Assignment #8: Problem Set 5 - Due Fri. 02/22/13

To practice with parallel I/O ports, write a program that creates and displays one of two patterns on the 4 LEDs on the Discovery board. Under the control of the user push button (the blue button).
LED3 (orange) = I/O port pin PD13
LED4 (green) = I/O port pin PD12
LED5 (red) = I/O port pin PD14
LED6 (blue) = I/O port pin PD15
User button (blue) = I/O port pin PA0

The program is to operate as follows.

1. Initially, all LEDs are off.
2. On the first press of the user button, the LEDs should be turned on with the following pattern: LED3 – LED4 – LED6 – LED5 – ALL OFF (each LED remains ON until ALL OFF) This pattern is to be repeated until the next button press. Note that you should see LEDs turn on in a counter-clockwise circle. Each step of the pattern is to be held for exactly one-half second.
3. On the next press of the user button, the LED pattern is to change to the following: LED3 – LED5 – LED6 – LED4 – ALL OFF (each LED remains ON until ALL OFF) This pattern is to be repeated until the next button press. Note that you should see LEDs turn on in a clockwise circle. Each step of the pattern is to be held for exactly one-half second.
4. On the next button press, return to step 1 (all LEDs off). Then repeat steps 1-4 continuously.

The program is to contain the following modules:

1. An output handler, written in ARM assembly language, which writes patterns to the LEDs.
2. An input handler, written in ARM assembly language, which tests the user button, and sets a global variable.
3. A system tick timer interrupt handler, written in C, which is activated every one-half second. This routine should call the output handler, if the LEDs are to be changed.
4. A main program, written in C, which executes in a continuous loop, calling the input handle every time through the loop.
5. The "startup code" for the STM32F4-Discovery board, as found in the Keil installation directory: C:/Keil/ARM/Boards/ST/STM32F4-Discovery/Blinky
6. The STM32F4xx microcontroller's "include file", found in the Keil installation directory: C:/Keil/ARM/INC/ST/STM32F4xx/stm32f4xx.h

Execution

This assignment was completed by writing two assembly language program files, two C files, and one header file. The project that these files where included in was an edited version of the Keil Blinky project. The contents of the "Blinky" directory were copied into a new directory. Then *blink.c* and *leds.c* were removed, and replaced by the five before-mentioned files. The "startup code" and "include file" were both left unchanged. The program works, though occasionally the "user" button seems to detect multiple key-presses when it has only been pressed once, which causes the system to skip one of the above mentioned operation states.

output_handler.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    Brian Arnberg                         ;;
;;    Problem Set #5 - Output Handler       ;;
;;     output_handler.s                     ;;
;;      - writes patterns to the LEDS       ;;
;;      - called by system tick timer       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      - takes no arguments                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      - global variables                  ;;
;;         - state (represents a pattern)   ;;
;;           1 - do nothing (initial)       ;;
;;           2 - counter-clockwise pattern;;
;;           3 - clockwise pattern          ;;
;;           4 - return to (1)              ;;
;;         - pressed (only reset here)      ;;
;;           1 - it was pressed             ;;
;;           0 - it was not pressed         ;;
;;         - step ( 0,4,8,12,16)            ;;
;;            indicates what to do          ;;
;;                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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 output_handler
        IMPORT state
        IMPORT pressed

output_handler
        LDR r3, =step           ;; get address of step
        LDR r2, [r3]            ;; r2 = step
        LDR r1, =pressed        ;; get address of pressed
        LDR r0, [r1]            ;; r0 = pressed
        CMP r0, #1              ;; if (pressed != 1) (i.e. pressed == 0)
        BNE step_check          ;; jump to step check
        MOV r2, #0              ;; then reset step
        MOV r0, #0              ;; then reset pressed (to leave it alone)
        STR r0, [r1]            ;; and store the value of pressed
        B clear_leds            ;; clear the LEDS for new state

step_check      ;; check the step
        CMP r2, #16             ;; if (step == 5)
        MOVEQ r2, #0            ;;  reset step
        BEQ clear_leds          ;;  clear_leds
                                ;; else
state_check     ;; check the state
        LDR r1, =state          ;; get address of state
        LDR r0, [r1]            ;; r0 = state
        CMP r0, #2              ;; compare state to #2
        BEQ counter_clockwise   ;; if (r0 == 2), counter_clockwise
        BGT clockwise           ;; else if (r0 > 2), clockwise
                                ;; else, clear_leds
clear_leds      ;; clear the LEDS
        MOV r4, #1              ;; make r4 = 0x0001
        MOV r5, #12             ;; make r5 = 0x000C
        MOV r4, r4, lsl r5      ;; bit shift r4 by r5
        LDR r6, =GPIOD          ;; address of GPIOD
loop_clear
        STR r4, [r6, #BSRRH]    ;; reset the pin
        MOV r4, r4, lsl #1      ;; bit shift r4 once
        CMP r4, #0x8000         ;; if (r4 <= 15)
        BLE loop_clear          ;; then branch to loop_clear
        B exit                  ;; branch to exit

