**ECEN 5813**

**Chutao Wei**

**Curry Buscher**

**PES Project 4 Code pdf**

**readme.md**

**# cu-ecen-5813-project-4**

**\*\*Title:\*\***

PES Project 4 Readme **<**br/**>**

**\*\*Name:\*\***

Curry Buscher, Chutao Wei **<**br/**>**

**\*\*Repository Comments:\*\*** **<**br/**>**

In documents folder: **<**br/**>**

There are PES Project 4.pdf, and state machine diagram.jpg**<**br/**>**

In source folder: **<**br/**>**

main.c: main function wrapper has two versions. One runs the test script without command line, one require user to put in command in console.**<**br/**>**

memory\_utility.c/h: contains all memory utility functions.**<**br/**>**

pattern\_gen.c/h: generate random byte array using linear feedback shit register**<**br/**>**

led.c/h: contains RGB LED control functions**<**br/**>**

timer.c/h: contains only blocking delay function for now**<**br/**>**

gpio.c/h: contains gpio control functions**<**br/**>**

state.c/h: state machine function**<**br/**>**

touch\_sen.c/h: contains touch sensor printing function**<**br/**>**

mma8451.c/h: contains mma8451 accelerameter function**<**br/**>**

test.c: contains test function for uCUnit testfunction**<**br/**>**

uCUnit.c/h: uCUnit test function**<**br/**>**

System.c/h: System for uCUnit**<**br/**>**

(see more details in PES Project 4.pdf) **<**br/**>**

**\*\*Project Comments:\*\***

Please use semihost **<**br/**>**

**### \*\*Installation/Execution/Editing Notes:\*\*<br/>**

**\*\*Language:\*\***

C**<**br/**>**

**\*\*Compiler:\*\***

GCC version 7.4.0**<**br/**>**

**\*\*IDE:\*\***

MCUExpresso**<**br/**>**

**\*\*Build Environment:\*\***

Ubuntu 16 or up**<**br/**>**

**\*\*Target Environment:\*\***

KL25Z/Linux**<**br/**>**

**\*\*License:\*\***

MIT**<**br/**>**

**Project\_4.c (main.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.

\*/

/\*\*

\* **@file** Project\_4.c

\* **@brief** Application entry point.

\*/

#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 "i2c.h"

#include "gpio.h"

#include "led.h"

#include "mma8451.h"

#include "touch\_sen.h"

#include "timer.h"

#include "state.h"

/\*

\* @brief Application entry point.

\*/

int main**(**void**)** **{**

/\* Init board hardware. \*/

BOARD\_InitBootPins**();**

BOARD\_InitBootClocks**();**

BOARD\_InitBootPeripherals**();**

/\* Init FSL debug console. \*/

BOARD\_InitDebugConsole**();**

/\* Init Systick \*/

Init\_SysTick**();**

/\* Init LED \*/

init\_LED**();**

turn\_LED\_blue**(**on**);**

/\* Init Touch Sensor \*/

Touch\_Init**();**

/\* Init i2c for MMA8451 \*/

i2c\_init**();**

PRINTF**(**"Hello, PES Project 4\n"**);**

/\* Init MMA8451 \*/

**if(**init\_mma8451**())**

**{**

PRINTF**(**"MMA8451 connection error\n"**);**

// if connection error, just halt the program

// and turn LED red

turn\_LED\_blue**(**off**);**

turn\_LED\_red**(**on**);**

**while(**1**);**

**}**

turn\_LED\_blue**(**off**);**

turn\_LED\_green**(**on**);**

/\* Enter an infinite loop \*/

**while(**1**)**

**{**

RunMachines**();**

**}**

**return** 0 **;**

**}**

**timer.c (main.c)**

/\*

\* timer.c

\*

\* Created on: Feb 11, 2020

\* Author: chutao

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <logger.h>

#include <stdint.h>

#include <stdbool.h>

#include "state.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Define \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define INCLUDE\_LOG\_DEBUG 1

#define CPU\_FREQ\_MHZ (48)

#define NUM\_ASSE\_FOR (7)

#define DELAY\_MS\_TO\_LOOP\_COUNT(msec)\

((uint32\_t)((msec\*(CPU\_FREQ\_MHZ\*1000))/(NUM\_ASSE\_FOR)))

//#define BLOCKWAITING

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const uint32\_t delay\_look\_up\_table**[]** **=** **{**

DELAY\_MS\_TO\_LOOP\_COUNT**(**500**),**

DELAY\_MS\_TO\_LOOP\_COUNT**(**1000**),**

DELAY\_MS\_TO\_LOOP\_COUNT**(**2000**),**

DELAY\_MS\_TO\_LOOP\_COUNT**(**3000**)};**

uint64\_t msec\_count **=** 0**;**

uint64\_t target\_msec\_count **=** 0**;**

bool delay\_flag **=** 0**;**

/\* Time out count for kWaitPollSlider state \*/

uint8\_t timeout\_count **=** 0**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Interrupt Hanlder \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SysTick\_Handler**(**void**)**

