```c
#include <msp430.h>
#include <stdbool.h>
#include <stdint.h>

#include "Initialize.h"
#include "Timers.h"

bool joystick_Flag = 0;
bool switch_Flag = 0;
bool TimerA0_Flag = 1;
bool TimerA1_Flag = 0;
bool TimerB0_Flag = 0;

uint16_t random_Time = 0;
uint16_t reaction_Time = 0;

enum Timer_States { random_Timer, LED_Timer, reaction_Timer } Timer_State;
void TickFct_Timer() {
        switch(Timer_State) {   // Transitions
                default:
                case random_Timer:                              // Run random timer
                        if (TimerA0_Flag) {
                                // stay
                                Timer_State = random_Timer;
                        } else {
                                Timer_State = LED_Timer;
                        }
                        break;

                case LED_Timer:
                        if (TimerB0_Flag) {                                     // if random timer goes off (B0)
                        Timer_State = reaction_Timer;
                        } else {
                                // stay
                        }
                        break;

                case reaction_Timer:
                        // do nothing go to interrupt                   timer A0 interrupt to turn on LED
                        Timer_State = random_Timer;
                        break;


        switch (Timer_State) {   // State actions
                default:
                case random_Timer:
                        initialize_TimerA0();       // initialize timer for A0 (continuous timer for random number)
                        TimerA0_Flag = 0;
                        break;

                case LED_Timer:
                        P1OUT |= BIT0;
                        break;

                case reaction_Timer:
                        write_Uart(reaction_Time, 0);                   // send reaction time through UART
                        break;
        } // State actions
        }
}

enum LA_States { wait_For_Start, reaction, UART_Transmission } LA_State;
void TickFct_Latch() {
        switch(LA_State) {   // Transitions
                default:
                case wait_For_Start:                // Wait for SW1
                        if (switch_Flag) {
                                LA_State = reaction;
                        } else {
                                 // do nothing
                        }
                        break;

                case reaction:
                        if (TimerB0_Flag) {                                             // if 3rd timer tripped
                                LA_State = UART_Transmission;
                        } else {
                                // stay
                        }
```

```
                        break;

                case UART_Transmission:
                        // do nothing go to interrupt                                   when done with UART
                        LA_State = wait_For_Start;
                        break;


        switch (LA_State) {   // State actions
                default:
                case wait_For_Start:
                        // wait for switch
                        break;

                case reaction:
                        // wait for user reaction
                        break;

                case UART_Transmission:
                        write_Uart(reaction_Time, 0);                   // send reaction time through UART
                        break;
                } // State actions
        }
}

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;        // Stop watchdog timer

    initialize_Joystick();           // initialize joystick
    initialize_Clocks();             // initialize clocks & export to test pins P11.0 to P11.2
    initialize_LED();                        // initialize LEDs
    initialize_Switches();           // initialize switches
    initialize_UART(0,0);            // initialize UART connection (for PC output)
    initialize_Interrupts();// sets up and enables all interrupts
    initialize_TimerA0();            // initialize timer for A1

    while (1) {                                         // run state machine
          TickFct_Timer();
          TickFct_Latch();
    }
}

// Port 2 interrupt service routine for Switch 1/joystick
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void) {
        switch (__even_in_range(P2IV, 14)) {
        case 14:                                                        // P2.6
                //start timer
                initialize_TimerB0();                           // initialize timer for A2
                TA0CTL = MC_0;                                          // pause random timer
                switch_Flag = 1;                                // go to Start_Experiment
                TimerA0_Flag = 0;                               // TimerA0 is off
        default:
                break;
        }
}

// Timer_B0 Interrupt Vector (TBIV) handler for LED timer
#pragma vector=TIMER0_B0_VECTOR
