Lab 4 Report

Matt Zimmerer

Danny Jennings

Questions:

1. Remove one shared data protection you put in place in your code and see if your program still works correctly. Try to explain why you observe the behavior you observe.

The program is still working correctly. There is a race condition that exists in our code, but it seems to be so improbable that it is not happening. The race condition that could happen exists when one of the two tasks is writing to the SSEG device, but is interrupted by a tick ISR. The tick ISR calls our tick hook, which writes the colon command character. If for instance the task that was interrupted writes a single character escape byte, the tick hook will write the colon command, which will be interpreted as a character to write to screen. This is all happening very fast, so even if garbage data is written to the screen, within a 5 or 10Hz period, it is overwritten with another command. Another reason why we may not be seeing any bad behavior is because while the tick ISR is happening at 500Hz, we use a software clock divider to do some logic at 1Hz. This means that the race condition only has a chance to fire every second, and if the chance for the data race to actually occur is very low, it may take thousands or even millions of seconds for the split second bug to occur.

Code:

main.c

```c
/*
 * Project: Lab4 SPI seven segment display, with multiple tasks
 *
 * Authors: Matt Zimmerer, Daniel Jennings
 *
 * Version: Lab04 version 1.0
 */

#define F_CPU 8000000L
#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>
#include "FreeRTOS.h"
#include "semphr.h"
#include "task.h"
#include "spi_sseg.h"

// LED definitions
#define LEDDDR  DDRF
#define LEDPORT PORTF
#define LED0    (1<<PF0)
#define LED2    (1<<PF2)

typedef const signed char * cscharp;

typedef struct taskJob {
   uint16_t period;  // The period to blink the LED (square wave period)
   uint8_t LED;      // The LED to blink
   uint8_t dispSide; // 0 is left, 1 is right
   uint8_t counter;  // The state transition count
} Job;

void Blink(void *tArgs);

static xSemaphoreHandle rSSEG;

int main(void)
{
   Job job5Hz = {200, LED0, 0, 0};  // LED0 Blinks at 5Hz, displays on left
   Job job10Hz = {100, LED2, 1, 0}; // LED2 Blinks at 10Hz, displays on right

   // Set data direction register for LEDs
   LEDDDR = LED0 | LED2;

   // Initialize SSEG display
   _delay_ms(100); // Give the SSEG a grace period
   SPI_MasterInit(); // Initialze ATMEGA spi interface
   SSEG_Set_Brightness(0); // Set brightness to max
   SSEG_Reset(); // Reset the SSEG device

   // Create SSEG resource semaphore
   vSemaphoreCreateBinary(rSSEG);

   // Creat the LED Blinky tasks
   xTaskCreate(Blink, (cscharp) "5HZ", 100, (void*) &job5Hz, 1, NULL);
   xTaskCreate(Blink, (cscharp) "10HZ", 100, (void*) &job10Hz, 1, NULL);

   // Kick off the scheduler
   vTaskStartScheduler();

   return 0;
}

// Tick rate is configured for 500Hz, so toggling the LED every 250th call
// will yield 1Hz
void vApplicationTickHook()
{
   // Higher priority task worken = false
   static portBASE_TYPE xHPTW = pdFALSE;
   static uint8_t colonState = 0;
   static uint16_t counter = 0;

   // Reduce the frequency of this code
   if (++counter == 250) {

      // Take the SSEG resource semaphore
      xSemaphoreTakeFromISR(rSSEG, &xHPTW);

      // Toggle colon state
      if (colonState ^= 1)
         SSEG_Write_Decimal_Point(4);
      else
         SSEG_Clear_Decimal_Point(4);

      // Release the SSEG resource semaphore
      xSemaphoreGiveFromISR(rSSEG, &xHPTW);

      counter = 0;
   }
}

void Blink(void *tArgs)
{
   Job *currJob = (Job*) tArgs;

   for (;;) {

      // Toggle target LED
      LEDPORT ^= currJob->LED;

      // Take the SSEG resources
      xSemaphoreTake(rSSEG, portMAX_DELAY);

      // Increment the state counter
      currJob->counter++;
      if (currJob->counter >= 100)
         currJob->counter = 0;

      // Display counter value on correct side of SSEG
      if (currJob->dispSide == 0)
         SSEG_Write_left_digits(currJob->counter);
      else
         SSEG_Write_right_digits(currJob->counter);

      // Release the SSEG resources
      xSemaphoreGive(rSSEG);

      // Delay a half period
      vTaskDelay((currJob->period/2)/portTICK_RATE_MS);
   }
}
```