**{**

msec\_count **++;**

**if** **(**delay\_flag **==** true**)**

**{**

**if** **(**msec\_count **==** target\_msec\_count**)**

**{**

delay\_flag **=** false**;**

timeout\_count**++;**

**if** **(**timeout\_count **==** 6**)**

**{**

timeout\_count **=** 0**;**

SetEvent**(**Timeout\_6**);**

**}**

**else**

**{**

SetEvent**(**Timeout\_1\_5**);**

**}**

**}**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Init\_SysTick**(**void**)** **{**

SysTick**->**LOAD **=** **(**48000L**-**1L**);**// count 1 msec

NVIC\_SetPriority**(**SysTick\_IRQn**,** 3**);** // enable NVIC

SysTick**->**VAL **=** **(**480000L**-**1L**);** // reset count value

SysTick**->**CTRL **=** SysTick\_CTRL\_CLKSOURCE\_Msk **|** SysTick\_CTRL\_TICKINT\_Msk **|** SysTick\_CTRL\_ENABLE\_Msk**;**

**}**

#ifdef BLOCK\_WAITING

// Block waiting function abandoned for now

void mdelay**(**uint32\_t msec**)**

**{**

LOG\_DEBUG**(**"Blocking wait for %d msec"**,** msec**);**

uint32\_t i **=** 0**;**

uint32\_t delay\_count **=** 0**;**

**if** **(**msec **==** 500**)**

**{**

delay\_count **=** delay\_look\_up\_table**[**0**];**

**}**

**else** **if** **(**msec **==** 1000**)**

**{**

delay\_count **=** delay\_look\_up\_table**[**1**];**

**}**

**else** **if** **(**msec **==** 2000**)**

**{**

delay\_count **=** delay\_look\_up\_table**[**2**];**

**}**

**else** **if** **(**msec **==** 3000**)**

**{**

delay\_count **=** delay\_look\_up\_table**[**3**];**

**}**

**else**

**{**

LOG\_ERROR**(**"Unexpected msec value, has to be 500, 1000, 2000, 3000"**);**

**}**

**for(**i**=**0**;** i**<**delay\_count**;** i**++);**

**}**

#else

// Interrupt waiting function4

void mdelay**(**uint32\_t msec**)**

**{**

LOG\_DEBUG**(**"Interrupt wait for %d msec"**,** msec**);**

// read current count

target\_msec\_count **=** msec\_count **+** msec**;**

delay\_flag **=** true**;**

**}**

#endif

**i2c.c**

/\*

\* i2c.c

\*

\* Created on: Mar 31, 2020

\* Author: user

\*/

//adapted from https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter\_8/I2C-Demo/

#include <MKL25Z4.h>

#include "i2c.h"

int lock\_detect**=**0**;**

int i2c\_lock**=**0**;**

//init i2c0

void i2c\_init**(**void**)**

**{**

//clock i2c peripheral and port E

SIM**->**SCGC4 **|=** SIM\_SCGC4\_I2C0\_MASK**;**

SIM**->**SCGC5 **|=** **(**SIM\_SCGC5\_PORTE\_MASK**);**

//set pins to I2C function

PORTE**->**PCR**[**24**]** **|=** PORT\_PCR\_MUX**(**5**);**

PORTE**->**PCR**[**25**]** **|=** PORT\_PCR\_MUX**(**5**);**

//set to 100k baud

//baud = bus freq/(scl\_div+mul)

