Assignment #9: Problem Set 6 - Due Wed. 02/27/13

1 Programming

While maintaining the same functionality as before, revise the previous program as follows.

- Trigger the input handler with an external interrupt signal, generated by a button press (rising edge.) Note that you may need to "debounce" the switch in software so that the program responds only once to any button press.
- Change the output handler so that it is executed in response to the SVC (supervisor call) instruction.

As before, the program can be tested in RAM or in flash memory, but the final version is to be programmed into the flash memory of the microcontroller, so that the program can be demonstrated without being connected to the Keil debugger. Print and submit the source program, and also email it to me, and bring your programmed board to my office to demonstrate the program.

1.1 Execution

This assignment was completed by editing the program files from the previous assignment. The files were edited based on example programs offered by *ST Microelectronics* for the "Discovery" board. These examples made use of the `stm32f4_discovery.h` library, so the program written also made use of this library. Indeed, some of the initialization code (in `main.c`) was based on what was listed in both the example programs and the libraries that they called.

The program worked as expected: Each button press produced a single state change, and each state change occured as expected.

1.2 Software Listing

output_handler.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    Brian Arnberg                       ;;
;;    Problem Set #6 - Output Handler     ;;
;;     output_handler.s                   ;;
;;      - provides two functionalities    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    enable_leds - turns an LED on       ;;
;;       - assumes a single argument      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    clear_leds - turns an LED off       ;;
;;       - assumes a single argument      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
GPIOD   EQU 0x40020C00           ;; general purpose I/O, port D
BSRRL   EQU 0x18                 ;; port D set
BSRRH   EQU 0x1A                 ;; port D reset

        AREA OUTPUT, CODE
        EXPORT enable_leds
        EXPORT clear_leds

enable_leds
        MOV r2, #1               ;; make r2 = 0x01
        MOV r2, r2, lsl r0       ;; left shift the bit by the argument
        LDR r3, =GPIOD           ;; get the address of GPIOD
        STR r2, [r3, #BSRRL]     ;; write the pattern in r2 to GPIOD->BSRRL
                                 ;;  this sets the pin value (passed in the argument)
        BX r14                   ;; return from output handler

clear_leds      ;; clear the LEDS
        MOV r2, #1               ;; make r2 = 0x01
        MOV r2, r2, lsl r0       ;; left shift the bit by the argument
        LDR r3, =GPIOD           ;; get the address of GPIOD
        STR r2, [r3, #BSRRH]     ;; write the pattern in r2 to GPIOD->BSRRH
                                 ;;  this clears the pin value (passed in the argument)
        BX r14                   ;; return from output handler

        END
```

input_handler.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;   Brian Arnberg                      ;;
;;   Problem Set #6 - Input Handler     ;;
;;    input_handler.s                   ;;
;;     - tests the user button          ;;
;;     - sets a global variable         ;;
;;     - triggered by a button press    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;     - global variables               ;;
;;        - state (represents a pattern) ;;
;;          1 - do nothing (initial)    ;;
;;          2 - counter-clockwise pattern;;
;;          3 - clockwise pattern       ;;
;;          4 - return to (1)           ;;
;;        - pressed (only set here)     ;;
;;          1 - it was pressed          ;;
;;          0 - it was not pressed      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

GPIOA   EQU 0x40020000
IDR     EQU 0x10

        AREA INPUT, CODE
        EXPORT input_handler
        IMPORT state
        IMPORT pressed

input_handler           ;; Test for user button
        ;; if pressed, go on and leave
        LDR r3, =pressed        ;; load address of pressed
        LDR r2, [r3]            ;; load pressed to r2
        CMP r2, #1      ;; if (r2 == 1)
        BEQ exit        ;; then exit (don't mess with stuff)
loop    ;; while button is pressed
        ;; load value of button
        LDR r1, =GPIOA          ;; load address of PORTA
        LDR r0, [r1, #IDR]      ;; r0 = PORTA
        AND r0, r0, #0x01       ;; only look at last bit
        ;; cmp value to zero
        CMP r0, #1      ;; if (r0 < 1)
        BLT exit        ;; then exit
        CMP r2, #1      ;; if (pressed == 1), it was set, continue to loop
        BEQ loop
        ;; else increment step
        LDR r1, =state  ;; load the address of state
        LDR r0, [r1]    ;; r0 = state
        ADD r0, r0, #1  ;; increment state
        CMP r0, #4      ;; if (r0 == 4)
        MOVEQ r0, #1    ;; then r0 = 1
        STR r0, [r1]    ;; state = r0
        ;; then indicate the button was pressed
        MOV r2, #1      ;; set r0 to 1
        STR r2, [r3]    ;; store 1 to pressed
        B loop          ;; remain in while loop
exit
        BX r14          ;; return from input_handler
        END
```

