CG2271 Real Time Operating Systems
Lab 4
Using an RTOS

1. Introduction

In this lab you will be using an RTOS for the first time, which will hopefully give you some idea of the differences between using an RTOS and other architectures you explored earlier, as well as give you an idea of the differences between an RTOS and other OSes like OSX and Linux.

2. Introducing ArdOS

ArdOS is a free RTOS written specifically for the Arduino series of microcontroller boards based on the Atmel ATMega 128, 328, 1280 and 2560 microcontrollers. ArdOS has almost 500 downloads and just 3 bug reports, so it appears to be fairly reliable, although it has never been through reliability audits unlike the much more established FreeRTOS, it is highly compact and easy to learn (and you will get a chance to work with FreeRTOS later on).

So we will be using ArdOS for this lab not because it was written by the course lecturer (definitely, not because of that!), but because it is small, easy to understand, reflective of how to use other RTOS, and apparently quite reliable.

You can obtain ArdOS from https://bitbucket.org/ctank/ardos-ide/downloads as a zip file incorporating the source code and documentation, or via git clone at https://bitbucket.org/ctank/ardos-ide.git. You may also fork the repository at https://bitbucket.org/ctank/ardos-ide/.

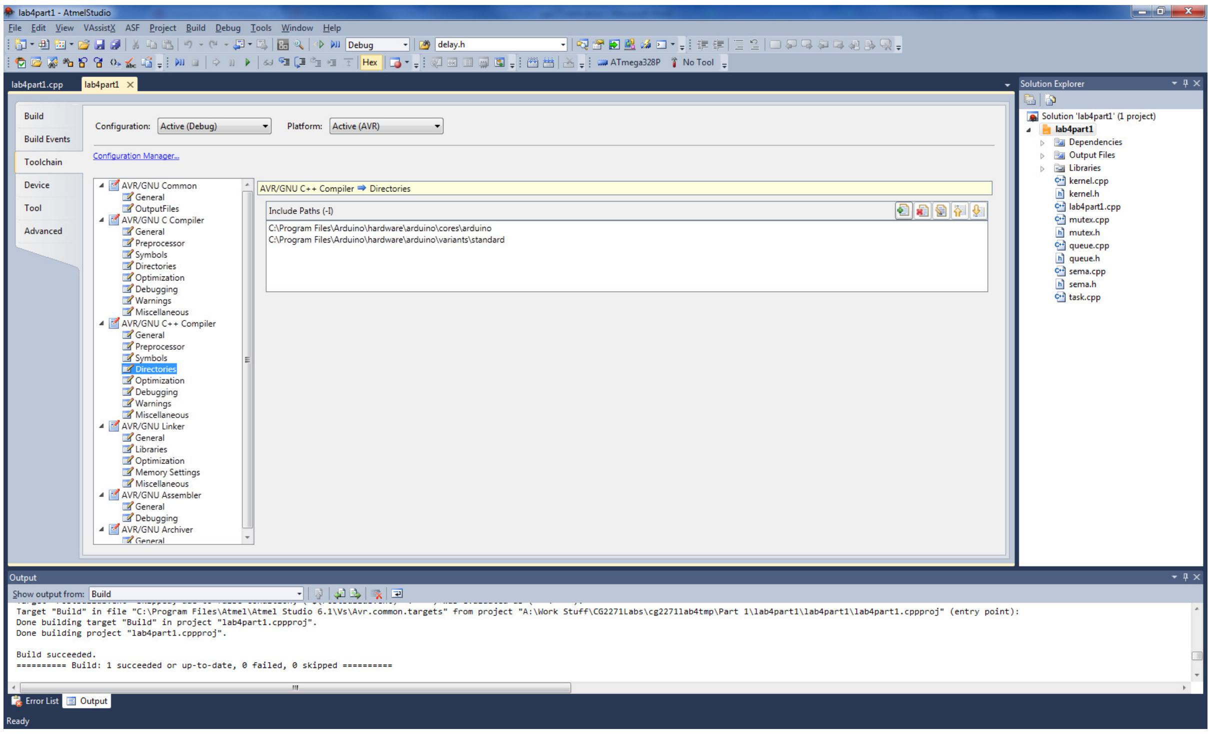
(Note: Do not use the Arduino delay function in this lab. It will cause ArdOS to crash due to the way it uses Timer 0, which ArdOS uses for its time slices).

3. Creating an ArdOS Project

Clone the ArdOS repository somewhere on your computer, or download and unzip the ArdOS zip file. Create a new GCC C++ Executable Project for the ATMega328P called lab4part1. Add in the usual include directories and Arduino library as you had done for previous labs.