//~400k = 24M/(64); icr=0x12 sets scl\_div to 64

I2C0**->**F **=** **(**I2C\_F\_ICR**(**0x10**)** **|** I2C\_F\_MULT**(**0**));**

//enable i2c and set to master mode

I2C0**->**C1 **|=** **(**I2C\_C1\_IICEN\_MASK**);**

// Select high drive mode

I2C0**->**C2 **|=** **(**I2C\_C2\_HDRS\_MASK**);**

**}**

void i2c\_busy**(**void**){**

// Start Signal

lock\_detect**=**0**;**

I2C0**->**C1 **&=** **~**I2C\_C1\_IICEN\_MASK**;**

I2C\_TRAN**;**

I2C\_M\_START**;**

I2C0**->**C1 **|=** I2C\_C1\_IICEN\_MASK**;**

// Write to clear line

I2C0**->**C1 **|=** I2C\_C1\_MST\_MASK**;** /\* set MASTER mode \*/

I2C0**->**C1 **|=** I2C\_C1\_TX\_MASK**;** /\* Set transmit (TX) mode \*/

I2C0**->**D **=** 0xFF**;**

**while** **((**I2C0**->**S **&** I2C\_S\_IICIF\_MASK**)** **==** 0U**)** **{**

**}** /\* wait interrupt \*/

I2C0**->**S **|=** I2C\_S\_IICIF\_MASK**;** /\* clear interrupt bit \*/

/\* Clear arbitration error flag\*/

I2C0**->**S **|=** I2C\_S\_ARBL\_MASK**;**

/\* Send start \*/

I2C0**->**C1 **&=** **~**I2C\_C1\_IICEN\_MASK**;**

I2C0**->**C1 **|=** I2C\_C1\_TX\_MASK**;** /\* Set transmit (TX) mode \*/

I2C0**->**C1 **|=** I2C\_C1\_MST\_MASK**;** /\* START signal generated \*/

I2C0**->**C1 **|=** I2C\_C1\_IICEN\_MASK**;**

/\*Wait until start is send\*/

/\* Send stop \*/

I2C0**->**C1 **&=** **~**I2C\_C1\_IICEN\_MASK**;**

I2C0**->**C1 **|=** I2C\_C1\_MST\_MASK**;**

I2C0**->**C1 **&=** **~**I2C\_C1\_MST\_MASK**;** /\* set SLAVE mode \*/

I2C0**->**C1 **&=** **~**I2C\_C1\_TX\_MASK**;** /\* Set Rx \*/

I2C0**->**C1 **|=** I2C\_C1\_IICEN\_MASK**;**

/\* wait \*/

/\* Clear arbitration error & interrupt flag\*/

I2C0**->**S **|=** I2C\_S\_IICIF\_MASK**;**

I2C0**->**S **|=** I2C\_S\_ARBL\_MASK**;**

lock\_detect**=**0**;**

i2c\_lock**=**1**;**

**}**

//#pragma no\_inline

void i2c\_wait**(**void**)** **{**

lock\_detect **=** 0**;**

**while(((**I2C0**->**S **&** I2C\_S\_IICIF\_MASK**)==**0**)** **&** **(**lock\_detect **<** 200**))** **{**

lock\_detect**++;**

**}**

**if** **(**lock\_detect **>=** 200**)**

i2c\_busy**();**

I2C0**->**S **|=** I2C\_S\_IICIF\_MASK**;**

**}**

//send start sequence

void i2c\_start**()**

**{**

I2C\_TRAN**;** /\* set to transmit mode \*/

I2C\_M\_START**;** /\* send start \*/

**}**

//send device and register addresses

//#pragma no\_inline

void i2c\_read\_setup**(**uint8\_t dev**,** uint8\_t address**)**

**{**

I2C0**->**D **=** dev**<<**1**;** /\* send dev address \*/

I2C\_WAIT /\* wait for completion \*/

I2C0**->**D **=** address**;** /\* send read address \*/

I2C\_WAIT /\*wait for completion \*/

I2C\_M\_RSTART**;** /\* repeated start \*/

I2C0**->**D **=** **(**dev**<<**1**)|**0x1**;** /\* send dev address (read) \*/

I2C\_WAIT /\* wait for completion \*/

I2C\_REC**;** /\* set to receive mode \*/

**}**

//read a byte and ack/nack as appropriate

// #pragma no\_inline

uint8\_t i2c\_repeated\_read**(**uint8\_t isLastRead**)**

**{**

uint8\_t data**;**

lock\_detect **=** 0**;**

**if(**isLastRead**)** **{**

NACK**;** /\* set NACK after read \*/

**}** **else** **{**

ACK**;** /\* ACK after read \*/

**}**

data **=** I2C0**->**D**;** /\* dummy read \*/

I2C\_WAIT /\* wait for completion \*/

**if(**isLastRead**)** **{**

I2C\_M\_STOP**;** /\* send stop \*/

**}**

data **=** I2C0**->**D**;** /\* read data \*/

**return** data**;**

**}**

//////////funcs for reading and writing a single byte

//using 7bit addressing reads a byte from dev:address

// #pragma no\_inline

uint8\_t i2c\_read\_byte**(**uint8\_t dev**,** uint8\_t address**)**

**{**

uint8\_t data**;**

I2C\_TRAN**;** /\* set to transmit mode \*/

I2C\_M\_START**;** /\* send start \*/

I2C0**->**D **=** **(**dev**<<**1**);** /\* send dev address \*/

I2C\_WAIT /\* wait for completion \*/

I2C0**->**D **=** address**;** /\* send read address \*/

I2C\_WAIT /\* wait for completion \*/

I2C\_M\_RSTART**;** /\* repeated start \*/

I2C0**->**D **=** **(**dev**<<**1**)|**0x1**;** /\* send dev address (read) \*/

I2C\_WAIT /\* wait for completion \*/

I2C\_REC**;** /\* set to recieve mode \*/

NACK**;** /\* set NACK after read \*/

data **=** I2C0**->**D**;** /\* dummy read \*/

I2C\_WAIT /\* wait for completion \*/

I2C\_M\_STOP**;** /\* send stop \*/

data **=** I2C0**->**D**;** /\* read data \*/

**return** data**;**

**}**

//using 7bit addressing writes a byte data to dev:address

//#pragma no\_inline

void i2c\_write\_byte**(**uint8\_t dev**,** uint8\_t address**,** uint8\_t data**)**

**{**

I2C\_TRAN**;** /\* set to transmit mode \*/

I2C\_M\_START**;** /\* send start \*/

I2C0**->**D **=** **(**dev**<<**1**);** /\* send dev address \*/

I2C\_WAIT /\* wait for ack \*/

I2C0**->**D **=** address**;** /\* send write address \*/

I2C\_WAIT /\* wait for ack \*/

I2C0**->**D **=** data**;** /\* send data \*/

I2C\_WAIT /\* wait for ack \*/

I2C\_M\_STOP**;** /\* send stop \*/

**}**

**touch\_sen.c**

/\*

\* touch\_sen.c

\*

\* Created on: Feb 11, 2020

\* Author: Ben Roloff

\* link: https://www.digikey.com/eewiki/display/microcontroller/Using+the+Capacitive+Touch+Sensor+on+the+FRDM-KL46Z