tick_timer.c

```
/*****************************************************/
/* Brian Arnberg                                     */
/* Problem Set #6 - System Interrupts                */
/*  tick_timer.c                                     */
/*****************************************************/
/*  SysTick_Handler - once every ms                  */
/*     - calls SVC_Handler every 500ms               */
/*  SVC_Handler - called by SysTick                  */
/*     - handles output based on global variables    */
/*  EXTI0_IRQHandler - triggered by a button press   */
/*     - configured in main.c                        */
/*     - executes the input handler (self-debounced) */
/*     - clears pending status                       */
/*****************************************************/
/*  ''state'' is a global variable.                  */
/*****************************************************/
#include "STM32F4xx.h"
#include "MAIN.h"

volatile uint32_t msTicks;                      /* counts 1ms timeTicks       */

void SysTick_Handler ( void ) {
    msTicks++;
    if (msTicks == 500) {
    msTicks = 0;                                // reset the msTicks
        __asm("SVC 0");                         // Supervisor Call
    }
}

void EXTI0_IRQHandler  (void) { // This simply calls the input_handler
        input_handler();
      EXTI->PR = EXTI_Line0;
    }

void SVC_Handler(void) {
    int i;
    if ((pressed == 1) || (step == 4)) { // if I've pressed a button or reached max step
            step = 0;   // reset the step counter
            pressed = 0;    // clear pressed
            for (i = 12; i <= 15; i++) { // clear the 4 leds
                clear_leds(i);
            }
            } else if (state == 2)  {   //counter clockwise output
            enable_leds(ccw[step]); //output to appropriate LED
            step++; //increment step
        } else if (state == 3) { //clockwise output
            enable_leds(cw[step]);  //output to appropriate LED
            step++; //increment step
        }
}
```

