```
================================================================================
===
Arpit Savarkar
Buffaiti Assignment
================================================================================
===

<cap_sensor.h>
================================================================================
===
/*
 * cap_sensor.h
 *
 *  Created on: Oct 3, 2020
 *      Author: root
 */

#ifndef CAP_SENSOR_H_
#define CAP_SENSOR_H_


#include "MKL25Z4.h"
#include "statemachine.h"
#include "stdint.h"
#include "stdio.h"
#include "fsl_debug_console.h"

/* ************************************************************************
 * Macros
 ************************************************************************/
#define SCAN_OFFSET 588  // Offset for scan range
#define SCAN_DATA TSI0->DATA & 0xFFFF // Accessing the bits held in
TSI0_DATA_TSICNT
#define NOISE_LOW 25
#define NOISE_HIGH 60000

/* ************************************************************************
 * Prototypes
 ************************************************************************/

/*   @brief  Initialization of Capacitive Sensor (Clocks Gating etc) */
void CAP_Init(void);


/**
 *   @brief  Helper Function to return back the Value sensed by the capacitive
sensor
 *
 *
 *   @param  void
 *
 *   @return  Boolean if Touch within threshold
 */
uint16_t CAP_Scan(void);


#endif /* CAP_SENSOR_H_ */
================================================================================
===
```

<cap_sensor.c>

```
================================================================================
===
/*
 * cap_sensor.c
 *
 *  Created on: Oct 3, 2020
 *      Author: Arpit Savarkar / arpit.savarkar@colorado.edu
 *
 *      Refereneces : (Textbook) Embedded Systems Fundamentals with Arm Cortex-M
based MicroControllers
 */


#include "MKL25Z4.h"
#include "cap_sensor.h"



/* ************************************************************************
 * Code
 ************************************************************************/

/*   @brief  Initialization of Capacitive Sensor (Clocks Gating etc) */
void CAP_Init(void) {

      // Enable clock for TSI PortB 16 and 17
            SIM->SCGC5 |= SIM_SCGC5_TSI_MASK;

            TSI0->GENCS = TSI_GENCS_OUTRGF_MASK |  // Out of range flag, set to 1
to clear
                  TSI_GENCS_MODE(0U) |  // Set at 0 for capacitive sensing.  Other
settings are 4 and 8 for threshold detection, and 12 for noise detection
                  TSI_GENCS_REFCHRG(0U) | // 0-7 for Reference charge
                  TSI_GENCS_DVOLT(0U) | // 0-3 sets the Voltage range
                  TSI_GENCS_EXTCHRG(0U) | //0-7 for External charge
                  TSI_GENCS_PS(0U) | // 0-7 for electrode prescaler
                  TSI_GENCS_NSCN(31U) | // 0-31 + 1 for number of scans per
electrode
                  TSI_GENCS_TSIEN_MASK | // TSI enable bit
                  TSI_GENCS_STPE_MASK | // Enables TSI in low power mode
                  TSI_GENCS_EOSF_MASK ; // End of scan flag, set to 1 to clear
#ifdef DEBUG
      PRINTF("\n\r Clock Gating and Initialization of Capacitive Sensor Complete
");
#endif
}


/**
 *   @brief  Helper Function to return back the Value sensed by the capacitive
sensor
 *
 *
 *   @param  void
 *
 *   @return  Boolean if Touch within threshold
 */
```

```
uint16_t CAP_Scan(void) {

      int scan = 0;
      int scan_flag = 0;
      TSI0->DATA = TSI_DATA_TSICH(9U); // Using channel 9 of the TSI
      TSI0->DATA |= TSI_DATA_SWTS_MASK; // Software trigger for scan
      while (!(TSI0->GENCS & TSI_GENCS_EOSF_MASK )) // waiting for the scan to
complete 32 times
            ;
      scan = SCAN_DATA;
      TSI0->GENCS |= TSI_GENCS_EOSF_MASK; // Reset end of scan flag
      scan_flag = scan - SCAN_OFFSET;
      if( scan_flag <= 20 ||  scan_flag > 60000) {
            return 0;
      }
      return 1;
}
```

==========

<led.c>

==========

```
/*
 * pwm_led.c
 *
 *  Created on: Sep 30, 2020
 *      Author: Arpit Savarkar , arpit.savarkar
 */

#include "led.h"
#include "fsl_debug_console.h"


/**
 *   @brief  This function initializes the PWM Functionalities and TPM settings for
PWM Control
 *                       and clock gating functionalities
 *
 *   @param  PWM Levels - Described as a period
 *   @return  none
 */
void Init_LED_PWM(uint16_t period) {