\* Editor: chutao

\*/

#include "MKL25Z4.h"

#include "touch\_sen.h"

#include "fsl\_debug\_console.h"

#include "logger.h"

//#define INCLUDE\_LOG\_DEBUG 1

tsi\_position\_t position**;**

#ifdef L\_M\_R\_POSITION\_MODE

const char **\*** position\_string**[**3**]** **={**"left"**,**"middle"**,**"right"**};**

#endif

#ifdef L\_R\_POSITION\_MODE

const char **\*** position\_string**[**3**]** **={**"left"**,**"right"**};**

#endif

// TSI initialization function

void Touch\_Init**()**

**{**

// 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\_ESOR\_MASK | // This is disabled to give an interrupt when out of range. Enable to give an interrupt when end of scan

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\_TSIIEN\_MASK | //TSI interrupt is disables

TSI\_GENCS\_STPE\_MASK **|** // Enables TSI in low power mode

//TSI\_GENCS\_STM\_MASK | // 0 for software trigger, 1 for hardware trigger

//TSI\_GENCS\_SCNIP\_MASK | // scan in progress flag

TSI\_GENCS\_EOSF\_MASK **;** // End of scan flag, set to 1 to clear

//TSI\_GENCS\_CURSW\_MASK; // Do not swap current sources

**}**

// Function to read touch sensor from low to high capacitance for left to right

int Touch\_Scan\_LH**(**void**)**

**{**

int scan**;**

TSI0**->**DATA **=** TSI\_DATA\_TSICH**(**10u**);** // Using channel 10 of The TSI

TSI0**->**DATA **|=** TSI\_DATA\_SWTS\_MASK**;** // Software trigger for scan

scan **=** SCAN\_DATA**;**

TSI0**->**GENCS **|=** TSI\_GENCS\_EOSF\_MASK **;** // Reset end of scan flag

**return** scan **-** SCAN\_OFFSET**;**

**}**

// Function to read touch sensor from high to low capacitance for left to right

int Touch\_Scan\_HL**(**void**)**

**{**

int scan**;**

TSI0**->**DATA **=** TSI\_DATA\_TSICH**(**9u**);** // Using channel 9 of the TSI

TSI0**->**DATA **|=** TSI\_DATA\_SWTS\_MASK**;** // Software trigger for scan

scan **=** SCAN\_DATA**;**

TSI0**->**GENCS **|=** TSI\_GENCS\_EOSF\_MASK **;** // Reset end of scan flag

**return** scan **-** SCAN\_OFFSET**;**

**}**

// Function to scan the position of slider

tsi\_position\_t Touch\_Scan\_Position**(**void**)**

**{**

LOG\_DEBUG**(**"Scan Capacitance"**);**

// read capacitance value

int LH\_value **=** Touch\_Scan\_LH**();**

int HL\_value **=** Touch\_Scan\_HL**();**

LOG\_DEBUG**(**"LH\_value: %d, HL\_value: %d"**,** LH\_value**,** HL\_value**);**

#ifdef L\_M\_R\_POSITION\_MODE

// Check the capacitance, and decide the position

**if** **((**LH\_value**>**50**)** **&&** **(**LH\_value**<**350**)** **&&** **(**HL\_value**>**50**)** **&&** **(**HL\_value**<**350**))**

**{**

position **=** left**;**

**}**

**else** **if** **((**LH\_value**>**450**)** **&&** **(**LH\_value**<**750**)** **&&** **(**HL\_value**>**450**)** **&&** **(**HL\_value**<**750**))**

**{**

position **=** middle**;**

**}**

**else** **if** **((**LH\_value**>**800**)** **&&** **(**LH\_value**<**1200**)** **&&** **(**HL\_value**>**800**)** **&&** **(**HL\_value**<**1200**))**

**{**

position **=** right**;**

**}**

LOG\_DEBUG**(**"Slider Position is %s"**,** position\_string**[**position**]);**

#endif

#ifdef L\_R\_POSITION\_MODE

// Check the capacitance, and decide the position

**if** **((**LH\_value**>**0**)** **&&** **(**LH\_value**<**1000**)** **&&** **(**HL\_value**>**0**)** **&&** **(**HL\_value**<**1000**))**

**{**

position **=** left**;**

**}**

**else** **if** **((**LH\_value**>**1100**)** **&&** **(**LH\_value**<**3000**)** **&&** **(**HL\_value**>**1100**)** **&&** **(**HL\_value**<**3000**))**

**{**

position **=** right**;**

**}**

**else**

**{**

position **=** unknown**;**

LOG\_DEBUG**(**"Slider Position is a little off, calibration needed, default to left"**);**

**}**

LOG\_DEBUG**(**"Slider Position is %s"**,** position\_string**[**position**]);**

#endif

**return** **(**position**);**

**}**

**state.c**

/\*

\* state.c

\*

\* Created on: Mar 31, 2020

\* Author: user

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include "state.h"

#include "timer.h"

#include "mma8451.h"

#include "touch\_sen.h"

#include "logger.h"

