Experiment 3: Semaphores

Matt Zimmerer
Daniel Jennings
4/23/2012

Questions:

1. The debounce is handled in the vISRHndlrTask which is in control of setting the global button state buttonState, and toggles LED6 on or off. This task delays 10ms and if the button was pressed and is still pressed, or was not pressed and still isn't after that time then we know the button press is real and allow the toggling, if it's not then we know that the button was not pressed enough for us to care.
2. The purpose of using a deferred handler interrupt scheme is to preserve task priorities and not have a task that is lower priority run over a task with higher priority just because it grabbed a semaphore. It also shortens the ISR which reduces the amount of un-handled ISRs.

Conclusions

Danny Jennings:

This lab took a little more time for me to understand the concepts involved compared to the previous labs. The idea of the semaphore was something quite new and the use of the handler task in unison with it took a little while. Once we figured out how to make the semaphore and handlers work I can see the value in their use, they are helpful and make running your tasks in the proper order easy. Once we got that working it was mostly a matter of getting the button to debounce properly, and finding the correct frequencies. The button problem seemed to not be an issue because the button seems to debounce pretty well on its own with no software implementation, we think that might be due to something on the hardware. But we put in the debounce logic in the ISR handler task and it works nicely. The frequencies for LED2 were easy because we had worked with timers before, but the frequency for LED7 was a little more difficult because we had set our CPU to 16Mhz, we used a toggle in the ISR to divide that clock.

Matt Zimerrer:

Conclusion

The new topic in this lab is deferred interrupt handling by the use of semaphores. For this particular application, the semaphores were used to synchronize a prioritizable task with a hardware interrupt event. This seemingly extra step is desirable because it shortens the ISR, and allows us to prioritize the code that is ran from ISRs by use of FreeRTOS's scheduler. Another thing was new to me in this lab. The debouncing logic we decided upon was great because it was able to be placed int he button ISR's handler, where we were able to use vTaskDelay. This made sure that we were not wasting any CPU resources while the debouncing delay was ticking away.

CODE:
main.c:

```
/*
*Project: Lab3 Semaphores
*
*Lab 3 controls LEDs 2, 6, and 7. When SW7 is not pressed,
*LED6 is off, LED 2 is at 2Hz and LED7 is at 7Hz.
*When pressed, LED6 turns on and LEDs 2 and 7 double their speeds.
*The button used an external interrupt and the tasks are controlled using
*semaphores unique to each task.
*
*Authors: Matt Zimmerer, Daniel Jennings
*
*Version: Lab03 version 1.0
*/
#define F_CPU 16000000L

#include <stdint.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "FreeRTOS.h"
#include "semphr.h"
#include "task.h"

#define _2HZ_HALF_PERIOD 250
#define _4HZ_HALF_PERIOD 125

#define DEBOUNCE_WAIT_MS 10

#define SW7 (1 << 7)
//defining our LEDs
#define LED2 (1 << 2)
#define LED6 (1 << 6)
#define LED7 (1 << 7)

void vISRHdlrTask(void *tArgs);
void vTIMHdlrTask(void *tArgs);
void vLEDTask(void *tArgs);

//The two semaphores used for the respective tasks
xSemaphoreHandle xISRSemaphore;
xSemaphoreHandle xTIMSemaphore;

// Active high buttonState mirror
static volatile unsigned char buttonState = 1;

static void init_isr()
{
  EICRA = 0x00; // INT0-6 not configured
  EICRB = 0x40; // INT7 falling edge triggered
  EIMSK = 0x80; // Enable INT7
}

static void init_timer()
{
  TCCR2A = 0xC0; // Set timer2 to CTC mode, sets OC2A on compare match
  TCCR2B = 0x0F; // Set prescaler to divide by 1024
  OCR2A = 162;   // Set output compare reg to 162
  TIMSK2 = 0x02; // Enables interrupt on compare match A
}

int main(void)
```

```
{
  DDRB |= LED2 | LED6 | LED7; //sets LEDs 2,6 and 7 to outputs
  DDRE &= ~SW7; //Sets our external interrupt SW7 to the correct E pin
  PORTB |= (LED2 | LED7); //Sets LED2 and 7 to on initially
  PORTB &= ~LED6; //Sets LED6 to off initially

  init_isr();//initializing ISR
  init_timer();//initializing timer
  sei();//enabling interrupts

  vSemaphoreCreateBinary(xISRSemaphore);
  vSemaphoreCreateBinary(xTIMSemaphore);

  xTaskCreate(vTIMHdlrTask, (const signed char *) "TIMHDLR", 100, NULL, 3, NULL);
  xTaskCreate(vISRHdlrTask, (const signed char *) "ISRHDLR", 100, NULL, 2, NULL);
  xTaskCreate(vLEDTask, (const signed char *) "LED", 100, NULL, 1, NULL);

  // Kick off the scheduler
  vTaskStartScheduler();

  return 0;
}

void vISRHdlrTask(void *tArgs)
{
  for (;;) {
    xSemaphoreTake(xISRSemaphore, portMAX_DELAY);

    // Debouncing logic
    vTaskDelay(DEBOUNCE_WAIT_MS / portTICK_RATE_MS);
    if ((buttonState == 0 && (PINE & SW7))
          || (buttonState == 1 && !(PINE & SW7)))
      continue;

    buttonState ^= 1;

    PORTB ^= LED6;
  }
}

void vTIMHdlrTask(void *tArgs)
{
  static volatile char toggle = 0;

  for (;;) {
    xSemaphoreTake(xTIMSemaphore, portMAX_DELAY);

    // If button is pressed
    if (buttonState == 0) {

      // Only toggle LED every other timer tick
      if (toggle ^= 0xFF)
        PORTB ^= LED7;

    // If button is not pressed
    } else {
      PORTB ^= LED7;
    }
  }
}

void vLEDTask(void *tArgs)
{
  static volatile char toggle = 0;
  for (;;) {

    // If button is pressed
    if (buttonState == 0) {

      // Only toggle LED every other timer tick
      if (toggle ^= 0xFF)
        PORTB ^= LED2;

      vTaskDelay(_2HZ_HALF_PERIOD / portTICK_RATE_MS);

    // If button is not pressed
    } else {
      PORTB ^= LED2;

      vTaskDelay(_4HZ_HALF_PERIOD / portTICK_RATE_MS);
    }
  }
}

ISR(INT7_vect)
{
  static portBASE_TYPE xHPTW = pdFALSE;

  xSemaphoreGiveFromISR(xISRSemaphore, &xHPTW);
}

ISR(TIMER2_COMPA_vect)
{
  static portBASE_TYPE xHPTW = pdFALSE;
  static volatile char toggle = 0;

  // At 16Mhz, we needed software support to provide an extra clock division
  if (toggle ^= 0xFF)
    return;

  xSemaphoreGiveFromISR(xTIMSemaphore, &xHPTW);
}
```