      // Enable Clock to PORTB and PORTD for (Red, Green) and Blue LED
      SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;
      SIM->SCGC6 |= SIM_SCGC6_TPM0_MASK | SIM_SCGC6_TPM2_MASK;

      // Enable the FlexibleTImer configs that enable PWM capabilities
      // Red
      PORTB->PCR[RED_LED_PIN_POS] &= ~PORT_PCR_MUX_MASK;
      PORTB->PCR[RED_LED_PIN_POS] |= PORT_PCR_MUX(3);

      // Green
      PORTB->PCR[GREEN_LED_PIN_POS] &= ~PORT_PCR_MUX_MASK;
      PORTB->PCR[GREEN_LED_PIN_POS] |= PORT_PCR_MUX(3);

      //Blue
```

```
      PORTD->PCR[BLUE_LED_PIN_POS] &= ~PORT_PCR_MUX_MASK;
      PORTD->PCR[BLUE_LED_PIN_POS] |= PORT_PCR_MUX(4);

      // Configure TPM
      // Setting Clock Source at CPU rate - 48 Mhz
      SIM->SOPT2 |= (SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK);

      TPM0->MOD = period-1;
      TPM2->MOD = period-1;

      // Prescalar set to 1, no division
      TPM0->SC = TPM_SC_PS(0);
      TPM2->SC = TPM_SC_PS(0);

      // Contiunue Operation in Debug Mode
      TPM0->CONF |= TPM_CONF_DBGMODE(3);
      TPM2->CONF |= TPM_CONF_DBGMODE(3);

      // Channel Based Setup to Edge-alligned active-low PWM
      TPM2->CONTROLS[0].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK;
      TPM2->CONTROLS[1].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK;
      TPM0->CONTROLS[1].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK;

      // Setting Initial Duty cycle to 0
      TPM2->CONTROLS[0].CnV = 0;
      TPM2->CONTROLS[1].CnV = 0;
      TPM0->CONTROLS[1].CnV = 0;

      // Start TPM
      TPM2->SC |= TPM_SC_CMOD(1);
      TPM0->SC |= TPM_SC_CMOD(1);
#ifdef DEBUG
      PRINTF("\n\r Clock Gating and Initialization of TPM for PORTB and PORTD
Complete ");
#endif
}


/**
 *   @brief  This function Sets the LED PWM converting from hex to PWM_PERIOD (0 -
48000)
 *
 *   @param  red : Hex Value of RED Led
 *   @param  green : Hex Value of GREEN Led
 *   @param  blue : Hex Value of BLUE Led
 *   @return  none
 */
void LED_SET(unsigned int red, unsigned int green,unsigned int blue) {

      BLUE_PWM = (blue * PWM_PERIOD) / 0xFF; // Blue
      RED_PWM = (red * PWM_PERIOD) / 0xFF; // Red
      GREEN_PWM = (green * PWM_PERIOD) / 0xFF; // Green

}
================================================================================
===

<led.h>
================================================================================
```

```
===
/*
 * pwm_led.h
 *
 *  Created on: Sep 30, 2020
 *      Author: Arpit Savarkar / arpit.savarkar@colorado.edu
 */

#ifndef LED_H_
#define LED_H_

#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include <stdbool.h>
#include <stdio.h>


/
*******************************************************************************
**************
                                  Macros
*******************************************************************************
**************/
#define PWM_PERIOD (48000)

#define BLUE_PWM TPM0->CONTROLS[1].CnV
#define RED_PWM TPM2->CONTROLS[0].CnV
#define GREEN_PWM TPM2->CONTROLS[1].CnV

#define RED_LED_PIN_POS (18)
#define GREEN_LED_PIN_POS (19)
#define BLUE_LED_PIN_POS (1)


/
*******************************************************************************
**************
                               Function Prototypes
*******************************************************************************
***************/

/**
 *   @brief  This function initializes the PWM Functionalities and TPM settings for
PWM Control
 *                       and clock gating functionalities
 *
 *   @param  PWM Levels - Described as a period
 *   @return  none
 */
void Init_LED_PWM(uint16_t period);


/**
 *   @brief  This function Sets the LED PWM converting from hex to PWM_PERIOD (0 -
48000)
 *
 *   @param  red : Hex Value of RED Led
 *   @param  green : Hex Value of GREEN Led
 *   @param  blue : Hex Value of BLUE Led
```

```
 *   @return  none
 */
void LED_SET(unsigned int red, unsigned int green, unsigned int blue);

#endif /* LED_H_ */
```

```
<statemachine.c>
```

```
/*
 * statemachine.c
 *
 *  Created on: Oct 2, 2020
 *      Author: Arpit Savarkar / arpit.savarkar@colorado.edu
 */


#include "statemachine.h"
#include "switch.h"
#include "temp_systick.h"
#include "cap_sensor.h"
#include "led.h"

#ifdef DEBUG
      #define MSG_DEBUG PRINTF
#else // non-debug mode - get rid of printing message
      #define MSG_DEBUG(...)