Now right click on “lab4part1” on the right window pane, select “Add->Existing Item”, then navigate to the directory where you unzipped or cloned ArdOS, and select every .cpp and .h file (leave out the PDF files, COPYING, COPYING.LESSER and the “examples” directory). Add all the .cpp and .h files to your project.

Your project should now look like this:


Configuration: Active (Debug) Platform: Active (AVR)

AVR/GNU C++ Compiler ⇒ Directories

Include Paths (-I)

```
C:\Program Files\Arduino\hardware\arduino\cores\arduino
C:\Program Files\Arduino\hardware\arduino\variants\standard
```

Output

Show output from: Build

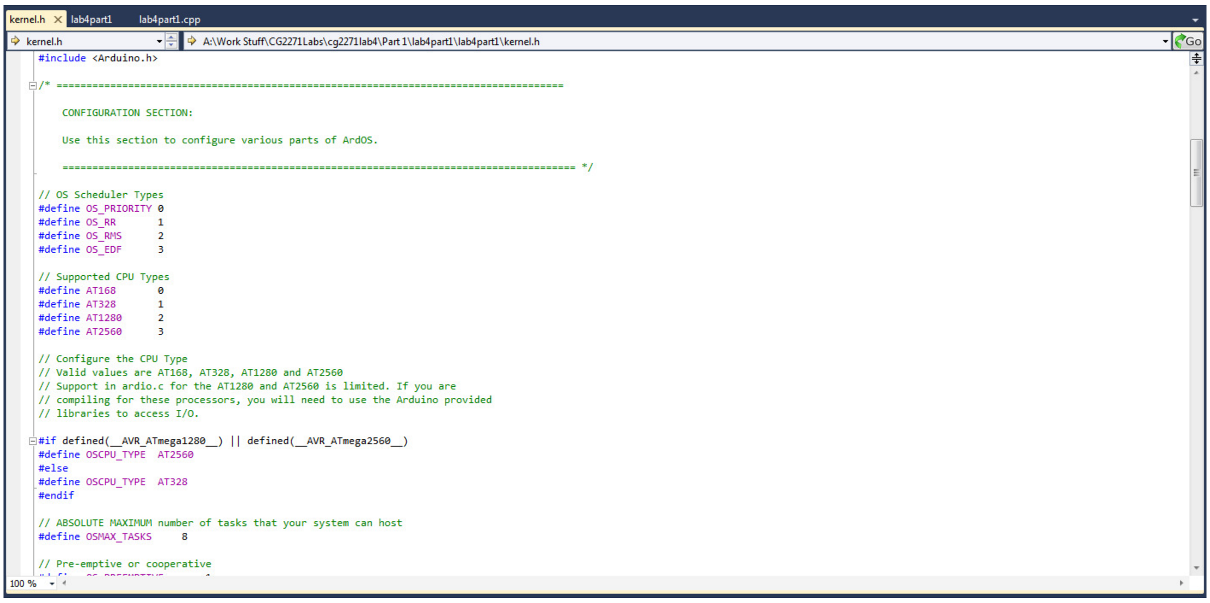
```
Target "Build" in file "C:\Program Files\Atmel\Atmel Studio 6.1\Vs\Avr.common.targets" from project "A:\Work Stuff\CG2271Labs\cg2271lab4tmp\Part 1\lab4part1\lab4part1\lab4part1.cppproj" (entry point):
Done building target "Build" in project "lab4part1.cppproj".
Done building project "lab4part1.cppproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Ensure that kernel.cpp, kernel.h, mutex.cpp, mutex.h, queue.cpp, queue.h, sema.cpp, sema.h and task.cpp are all in the project, or compilation might fail.

4. Configuring ArdOS

Typical of most RTOS like microCOS, FreeRTOS and OSEK, ArdOS comes in source code form so that you can decide which components should be compiled into your applications and which should not, to minimize memory requirements. ArdOS is available for the ATMega128/328 as well as ATMega1280/2560 microcontrollers, and these utilize the stack differently. By making ArdOS available in source code form you will be able to choose which processor to use.

All configuration information is in the kernel.h file. Double click kernel.h and scroll down to the Configuration Section:


kernel.h
lab4part1
lab4part1.cpp
kernel.h
A:\Work Stuff\CG2271Labs\cg2271lab4\Part 1\lab4part1\lab4part1\kernel.h
Go
100 %

```
#include <Arduino.h>

/* ====================================================================================

    CONFIGURATION SECTION:

    Use this section to configure various parts of ArdOS.