#include "led.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* State machine relted varibles \*/

tEvent event **=** RightSlider**;**

tMachine machine**=**mStateCentric**;**

tState currentState**=** kReadXYZ**;**

/\* State Table used by RunTableDriven() function \*/

struct sStateTableEntry stateTable**[]** **=** **{**

/\* CurrentState Action function Event:complete timeout\_1\_5, timout\_6, left\_slider, right\_slider \*/

/\* kReadXYZ \*/ **{**state\_ReadXYZ**,** **{**kProcessDisplay**,** kError**,** kError**,** kError**,** kError **}},**

/\* kProcessDisplay \*/ **{**state\_Display**,** **{**kWaitPollSlider**,** kError**,** kError**,** kError**,** kError **}},**

/\* kWaitPollSlider \*/ **{**state\_WaitPoll**,{**kError**,** kReadXYZ**,** kReadXYZ**,** kReadXYZ**,** kEnd **}},**

/\* kend \*/ **{**state\_End**,** **{**kEnd**,** kEnd**,** kEnd**,** kEnd**,** kEnd **}},**

/\* kerror \*/ **{**state\_Error**,** **{**kError**,** kError**,** kError**,** kError**,** kError **}}**

**};**

const char **\*** state\_machine\_type\_string**[**2**]** **={**"StateCentric"**,**"TableDriven"**};**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* State Machine action functions \*/

void state\_ReadXYZ**(){**

read\_full\_xyz**();**

SetEvent**(**Complete**);**

**}**

void state\_Display**(){**

display\_dataset**();**

SetEvent**(**Complete**);**

**}**

void state\_WaitPoll**(){**

mdelay**(**3000**);**

tsi\_position\_t final\_pos**;**

// wait until SysTick\_Handler set timeout event flag

uint32\_t left\_pos\_count **=** 0**;**

uint32\_t right\_pos\_count **=** 0**;**

**while(**event **==** Complete**)**

**{**

// keep polling slider position

tsi\_position\_t pos **=** Touch\_Scan\_Position**();**

**if** **(**pos **==** left**)**

left\_pos\_count**++;**

**else** **if** **(**pos **==** right**)**

right\_pos\_count**++;**

**}**

**if** **(**left\_pos\_count **>** right\_pos\_count**)**

final\_pos **=** left**;**

**else** **if** **(**left\_pos\_count **<** right\_pos\_count**)**

final\_pos **=** right**;**

**else**

final\_pos **=** left**;**

// if slider is right or event is timeout\_6

**if** **((**final\_pos **==** right**)** **||** **(**event **==** Timeout\_6**))**

**{**

// change to the other state machine

machine **=** machine **^** 0x1**;**

**}**

**}**

void state\_End**(){**

// halt the program

turn\_LED\_green**(**off**);**

turn\_LED\_red**(**on**);**

**while(**1**);**

**}**

void state\_Error**(){**

// halt the program

**while(**1**);**

**}**

void SetEvent**(**tEvent evt**){**

event **=** evt**;**

**}**

/\* state machine related functions \*/

void RunMachines**(**void**){**

LOG\_INFO**(**"current state machine is %s"**,**state\_machine\_type\_string**[**machine**]);**

**switch(**machine**){**

**case** mStateCentric**:**

RunStateCentric**();**

**break;**

**case** mTableDriven**:**

RunTableDriven**();**

**break;**

**}**

**}**

void RunStateCentric**(**void**){**

//sStateTableEntry \*currentState= stateTable[kReadXYZ];

/\* Do the action in this state \*/

stateTable**[**currentState**].**func\_p**();**

/\* Set the state to go next \*/

**switch(**currentState**){**

**case** kReadXYZ**:**

**if** **(**event**==**Complete**){**

HandleEventComplete**();**

**}**

**else{**

printf**(**"last event is %d"**,(**int**)**event**);**

**}**

**break;**

**case** kProcessDisplay**:**

**if** **(**event**==**Complete**){**

HandleEventComplete**();**

**}**

**else{**

printf**(**"last event is %d"**,(**int**)**event**);**

**}**

**break;**

**case** kWaitPollSlider**:**

**if** **(**event**==**Timeout\_1\_5**){**

HandleEventTimeout\_1\_5**();**

**}**

**else** **if** **(**event**==**Timeout\_6**){**

HandleEventTimeout\_6**();**

**}**

**else** **if** **(**event**==**LeftSlider**){**

HandleEventLeftSlider**();**

**}**

**else** **if** **(**event**==**RightSlider**){**

HandleEventRightSlider**();**

**}**

**else{**

LOG\_ERROR**(**"last event is %d"**,(**int**)**event**);**

**}**

**break;**

**default:**

LOG\_ERROR**(**"Error state reached"**);**

LOG\_ERROR**(**"last event is %d"**,(**int**)**event**);**

**break;**

**}**

**}**

//kReadXYZ, kProcessDisplay, kSensorDisconnect, kWaitPollSlider

void RunTableDriven**(**void**){**

//sStateTableEntry \*currentState= stateTable[kReadXYZ];