#endif

/
**************************************************************************************
**************
                                  Global Flags
**************************************************************************************
**************/
volatile double val;
volatile int flag_250msec;
volatile int flag_750msec;
volatile int flag_Switch;


/
**************************************************************************************
**************
                                  Functions
**************************************************************************************
**************/

/* Structure for Handling */
struct traffic_light_t{
      color_t color_go;
      color_t color_stop;
      color_t color_warn;
      color_t color_crosswalk;
      state_t state;
      event_t event;
} traffic_light_t  = {
```

```
      .color_go.red   = ((HEX_GO >> 16) & 0xFF),
      .color_go.green = ((HEX_GO >> 8) & 0xFF),
      .color_go.blue  = (HEX_GO & 0xFF),
      .color_stop.red   = ((HEX_STOP >> 16) & 0xFF),
      .color_stop.green = ((HEX_STOP >> 8) & 0xFF),
      .color_stop.blue  = (HEX_STOP & 0xFF),
      .color_warn.red   = ((HEX_WARNING >> 16) & 0xFF),
      .color_warn.green = ((HEX_WARNING >> 8) & 0xFF),
      .color_warn.blue  = (HEX_WARNING & 0xFF),
      .color_crosswalk.red   = ((HEX_CROSSWALK >> 16) & 0xFF),
      .color_crosswalk.green = ((HEX_CROSSWALK >> 8) & 0xFF),
      .color_crosswalk.blue  = (HEX_CROSSWALK & 0xFF),
      .state       = s_STOP,
      .event       = e_Void,
};


/**
 *   @brief - Function to update event incase of succesfull Capacitive Touch
 *
 *   @param  goal:  (color_t) goal color to be set
 *   @param  goal:  (event_t) event of the statemachine
 *   @return  (int) 1 if sucessfull touch , else 0
 */
int cap_touch_action(color_t* goal, event_t* event) {
      int flag = 0;
      /* Sets flag if touch detected */
      flag = CAP_Scan();
      if(flag) {
            MSG_DEBUG("\n\r Capacitive Touch Detected at sec_time: %d",
now()/1000 );
            /* Updates the goal color to CROSSWALK color state */
            *goal = traffic_light_t.color_crosswalk;
            /* Update event*/
            *event = e_TransitionTimeout;
            flag = 0;
            return 1;
      }
      return 0;
}


/**
 *   @brief - Function to update event incase of succesfull Button Press
 *
 *   @param  goal:  (color_t) goal color to be set
 *   @param  goal:  (event_t) event of the statemachine
 *   @return  (int) 1 if sucessfull touch , else 0
 */
int switch_action(color_t* goal, event_t* event) {

      if(flag_Switch) {
            MSG_DEBUG("\n\r Switch Button Detected at sec_time: %d", now()/1000 );
            /* Update goal color*/
            *goal = traffic_light_t.color_crosswalk;
            /* Update event*/
            *event = e_TransitionTimeout;
            PORTA->ISFR = 0xffffffff;
            flag_Switch = 0;
```

```
            return 1;
      }
      flag_Switch = 0;
      return 0;
}


/**
 *   @brief - Compares if two color sets are same
 *
 *   @param  color1:  (color_t) First color set
 *   @param  goal:  (color_t) Second Color Set
 *   @return  (int) 1 if sucessfull touch , else 0
 */
int compare_color(color_t color1, color_t color2) {

      return (color1.red == color2.red && color1.green == color2.green &&
color1.blue == color2.blue);
}


/**
 *   @brief - State Machine Function,
 *                       1) Updates the state and events in accordance to the
Timeout for a Traffic Signal
 *                              with Cross walk.
