**#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

}

/\*

\* 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;

}