    ==================================================================================== */

// OS Scheduler Types
#define OS_PRIORITY 0
#define OS_RR       1
#define OS_RMS      2
#define OS_EDF      3

// Supported CPU Types
#define AT168       0
#define AT328       1
#define AT1280      2
#define AT2560      3

// Configure the CPU Type
// Valid values are AT168, AT328, AT1280 and AT2560
// Support in ardio.c for the AT1280 and AT2560 is limited. If you are
// compiling for these processors, you will need to use the Arduino provided
// libraries to access I/O.

#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define OSCPU_TYPE  AT2560
#else
#define OSCPU_TYPE  AT328
#endif

// ABSOLUTE MAXIMUM number of tasks that your system can host
#define OSMAX_TASKS     8

// Pre-emptive or cooperative
```

The section begins with a set of constant definitions, and then a set of configuration definitions. These allow you to choose your microcontroller type, scheduler type, and whether or not to implement various parts of the operating system.

In our lab we will explore the use of sleep functions and semphores, and will therefore implement these and nothing else.

Symbol	**Explanation**	**Set this to:**
OSCPU_TYPE	Sets the CPU type for your platform (168/328 or 1280/3280)	Nothing. This is automatically set for you.
OSMAX_TASKS	Sets the absolute maximum number of tasks you intend to create	3
OS_PREEMPTIVE	Set to 1 for pre-emptive multitasking, 0 for cooperative.	1
OSSCHED_TYPE	Sets the scheduler type. Currently only fixed priority is supported.	OS_PRIORITY
OSSTACK_SIZE	Sets the default stack size (in 4-byte words) for each task.	30 (Increase to 50 or more if board keeps rebooting)
OSUSE_SLEEP	1: Implement the “sleep” function. 0: Don’t implement.	1
OSUSE_SEMA	1: Implements binary and counting semaphores. 0: Don’t implement.	1
OSUSE_QUEUES	1: Implements FIFO queues. 0: Don’t implement.	0
OSUSE_PRIOQUEUES	1: Implements priority queues. 0: Don’t implement.	0
OSUSE_MUTEXES	1: Implements mutex locks. 0: Don’t implement	0
OSUSE_CONDITIONALS	1: Implements conditional variables. 0: Don’t implement	0

5. <u>Writing your First ArdOS Application</u>

We will continue to use the same circuit as we did in past labs, and we will now create a simple ArdOS application that does what you did section 4 of Lab 3: flash the LED at pin 6 at 5Hz, and the LED at pin 7 at 1 Hz.

Open lab4part1.cpp, highlight the contents in the editor window and press “Delete” to delete all the contents. Then type in the following code:

```
#include <avr/io.h>
#include <Arduino.h>
#include "kernel.h"

#define NUMTASKS     2      // Number of tasks to create.

void task1(void *arg)
{
       // 5Hz = 200ms cycle time. Hence each half is 100ms

       while(1)
       {
              digitalWrite(6, HIGH);
              OSSleep(100);
              digitalWrite(6, LOW);
              OSSleep(100);
       }
}

void task2(void *arg)
{

       while(1)
       {
              // 1HZ = 1000ms cycle time. Each half is 500ms
              digitalWrite(7, HIGH);
              OSSleep(500);
              digitalWrite(7, LOW);
              OSSleep(500);
       }
}

void setup()
{
       // Set pins 6 and 7 as output
       pinMode(6, OUTPUT);
       pinMode(7, OUTPUT);

       // Initialize the OS
       OSInit(NUMTASKS);

       // Add in the tasks
       OSCreateTask(0, task1, NULL);
       OSCreateTask(1, task2, NULL);

       // Start execution
       OSRun();
}


void loop()
{
       // Empty
}
```

```
// Do not modify this section!
int main(void)
{
	init();
	setup();
    while(1)
    {
		loop();
		if(serialEventRun)
			serialEventRun();
    }
}
```

Compile your program and upload it to the Arduino, then answer the following questions:

Question 1 (3 marks)
Consult the ArdOS Reference Guid and describe what OSInit, OSCreateTask and OSRun do.

Question 2 (3 marks)
Describe, IN YOUR OWN WORDS, what the parameters for OSInit and OSCreateTask do.

Question 3 (3 marks)
Notice that the loop() function is empty, unlike the loop() function in earlier labs. Explain why we do not have to put any code inside loop().

Question 4 (3 marks)
In addition notice that the code within tasks 1 and 2 execute within infinite loops. Explain why RTOS tasks typically execute inside infinite loops.

Question 5 (5 marks)
Consult the ArdOS Reference Guide and explain, IN YOUR OWN WORDS, what OSSleep does and what sort of parameters it takes. Explain as well how you think OSSleep works.

6. Coordinating Tasks using Semaphores

Tasks can be coordinated using special variables known as semaphores. To understand how semaphores work, create a new project called lab4part2, and key in the following program. As before add in the include and library directories for Arduino and insert all the ArdOS *.cpp and *.h files into the project.

