EEC172 Prelab 3

4/22/2024

Name:

SID:

Difficulty Rating

Question 1.

(Ordering) Assume that an ARM Cortex-M based MCU is in an idle main loop and a GPIO interrupt has been configured on pin X to trigger when it detects a falling edge. No other interrupts have been configured. At some point, a single falling edge occurs on pin X. Place the following sequence of events in order they should occur.

A) Processor begins execution in ISR context
B) MCU hardware saves current register state to stack
C) Processor encounters ISR return instruction
D) MCU hardware restores pre-interrupt register state
E) Processor pauses execution
F) Interrupt flag is cleared in software
G) MCU hardware loads the corresponding ISR address to the PC from the interrupt vector table
H) Processor resumes execution of original context

Difficulty Rating

Question 2.

(Free Response) In your own words, briefly explain what the `volatile` C keyword does and when it is used.

Difficulty Rating

Question 3.

(Free Response) In your own words, briefly explain what the resistor and capacitor are doing in the IR receiver application circuit and how they are accomplishing it.

Difficulty Rating

Question 4.

(Debugging) Imagine we are trying to build a system that responds to a button press by printing out a message to the debug console. We use an ARM Cortex-M based MCU and configure it with a GPIO interrupt on pin X that invokes a the handler function `button_handler` when it detects a falling edge. No other interrupts will be configured. At some point, the button is pressed, causing a falling edge to occur on pin X. However, the expected message does not get printed. Assuming that the circuit has been connected properly, given the following code, identify the bug causing the problem.

```
1  // main.c
2  #include <stdbool.h>
3  #include <stdio.h>
4
5  bool button_pressed = false;
6
7  // button interrupt handler
8  void button_handler(void) {
9      clear_int_flag();
10     button_pressed = true;
11     return;
12 }
13
14 int main(void) {
15
16     // initialization steps ...
17
18     while(1) {
19         if (button_pressed) {
20             // assume printf prints to debug console
21             button_pressed = false;
22             printf("button pressed!");
23         }
24     }
25 }
```

Difficulty Rating

Question 5.

(Debugging) As part of this lab, you will likely need to use SysTick module to track time when measuring pulse widths. One scenario that you may need to consider is when the SysTick countdown register wraps around. This occurs when the SysTick counter hits 0, at which point the registered SysTick handler is called and the counter resets itself back to the configured reload value. The SysTick module exposes the counter register for timekeeping, and triggers an interrupt handler that can be used as a simple periodic timer.

The following firmware should print out the systick register value for every falling edge that was detected on the configured gpio pin.

```
1  // main.c
2  #include <stdbool.h>
3  #include <stdint.h>
4  #include <stdio.h>
5
6  static uint8_t buffer[1024];
7  volatile uint16_t buf_idx = 0;  // current buffer index
8  volatile bool systick_expired = false; // systick expiry flag
9
10 void gpio_handler(void) {
11     // clear interrupt status flag
12     clear_int_flag();
13
14     // store current systick register value in buffer and increment idx
15     buffer[buf_idx] = systick_get();
16     buf_idx++;
17
18     return;
19 }
20
21 void systick_handler(void) {
22     // no flags need to be cleared for systick interrupt
23     // clear and reset the buffer
24     for (int i = 0; i < buf_idx; i++) {
25         buffer[i] = 0;
26     }
27     buf_idx = 0;
28     systick_expired = true;
29     return;
30 }
31
32 int main(void) {
33
34     // initialization steps ...
35     // gpio handler registered to occur on falling edge
36     // systick interrupt enabled and period set
37
38     uint16_t prev_idx = buf_idx;
39     while (1) {
40         if (systick_expired) {
41             systick_expired = false;
42             printf("\nsystick reset\n\n\n");
43         }
44
```

```
45          if (prev_idx != buf_idx) {
46              // print the message whenever a falling edge is detected
47              printf("falling edge at %d ticks\n", buffer[prev_idx]);
48              prev_idx = buf_idx;
49          }
50      }
51  }
```

Assume that the following is true:

- The GPIO handler writes the current SysTick value into a buffer, which will be printed in the main loop.
- At regular intervals, the SysTick handler is used to clear the buffer so that it doesn't overflow.
- The buffer is large enough that it will not overflow before the timer expires.
- The GPIO handler and the SysTick handler do not preempt one another, but will instead wait for the other to finish before executing.

As you test the system, you connect the GPIO pin to an oscilloscope and compare the printed output against the measured signal. When you do this, you find 2 issues:

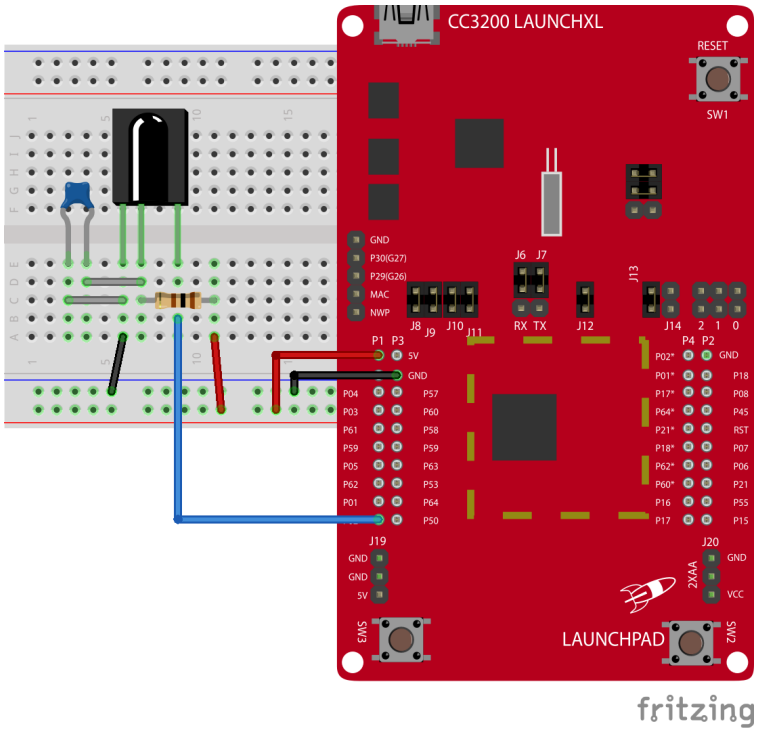
1) there are more falling edges in the oscilloscope signal than were printed by the firmware
2) 0 values are frequently being reported.

Describe what could be causing each of these issues.

Difficulty Rating

Question 6.

(Debugging) In this problem, the CC3200 is meant to measure and print out <40ms pulse widths detected from the IR receiver, where a pulse width is defined as the time elapsed between a rising edge and a falling edge. **There are errors in the circuit, the main while loop, the global variables, and the gpio handler. Identify as many as you can.**




```c
// main.c
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

#include "gpio.h"
#include "systick.h"

// some helpful macros for systick

#define SYSCLKFREQ 80000000ULL
#define TICKS_TO_US(ticks) \
    ((((ticks) / SYSCLKFREQ) * 1000000ULL) + \
    ((((ticks) % SYSCLKFREQ) * 1000000ULL) / SYSCLKFREQ))\

#define US_TO_TICKS(us) ((SYSCLKFREQ / 1000000ULL) * (us))

// systick reload value set to 40ms period
// (PERIOD_SEC) * (SYSCLKFREQ) = PERIOD_TICKS
#define SYSTICK_RELOAD_VAL 3200000UL

```

```
22 // track systick counter periods elapsed
23 // if it is not 0, we know the transmission ended
24 volatile int systick_expired = 0;
25
26 /**
27  * Function to reset SysTick Counter
28  */
29 static inline void SysTickReset(void) {
30     HWREG(NVIC_ST_CURRENT) = 1;
31     systick_expired = 0;
32 }
33
34 // pin 50 info
35 #define IR_GPIO_PORT    GPIOA0_BASE
36 #define IR_GPIO_PIN     0x1
37
38 uint64_t ulsystick_delta_us = 0;
39
40 /**
41  * Interrupt handler for GPIOA0 port
42  *
43  * Only used here for decoding IR transmissions
44  */
45 static void GPIOA0IntHandler(void) {
46     static bool prev_state = 1;
47
48     // get and clear status
49     unsigned long ulStatus;
50     ulStatus = MAP_GPIOIntStatus(IR_GPIO_PORT, true);
51     MAP_GPIOIntClear(IR_GPIO_PORT, ulStatus);
52
53     // check in interrupt occured on pin 50
54     if (ulStatus & IR_GPIO_PIN) {
55         if (prev_state) {
56             // previous state was high -> falling edge
57
58             if (!systick_expired) {
59                 // if systick expired, the pulse was longer than 40ms
60                 // don't measure it in that case
61
62                 // calculate the pulse width
63                 ulsystick_delta_us = TICKS_TO_US(MAP_SysTickValueGet());
64             }
65         } else {
66             // previous state was low  -> rising edge
67
68             // begin measuring a new pulse width, reset the delta and systick
69             ulsystick_delta_us = 0;
70             SysTickRestart();
71         }
72     }
73
74     return;
75 }
76
77 /**
```

```
 * SysTick Interrupt Handler
 *
 * Keep track of whether the systick counter expired
 */
static void SysTickHandler(void) {
    systick_expired = 1;
}

int main(void) {

    // ... other board initializations ...

    systick_wrapped = 1;
    MAP_SysTickPeriodSet(SYSTICK_RELOAD_VAL);
    MAP_SysTickIntRegister(SysTickHandler);
    MAP_SysTickIntEnable();
    MAP_SysTickEnable();

    // Configure PIN_50 for GPIO Input
    PinTypeGPIO(PIN_50, PIN_MODE_0, false);
    GPIODirModeSet(IR_GPIO_PORT, IR_GPIO_PIN, GPIO_DIR_MODE_IN);

    // Register the interrupt handlers
    MAP_GPIOIntRegister(IR_GPIO_PORT, GPIOA0IntHandler);

    // Configure interrupts on rising and falling edges
    MAP_GPIOIntTypeSet(IR_GPIO_PORT, IR_GPIO_PIN, GPIO_BOTH_EDGES); // read ir_output
    uint64_t ulStatus = MAP_GPIOIntStatus(IR_GPIO_PORT, false);
    MAP_GPIOIntClear(IR_GPIO_PORT, ulStatus);           // clear interrupts on GPIOA2

    // Enable interrupts on ir_output
    MAP_GPIOIntEnable(IR_GPIO_PORT, IR_GPIO_PIN);


    while(1) {
        if (ulsystick_delta_us) {
            // a pulse width was measured by the gpio handler

            // print the measured pulse width
            printf("measured pulse width: %d us", ulsystick_delta_us);
        }
    }
}
```