spi_sseg.c

```
#include <avr/io.h>
#include "spi_sseg.h"

// Generic structure for LUT
typedef struct Command {
  uint8_t input;
  uint8_t command;
} Command;

// Single digit command LUT
Command digitCommands[] = {
  {1, SSEG_DIG1},
  {2, SSEG_DIG2},
  {3, SSEG_DIG3},
  {4, SSEG_DIG4},
  {0, 0}
};

// Single digit value LUT
Command valueCommands[] = {
  {0, SSEG_0},
  {1, SSEG_1},
  {2, SSEG_2},
  {3, SSEG_3},
  {4, SSEG_4},
  {5, SSEG_5},
  {6, SSEG_6},
  {7, SSEG_7},
  {8, SSEG_8},
  {9, SSEG_9},
  {10, SSEG_A},
  {11, SSEG_B},
  {12, SSEG_C},
  {13, SSEG_D},
  {14, SSEG_E},
  {15, SSEG_F},
  {0, 0}
};

// Single decimal command LUT
Command decimalCommands[] = {
  {0, SSEG_DP_0},
  {1, SSEG_DP_1},
  {2, SSEG_DP_2},
  {3, SSEG_DP_3},
  {4, SSEG_DP_4},
```

```c
  {5, SSEG_DP_5},
  {0, 0}
};

static uint8_t decimalPoints;

void SPI_MasterInit(void)
{
  // Set all required pins as outputs, no inputs
  SPI_DDR = (1<<SPI_SS) | (1<<SPI_MOSI) | (1<<SPI_SCK);

  // Enable spi, set master mode, and sck = fosc/64
  SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR1);
}

void SPI_MasterTransmit(uint8_t data)
{
  SPDR = data;
  while(!(SPSR & (1<<SPIF)));
}

void SSEG_Set_Brightness(uint8_t val)
{
  //Special command for brightness
  SPI_MasterTransmit(SSEG_BRIGHTNESS);

  //Brightness value between 0 and 254
  if (val > 254)
    val = 254;

  SPI_MasterTransmit(val);
}

void SSEG_Reset(void)
{
  //Special command for reset display
  SPI_MasterTransmit(SSEG_RESET);
}

void SSEG_Write_digit(uint8_t digit, uint8_t val)
{
  Command *dc;
  Command *vc;
  uint8_t command = 0;
  uint8_t value = SSEG_BLANK;

  // Iterate through command LUT and find the correct command per digit
  for (dc = digitCommands; dc && (dc->input || dc->command); dc++)
```

```c
        if (dc->input == digit)
            command = dc->command;

    // Iterate through value LUT and find the correct command per value
    for (vc = valueCommands; vc && (vc->input || vc->command); vc++)
        if (vc->input == val)
            value = vc->command;

    // If either is not found, forget about it!
    if (!command)
        return;

    SPI_MasterTransmit(command);
    SPI_MasterTransmit(value);
}

void SSEG_Write_4vals_array(uint8_t *vals)
{
    SSEG_Write_digit(DIGIT_1, vals[0]);
    SSEG_Write_digit(DIGIT_2, vals[1]);
    SSEG_Write_digit(DIGIT_3, vals[2]);
    SSEG_Write_digit(DIGIT_4, vals[3]);
}

void SSEG_Write_left_digits(uint8_t val)
{
    SSEG_Write_digit(DIGIT_1, val / 10);
    SSEG_Write_digit(DIGIT_2, val % 10);
}

void SSEG_Write_right_digits(uint8_t val)
{
    SSEG_Write_digit(DIGIT_3, val / 10);
    SSEG_Write_digit(DIGIT_4, val % 10);
}

void SSEG_Write_Decimal_Point(uint8_t val)
{
    Command *dc;

    // Iterate through decimal LUT and find the correct command per decimal
    for (dc = decimalCommands; dc && (dc->input || dc->command); dc++)
        if (dc->input == val)
            decimalPoints |= dc->command;

    SPI_MasterTransmit(SSEG_DEC_PNT);
    SPI_MasterTransmit(decimalPoints);
}
```

```
void SSEG_Clear_Decimal_Point(uint8_t val)
{
  Command *dc;

  // Iterate through decimal LUT and find the correct command per decimal
  for (dc = decimalCommands; dc && (dc->input || dc->command); dc++)
    if (dc->input == val)
      decimalPoints &= ~dc->command;

  SPI_MasterTransmit(SSEG_DEC_PNT);
  SPI_MasterTransmit(decimalPoints);
}
```

Conclusions:

Danny Jennings:

This lab reminded me a lot of the 329 assignments working with the sseg. Writing the C file for spi_sseg was easy because of the very short and succinct datasheet for the seven segment display, and the good SPI section of the datasheet for our processor. The whole interfacing and SPI part in general was enjoyable, but pretty easy and straightforward. The tick function didn't present too much of a problem for us because of Matt figuring it out really fast for some reason. The datasheet is helpful for any debugging. The fact that if we don't protect our system it will start throwing garbage is interesting, but makes sense when we think about it a little.

Matt Zimmerer:

The most trouble we encountered with the SSEG driver was the wiring. For about an hour we used the rx pin on the device, but later figured that it was only for the UART interface that is also supported. After correctly wiring the SSEG to our development board, actually implementing the code was easy. The provided header file was very detailed in how we should go about implementing the driver. One point of note, the pin definitions that were included in the header file were wrong. Looking at the data-sheet for the ATMEGA 2560, we found the correct pins to use for SPI protocol. Also, adding the tick hook was not hard. The only step that may have been troublesome was finding the prototype for the tick hook function. This was found by greping the FreeRTOS source (which is awesomely small) for various keywords. Lastly, the semaphores used to protect the resource from tick interrupts worked on the first try. FreeRTOS is just great.