EEC172 Lab 4 Report

Honglei Liu

Bingxuan Ying

Objectives:

In this lab, we want to design a circuit which is pretty much similar to the one in Lab 3: the ultimate goal for this lab is, instead of transferring the data which is received from AT&T controller, transferring the data that is received from a microphone between two CC3200 Launchpad using UART

Codes:

Main.c

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Application Name - int\_sw

// Application Overview - The objective of this application is to demonstrate

// GPIO interrupts using SW2 and SW3.

// NOTE: the switches are not debounced!

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup int\_sw

//! @{

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Standard includes

#include <stdio.h>

#include "string.h"

// Driverlib includes

#include "hw\_types.h"

#include "hw\_ints.h"

#include "hw\_memmap.h"

#include "hw\_common\_reg.h"

#include "interrupt.h"

#include "hw\_apps\_rcm.h"

#include "prcm.h"

#include "rom.h"

#include "rom\_map.h"

#include "prcm.h"

#include "gpio.h"

#include "utils.h"

#include "timer.h"

#include "systick.h"

#include "spi.h"

#include "uart.h"

// Common interface includes

#include "uart\_if.h"

#include "pinmux.h"

#include "timer\_if.h"

#include "gpio\_if.h"

#define MASTER\_MODE 1

#define SPI\_IF\_BIT\_RATE 400000

#define TR\_BUFF\_SIZE 100

#define BLACK 0x0000

#define WHITE 0xFFFF

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GLOBAL VARIABLES -- Start

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#if defined(ccs)

extern void (\* const g\_pfnVectors[])(void);

#endif

#if defined(ewarm)

extern uVectorEntry \_\_vector\_table;

#endif

volatile static tBoolean bRxDone;

long int coeff\_array[7] = {31548, 31281, 30951, 30556, 29144, 28361, 27409};

long int power\_all[7];

unsigned long A0TICK = (80000000 / 16000);

unsigned long A1TICK = 80000000;

unsigned char isSampling;

unsigned char isProcessing;

unsigned char isNewChar;

int isNew;

unsigned char UART\_flag;

unsigned short sample\_num;

int buf[400];//400

char Str4UART[60];

int new\_digit = 0;

int maxLength;

int j;

signed int k;

int x;

int y;

int Rx;

int Ry;

char checkList[12][4] = {

{' '}, //Button 0

{',', '.', '!'}, //Button 1

{'a', 'b', 'c'}, //Button 2

{'d', 'e', 'f'}, //Button 3

{'g', 'h', 'i'}, //Button 4

{'j', 'k', 'l'}, //Button 5

{'m', 'n', 'o'}, //Button 6

{'p', 'q', 'r', 's'}, //Button 7

{'t', 'u', 'v'}, //Button 8

{'w', 'x', 'y', 'z'}, //Button 9

{'#'}, //Button # -> send

{'\*'}, //Button \* -> delete

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GLOBAL VARIABLES -- End

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Handler

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Timer A0

// Used to control sampling

static void

TimerA0IntHandler(void)

{

unsigned long ulStatus;

ulStatus = MAP\_TimerIntStatus(TIMERA0\_BASE, true);

MAP\_TimerIntClear(TIMERA0\_BASE, ulStatus);

sample\_num++;

isSampling = 1;

if (sample\_num == 400)

isProcessing = 1;

}

// Timer A1

// Used for inter-button

static void

TimerA1IntHandler(void)

{

unsigned long ulStatus;

ulStatus = MAP\_TimerIntStatus(TIMERA1\_BASE, true);

MAP\_TimerIntClear(TIMERA1\_BASE, ulStatus);

isNewChar = 1;

}

static void UARTIntHandler(void) {

unsigned char transChar;

MAP\_UARTIntDisable(UARTA1\_BASE,UART\_INT\_RX);

while(UARTCharsAvail(UARTA1\_BASE))

{

transChar = MAP\_UARTCharGet(UARTA1\_BASE);

//printf("%c", transChar);

if(isNew)

{

fillRect(0, 65, 127, 127, BLACK);

isNew = 0;

}

if(Rx > 127) // About to go off right edge.

{

Rx = 0;

Ry += 8; // go to next line.

}

if(transChar == '\0')

{

//printf("\n\n");

isNew = 1;

Rx = 0;

Ry = 65;

}

else

{

drawChar(Rx, Ry, transChar, WHITE, BLACK, 1);

Rx += 8;

}

}

MAP\_UARTIntClear(UARTA1\_BASE,UART\_INT\_RX);

MAP\_UARTIntEnable(UARTA1\_BASE,UART\_INT\_RX);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// LOCAL FUNCTION PROTOTYPES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void BoardInit(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Board Initialization & Configuration

//!

//! \param None

//!