main.c

```
/*****************************************************/
/* Brian Arnberg                                     */
/* Problem Set #6 - Main Program                     */
/*  main.c                                           */
/*****************************************************/
/*  Continuous Loop                                  */
/*  Calls input_handler each loop                    */
/*****************************************************/
/*  ''state'' is a global variable.                  */
/*  ''pressed'' is also a global variable            */
/*****************************************************/
#include <stdio.h>
#include "STM32F4xx.h"
#include "main.h"
volatile uint32_t state;                              /* keep track of the system state
                */
volatile uint32_t pressed;                          /* keep track of whether a button was
    pressed */
volatile uint32_t step;                               /* keep track of the system step
                */
uint32_t ccw[4] = {13,12,15,14};        /* counter clock-wise pin order              */
uint32_t cw[4] = {13,14,15,12};         /* clock-wise pin order                      */

/**   From stm32f4xx_syscfg.c
  * @brief  Selects the GPIO pin used as EXTI Line.
  * @param  EXTI_PortSourceGPIOx : selects the GPIO port to be used as source for
  *          EXTI lines where x can be (A..I).
  * @param  EXTI_PinSourcex: specifies the EXTI line to be configured.
  *           This parameter can be EXTI_PinSourcex where x can be (0..15, except
  *           for EXTI_PortSourceGPIOI x can be (0..11).
  * @retval None
  */
void SYSCFG_EXTILineConfig(uint8_t EXTI_PortSourceGPIOx, uint8_t EXTI_PinSourcex)
{
  uint32_t tmp = 0x00;

  tmp = ((uint32_t)0x0F) << (0x04 * (EXTI_PinSourcex & (uint8_t)0x03));
  SYSCFG->EXTICR[EXTI_PinSourcex >> 0x02] &= ~tmp;
  SYSCFG->EXTICR[EXTI_PinSourcex >> 0x02] |= (((uint32_t)EXTI_PortSourceGPIOx) << (0x04 * (
      EXTI_PinSourcex & (uint8_t)0x03)));
}

/*-----------------------------------------------------------------------------
  Function that initializes Button pins
 *-----------------------------------------------------------------------------*/
void BTN_Init(void) {

  RCC->AHB1ENR  |= ((1UL <<  0) );              /* Enable GPIOA clock          */

  GPIOA->MODER    &= ~((3UL << 2*0)  );         /* PA.0 is input               */
  GPIOA->OSPEEDR  &= ~((3UL << 2*0)  );         /* PA.0 is 50MHz Fast Speed    */
  GPIOA->OSPEEDR  |=  ((2UL << 2*0)  );
  GPIOA->PUPDR    &= ~((3UL << 2*0)  );         /* PA.0 is no Pull up          */
}

/*-----------------------------------------------------------------------------
  initialize LED Pins
 *-----------------------------------------------------------------------------*/
void LED_Init (void) {

  RCC->AHB1ENR  |= ((1UL <<  3) );          /* Enable GPIOD clock                */

  GPIOD->MODER    &= ~((3UL << 2*12) |
                       (3UL << 2*13) |
                       (3UL << 2*14) |
                       (3UL << 2*15)  );    /* PD.12..15 is output               */
  GPIOD->MODER    |=  ((1UL << 2*12) |
                       (1UL << 2*13) |
                       (1UL << 2*14) |
                       (1UL << 2*15)  );
  GPIOD->OTYPER   &= ~((1UL <<  12) |
                       (1UL <<  13) |
                       (1UL <<  14) |
                       (1UL <<  15)  );    /* PD.12..15 is output Push-Pull     */
```

```
71    GPIOD->OSPEEDR  &= ~((3UL << 2*12) |
72                         (3UL << 2*13) |
73                         (3UL << 2*14) |
74                         (3UL << 2*15)  );   /* PD.12..15 is 50MHz Fast Speed     */
75    GPIOD->OSPEEDR  |=  ((2UL << 2*12) |
76                         (2UL << 2*13) |
77                         (2UL << 2*14) |
78                         (2UL << 2*15)  );
79    GPIOD->PUPDR    &= ~((3UL << 2*12) |
80                         (3UL << 2*13) |
81                         (3UL << 2*14) |
82                         (3UL << 2*15)  );   /* PD.12..15 is Pull up              */
83    GPIOD->PUPDR    |=  ((1UL << 2*12) |
84                         (1UL << 2*13) |
85                         (1UL << 2*14) |
86                         (1UL << 2*15)  );
87  }
88
89
90  void EXTILine0_Config(void)
91  {
92      uint32_t tmp = 0;
93      uint8_t tmppriority = 0x00, tmppre = 0x00, tmpsub = 0x0F;
94      /* Connect EXTI Line0 to PA0 pin */
95      SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
96
97
98      tmp = (uint32_t)EXTI_BASE;
99
100     /* Clear EXTI line configuration */
101     EXTI->IMR &= EXTI_Line0;
102     EXTI->EMR &= EXTI_Line0;
103
104     // EXTI_Mode_Interrupt = 0x00
105     tmp += 0x00;
106     *(__IO uint32_t *) tmp |= EXTI_Line0;
107
108     /* Clear Rising Falling edge configuration */
109     EXTI->RTSR &= EXTI_Line0;
110     EXTI->FTSR &= EXTI_Line0;
111
112     // EXTI_Trigger_Rising
113     tmp = (uint32_t)EXTI_BASE;
114     tmp += 0x08;
115     *(__IO uint32_t *) tmp |= EXTI_Line0;
116
117      /* Enable and set EXTI Line0 Interrupt to the lowest priority */
118
119     /* Compute the Corresponding IRQ Priority --------------------------------*/
120     tmppriority = (0x700 - ((SCB->AIRCR) & (uint32_t)0x700))>> 0x08;
121     tmppre = (0x4 - tmppriority);
122     tmpsub = tmpsub >> tmppriority;
123
124     //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
125     tmppriority = 0x01 << tmppre;
126     //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;
127     tmppriority |=  (uint8_t)(0x01 & tmpsub);
128
129     tmppriority = tmppriority << 0x04;
130
131     //NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
132     NVIC->IP[EXTI0_IRQn] = tmppriority;
133
134     /* Enable the Selected IRQ Channels --------------------------------------*/
135     NVIC->ISER[EXTI0_IRQn >> 0x05] =
136     (uint32_t)0x01 << (EXTI0_IRQn & (uint8_t)0x1F);
137 }
138
139 /*****************************************
140  * Generates SWI on particular line
141  *****************************************/
142 void EXTI_GenerateSWInterrupt(uint32_t EXTI_Line)
143 {
144     EXTI->SWIER |= EXTI_Line;
145 }
146
```

```
147 int main (void) {
148     state = 1;              // Initialize State
149     pressed = 0;            // Make sure pressed is clear
150
151   SystemCoreClockUpdate();                          /* Get Core Clock Frequency   */
152   if (SysTick_Config(SystemCoreClock / 1000 )) { /* SysTick 1 msec interrupts  */
153     while (1);                                      /* Capture error              */
154   }
155
156     /* Initialize the LEDS and the buttons */
157     LED_Init();
158     BTN_Init();
159     EXTILine0_Config();
160     EXTI_GenerateSWInterrupt(EXTI_Line0);
161
162
163     for (;;) {          // loop forever
164     }
165 }
```

