CG2271 Real Time Operating Systems
Lab 2 – Real Time Architectures

1. Introduction

In the previous lab you learnt how to use Atmel Studio and how to compile the Arduino libraries, as well as how to use the debugging features in Atmel Studio.

In this lab we will build a simple circuit to explore two ways that we can build real-time software, without using an RTOS. We will use a push-button to control reading an analog-to-digital converter channel and use the value read to control the brightness of an LED.

This lab is worth 40 marks for the answerbook at 5 marks for the demo, totaling 45 marks.

2. Circuit for This Lab

i. Wire up a pushbutton at digital pin 2 on the Arduino. This pushbutton should pull pin 2 high when pressed. Pin 2 should be low when the button is not pressed. Use the 10K resistor for pull-down.

ii. In the same way wire up a pushbutton to digital pin 3.

iii. Connect a potentiometer to analog input A0.

iv. Connect one LED to digital pin 6 and one LED to digital pin 7. **Use the 330 ohm resistors to limit current through the LEDs**.

All electronic devices have a tiny amount of magic smoke inserted into them that makes them work. Failing to use a current limiting resistor will cause the magic smoke to overheat, expand too quickly and escape from the device. There is no way to put magic smoke back into a device and the device will be irrecoverably damaged.

In addition your Aduino kit comes with a touch sensor that looks like this:

v. Connect this to Analog Input A1 in the same way that you connect a potentiometer

If you don't know how to use a potentiometer, read this: http://www.arduino.cc/en/Tutorial/Potentiometer. You can also refer to the "electronics.pdf" file that has been uploaded to the lecture notes directory on IVLE.

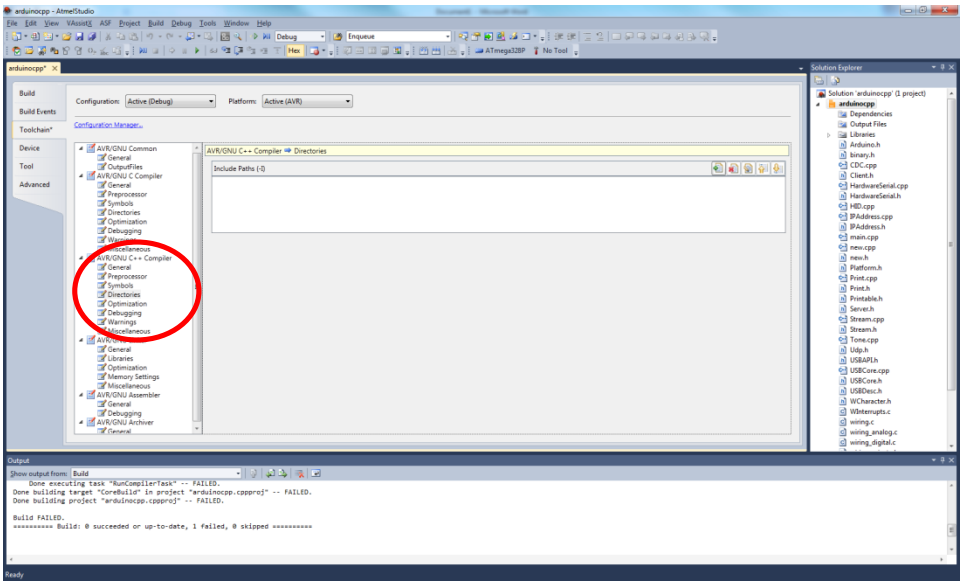
Question 1 (3 marks) Sketch your circuit in the Answer Book. You can use a tool like Fritzing (http://fritzing.org/download/) to help you.

Note: You should never leave a digital input "floating". I.e. it must be tied low when the pushbutton is not pressed. **You need a current limiting resistor with your LED or magic smoke will escape from your LED and it won't work anymore**.

3. Rebuilding the Arduino Library

In the last lab you built the Arduino library as a static C library. We now need to rebuild the library as a static C++ library in order to use cool stuff like the Serial library. To do this:

i. Create a new "GCC C++ Static Library Project".
ii. Select the Atmega328P as before.
iii. Call the new library "arduinocpp"
iv. Follow the rest of the instructions in Lab 1 on adding in the source files. Add in the required include directories, but to "AVR/GNU C++ Compiler" instead of "AVR/GNU C Compiler".



v. Locate on the right panel the file called "Platform.h". Double click on it to edit it.
vi. Locate the line "#include <util/delay.h> and add in "#define F_CPU 16000000UL" just above that line. Your Platform.h file should look like this:

```
ifndef __PLATFORM_H__
#define __PLATFORM_H__

#include <inttypes.h>
#include <avr/pgmspace.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#define F_CPU     16000000UL
#include <util/delay.h>

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned long u32;

… Rest of file removed for brevity.
```

vii. Compile the library as before.

