Matt Zimmerer
Daniel Jennings
Lab1
RTOS Spring 2013

```
/*
*Lab02 Blink LED0 @ 1Hz, LED2 @ 2Hz, LED4 @ 4Hz, and LED6 @ 8Hz
*
*Authors: Matt Zimmerer, Daniel Jennings
*
*Version: Lab02 version 1.0
*/
#define F_CPU 8000000

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

#include "FreeRTOS.h"
#include "task.h"

#define LED0 0x01   //Definition of LED pins
#define LED2 0x04
#define LED4 0x10
#define LED6 0x40

struct task_args {
  uint16_t period_ms;        //Struct for period of blinking, and LED character
  uint8_t led;
};

void vTask(void *tArgs);//Prototype for vTask

int main( void )
{                            //Struct initialization with periods and respective LEDs
  struct task_args tArgs[4] = {{1000,LED0}, {500,LED2}, {250,LED4}, {125,LED6}};

  DDRB = 0xFF;      //Sets LED port to output, and off
  PORTB = 0xFF;
                            //Task calls for all 4 LEDs
  xTaskCreate(vTask, (const char *) "1Hz", 100, (void *) &tArgs[0], 1, NULL);
  xTaskCreate(vTask, (const char *) "2Hz", 100, (void *) &tArgs[1], 1, NULL);
  xTaskCreate(vTask, (const char *) "4Hz", 100, (void *) &tArgs[2], 1, NULL);
  xTaskCreate(vTask, (const char *) "8Hz", 100, (void *) &tArgs[3], 1, NULL);

  vTaskStartScheduler();//Starts the Tasks running

  return 0;
```

```
}

void vTask(void *tArgs)      //Task Definition for vTask
{
                             //Struct for using tArgs in vTask
  struct task_args *args = (struct task_args *) tArgs;

  for (;;)  {
    PORTB ^= args->led;//Toggle LED, wait the given period
    vTaskDelay( (args->period_ms / 2) / portTICK_RATE_MS );
  }
}
```

```
###############################################################################
# Filename: Makefile
# Revision log: Let there be light.
###############################################################################

PROJNAME=main

CFLAGS=-funsigned-char -funsigned-bitfields -O3
CFLAGS+=-fpack-struct -fshort-enums -Wall -c -std=gnu99
CFLAGS+=-I../Source/portable
CFLAGS+=-I../Source/include
CFLAGS+=-I../Source/MemMang
CFLAGS+=-mmcu=atmega2560

LDFLAGS=--Map=$(PROJNAME).map -lm
LDFLAGS+=-L../Source/include
LDFLAGS+=-L../Source/portable
LDFLAGS+=-L../Source/MemMang
LDFLAGS+=-mmcu=atmega2560

HEXFLAGS=-O ihex -R .eeprom -R .fuse -R .lock -R .signature

C_SRCS = main.c \
croutine.c \
heap_1.c \
list.c \
port.c \
queue.c \
tasks.c \
timers.c

OBJS=$(C_SRCS:.c=.o)

%.o: %.c
	avr-gcc $(CFLAGS) -o$@ $<

all: $(PROJNAME).hex

$(PROJNAME).hex: $(OBJS)
	avr-gcc -o$(PROJNAME).elf $(OBJS) $(LDFLAGS)
	avr-objcopy $(HEXFLAGS) $(PROJNAME).elf $(PROJNAME).hex
	avr-size $(PROJNAME).hex

install: $(PROJNAME).hex
	sudo avrdude -c stk600 -p atmega2560 -P usb -v -v -U flash:w:$(PROJNAME).hex

clean:
	rm -rf *~ ./*o ./*d *.elf *.a *.hex *.lss *.eep *.map *.srec
```

Questions

1. **What rate did you set your tick interrupt to? Why? Is there a rate that would not be suitable to ensure proper operation of your program? (you can test out different rates with your program to observe the behavior)**

We left configTICK_RATE_HZ defined as 500Hz. A broad range of rates will cause improper operation of the program. For instance, if we up the rate closer to F_CPU, the scheduler will be starving CPU resources as it works more often than it should. Conversely, if we lower the tick rate closer to 0, our frequencies will begin to deviate from the desired values until at no interrupts, there is not change on the LED states.

2. **Why does FreeRTOSConfig.h allow you to include or exclude API functions in your RTOS?**

This is so that the developer can have control over the size of the output .hex file. If the .hex file that is to be loaded onto target is too large for the target's memory bank, The target will not work properly.

Conclusion (Matt Zimmerer)

Using the FreeRTOS api is pretty straight forward as far as this project goes. The ability to delay a task on top of a real time scheduler by calling a single function was desirable. Besides the ease of using API functions, it is fairly easy to compile the FreeRTOS objects and the output hex is small (28,859bytes). Modifying the FreeRTOS config file proved useful here, allowing us to pass clock parameters into the FreeRTOS library through the cherished F_CPU macro.

Conclusion (Danny Jennings)

The APIs that exist in the RTOS Kernel are useful for many different things, definition and deletion of tasks, as well as scheduling tasks for different times with different priority seems to be a vital part of creating an RTOS program. For this lab we only defined one task, but got to see how to schedule them and how to use the various APIs provided, as well as how to edit the FreeRTOSConfig.h to set things like TickRate and edit what APIs we would like included. Overall, an informative introduction to RTOS programming.