MAIN.h

```
1  /*****************************************/
2  /*   Brian Arnberg                       */
3  /*    main.h: primary include file for this   */
4  /*            project                    */
5  /*****************************************/
6
7  #ifndef __MAIN_H
8  #define __MAIN_H
9  #define EXTI_Line0        ((uint32_t)0x00001)     /*!< External interrupt line 0 */
10 #define EXTI_PortSourceGPIOA        ((uint8_t)0x00)
11 #define EXTI_PinSource0             ((uint8_t)0x00)
12
13 #include "stm32f4xx.h"
14 #include "stm32f4_discovery.h"
15
16 extern volatile uint32_t state;             // initialize state
17 extern volatile uint32_t step;              // initialize step
18 extern volatile uint32_t pressed;           // initialize pressed
19 extern uint32_t ccw[4];                  // initialize counter clockwise array
20 extern uint32_t cw[4];                   // initialize clockwise array
21 extern volatile uint32_t msTicks;     /* counts 1ms timeTicks       */
22
23
24 extern void EXTI0_IRQHandler(void);     // External Interupt Handler
25 extern void input_handler(void);    // declare input handler
26 extern void enable_leds(int PIN);   // declare enable_leds (in output_handler.s)
27 extern void clear_leds(int PIN);    // declare disable_leds (in output_handler.s)
28
29 #endif
```

2 Book Questions

From the end of Chapter 3, answer questions Q3-24, Q3-25 and Q3-26. These deal with cache memory.

Q3 - 24: Provide exmples of how each of the following can occur in a typical program:

compulsory miss This can occur at system start-up. At system start-up, no locations have been used, so all memory requests for non-contiguous memory locations will be compulsory misses.

capacity miss A capacity miss can occur whenever the CPU needs access to more memory locations than the cache can store. This type of miss will occur whenever the cache is completely occupied. When the CPU needs access to memory not currently in the cache, there will be a capacity miss.

conflict miss A conflict miss can occur when a memory location in the cache is overwritten prematurely. Location A, for instance, is stored in the cache. The CPU then looks for location B, which is not in the cache, and then loads it where location A was stored. Then the CPU looks for location A, which is no longer in the cache, but was where location B now is. A conflict miss has occured.

Q3 - 25: What is the average memory access time of a machine whose hit rate is 96%, with a cache access time of 3ns and a main memory access time of 70ns?

$$H_c = .96 \colon T_c = 3ns \colon T_m = 70ns \colon T_a = ?$$

$$T_a = H_c T_c + (1 - H_c)(T_m)$$

$$T_a = .96 \times 3ns + (1 - .96)(70ns)$$

$$T_a = 5.68ns$$

Q3 - 26: If we want an average memory access time of 6.5ns, our cache access time is 5ns, and our main memory access time is 80ns, what cache hit rate must we achieve?

$$T_a = 6.5ns \colon T_c = 5ns \colon T_m = 80ns \colon H_c = ?$$

$$T_a = H_c T_c + (1 - H_c)(T_m)$$

$$H_c = \frac{T_a - T_m}{T_C - T_m}$$

$$H_c = \frac{6.5ns - 80ns}{5ns - 80ns}$$

$$H_c = .98 = 98\%$$