Embedded System Design - ECE 561

Challenge Assignment 1 : 2/16/2016

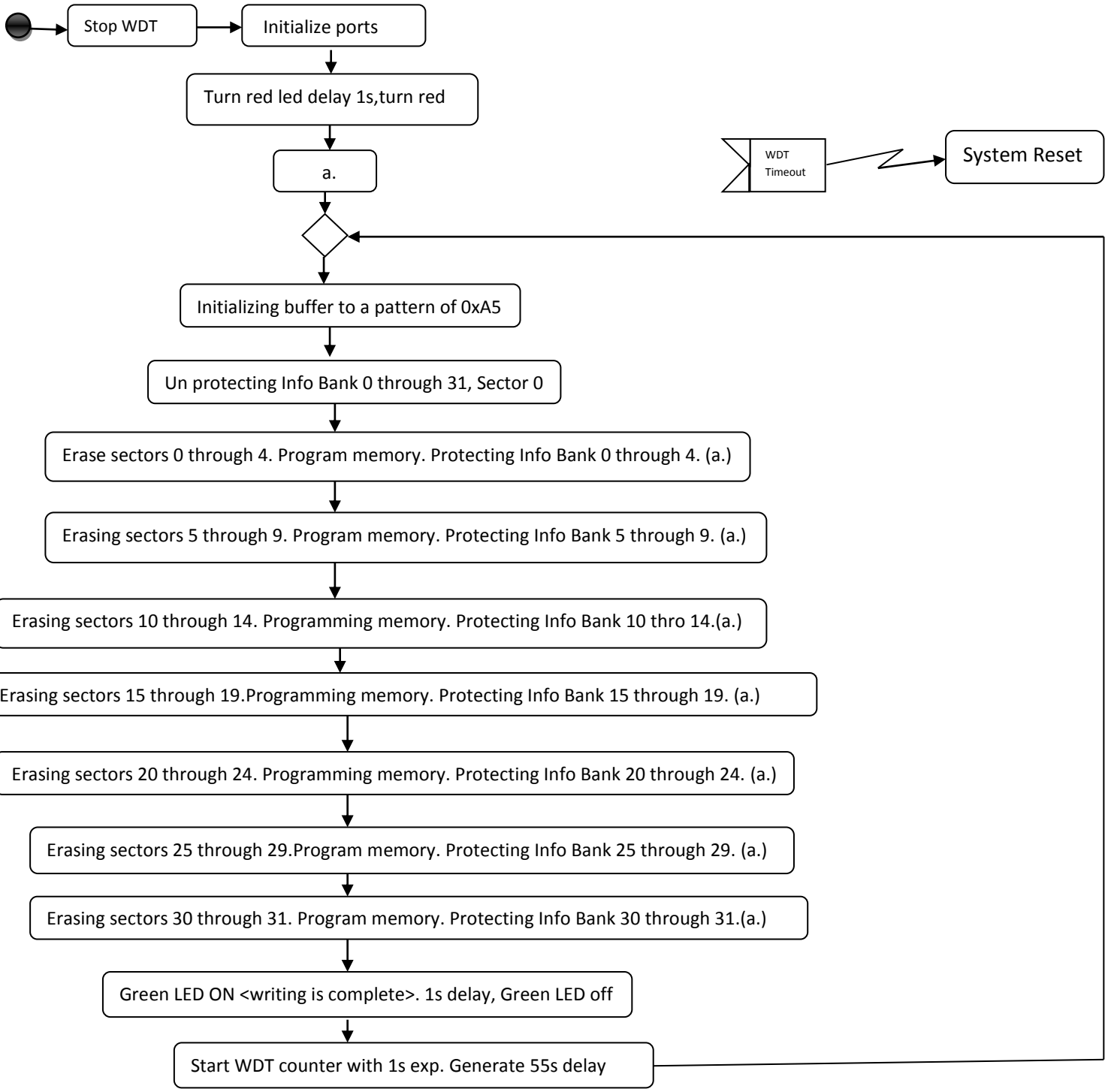
ID: 200108828

Anusha Ravilla

Instructor: Dr. Corey Graves

Youtube Link: https://youtu.be/6-lamyfc4A4

UML Activity Diagram

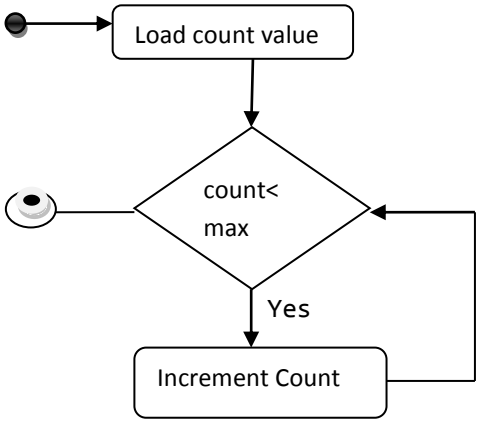


a. start WDT counter with a 1 s expiration. Repeat empty loop if the S2 button is pressed