/\* Do the action in this state \*/

stateTable**[**currentState**].**func\_p**();**

/\* Set the state to go next \*/

currentState**=**stateTable**[**currentState**].**next\_state**[**event**];**

**}**

void HandleEventComplete**(**void**){**

currentState**=**stateTable**[**currentState**].**next\_state**[**Complete**];**

**}**

void HandleEventTimeout\_1\_5**(**void**){**

currentState**=**stateTable**[**currentState**].**next\_state**[**Timeout\_1\_5**];**

**}**

void HandleEventTimeout\_6**(**void**){**

currentState**=**stateTable**[**currentState**].**next\_state**[**Timeout\_6**];**

**}**

void HandleEventLeftSlider**(**void**){**

currentState**=**stateTable**[**currentState**].**next\_state**[**LeftSlider**];**

**}**

void HandleEventRightSlider**(**void**){**

currentState**=**stateTable**[**currentState**].**next\_state**[**RightSlider**];**

**}**

**mma8451.c**

/\*

\* mma8451.c

\*

\* Created on: Apr 1, 2020

\* Author: user

\*/

//adapted from https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter\_8/I2C-Demo/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <MKL25Z4.h>

#include "mma8451.h"

#include "i2c.h"

#include "timer.h"

#include "logger.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MAX\_DATA\_SET\_COUNT 3

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// data variables

uint32\_t data\_set\_count **=** 0**;**

uint32\_t current\_data\_set\_num **=** 0**;**

int16\_t acc\_x**[**MAX\_DATA\_SET\_COUNT**];** // syntax for every entry to be 0

int16\_t acc\_y**[**MAX\_DATA\_SET\_COUNT**];**

int16\_t acc\_z**[**MAX\_DATA\_SET\_COUNT**];**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t average**(**int16\_t arr**[],** uint32\_t n**)**

**{**

int i**;**

int average **=** arr**[**0**];**

**for** **(**i **=** 0**;** i **<** n**;** i**++)**

average **=** average **+** arr**[**i**];**

**return** **(**average**/**n**);**

**}**

// source: https://www.geeksforgeeks.org/c-program-find-largest-element-array/

int16\_t max**(**int16\_t arr**[],** uint32\_t n**)**

**{**

int i**;**

int max **=** arr**[**0**];**

**for** **(**i **=** 1**;** i **<** n**;** i**++)**

**if** **(**arr**[**i**]** **>** max**)**

max **=** arr**[**i**];**

**return** max**;**

**}**

int16\_t min**(**int16\_t arr**[],** uint32\_t n**)**

**{**

int i**;**

int min **=** arr**[**0**];**

**for** **(**i **=** 1**;** i **<** n**;** i**++)**

**if** **(**arr**[**i**]** **<** min**)**

min **=** arr**[**i**];**

**return** min**;**

**}**

int init\_mma8451**()**

**{**

// make sure device is connected

uint8\_t whoami\_val **=** 0**;**

whoami\_val **=** i2c\_read\_byte**(**MMA\_ADDR**,** REG\_WHOAMI**);**

**if** **(**whoami\_val **!=** WHOAMI**)**

**{**

// error, either not connected

// or corrupted data

**return** 1**;**

**}**

// set active mode, 14 bit samples and 800 Hz ODR

i2c\_write\_byte**(**MMA\_ADDR**,** REG\_CTRL1**,** 0x03**);**

**return** 0**;**

**}**

int verify\_connect\_mma8451**()**

**{**

// make sure device is connected

uint8\_t whoami\_val **=** 0**;**

whoami\_val **=** i2c\_read\_byte**(**MMA\_ADDR**,** WHOAMI**);**

**if** **(**whoami\_val **!=** 0x1A**)**

**{**

// error, either not connected

// or corrupted data

**return** 1**;**

**}**

**return** 0**;**

**}**

void read\_full\_xyz**()**

**{**

int i**;**

uint8\_t data**[**6**];**

int16\_t temp**[**3**];**

i2c\_start**();**

i2c\_read\_setup**(**MMA\_ADDR **,** REG\_XHI**);**

// Read five bytes in repeated mode

**for(** i**=**0**;** i**<**5**;** i**++)** **{**

data**[**i**]** **=** i2c\_repeated\_read**(**0**);**

**}**

// Read last byte ending repeated mode

data**[**i**]** **=** i2c\_repeated\_read**(**1**);**

**for** **(** i**=**0**;** i**<**3**;** i**++** **)** **{**

temp**[**i**]** **=** **(**int16\_t**)** **((**data**[**2**\***i**]<<**8**)** **|** data**[**2**\***i**+**1**]);**