counter_clockwise
        MOV r4, #1              ;; make r4 = 0x0001
        LDR r7, =ccw            ;; r7 points to ccw
        LDR r5, [r7, r2]        ;; r5 is the step value for ccw
        MOV r4, r4, lsl r5      ;; bit shift r4 by r5
        LDR r6, =GPIOD          ;; address of GPIOD
```

```
        STR r4, [r6, #BSRRL]     ;; set the pin
        ADD r2, r2, #4           ;; increment step
        B exit                   ;; branch to exit

clockwise
        MOV r4, #1               ;; make r4 = 0x0001
        LDR r7, =cw              ;; r7 points to ccw
        LDR r5, [r7, r2]         ;; r5 is the step value for ccw
        MOV r4, r4, lsl r5       ;; bit shift r4 by r5
        LDR r6, =GPIOD           ;; address of GPIOD
        STR r4, [r6, #BSRRL]     ;; set the pin
        ADD r2, r2, #4           ;; increment step
        B exit                   ;; branch to exit

exit
        STR r2, [r3]             ;; store the final step value
        BX r14                   ;; return from output handler
        ;; define steps
ccw     DCD     13, 12, 15, 14   ;; steps for counter-clockwise
cw      DCD     13, 14, 15, 12   ;; steps for clockwise
        ;; store variables
        AREA data1, DATA
step    SPACE 4                  ;; set space aside to store step
        END
```

input_handler.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    Brian Arnberg                       ;;
;;    Problem Set #5 - Input Handler      ;;
;;     input_handler.s                    ;;
;;      - tests the user button           ;;
;;      - sets a global variable          ;;
;;      - called each time the main       ;;
;;        loop is executed                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      - 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 #5 - System Tick Timer                */
/*  tick_timer.c                                     */
/*****************************************************/
/*  Activates once every half second                 */
/*  Calls output_handler (iff LEDS should change)    */
/*****************************************************/
/*  ''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
        if ((state != 1) || (pressed == 1)) {   // if not state 1
            output_handler();                   // write to LEDS
        }                                       // else, return

    }
}
```

main.c

```
/*****************************************************/
/* Brian Arnberg                                     */
/* Problem Set #5 - 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 */

/*-------------------------------------------------------------------------
  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
 *-------------------------------------------------------------------------*/
```

```c
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      */
  GPIOD->OSPEEDR  &= ~((3UL << 2*12) |
                       (3UL << 2*13) |
                       (3UL << 2*14) |
                       (3UL << 2*15)  );   /* PD.12..15 is 50MHz Fast Speed      */
  GPIOD->OSPEEDR  |=  ((2UL << 2*12) |
                       (2UL << 2*13) |
                       (2UL << 2*14) |
                       (2UL << 2*15)  );
  GPIOD->PUPDR    &= ~((3UL << 2*12) |
                       (3UL << 2*13) |
                       (3UL << 2*14) |
                       (3UL << 2*15)  );   /* PD.12..15 is Pull up               */
  GPIOD->PUPDR    |=  ((1UL << 2*12) |
                       (1UL << 2*13) |
                       (1UL << 2*14) |
                       (1UL << 2*15)  );
}


int main (void) {
    state = 1;              // Initialize State
    pressed = 0;            // Make sure pressed is clear

  SystemCoreClockUpdate();                          /* Get Core Clock Frequency   */
  if (SysTick_Config(SystemCoreClock / 1000 )) { /* SysTick 1 msec interrupts  */
    while (1);                                      /* Capture error              */
  }

    /* Initialize the LEDS and the buttons */
    LED_Init();
    BTN_Init();

    for (;;) {          // loop forever
    input_handler();        // call input_handler
    }
}
```

MAIN.h

```c
/*****************************************/
/*   Brian Arnberg                       */
/*    main.h: primary include file for this   */
/*            project                     */
/*****************************************/

#ifndef __MAIN_H
#define __MAIN_H

extern volatile uint32_t state;                     // initialize state
extern volatile uint32_t pressed;                   // initialize pressed
extern void input_handler(void);                    // declare input handler
extern void output_handler(void);                   // declare output handler
extern volatile uint32_t msTicks;                      /* counts 1ms timeTicks       */


#endif
```