The reason you have to do this is that this week we will use the Serial library to help us debug our code. This requires us to compile our project as a C++ project rather than a C project, and the current arduino library as compiled last week is not usable.

4. Adjusting the Range of the Potentiometer

We now want to program our Arduino so that we take a reading from the potentiometer when we press on the pushbutton, and use the value to control the brightness of the LED.

Create a new project called “lab2part1”, and key in the following program into Atmel Studio (Be sure you choose “GCC C++ Executable Project” and not “GCC C++ Static Library Project”):

```
/*
 * Lab2Part1.cpp
 *
 * Created: 28/8/2013 2:39:27 PM
 *  Author: You
 */


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

#define polledPin      2
#define analogOut      6
#define analogChannel  0

void setup()
{
       pinMode(2, INPUT);
       pinMode(7, OUTPUT);
       Serial.begin(9600);
}
```

```
void flashPin7(int delayVal)
{
       digitalWrite(7, HIGH);
       delay(delayVal);
       digitalWrite(7, LOW);
       delay(delayVal);
}

void loop()
{
       int val=analogRead(0);
       int touch=analogRead(1);

       Serial.print(val);
       Serial.print(" ");
       Serial.print(touch);
       Serial.println();

       analogWrite(6, val);
       delay(500);
}

// Note: Do not modify main. It has been written to work correctly with
// the Arduino library. Modify only setup() and loop(), though you may
// add new functions.

int main(void)
{
       init();
       setup();
    while(1)
    {
              loop();
              if(serialEventRun)
                     serialEventRun();
    }

       return 0;
}
```

Set up the "include" directories for "AVR/GNU C++ Compiler". Add in "libarduinocpp" to "Libraries" in AVR/GNU Linker, as well as the directory that you built libarduinocpp in, into the "Library search path". Again refer to Lab 1 if you are not clear what to do, noting that you should be setting up directories for "AVR/GNU C++ Compiler" instead of "AVR/GNU C Compiler". Upload the resulting hex file to the Arduino using avrdude.

To test your program turn the potentiometer. You will find that the potentiometer controls the brightness of the LED connected to pin 6.

Question 2 (3 marks)

Turn the potentiometer all the way to the left. Then turn it all the way to the right. Notice that rather than going from fully dark to fully bright as you turn the potentiometer, the LED actually goes from dark to bright then back to dark, and then back to bright, etc.

This behaviour is caused by the ADC returning a value between 0 and 1023, while the PWM generator at pin 6 can only understand values of between 0 and 255. Explain why this mismatch causes the behaviour observed in this question.

Question 3 (5 marks)

Write a function called “remap” with the following prototype to remap the values return by the ADC from 0 to 1023, to 0 to 255. Modify your code to use this function. Cut and paste your “remap” function to your answer book and explain why it solves the problem in Question 2.

```
int remap(int val);
```

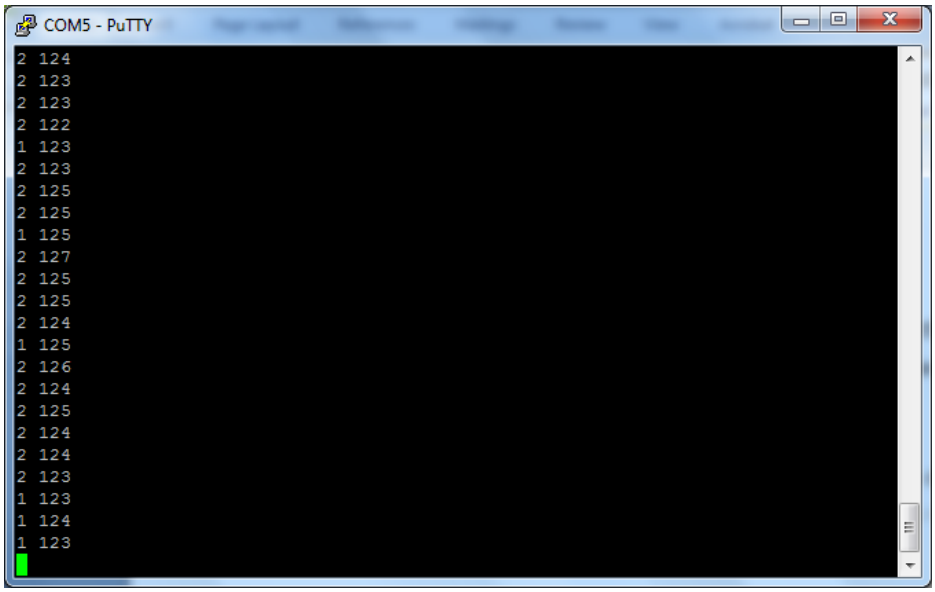
5. Adjusting the Range of the Touch Sensor

Adjusting the touch sensor is trickier than the potentiometer. This is because the potentiometer will generally return a value of between 0 and 1023 (your own potentiometer might vary slightly but not significantly).