**}**

// Align for 14 bits

// assign to the current array

acc\_x**[**current\_data\_set\_num**]** **=** temp**[**0**]>>**2**;**

acc\_y**[**current\_data\_set\_num**]** **=** temp**[**1**]>>**2**;**

acc\_z**[**current\_data\_set\_num**]** **=** temp**[**2**]>>**2**;**

// update data\_set\_count

**if** **(**data\_set\_count **!=** MAX\_DATA\_SET\_COUNT**)**

data\_set\_count **++;**

// update current data\_set

**if** **(**current\_data\_set\_num **==** 2**)**

current\_data\_set\_num **=** 0**;**

**else** **if** **((**data\_set\_count**==**1**)||(**current\_data\_set\_num**==**0**))**

current\_data\_set\_num **=** current\_data\_set\_num**;**

**else**

current\_data\_set\_num **++;**

**}**

void read\_xyz**(**void**)**

**{**

// sign extend byte to 16 bits - need to cast to signed since function

// returns uint8\_t which is unsigned

acc\_x**[**current\_data\_set\_num**]** **=** **(**int8\_t**)** i2c\_read\_byte**(**MMA\_ADDR**,** REG\_XHI**)<<**6**;**

mdelay**(**100**);**

acc\_y**[**current\_data\_set\_num**]** **=** **(**int8\_t**)** i2c\_read\_byte**(**MMA\_ADDR**,** REG\_YHI**)<<**6**;**

mdelay**(**100**);**

acc\_z**[**current\_data\_set\_num**]** **=** **(**int8\_t**)** i2c\_read\_byte**(**MMA\_ADDR**,** REG\_ZHI**)<<**6**;**

// update data\_set\_count

**if** **(**data\_set\_count **!=** MAX\_DATA\_SET\_COUNT**)**

data\_set\_count **++;**

// update current data\_set

**if** **(**current\_data\_set\_num **==** 2**)**

current\_data\_set\_num **=** 0**;**

**else** **if** **((**data\_set\_count**==**1**)||(**current\_data\_set\_num**==**0**))**

current\_data\_set\_num **=** current\_data\_set\_num**;**

**else**

current\_data\_set\_num **++;**

**}**

void display\_dataset**()**

**{**

// Last X, Y, Z readings and the state entry count

LOG\_INFO**(**"Last: acc\_x = %d, acc\_y = %d, acc\_z = %d"**,**

acc\_x**[**current\_data\_set\_num**],**

acc\_y**[**current\_data\_set\_num**],**

acc\_z**[**current\_data\_set\_num**]);**

// Average X, Y, Z readings

LOG\_INFO**(**"Average: acc\_x = %d, acc\_y = %d, acc\_z = %d"**,**

average**(**acc\_x**,**MAX\_DATA\_SET\_COUNT**),**

average**(**acc\_y**,**MAX\_DATA\_SET\_COUNT**),**

average**(**acc\_z**,**MAX\_DATA\_SET\_COUNT**));**

// Low X, Y, Z readings

LOG\_INFO**(**"Low: acc\_x = %d, acc\_y = %d, acc\_z = %d"**,**

min**(**acc\_x**,**MAX\_DATA\_SET\_COUNT**),**

min**(**acc\_y**,**MAX\_DATA\_SET\_COUNT**),**

min**(**acc\_z**,**MAX\_DATA\_SET\_COUNT**));**

// High X, Y, Z readings

LOG\_INFO**(**"High: acc\_x = %d, acc\_y = %d, acc\_z = %d"**,**

max**(**acc\_x**,**MAX\_DATA\_SET\_COUNT**),**

max**(**acc\_y**,**MAX\_DATA\_SET\_COUNT**),**

max**(**acc\_z**,**MAX\_DATA\_SET\_COUNT**));**

**}**

**gpio.c**

/\*

\* gpio.c

\*

\* Created on: Feb 11, 2020

\* Author: chutao

\*

\* Minic the functions from fsl\_gpio.c

\* Still use MKL25Z4.h for hardware addresses

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdint.h>

#include "gpio.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void set\_GPIO\_Pinout**(**GPIO\_Type **\***port**,** uint32\_t pin**)**

**{**

port**->**PSOR **=** **(**0x1 **<<** pin**);**

**}**

void clear\_GPIO\_Pinout**(**GPIO\_Type **\***port**,** uint32\_t pin**)**

**{**

port**->**PCOR **=** **(**0x1 **<<** pin**);**

**}**

void toggle\_GPIO\_Pinout**(**GPIO\_Type **\***port**,** uint32\_t pin**)**

**{**

port**->**PTOR **=** **(**0x1 **<<** pin**);**

**}**

void init\_GPIO\_Pin**(**GPIO\_Type **\***port**,** uint32\_t pin**,**

gpio\_pin\_direct\_t pin\_direction**,** uint8\_t pin\_data**)**

**{**

**if** **(**pin\_direction **==** GPIO\_DigitalInput**)**

**{**

// Set pin to input direction

port**->**PDDR **&=** **~(**0x1 **<<** pin**);**

**}**

**else** **if** **(**pin\_direction **==** GPIO\_DigitalOutput**)**

**{**

// Set pin to output direction

port**->**PDDR **|=** **(**0x1 **<<** pin**);**

**if** **(**pin\_data**)**

**{**

set\_GPIO\_Pinout**(**port**,**pin**);**

**}**

**else**

**{**

clear\_GPIO\_Pinout**(**port**,**pin**);**

**}**

**}**

**else**

**{**

#ifdef LOGGING\_DEBUG

// TODO: Debug message

#endif

**}**

**}**

**led.c**

/\*

\* led.c

\*

\* Created on: Feb 11, 2020

\* Author: chutao

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <logger.h>

#include <stdint.h>

#include "gpio.h"

#include "led.h"

#include "timer.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Varibles \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

led\_color\_t color **=** red**;**

const char **\*** led\_color\_string**[**3**]** **={**"on"**,**"off"**,**"toggle"**};**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init\_LED**(**void**)**