* while erasing { Within this function, the API will automatically try to erase the maximum number of tries. If it fails, trap in an infinite loop }

* Within this function, the API will automatically try to program the maximum number of tries. If it fails, trap inside an infinite loop */

Delay Loop



Block Diagram:

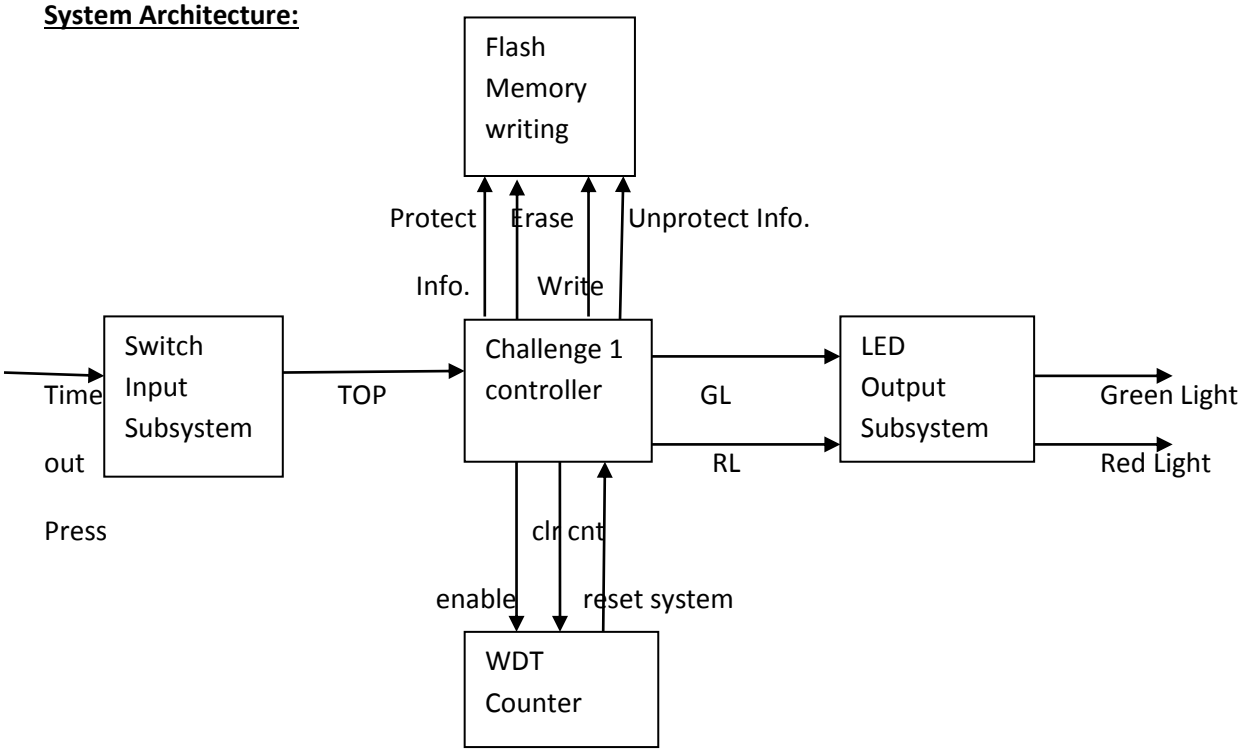
Level 0:

Functional Requirement Table for Level 0 block:

Module	Challenge1
Inputs	Button Press: To cause a WDT Timeout
Outputs	Red LED: Indicating System Reset Green LED: Indicating the completion of writing
Functionality	Erases and writes 128 kB Flash main memory locations once every 1 minute. In doing this, the WDT_A is configured to reset the system every 1 second, unless software prevents this reset, by clearing the WDT counter. Pushing the button leads to trapping in a while loop which will lead to a system reset.

Level 1:

System Architecture:



Module	**Flash Memory writing**
Inputs	Protect Info. - Protects the information in the sector preventing writes to the sector. Unprotect Info. - Unprotect the information in the sector facilitating writes to the sector. Erase - Erase the information in the memory location specified Program - Program the information into the specified memory location.
Outputs	-
Functionality	Managing the Flash memory by unprotecting the memory, erasing the memory and writing into it and protecting the data written into it.

Module	**Switch Input Subsystem**
Inputs	Button Press: Button Press
Outputs	Time out press
Functionality	Due to the Time out press, the program is trapped in an infinite while loop leading to a WDT time out and hence a system reset.