The touch sensor on the other hand returns much dirtier data. To do the remapping properly you need to know the minimum and maximum values returned by the touch sensor. To figure this out:

i. Start up a terminal program like puTTY or the “Serial Monitor” in the Arduino environment.
ii. Set up your terminal program so that it reads from the Arduino’s COM port at a rate of 9600 bps.

You will see that your program on the Arduino is returning two columns of numbers as shown below:



Turn the potentiometer. You will notice the numbers in the first column changing. Similarly touch the touch sensor, and you will see the numbers in the second column changing.

What is happening here is that your program is using the Serial class to write values read to your computer through the COM port. The Serial class is an extremely useful tool for debugging programs on the Arduino.

Question 4 (3 marks)

Place your finger at various points on the touch sensor, both at the extreme top of the sensor and extreme bottom, and points in between. Write down the maximum and minimum values you observed.

Question 5 (5 marks)

Write a function called "remapTouch" that remaps the maximum and minimum values you observed in Question 3 to the range of 125 to 500. The prototype for remapTouch is:

```
int remapTouch(int value);
```

Question 6 (3 marks)

Modify your program so that the touch sensor now controls how quickly the LED connected to pin 7 blinks using the flashPin7 function provided. The minimum value passed to this function should be 125 and the maximum value should be 500. Cut and paste your code into your answer book.

6. Creating a Round Robin Application

Currently the LEDs respond immediately to changes in the potentiometer or touch sensor. So turning the potentiometer will cause the LED at pin 6 to brighten or darken, and moving your finger along the touch sensor will change the rate at which the LED at pin 7 is blinking.

We want to modify our program so that the brightness of the LED at pin 6 when the button at pin 2 is pressed, and the rate of blinking of the LED at pin 7 when the button at pin 3 is depressed.

Create a new project called lab2part2. Copy and paste your code from question 6 into the lab2part2.cpp file. Delete the line:

delay(500);

from the loop function.

Remember to set up all the include and library directories and to include the libarduinocpp library.

Question 7 (6 marks)

Modify your lap2part2.cpp file so that the brightness of the LED at pin 6 changes only when the button at pin 2 is pressed, and likewise the blinking rate of the LED at pin 7 changes only when the button at pin 3 is pressed (We are of course assuming that you turned the potentiometer to a new position, or moved your finger on the touch sensor BEFORE pressing the respective button).

Cut and paste your code into the answer book. Explain your modifications, and how these modifications are an example of the round-robin architecture.

7. Using Interrupts in the Arduino

The code below shows a very simple example of how to use interrupts. This example uses INT0 (digital pin 2) to switch an LED on and off. An ISR (called isr()) is created to respond to INT0. The interrupt is configured to respond to a LOW->HIGH transition at digital pin 2.

```
/*
 * testInt.cpp
 *
 * Created: 31/8/2013 11:13:39 AM
 *  Author: dcstanc
 */


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

unsigned char flag=0;

void isr()
{
	flag=!flag;
}

void setup()
{
	attachInterrupt(0, isr, RISING);
	pinMode(6, OUTPUT);
}

void loop()
{
	digitalWrite(6, flag);
}
```

```
int main(void)
{
       init();
       setup();

    while(1)
    {
              loop();

              if(serialEventRun)
                     serialEventRun();
    }

       return 0;
}
```

One thing about this program: It works fine *in theory*, but if you actually upload this program to the Arduino, you will find that the LED does not light up reliably. This is because the switch is mechanically imperfect and causes a phenomenon called "bouncing". This is fine for our purposes so we will ignore this problem for now.

Create a new project called lab2part3. Copy and paste your code from question 7 into the lab2part3.cpp file. Remember to set up all the include and library directories and to include the libarduinocpp library.

Question 8 (12 marks)

Explain how to modify lab2part3.cpp so that your program is now using round-robin with interrupts instead of round-robin.

Cut and paste your modified code into your answer book.

8. Demo Session

   You will demonstrate your lab2part2 and lab2part3 to your lab TA at the start of your lab session for Lab 3. Please be punctual for your demonstration. Your demonstration is worth 5 marks.