**{**

init\_GPIO\_Pin**(**LED3\_RED\_PORT**,** LED3\_RED\_PIN**,** GPIO\_DigitalOutput**,** 1**);**

init\_GPIO\_Pin**(**LED3\_GREEN\_PORT**,** LED3\_GREEN\_PIN**,** GPIO\_DigitalOutput**,** 1**);**

init\_GPIO\_Pin**(**LED3\_BLUE\_PORT**,** LED3\_BLUE\_PIN**,** GPIO\_DigitalOutput**,** 1**);**

**}**

void turn\_LED**(**led\_state\_t LED\_state**)**

**{**

**if** **(**color **==** red**)**

**{**

turn\_LED\_red**(**LED\_state**);**

**}**

**else** **if** **(**color **==** green**)**

**{**

turn\_LED\_green**(**LED\_state**);**

**}**

**else** **if** **(**color **==** blue**)**

**{**

turn\_LED\_blue**(**LED\_state**);**

**}**

**else**

**{**

LOG\_ERROR**(**"Unexpected led\_state\_t"**);**

**}**

**}**

void change\_LED\_color**(**led\_color\_t LED\_color**)**

**{**

color **=** LED\_color**;**

**}**

void turn\_LED\_red**(**led\_state\_t LED\_state**)**

**{**

LOG\_DEBUG**(**"Turn LED red %s"**,**led\_color\_string**[**LED\_state**]);**

color **=** red**;**

**if** **(**LED\_state **==** off**)**

**{**

set\_GPIO\_Pinout**(**LED3\_RED\_PORT**,** LED3\_RED\_PIN**);**

**}**

**else** **if** **(**LED\_state **==** on**)**

**{**

clear\_GPIO\_Pinout**(**LED3\_RED\_PORT**,** LED3\_RED\_PIN**);**

**}**

**else** **if** **(**LED\_state **==** toggle**)**

**{**

toggle\_GPIO\_Pinout**(**LED3\_RED\_PORT**,** LED3\_RED\_PIN**);**

**}**

**else**

**{**

LOG\_ERROR**(**"Unexpected led\_state\_t"**);**

**}**

**}**

void turn\_LED\_green**(**led\_state\_t LED\_state**)**

**{**

LOG\_DEBUG**(**"Turn LED green %s"**,**led\_color\_string**[**LED\_state**]);**

color **=** green**;**

**if** **(**LED\_state **==** off**)**

**{**

set\_GPIO\_Pinout**(**LED3\_GREEN\_PORT**,** LED3\_GREEN\_PIN**);**

**}**

**else** **if** **(**LED\_state **==** on**)**

**{**

clear\_GPIO\_Pinout**(**LED3\_GREEN\_PORT**,** LED3\_GREEN\_PIN**);**

**}**

**else** **if** **(**LED\_state **==** toggle**)**

**{**

toggle\_GPIO\_Pinout**(**LED3\_GREEN\_PORT**,** LED3\_GREEN\_PIN**);**

**}**

**else**

**{**

LOG\_ERROR**(**"Unexpected led\_state\_t"**);**

**}**

**}**

void turn\_LED\_blue**(**led\_state\_t LED\_state**)**

**{**

LOG\_DEBUG**(**"Turn LED blue %s"**,**led\_color\_string**[**LED\_state**]);**

color **=** blue**;**

**if** **(**LED\_state **==** off**)**

**{**

set\_GPIO\_Pinout**(**LED3\_BLUE\_PORT**,** LED3\_BLUE\_PIN**);**

**}**

**else** **if** **(**LED\_state **==** on**)**

**{**

clear\_GPIO\_Pinout**(**LED3\_BLUE\_PORT**,** LED3\_BLUE\_PIN**);**

**}**

**else** **if** **(**LED\_state **==** toggle**)**

**{**

toggle\_GPIO\_Pinout**(**LED3\_BLUE\_PORT**,** LED3\_BLUE\_PIN**);**

**}**

**else**

**{**

LOG\_ERROR**(**"Unexpected led\_state\_t"**);**

**}**

**}**

**logger.c**

/\*

\* log.c

\*

\* Created on: Dec 18, 2018

\* Author: Chutao Wei

\*

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdint.h>

#include "logger.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

log\_status\_t log\_status **=** disable**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Log\_enable**(**void**)**

**{**

log\_status **=** enable**;**

**}**

void Log\_disable**(**void**)**

**{**

log\_status **=** disable**;**

**}**

log\_status\_t Log\_status**(**void**)**

**{**

**return** log\_status**;**

**}**

void Log\_data**(**uint32\_t data**)**

**{**

**if(**log\_status **==** enable**)**

**{**

printf**(**"Addr: 0x%08x, Data: 0x%08x"**,&**data**,**data**);**

**}**

**}**

void Log\_string**(**const char **\*** str**)**

**{**

**if(**log\_status **==** enable**)**

**{**

printf**(**"%s\n"**,** str**);**

**}**

**}**

void Log\_integer**(**int integer**)**

**{**

**if(**log\_status **==** enable**)**

**{**

printf**(**"%d\n"**,** integer**);**

**}**

**}**

**test.c**