Module	**WDT Counter**
Inputs	Clear Count: Clear the WDT counter Enable: Enable the WDT counter
Outputs	System Reset
Functionality	WDT_A is configured to reset the system after a time lapse specified. (1 sec)

Module	**LED Output Subsystem**
Inputs	GL: The pin for Green one is on RL: The pin for Red one is on
Outputs	Green Light : Is switched on to indicate completion of a write to 128 kb memory Red Light : Is Switched on to indicate system reset.
Functionality	Green and Red Lights are switch on for one second and then switched off to indicate the respective completion and system reset.

Module	**Challenge 1 controller**
Inputs	TOP: Generated from push button. Reset System: Generated due to a WDT timeout
Outputs	Enable WDT: Switch on WDT counter with a time lapse Clear WDT count: Reset the WDT counter Protect Info.: Protect the sector in order to prevent further writes Unprotect Info: Unprotect the sector in order to facilitate further writes Erase: Erase the memory Write: Write to the memory GL: Give a signal to enable the green light to glow RL: Give a signal to enable the red light to glow
Functionality	Generate unprotect signal, followed by erase , program and protect signals. Send GL signal to signify the switching ON of green LED. Enable WDT and keep clearing WDT to prevent system reset. In the event of system reset, send RL signal.

Main Code:

```
/* DriverLib Includes */
#include "driverlib.h"
/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "msp.h"
#define CALIBRATION_START1 0x00020000 // sector 0 of Bank 0 starts at 20000
#define CALIBRATION_START2 0x00025000 // sector 6 of Bank 0 starts at 25000
#define CALIBRATION_START3 0x0002A000 // sector 11 of Bank 0 starts at 2A000
#define CALIBRATION_START4 0x0002F000 // sector 16 of Bank 0 starts at 2F000
#define CALIBRATION_START5 0x00034000 // sector 21 of Bank 0 starts at 34000
#define CALIBRATION_START6 0x00039000 // sector 26 of Bank 0 starts at 39000
#define CALIBRATION_START7 0x0003e000 // sector 31 of Bank 0 starts at 3e000
void port_init()                //      port initialization function
{       P1DIR |= BIT0;                  // make P1.0 an output
        P1DIR &= ~BIT4;                 // make P1.4 and input
        P1REN |= BIT4;                  // enable pull resistor on P1.4
        P1OUT |= BIT4;                  // make it a pull-up resistor
        P2DIR |= BIT1;                  // make P2.1 an output
}
void delay_1sec()       // 1 second delay for 1.5 MHz clock
{
        int  dcntr;                     // delay counter variable
        for (dcntr=0;dcntr<57292;dcntr++);              //~1 second delay loop
        return;
}
void delay_200ms()                                      // 1 ms delay for 1.5 MHz system clock
{       int  dcntr;                                             // delay counter variable
        for (dcntr=0;dcntr<(57092/4);dcntr++);         // ~0.20 second delay loop
        return;
}
/* Statics */
uint8_t simulatedCalibrationData1[20480];
uint8_t simulatedCalibrationData2[8184];

int main(void)
{
           /* Since this pr ogram has a huge buffer that simulates the calibration data,
                * halting the watch dog is done in the reset ISR to avoid a watchdog
                * timeout during the zero     */
     WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer
          int delays;
     port_init();                               // initialize needed port pins
          P1OUT |= BIT0;                            // turn red LED on
                 delay_1sec();                          // wait 1 second
                 P1OUT &= ~BIT0;                        // turn red LED off
        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;             //start WDT counter with a 1 s
expiration
   while (1)
            { delays=0;                 //counter for the overall delays
          while ((P1IN & BIT4)==0);             //repeat empty loop if S2 button is pressed
    /* Initializing our buffer to a pattern of 0xA5 */
    memset(simulatedCalibrationData1, 0xA5, 20480);
    memset(simulatedCalibrationData2, 0xA5, 8184);
    /* Unprotecting Info Bank 0 through 31, Sector 0  */
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR0);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR1);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR2);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR3);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR4);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR5);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR6);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR7);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR8);
    MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR9);
```

```
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR10);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR11);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR12);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR13);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR14);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR15);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR16);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR17);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR18);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR19);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR20);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR21);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR22);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR23);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR24);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR25);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR26);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR27);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR28);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR29);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR30);
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR31);
WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;          //start WDT counter with a 1 s expiration
     while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is pressed
/* Trying to erase the sector. Within this function, the API will
    automatically try to erase the maximum number of tries. If it fails,
     trap in an infinite loop */
if(!MAP_FlashCtl_eraseSector(CALIBRATION_START1))
    while(1);
/* Trying to program the memory. Within this function, the API will
    automatically try to program the maximum number of tries. If it fails,
    trap inside an infinite loop */
if(!MAP_FlashCtl_programMemory(simulatedCalibrationData1,
        (void*) CALIBRATION_START1, 20480))
            while(1);
WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;          //start WDT counter with a 1 s expiration
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR0);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR1);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR2);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR3);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR4);