 *                       2) Initializes the State with the Stop State
 *
 *   @param  none
 *   @return  none
 */
void state_machine(void) {
      /* Intialzing start, goal, current color and new state
       * These variables are used all over the state machine to keep track of state
       * color - represets the current color set being lit up on the LED */
      state_t new_state = traffic_light_t.state;
      event_t event = traffic_light_t.event;
      color_t start = traffic_light_t.color_stop;
      color_t goal = traffic_light_t.color_go;
      color_t color = start;
      flag_Switch = 0;

      MSG_DEBUG("\n\r Initializing Traffic Signal Loop with State: STOP");

      // State Machine Infinite Loop Begin
      while(1) {
            switch(new_state) {
                  // STOP state
                  case s_STOP:
                        // Resets the Timer before State Functionality
                        reset_timer();
                        flag_Switch = 0;
                        MSG_DEBUG("\n\r Entering State 'STOP' at sec_time: %d",
now()/1000 );
                        start = traffic_light_t.color_stop; // Updates start color
to - Stop Color Set
                        goal = traffic_light_t.color_go; // Updates goal color to
possible next state color
                        color = traffic_light_t.color_stop; // Current Color to be
```

```
seen on the LED
                        LED_SET(color.red, color.green, color.blue); // Sets the
Color
                        new_state = s_TRANS; // Next State
                        event = e_StopTimeout; // Current Event
                        // Timeout Constraint
                        // Exists the timeout incase of any touch
(capacitive/button) with updated event
                        while ((get_timer() < ROUTINE_TIMEOUT) && (event ==
e_StopTimeout)) {
                              if(get_timer() % 100 == 0) {
                                    cap_touch_action(&goal, &event);
//                                  flag_Switch =0;
                                    switch_action(&goal, &event);
                                    }
                              }
                        break;

                  // GO state
                  case s_GO:
                        // Resets the Timer before State Functionality
                        reset_timer();
                        flag_Switch = 0;
                        MSG_DEBUG("\n\r Entering State 'GO' at sec_time: %d",
now()/1000 );
                        start = traffic_light_t.color_go; // Updates start color to
- GO Color Set
                        color = traffic_light_t.color_go; // Current Color to be
seen on the LED
                        LED_SET(color.red, color.green, color.blue); // Sets the
Color
                        new_state = s_TRANS; // Next State
                        goal = traffic_light_t.color_warn;  // Updates goal color
to possible next state color
                        event = e_GoTimeout; // Current Event
                        // Timeout Constraint
                        // Exists the timeout incase of any touch
(capacitive/button) with updated event
                        while ((get_timer() < ROUTINE_TIMEOUT) && (event ==
e_GoTimeout)) {
                              if(get_timer() % 100 == 0) {
                                    cap_touch_action(&goal, &event);
//                                  flag_Switch = 0;
                                    switch_action(&goal, &event);
                                    }
                              }
                        break;

                  // WARNING state
                  case s_WARNING:
                        // Resets the Timer before State Functionality
                        reset_timer();
                        flag_Switch = 0;
                        MSG_DEBUG("\n\r Entering State 'WARNING' at sec_time: %d",
now()/1000 );
                        start = traffic_light_t.color_warn; // Updates start color
to - WARNING Color Set
                        color = traffic_light_t.color_warn; // Current Color to be
seen on the LED
```

```
                        LED_SET(color.red, color.green, color.blue); // Sets the
Color
                        new_state = s_TRANS; // Next State
                        goal = traffic_light_t.color_stop;  // Updates goal color
to possible next state color
                        event = e_WarnTimeout;  // Current Event
                        // Timeout Constraint
                        // Exists the timeout incase of any touch
(capacitive/button) with updated event
                        while ((get_timer() < WARN_TIMEOUT) && (event ==
e_WarnTimeout)) {
                              if(get_timer() % 100 == 0) {
                                    cap_touch_action(&goal, &event);
//                                  flag_Switch = 0;
                                    switch_action(&goal, &event);
                                    }
                              }
                        break;

                  // CROSSWALK state
                  case s_CROSSWALK:
                        // Resets the Timer before State Functionality
                        reset_timer();
                        new_state = s_TRANS; // Next State
                        start = traffic_light_t.color_crosswalk; // Updates start
color to - CROSSWALK Color Set
                        color = traffic_light_t.color_crosswalk; // Current Color
to be seen on the LED
                        goal = traffic_light_t.color_go; // Updates goal color to
possible next state color
                        MSG_DEBUG("\n\r Entering State 'CROSSWALK' at sec_time:
%d", now()/1000 );

                        // Flashing CROSSWALK color state until 10 seconds
                        while(get_timer() < CROSSWALK_TIMEOUT) {
                              LED_SET(color.red, color.green, color.blue);
                              Delay(750);
                              LED_SET(0x00, 0x00, 0x00);
                              Delay(250);
                        }

//                      LED_SET(color.red, color.green, color.blue);

                        // Exists to GO_state color set .
