanf("%s", name);

    printf("Alright %s. What is a number?\n", name);

    int number;

    if (scanf("%d", &number) == 1){
        printf("OK %s, did you pick %d?\n", name, number);
    } else {
        printf("Look %s - I said a number. Try again.\n");
    }
}
```

Record the usage of SRAM and FLASH across each version. The SRAM and FLASH usage is printed when Atmel Studio compiles your code. Switch to the ‘Output’ tab and scroll up slightly, for example something like this is printed, which shows Program Memory (FLASH) usage and Data Memory (SRAM) usage:

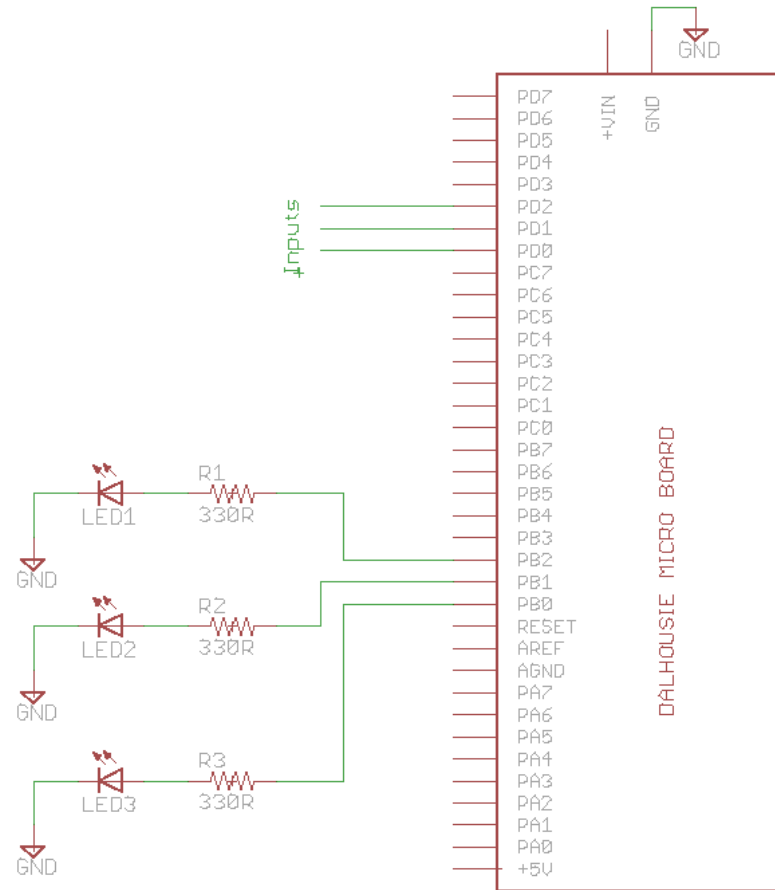
```
Done executing task "RunCompilerTask".
Using "RunOutputFileVerifyTask" task from assembly "C:\Program Files (x86)\Atmel\
Task "RunOutputFileVerifyTask"
    Program Memory Usage      :   1990 bytes   3.0 % Full
    Data Memory Usage         :    72 bytes   1.8 % Full
Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "Lab4.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '')
Target "Build" in file "C:\Program Files (x86)\Atmel\Atmel Studio 6.2\Vs\Avr\common.t
```

Error List Output Find Results 1

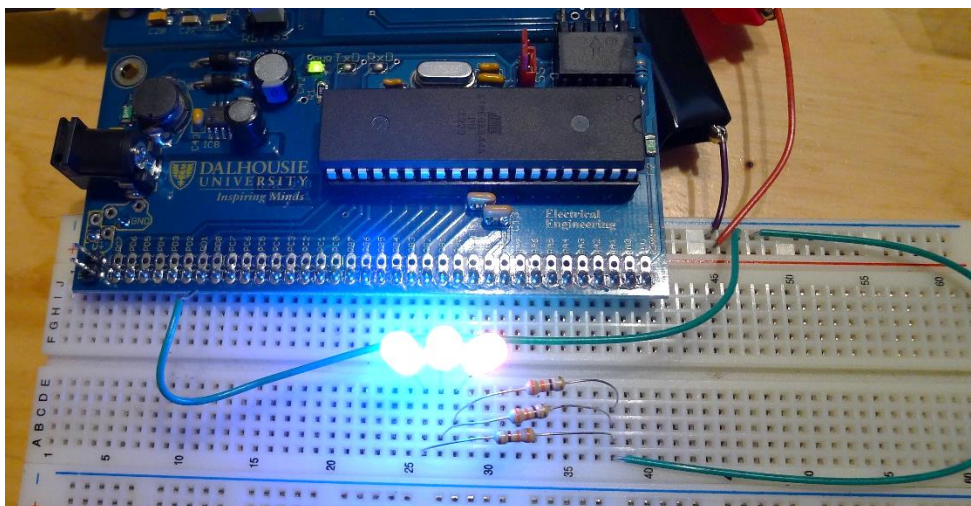
In this example the compiled program used 1990 bytes of Program Memory and 72 bytes of Data Memory. Also note the AVR has a limited amount of both – the program here is only 3% full of program memory for example. If you ever reach 100% things will stop working!

Part 3: Simple Processing

1. Connect LEDs (along with current-limiting resistors) to PB0, PB1, and PB2 as in the following schematic:



As an example your system might look like this (in this case the LEDs are on, yours should be off):



2. We want to program some code which plays a game with the user. The game does the following:
 - a. Turns on all LEDs, and waits for the user to ground pins PD0, PD1, and PD2
 - b. Turns on a single LED randomly.
 - c. Waits for the user to ground the corresponding input pin (i.e. if LED on PB0 is on, the user must ground PD0).
 - d. Displays a new LED randomly, and again waits for the user to ground the corresponding pin.
 - e. Repeats the pattern...

This will be accomplished with a state machine, and also making use of the *enumerated type* in C. You can replace the main() function of the previous code with the following example:

```
typedef enum {
    state_init,
    state_wait,
    state_ledred,
    state_ledgreen,
    state_ledblue
} states_t;

states_t random_state(void) {
    int randtest = rand() % 3;

    if (randtest == 0) return state_ledred;
    if (randtest == 1) return state_ledgreen;
    return state_ledblue;
}

int main(void)
{
    init_uart();
    printf_P(PSTR("System Booted, built %s on %s\n"), __TIME__, __DATE__);

    //PORTD is all inputs, pull-ups enabled
    DDRD = 0;
    PORTD = 0xFF;

    //PORTB 0,1,2 is outputs. Example of _BV() macro
    DDRB |= _BV(0) | _BV(1) | _BV(???);
    //Turn on LEDs
    PORTB |= _BV(0) | _BV(1) | ???

    //State machine
    states_t state = state_init;
    states_t oldstate = state_init;

    while(1){
        //Initial state - nothing done, transition to waiting
        if (state == state_init){
            state = state_wait;

            //Wait for all lines to go low
        } else if (state == state_wait){
            if ((PIND & (0b0000111)) == 0){
                state = random_state();
            }

        } else if (state == state_ledred){
```

```

        PORTB |= _BV(0);
        //Turn off LED 1 and 2
        PORTB &= ???

        //Check if PD.0 is cleared
        if ((PIND & _BV(0)) == 0){
            state = random_state();
        }

    } else if (state == state_ledgreen){
        PORTB |= _BV(1);
        //Turn off LED 0 and 2
        PORTB &= ~(_BV(0) | _BV(2));

        //Check if PD.1 is cleared
        if ((PIND & _BV(1)) == 0){
            state = random_state();
        }

    } else if (state == state_ledblue){
        PORTB |= _BV(2);
        PORTB &= ~(_BV(0) | _BV(1));

        //Check if PD.2 is cleared
        if ((????) == 0){
            state = random_state();
        }

    }

    if (oldstate != state){
        printf("State Change: %d->%d\n", oldstate, state);
        oldstate = state;
    }

}

return 0;
}

```

3. Some of the sections are marked with ??? indicating you need to fill them in. By inspecting the code you should see how to fill these sections in (noting much of the code is the same), the comments will guide you with what needs to be done.
4. Program the example (ensure the serial console is still running). All three LEDs should be on initially – and then temporarily connect pins PD0, PD1, and PD2 to ground, then remove the connection. This should leave a single LED lit up.
5. Play the game of grounding the corresponding input (i.e. ground PD0 if LED associated with PB0 lights up).
6. You will see printed in the serial console information on the State Changes. Determine the numerical value of each of these states:
 - state_ledred
 - state_ledgreen
 - state_ledblue

Lab Questions

Be sure to include/answer the following:

1. What were the two “Program Memory” and “Data Memory” sizes in Part 2 for using the PSTR() macro and when not using it?
2. What were the numerical values of the three states from Step 6 of Part 3?
3. Include your source code for Part 3.