__interrupt void TIMERB0_ISR(void) {

        TA0CTL = MC_0;                                          // pause A0 timer
        TB0CTL = MC_0;                                          // pause B0 timer
        P1OUT &= ~BIT1;                                         // turn off LED2
        P1OUT |= BIT0;                                          // turn on LED1
        initialize_TimerA1();                           // start capture timer
}

// Timer A0 interrupt
// service routine for random timer
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMERA0_ISR(void) {
  P1OUT ^= 0x02;                            // Toggle LED2
}


// Timer A1 interrupt service routine for random timer
#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMERA1_ISR(void) {
```

```
        TA1CTL = MC_0;                                              // pause A1 timer
        reaction_Time = TA1R;                                     // capture the reaction time

        switch (__even_in_range(TA1IV, 14)) {

        case 0:    break;                          // No interrupt
        case 2:    break;                          // CCR1 not used
        case 4:    break;                          // CCR2 not used
        case 6:    break;                          // reserved
        case 8:    break;                          // reserved
        case 10: break;                           // reserved
        case 12: break;                           // reserved
        case 14:
                 P1OUT ^= 0x01;                                  // overflow
                 break;
        default:
                 break;
        }
}
```

```
/*
 * Initialize.h
 *
 *  Created on: Nov 11, 2016
 *      Author: aaronewing
 */

// contains all functions for Initializing MSP430

#ifndef INITIALIZE_H_
#define INITIALIZE_H_

void initialize_Joystick(void);
void initialize_Ports(void);
void initialize_Clocks(void);
void initialize_LED(void);
void initialize_Switches(void);
void initialize_Interrupts(void);

#endif /* INITIALIZE_H_ */



/*
 * Initialize.c
 *
 *  Created on: Nov 3, 2016
 *      Author: aaronewing
 */

// contains all functions for Initializing MSP430

#include <msp430.h>
#include <stdbool.h>
#include <stdint.h>
#include "Initialize.h"

#define LED1 BIT0
#define LED2 BIT1

#define LEFT 0xFD
#define RIGHT 0xFB
#define CENTER 0xF7
#define UP 0xEF
#define DOWN 0xDF
#define no_Input 0xFF

void initialize_Clocks(void) {                          // Sets all clocks to standard position

          P11DIR |= 0x07;                             // ACLK, MCLK, SMCLK set out to pins
          P11SEL |= 0x07;                             // P11.0,1,2 for debugging purposes.

// Initialize LFXT1
         P7SEL |= 0x03;                              // Select XT1
         UCSCTL6 &= ~(XT1OFF);                       // XT1 On
         UCSCTL6 |= XCAP_3;                          // Internal load cap

// Loop until XT1 fault flag is cleared
         /*
         do {
                  UCSCTL7 &= ~XT1LFOFFG;                  // Clear XT1 fault flags
         } while (UCSCTL7 & XT1LFOFFG);               // Test XT1 fault flag
} */

// Initialize DCO to 16MHz
         __bis_SR_register(SCG0);                        // Disable the FLL control loop
         UCSCTL0 = 0x0000;                     // Set lowest possible DCOx, MODx
         UCSCTL1 = DCORSEL_3;                   // Set RSELx for DCO = 4.9 MHz
         UCSCTL2 = FLLD_1 + 478;                                 // Set DCO Multiplier for 16MHz
                                                                         // (N + 1) * FLLRef = Fdco
                                                                    // (478 + 1) * 32768 = 15.99MHz
                                                                    // Set FLL Div = fDCOCLK/2
         __bic_SR_register(SCG0);              // Enable the FLL control loop

         // Worst-case settling time for the DCO when the DCO range bits have been
         // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
         // UG for optimization.
         // 32 x 32 x 2.45 MHz / 32,768 Hz = 76563 = MCLK cycles for DCO to settle
```

```
        __delay_cycles(76563);

        // Loop until XT1,XT2 & DCO fault flag is cleared
/*      do {
                UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + XT1HFOFFG + DCOFFG); // Clear XT2,XT1,DCO fault flags
                SFRIFG1 &= ~OFIFG;// Clear fault flags
        } while (SFRIFG1&OFIFG);                   // Test oscillator fault flag */
}

void initialize_LED(void) {
        P1DIR |= BIT0 | BIT1;               // Sets P1.0 and P1.1 as output (LED1 and LED2)
        P1OUT &= ~(BIT0 | BIT1);    // Turns LEDs off
}

void initialize_Switches(void) {
        P2DIR &= ~(BIT6 | BIT7);    // Init P2.6 and P2.7 as inputs
}

void initialize_Joystick(void) {
        P2DIR &= ~(LEFT | RIGHT | CENTER | UP | DOWN); // Sets up joystick as input
        // P2.1 - LEFT, P2.2 - RIGHT, P2.3 - CENTER, P2.4 - UP, DOWN - P2.5
        // 0 if pushed, 1 if not.