```
#include <avr/io.h>
#include <Arduino.h>
#include "kernel.h"
#include "sema.h"

OSSema task1Go, task2Go;

void task1(void *param)
{
	while(1)
	{
	// Wait for task1Go
	OSTakeSema(&task1Go);
	for(int i=0; i<5; i++)
		{
			digitalWrite(6, HIGH);
			OSSleep(250);
			digitalWrite(6, LOW);
			OSSleep(250);
		}

		// Release task 2
		OSGiveSema(&task2Go);
	}
}

void task2(void *param)
{
  while(1)
  {

    // Wait on task2Go
    OSTakeSema(&task2Go);
    for(int i=0; i<2; i++)
    {
      digitalWrite(7, HIGH);
      OSSleep(500);
      digitalWrite(7, LOW);
      OSSleep(500);
    }

    // Release task1Go
    OSGiveSema(&task1Go);
  }
}

void setup()
{
  // Set pins 6 and 7 as output
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);

  // Initialize the OS
  OSInit(2);

  // Create the binary semaphores
  // Set task1Go to initial value of 1 so that Task1 can run
  OSCreateSema(&task1Go, 1, 1);
  OSCreateSema(&task2Go, 0, 1);
// Add in the tasks
  OSCreateTask(0, task1, NULL);
  OSCreateTask(1, task2, NULL);

  // Launch the OS
  OSRun();

}

void loop()
{
  // Empty
}

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

	while(1)
	{
		loop();
		if(serialEventRun)
			serialEventRun();
	}
}
```

Compile and run this program and answer the following questions:

Question 6 (4 marks)
Observe how the LEDs flash. Based on your observations and on what you can find in the ArdOS Reference Guide, describe how you think OSTakeSema operates (e.g. what happens if you call OSTakeSema on a semaphore with a value of 0?) Describe as well how you think OSGiveSema operates.

We will now implement the function-queue-scheduling program you wrote in Lab 3, using an RTOS, and compare the behavior of the two programs.

Create a new project called lab4part3, delete lab4part3.cpp, and insert the lab4part3.cpp file included in cg2271lab4.zip. This contains the skeleton that you will use to implement your code. As before:

i. You will need to do switch debouncing.
ii. Pressing the button connected to INT0 will cause the LED at pin 6 to blink 5 times.
iii. Pressing the button connected to INT1 will cause the LED at pin 7 to blink 5 times.
iv. LEDs should blink at a rate of 2Hz (i.e. ON for 250ms, OFF for 250ms, repeated 5 times)

The major change is that right now instead of using function queue scheduling, you will implement this using ArdOS. Some hints for you:

i. You can use one task to blink the LED at pin 6 five times, and another to blink the LED at pin 7 five times.

ii. You can use a semaphores to coordinate between the ISR for INT0 and the task for LED at pin 6, and another semphore to coordiante between the ISR for INT1 and the task the LED at pin 7.

iii. Set the task for INT0 (LED at pin 6) to have a higher priority.

iv. You DO NOT NEED the priority queue routines from Lab 3, but you still need to do swith debouncing.

Complete the skeleton given to you, compile it and run it to make sure it works properly, then answer the following questions:

Question 7 (10 marks)
Cut and paste your code into your answer book. Ensure that there is sufficient comments to that we can understand your code (If we don't understand it, we won't grade it)

Question 8 (4 marks)
Press the button at INT0 first then press the button at INT1 after the LED has flashed twice. Describe and explain your observations of how the LEDs flash.

Question 9 (4 marks)
Press the button at INT1 first, then press the button at INT0 after the LED has flashed twice. Describe and explain your observations of how the LEDs flash.

Question 10 (6 marks)
From your observations in Questions 10 and 11, explain how your program implemented using ArdOS is similar, and how it is different, from the same program implemented using function queue scheduling instead. In particular explain why the LEDs flash differently from the Function Queue Scheduling version.

(Hint: It's more than just because of pre-emption.)