while ((P1IN & BIT4)==0);           //repeat empty loop if the S2 button is pressed

if(!MAP_FlashCtl_eraseSector(CALIBRATION_START2))
    while(1);
if(!MAP_FlashCtl_programMemory(simulatedCalibrationData1,
          (void*) CALIBRATION_START2, 20480))
              while(1);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR5);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR6);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR7);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR8);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR9);
WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;          //start WDT counter with a 1 s expiration
while ((P1IN & BIT4)==0);           //repeat empty loop if the S2 button is pressed
if(!MAP_FlashCtl_eraseSector(CALIBRATION_START3))
       while(1);
if(!MAP_FlashCtl_programMemory(simulatedCalibrationData1,
             (void*) CALIBRATION_START3, 20480))
                 while(1);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR10);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR11);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR12);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR13);
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR14);
WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;          //start WDT counter with a 1 s expiration
if(!MAP_FlashCtl_eraseSector(CALIBRATION_START4))
       while(1);
if(!MAP_FlashCtl_programMemory(simulatedCalibrationData1,
```

```
                (void*) CALIBRATION_START4, 20480))
                    while(1);
    while ((P1IN & BIT4)==0);           //repeat empty loop if the S2 button is pressed
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR15);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR16);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR17);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR18);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR19);

        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;             //start WDT counter with a 1 s
expiration

        if(!MAP_FlashCtl_eraseSector(CALIBRATION_START5))
            while(1);
    if(!MAP_FlashCtl_programMemory(simulatedCalibrationData1,
                (void*) CALIBRATION_START5, 20480))
                    while(1);
    while ((P1IN & BIT4)==0);           //repeat empty loop if the S2 button is pressed
    MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR20);
    MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR21);
    MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR22);
    MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR23);
    MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR24);

    WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;          //start WDT counter with a 1 s expiration
    if(!MAP_FlashCtl_eraseSector(CALIBRATION_START6))
            while(1);
    if(!MAP_FlashCtl_programMemory(simulatedCalibrationData1,
                (void*) CALIBRATION_START6, 20480))
                    while(1);
    while ((P1IN & BIT4)==0);           //repeat empty loop if the S2 button is pressed
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR25);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR26);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR27);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR28);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR29);

        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;             //start WDT counter with a 1 s
expiration
        if(!MAP_FlashCtl_eraseSector(CALIBRATION_START7))
                while(1);
        if(!MAP_FlashCtl_programMemory(simulatedCalibrationData2,
                    (void*) CALIBRATION_START6, 8184))
                        while(1);
        while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is pressed
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR30);
        MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR31);

        delays=0;
                P2OUT |= BIT1;                          // turn green LED on
               while(delays<5)                                          //repeat the 0.2 second
delays 5 times before toggling green LED
                       {       WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;                    //start
WDT counter with a 1 s expiration
                delay_200ms();                                          // delay for 0.2 seconds
                while ((P1IN & BIT4)==0);                                   //repeat empty loop if
the S2 button is pressed
                delays++;                                                   // increment the 0.2
second delay count
                       }
               WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;                    //start WDT counter with
a 1 s expiration
                                P2OUT &= ~BIT1;                                                 // turn
green LED off
        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;             //start WDT counter with a 1 s
expiration
        delays=0;
        ////////long delay here
```

```
                while(delays<36)        //repeat the 1.8 second delays 30 times to generate approx delay
of 50 sec
                {
                        delay_200ms();          // delay for 0.2 seconds
                        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;              //start WDT counter with
a 1 s expiration
                        while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is
pressed
                        delays++;                       // increment the 0.2 second delay count
                        delay_200ms();          // delay for 0.2 seconds
                        delay_200ms();          // delay for 0.2 seconds
                        while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is
pressed
                        delay_200ms();          // delay for 0.2 seconds
                        delay_200ms();          // delay for 0.2 seconds
                        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;              //start WDT counter with
a 1 s expiration
                        delay_200ms();          // delay for 0.2 seconds
                        delay_200ms();          // delay for 0.2 seconds
                        while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is
pressed
                        delay_200ms();          // delay for 0.2 seconds
                        WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;              //start WDT counter with
a 1 s expiration
                                        delay_200ms();          // delay for 0.2 seconds
                        while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is
pressed
                }
                WDTCTL = WDTPW |WDTSSEL_3|WDTIS_4|WDTCNTCL;             //start WDT counter with a 1 s
expiration
                delays=0;
                while ((P1IN & BIT4)==0);               //repeat empty loop if the S2 button is pressed

    }
}
```