                        break;

                  // TRANSITION state
                  case s_TRANS:
                        // Resets the Timer before State Functionality
                        reset_timer();
                        MSG_DEBUG("\n\r Smooth Transition Begins at sec_time: %d",
now()/1000 );
                        /* Following Functionality Updates the Color set to
smoothly transition from the current color set
                         * Goal Color set, val is updated in the interrupt every
100 Hz
                         */
                        val = 0;
                        while( val <= 1 ) {
```

```
                              color.blue = (goal.blue - start.blue) * (val) +
start.blue ;  // Blue Color TPM updated
                              color.red =  (goal.red - start.red) * (val) +
start.red ; // Green Color TPM Updated
                              color.green = (goal.green - start.green) * (val) +
start.green ; // Green Color TPM Updated
                              LED_SET(color.red, color.green, color.blue); // Sets
the updated colors
                              // Exit Condition Satisfied
                              if (compare_color(color, goal)) {
                                    val = 0;
                                    break;
                              }

                        }

                        /* The Following Conditional statements updates the next
state depending on the exit color state post
                         * Transition Loop above
                         */
                        // Setting Up the new state based on the color set
                        // If current color set is same as GO state
                        if(compare_color(color, traffic_light_t.color_go)) {
                                    new_state = s_GO;
                                    MSG_DEBUG("\n\r Smooth Transiton COMPLETE at
sec_time: %d, onto State 'GO' ", now()/1000 );
                              }
                        // If current color set is same as STOP state
                        else if(compare_color(color, traffic_light_t.color_stop)) {
                              new_state = s_STOP;
                              MSG_DEBUG("\n\r Smooth Transiton COMPLETE at
sec_time: %d, onto State 'STOP' ", now()/1000 );
                        }
                        // If current color set is same as WARN state
                        else if(compare_color(color, traffic_light_t.color_warn))
{
                              new_state = s_WARNING;
                              MSG_DEBUG("\n\r Smooth Transiton COMPLETE at
sec_time: %d, onto State 'WARNING' ", now()/1000 );
                        }
                        // If current color set is same as CROSSWALK state
                        else if(compare_color(color,
traffic_light_t.color_crosswalk)) {
                              new_state = s_CROSSWALK;
                              MSG_DEBUG("\n\r Smooth Transiton COMPLETE at
sec_time: %d, onto State 'CROSSWALK' ", now()/1000 );
                        }

                        break;
                  // Failure Conditon
                  default :
                        MSG_DEBUG("\n\r State Unknown Failure Condition");
                        break;

            }
      }

}
```

```
================================================================================
===

<statemachine.h>
================================================================================
===
/*
 * statemachine.h
 *
 *  Created on: Oct 2, 2020
 *      Author: root
 */

#ifndef STATEMACHINE_H_
#define STATEMACHINE_H_

#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "stdint.h"
#include "stdbool.h"


/
**********************************************************************************
**************
                                    MACROS
**********************************************************************************
**************/
#ifdef DEBUG
#define CROSSWALK_TIMEOUT 10000
#define ROUTINE_TIMEOUT 5000
#define WARN_TIMEOUT 3000
#endif

#ifdef PRODUCTION
#define CROSSWALK_TIMEOUT 10000
#define ROUTINE_TIMEOUT 20000
#define WARN_TIMEOUT 5000
#endif

#define MASK(x) (1UL << (x))

#define HEX_STOP 0x611E3C
#define HEX_GO 0x229622
#define HEX_WARNING 0xFFB200
#define HEX_CROSSWALK 0x001030


/
**********************************************************************************
**************
                                  Global Variables
**********************************************************************************
**************/
extern volatile double val;
extern volatile int flag_Switch;