        P2SEL |= BIT1 | BIT2;
        P2REN |= LEFT | RIGHT | CENTER | UP | DOWN;
        P2OUT |= LEFT | RIGHT | CENTER | UP | DOWN;
}

void initialize_Interrupts(void) {
        P2IE |= BIT1 + BIT2 + BIT6;          // BIT1 = joystick left, BIT2 = joystick right, BIT6 = switch 1

        __bis_SR_register(GIE);             // enable interrupt
}
```

```
/*
 * Timers.h
 *
 *  Created on: Nov 9, 2016
 *      Author: aaronewing
 */

#ifndef TIMERS_H_
#define TIMERS_H_

void initialize_TimerA1(void);
void initialize_TimerA0(void);
void initialize_TimerB0(void);


#endif /* TIMERS_H_ */

/*
 * Timer.c
 *
 *  Created on: Nov 9, 2016
 *      Author: aaronewing
 */

#include <msp430.h>
#include <stdbool.h>
#include <stdint.h>
#include "Timers.h"

// timer for "random" (continuous)
void initialize_TimerA0(void) {
         TA0CCTL0 = CCIE;                          // CCR0 interrupt enabled
         TA0CCR0 = 0xFFFF;
         TA0CTL = TASSEL_1 + MC_2 + TACLR;         // SMCLK, contmode, clear TAR
}

// timer for LED (up)
void initialize_TimerB0(void) {
         TB0CCTL0 = CCIE;                                 // CCR0 interrupt enabled
         TB0CCR0 = (32768 + TA0R);   // count up to 1 second + whatever TimerA1 counted to before switch 1 was pressed
         TB0CTL = TASSEL_1 + MC_1 + TACLR + TBIE;    // ACLK, up mode, clear TAR, enable TB interrupt
}

// timer for getting reaction time (up)
void initialize_TimerA1(void) {
                  TA1CCTL0 = CCIE + CAP + SCS + CCIS_0 + CM_2;            // CCR0 interrupt enabled, capture mode, sync
                                                                          input, CCI0A (joystick), falling edge
         TA1CCR0 = 0xFFFF;                                        // count up to 2 second (enough time for reaction timer)
         TA1CTL = TASSEL_1 + MC_1 + TACLR;                        // ACLK, up mode, clear TAR
}
```

```c
/*
 * UART.h
 *
 *  Created on: Nov 10, 2016
 *      Author: aaronewing
 */

#ifndef UART_H_
#define UART_H_

void init_UART (bool baud_Rate, bool pin_Setting);              // initalizes UART clk rate and which pins are being
used
void write_UART (uint8_t TX_Data, uint8_t pin_Setting); // writes 8 bits with UART
uint8_t read_UART (void);

#endif /* UART_H_ */



/*
 * UART.c
 *
 *  Created on: Nov 10, 2016
 *      Author: aaronewing
 */

#include <msp430.h>
#include <stdbool.h>
#include <stdint.h>
#include "UART.h"

uint8_t RX_Data = 0;

void initialize_UART(bool baud_Rate, uint8_t pin_Setting) {
         switch (pin_Setting) {
         default:
         case 0:
                  // Configure Secondary Function Pins
                  P3SEL |= BIT4 | BIT5;                                          // P3.4 - TX, P3.5 - RX

                  // assuming clk is set up already at 16MHz

                  // Configure USCI_A0 for SPI operation
                  UCA0CTL1 |= UCSWRST;                  // **Put state machine in reset**

                  switch (baud_Rate) {
                  case 0:
                           // Configure Timer for 9600 Baud
                           UCA0CTL1 = UCSSEL__ACLK;              // Set ACLK = 32768 as UCBRCLK
                           UCA0BR0 = 3;                             // 9600 baud
                           UCA0MCTL |= 0x5300;            // 32768/9600 - INT(32768/9600)=0.41
                                                          // UCBRSx value = 0x53 (See UG)
                           UCA0BR1 = 0;
                           break;

                  default:
                  case 1:
                           // Configure Timer for 38400 Baud
                           UCA0CTL1 = UCSSEL__SMCLK;          // Set SMCLK = 1000000 as UCBRCLK
                           UCA0BR0 = 0x1A;                                                    // 9600 baud
                           UCA0MCTL |= 0x0100;      // 1000000/38400 - INT(1000000/38400)=0.04
                                                                                 // UCBRSx value = 0x01 (See UG)
                           // N = 0.0529, effectively 38,383.4 Baud
                           UCA0BR1 = 0;
                           break;
                  }
                  UCA0CTL1 &= ~UCSWRST; // release from reset                     // **Initialize USCI state machine**

                  break;

         case 1:
                  // Configure Secondary Function Pins
                  P5SEL |= BIT6 | BIT7;                                          // P5.6 - TX, P5.7 - RX

                  // assuming clk is set up already at 16MHz

                  // Configure USCI_A0 for SPI operation
                  UCA1CTL1 |= UCSWRST;                  // **Put state machine in reset**
```

```
            switch (baud_Rate) {
            case 0:
                    // Configure Timer for 9600 Baud
                    UCA1CTL1 = UCSSEL__ACLK;              // Set ACLK = 32768 as UCBRCLK
                    UCA1BR0 = 3;                             // 9600 baud
                    UCA1MCTL |= 0x5300;            // 32768/9600 - INT(32768/9600)=0.41
                                                          // UCBRSx value = 0x53 (See UG)
                    UCA1BR1 = 0;
                    break;

            default:
            case 1:
                    // Configure Timer for 38400 Baud