extern volatile uint8_t flag;
```

```
/
*******************************************************************************
**************
                                Typededs
*******************************************************************************
**************/
typedef enum {

      s_STOP,
      s_GO,
      s_WARNING,
      s_CROSSWALK,
      s_TRANS
} state_t;

typedef enum {

      e_Void,
      e_TransitionTimeout,
      e_GoTimeout,
      e_StopTimeout,
      e_WarnTimeout
} event_t;

typedef struct color_t{
      int red;
      int green;
      int blue;
} color_t;


/
*******************************************************************************
**************
                                Function Prototypes
*******************************************************************************
**************/
void state_machine(void);


/**
 *   @brief - Function to update event incase of succesfull Capacitive Touch
 *
 *   @param  goal:  (color_t) goal color to be set
 *   @param  goal:  (event_t) event of the statemachine
 *   @return  (int) 1 if sucessfull touch , else 0
 */
int cap_touch_action(color_t* goal, event_t* event);


/**
 *   @brief - Function to update event incase of succesfull Button Press
 *
 *   @param  goal:  (color_t) goal color to be set
 *   @param  goal:  (event_t) event of the statemachine
 *   @return  (int) 1 if sucessfull touch , else 0
 */
int switch_action(color_t* goal, event_t* event);
```

```
/**
 *   @brief - Compares if two color sets are same
 *
 *   @param  color1:  (color_t) First color set
 *   @param  goal:  (color_t) Second Color Set
 *   @return  (int) 1 if sucessfull touch , else 0
 */
int compare_color(color_t color1, color_t color2);
#endif /* STATEMACHINE_H_ */
```

==========

<switch.c>

==========

```
/*
 * switch.c
 *
 *  Created on: Oct 5, 2020
 *      Author: root
 */

#include "switch.h"
#include "statemachine.h"


/**
 *   @brief  This function initializes the Port A clock and MUX for GPIO
 *
 *   @param  PWM Levels - Described as a period
 *   @return  none
 */
void Init_Switch() {


      // SWITCH CAPABILITIES Initialization
      // Push Button Switch
      // Select port A on pin mux, enable pull-up resistors
      PORTA->PCR[SW1_POS] = PORT_PCR_MUX(1) | PORT_PCR_PS_MASK | PORT_PCR_PE_MASK |
PORT_PCR_IRQC(11);
      // Clear switch bits to input
      PTA->PDDR &= ~MASK(SW1_POS);
      // Enabling Interrupts
      /* Configure NVIC */
      NVIC_SetPriority(PORTA_IRQn, 3);
      NVIC_ClearPendingIRQ(PORTA_IRQn);
      NVIC_EnableIRQ(PORTA_IRQn);

      /* Configure PRIMASK */
      __enable_irq();

}


/**
 *   @brief  Interrupt Controller for PORT A
```

```
 *
 *   @param  none
 *   @return  none
 */
void PORTA_IRQHandler(void) {

      flag_Switch = 0;

      // ISFR success
      if ((PORTA->ISFR & MASK(SW1_POS))) {
            flag_Switch = 0;
                  if (SWITCH_PRESSED(SW1_POS)) { // crosswalk state matched
                        flag_Switch = 1;
                  }
            }
      // clear status flags
      PORTA->ISFR = 0xffffffff;
}
================================================================================
===

<switch.h>
================================================================================
===
/*
 * switch.h
 *
 *  Created on: Oct 5, 2020
 *      Author: root
 */

#ifndef SWITCH_H_
#define SWITCH_H_

#include "MKL25Z4.h"
#include "statemachine.h"

/
*******************************************************************************
**************
                                  Macros
*******************************************************************************
**************/
#define SW1_SHIFT (5)   // on port A
#define SW1_POS (5)
#define MASK(x) (1UL << (x))
#define SWITCH_PRESSED(x) (!(PTD->PDIR & (MASK(x))))


/
*******************************************************************************
**************
                               Function Prototypes
*******************************************************************************
***************/
/**
 *   @brief  This function initializes the Port A clock and MUX for GPIO
 *
 *   @param  PWM Levels - Described as a period
```

```
 *   @return  none
 */
void Init_Switch();


/**
 *   @brief  Interrupt Handler
 *
 *   @param  none
 *   @return  none
 */
void PORTA_IRQHandler(void);

#endif /* SWITCH_H_ */
```

```
<temp_systick.c>
```

```
/*
 * temp_systick.c
 *
 *  Created on: Oct 3, 2020
 *      Author: root
 */

#include "temp_systick.h"


#ifdef DEBUG
      #define MSG_DEBUG PRINTF
#else // non-debug mode - get rid of printing message
      #define MSG_DEBUG(...)
#endif

/
**********
                                  Global Variables
**********
**********/
volatile ticktime_t trans_tick;
volatile ticktime_t trans_secs;
volatile ticktime_t Timer_U32;

volatile ticktime_t g_program_start;
volatile ticktime_t g_timer_start;

/
**********
                                  Functions
**********
**********/


/**
 *   @brief  This function initializes the systick timer with 1ms tick time.
```

```
 *              1ms timer value for 100Mhz clock.
 *              COUNT_PER_MS = 48Mhz / 1000(ticks/sec) - 1 = 48000000/1000 - 1 =
47999;
 *
 *   @param  none
 *   @return  none
 */
void SysTick_Init(void) {

      SysTick->LOAD = (COUNT_PER_MS);  // 1000 Hz
      NVIC_SetPriority(SysTick_IRQn, 3); // NVIC Interrupt Priority // 3
      NVIC_ClearPendingIRQ(SysTick_IRQn); // Clear Pending IRq's
      NVIC_EnableIRQ(SysTick_IRQn);
      SysTick->VAL = 0; // Clear Timer
      SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk |
SysTick_CTRL_CLKSOURCE_Msk ;  // Mask to Initialize TIcks, Enamble CTRL Mask and
use Processer CLock Source of 48 Mhz

      trans_tick = 0; // Extra Precaution during Initialization
      Timer_U32 = 0; // Overall CLock - Initialization Precauton
      g_program_start = g_timer_start = 0;
      MSG_DEBUG("\n\r Clock Gating and Initialization of SysTick Complete ");

}


/**
 *   @brief   This functions returns the time in ms since the power on.
                Max time=0xffffffff ms after that it rolls back to 0.
 *
 *   @param  none
 *   @return  none
 */
ticktime_t now() {
      return Timer_U32 - g_program_start;
}


/**
 *   @brief  Resets the Flags and Trans_tick to DEFAULT(0)
 *
 *   @param  none
 *   @return  none
 */
void reset_timer() {
      g_timer_start = Timer_U32;
}


/**
 *   @brief  Returns the number of ticks from reset
 *
 *   @param  none
 *   @return  Integer - Number of Ticks
 */
ticktime_t get_timer() {

      return (Timer_U32 - g_timer_start);
}
```

```
/**
 *   @brief  Interrupt Handler Function. A counter will be incremented to keep
track of Ms.
 *           Handles Smooth Trasition increments and Flags
 *
 *   @param  none
 *   @return  none
 */
void SysTick_Handler(){

      Timer_U32++; // Keep Track of the total timer

      /* A functionality which helps for smooth transition,
       * I found this gave me the smoothest transition
       */
      if(Timer_U32 % 100 == 0){
            val += 0.1;
            if(val > 1) {
                  val = 1;
            }
      }

}

void Delay (uint32_t Ticks) {
  uint32_t curr;

  curr = Timer_U32;
  while ((Timer_U32 - curr) < Ticks);
}
```

```
================================================================================
===

<temp_systick.h>
================================================================================
===
/*
 * temp_systick.h
 *
 *  Created on: Oct 3, 2020
 *      Author: root
 */

#ifndef TEMP_SYSTICK_H_
#define TEMP_SYSTICK_H_

#include "MKL25Z4.h"
#include "stdio.h"
#include "stdint.h"
#include "stdbool.h"
#include "statemachine.h"
```

```
/
*******************************************************************************
**************
                                #typedef's
*******************************************************************************
***************/
typedef uint32_t ticktime_t; // Stores in the resolution of 1000 Hz


/
*******************************************************************************
**************
                                Macros
*******************************************************************************
***************/
/* 48Mhz / 1000 Hz -1 47999 that runs at 1ms */
#define COUNT_PER_MS 47999


/
*******************************************************************************
**************
                                Global Variables
*******************************************************************************
***************/
extern volatile ticktime_t trans_tick;
extern volatile ticktime_t Timer_U32;


/
*******************************************************************************
**************
                                   Function Prototypes
*******************************************************************************
**************/

/**
 *   @brief  This function initializes the systick timer with 1ms tick time.