                    UCA1CTL1 = UCSSEL__SMCLK;          // Set SMCLK = 1000000 as UCBRCLK
                    UCA1BR0 = 0x1A;                                                    // 9600 baud
                    UCA1MCTL |= 0x0100;      // 1000000/38400 - INT(1000000/38400)=0.04
                                                                  // UCBRSx value = 0x01 (See UG)
                    // N = 0.0529, effectively 38,383.4 Baud
                    UCA1BR1 = 0;
                    break;
            }
            UCA1CTL1 &= ~UCSWRST; // release from reset                      // **Initialize USCI state machine**
            break;

    case 2:
            // Configure Secondary Function Pins
            P9SEL |= BIT4 | BIT5;                                           // P9.4 - TX, P9.5 - RX

            // assuming clk is set up already at 16MHz

            // Configure USCI_A0 for SPI operation
            UCA2CTL1 |= UCSWRST;                   // **Put state machine in reset**

            switch (baud_Rate) {
            case 0:
                    // Configure Timer for 9600 Baud
                    UCA2CTL1 = UCSSEL__ACLK;              // Set ACLK = 32768 as UCBRCLK
                    UCA2BR0 = 3;                             // 9600 baud
                    UCA2MCTL |= 0x5300;            // 32768/9600 - INT(32768/9600)=0.41
                                                          // UCBRSx value = 0x53 (See UG)
                    UCA2BR1 = 0;
                    break;

            default:
            case 1:
                    // Configure Timer for 38400 Baud
                    UCA2CTL1 = UCSSEL__SMCLK;          // Set SMCLK = 1000000 as UCBRCLK
                    UCA2BR0 = 0x1A;                            // 9600 baud
                    UCA2MCTL |= 0x0100;      // 1000000/38400 - INT(1000000/38400)=0.04
                                                                  // UCBRSx value = 0x01 (See UG)
                    // N = 0.0529, effectively 38,383.4 Baud
                    UCA2BR1 = 0;
                    break;
            }
            UCA2CTL1 &= ~UCSWRST; // release from reset                      // **Initialize USCI state machine**
            break;

    case 3:
            // Configure Secondary Function Pins
            P10SEL |= BIT4 | BIT5;                                          // P10.4 - TX, P10.5 - RX

            // assuming clk is set up already at 16MHz

            // Configure USCI_A0 for SPI operation
            UCA3CTL1 |= UCSWRST;                   // **Put state machine in reset**

            switch (baud_Rate) {
            case 0:
                    // Configure Timer for 9600 Baud
                    UCA3CTL1 = UCSSEL__ACLK;              // Set ACLK = 32768 as UCBRCLK
                    UCA3BR0 = 3;                             // 9600 baud
                    UCA3MCTL |= 0x5300;            // 32768/9600 - INT(32768/9600)=0.41
                                                          // UCBRSx value = 0x53 (See UG)
                    UCA3BR1 = 0;
                    break;

            default:
            case 1:
                    // Configure Timer for 38400 Baud
```

```
                        UCA3CTL1 = UCSSEL__SMCLK;          // Set SMCLK = 1000000 as UCBRCLK
                        UCA3BR0 = 0x1A;             // 9600 baud
                        UCA3MCTL |= 0x0100;      // 1000000/38400 - INT(1000000/38400)=0.04
                                                                        // UCBRSx value = 0x01 (See UG)
                        // N = 0.0529, effectively 38,383.4 Baud
                        UCA3BR1 = 0;
                        break;
                }
                UCA3CTL1 &= ~UCSWRST; // release from reset                      // **Initialize USCI state machine**
                break;
        }
}

void write_UART(uint8_t TX_Data, uint8_t pin_Setting) {
        switch (pin_Setting) {
        default:
        case 0:
                while (!(UCA0IFG & UCTXIFG)) {};                                        // If able to TX
                UCA0TXBUF = TX_Data;                                                    // 8 bits transmitted
                break;

        case 1:
                while (!(UCA1IFG & UCTXIFG)) {};                                        // If able to TX
                UCA1TXBUF = TX_Data;                                                    // 8 bits transmitted
                break;

        case 2:
                while (!(UCA2IFG & UCTXIFG)) {};                                        // If able to TX
                UCA2TXBUF = TX_Data;                                                    // 8 bits transmitted
                break;

        case 3:
                while (!(UCA3IFG & UCTXIFG)) {};                                        // If able to TX
                UCA3TXBUF = TX_Data;                                                    // 8 bits transmitted
                break;
        }
}

//////////////////// UART READ POLLING ////////////////////////////////
uint8_t read_UART(void) {
        while (!(UCA0IFG & UCRXIFG)) {};                        // While RX flag is high
        RX_Data = UCA0RXBUF;                                                    // Receive Data
        return RX_Data;
}
```