 *              1ms timer value for 100Mhz clock.
 *              COUNT_PER_MS = 48Mhz / 1000(ticks/sec) - 1 = 48000000/1000 - 1 =
47999;
 *
 *   @param  none
 *   @return  none
 */
void SysTick_Init(void);

/**
 *   @brief  Interrupt Handler Function. A counter will be incremented to keep
track of Ms.
 *           Handles Smooth Trasition increments and Flags
 *
 *   @param  none
 *   @return  none
 */
void SysTick_Handler();

//
//void SysTick_E(void);   // Enables the Systick Functionalities
```

```
//void SysTick_D(void); // Disables the Functionalities


/**
 *   @brief   This functions returns the time in ms since the power on.
 *              Max time=0xffffffff ms after that it rolls back to 0.
 *
 *   @param  none
 *   @return  none
 */
ticktime_t now(); // returns time since startup


/**
 *   @brief  Resets the Flags and Trans_tick to DEFAULT(0)
 *           Doesn't affect now() values
 *
 *   @param  none
 *   @return  none
 */
void reset_timer();


/**
 *   @brief  Returns the number of ticks from reset
 *
 *   @param  none
 *   @return  Integer - Number of Ticks
 */
ticktime_t get_timer();


/**
 *   @brief  Delays for a particular period of time
 *
 *   @param  none
 *   @return  Integer - Number of Ticks
 */
void Delay (uint32_t Ticks);

#endif /* TEMP_SYSTICK_H_ */


================================================================================
===

<Assignment_4_Buffaiti.c>
================================================================================
===
/*
 * Copyright 2016-2020 NXP
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this
list
 *   of conditions and the following disclaimer.
```

```
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


/*****************************************************************************
*Copyright (C) 2020 by Arpit Savarkar
*Redistribution, modification or use of this software insource or binary
*forms is permitted as long as the files maintain this copyright. Users are
*permitted to modify this and use it to learn about the field of embedded
*software. Arpit Savarkar and the University of Colorado are not liable for
*any misuse of this material.
*
*****************************************************************************/
/**
 * @file Assignment_4_Buffaiti.c
 * @brief Application entry point.
 *
 * This file provides functions and abstractions for Initializing Systick,
Capacitive Touch,
 * Switch and Calls State Machine Function
 *
 * @author Arpit Savarkar
 * @date September 10 2020
 * @version 1.0
 *
 *
  Sources of Reference :
  Textbooks : Embedded Systems Fundamentals with Arm Cortex-M based
MicroControllers
  I would like to thank the SA's of the course Rakesh Kumar, Saket Penurkar and
Howdy Pierece for their
  support to gain a deeper insight into State Machine Application
*/



#include <led.h>
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
```

```
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

#include "temp_systick.h"
#include "statemachine.h"
#include "switch.h"
#include "cap_sensor.h"

/* TODO: insert other definitions and declarations here. */
/*
 * @brief   Application entry point.
 */
int main(void) {

      /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();
#endif
    /* Initializes Clock and related functionalities for SysTick */
    SysTick_Init();
    /* Initializes Clock and related functionalities for using Switch to transition
between states */
    Init_Switch();
    /* Initializes Clock and related functionalities for using Capacitive Slider to
transition between states */
    CAP_Init();
    /* Initializes Clock and related functionalities and TPM Setting s for using
PORTB and PORTD under PWM settings to
     * operate the Red, Green and Blue LEDS's */
    Init_LED_PWM(PWM_PERIOD);
#ifdef DEBUG
      PRINTF("\n\r Entering Into the State Machine");
#endif
      // Enters into the State Machine Function
    state_machine();
    return 0 ;
}
```