//! \return None

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void

BoardInit(void) {

/\* In case of TI-RTOS vector table is initialize by OS itself \*/

#ifndef USE\_TIRTOS

//

// Set vector table base

//

#if defined(ccs)

MAP\_IntVTableBaseSet((unsigned long)&g\_pfnVectors[0]);

#endif

#if defined(ewarm)

MAP\_IntVTableBaseSet((unsigned long)&\_\_vector\_table);

#endif

#endif

// Enable Processor

//

MAP\_IntMasterEnable();

MAP\_IntEnable(FAULT\_SYSTICK);

PRCMCC3200MCUInit();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Init

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void SPI\_Init(void)

{

// Enable the SPI module clock

MAP\_PRCMPeripheralClkEnable(PRCM\_GSPI,PRCM\_RUN\_MODE\_CLK);

// Reset the peripheral

MAP\_PRCMPeripheralReset(PRCM\_GSPI);

// Reset SPI

MAP\_SPIReset(GSPI\_BASE);

// Configure SPI interface

MAP\_SPIConfigSetExpClk(GSPI\_BASE,MAP\_PRCMPeripheralClockGet(PRCM\_GSPI),

SPI\_IF\_BIT\_RATE,SPI\_MODE\_MASTER,SPI\_SUB\_MODE\_0,

(SPI\_SW\_CTRL\_CS |

SPI\_4PIN\_MODE |

SPI\_TURBO\_OFF |

SPI\_CS\_ACTIVEHIGH |

SPI\_WL\_8));

MAP\_SPIEnable(GSPI\_BASE);

}

static void Timer\_Init(void)

{

unsigned long ulStatus;

// Init Timer A0

PRCMPeripheralClkEnable(PRCM\_TIMERA0, PRCM\_RUN\_MODE\_CLK);

PRCMPeripheralReset(PRCM\_TIMERA0);

TimerIntRegister(TIMERA0\_BASE, TIMER\_A, TimerA0IntHandler);

TimerConfigure(TIMERA0\_BASE, TIMER\_CFG\_PERIODIC);

TimerLoadSet(TIMERA0\_BASE, TIMER\_A, A0TICK);

ulStatus = TimerIntStatus(TIMERA0\_BASE, false);

TimerIntClear(TIMERA0\_BASE, ulStatus);

// Init Timer A1

PRCMPeripheralClkEnable(PRCM\_TIMERA1, PRCM\_RUN\_MODE\_CLK);

PRCMPeripheralReset(PRCM\_TIMERA1);

TimerIntRegister(TIMERA1\_BASE, TIMER\_A, TimerA1IntHandler);

TimerConfigure(TIMERA1\_BASE, TIMER\_CFG\_ONE\_SHOT);

TimerLoadSet(TIMERA1\_BASE, TIMER\_A, A1TICK);

ulStatus = TimerIntStatus(TIMERA1\_BASE, false);

TimerIntClear(TIMERA1\_BASE, ulStatus);

}

void UART\_Setup(void)

{

MAP\_UARTConfigSetExpClk(UARTA1\_BASE,MAP\_PRCMPeripheralClockGet(PRCM\_UARTA1),

115200, (UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |

UART\_CONFIG\_PAR\_NONE));

MAP\_UARTIntRegister(UARTA1\_BASE,UARTIntHandler);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Function

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned short

readExternalADC(void)

{

unsigned char data0;

unsigned char data1;

GPIOPinWrite(GPIOA1\_BASE, 0x40, 0x00);

MAP\_SPITransfer(GSPI\_BASE, 0, &data0, 0x1, SPI\_CS\_ENABLE);

MAP\_SPITransfer(GSPI\_BASE, 0, &data1, 0x1, SPI\_CS\_DISABLE);

GPIOPinWrite(GPIOA1\_BASE, 0x40, 0x40);

unsigned short data = 0x1f & data0;

data = (data << 5) | ((0xf8 & data1) >> 3);

return data;

}

long int goertzel(long int coeff)

{

//initialize variables to be used in the function

int Q, Q\_prev, Q\_prev2,i;

long prod1,prod2,prod3,power;

Q\_prev = 0; //set delay element1 Q\_prev as zero

Q\_prev2 = 0; //set delay element2 Q\_prev2 as zero

power=0; //set power as zero

for (i=0; i<400; i++) // loop SAMPLE\_SPACE times and calculate Q, Q\_prev, Q\_prev2 at each iteration

{

Q = (buf[i]) + ((coeff\* Q\_prev)>>14) - (Q\_prev2); // >>14 used as the coeff was used in Q15 format

Q\_prev2 = Q\_prev; // shuffle delay elements

Q\_prev = Q;

}

//calculate the three products used to calculate power

prod1=((long) Q\_prev\*Q\_prev);

prod2=((long) Q\_prev2\*Q\_prev2);

prod3=((long) Q\_prev \*coeff)>>14;

prod3=(prod3 \* Q\_prev2);

power = ((prod1+prod2-prod3))>>8; //calculate power using the three products and scale the result down

return power;

}

char decode(void) // post\_test() function from the Github example

{

//initialize variables to be used in the function

int max\_power,i, row, col;

char buttonList[4][4] = {

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'},

};

// find the maximum power in the row frequencies and the row number

max\_power=0; //initialize max\_power=0

for(i=0;i<4;i++) { //loop 4 times from 0>3 (the indecies of the rows)

if (power\_all[i] > max\_power) { //if power of the current row frequency > max\_power

max\_power=power\_all[i]; //set max\_power as the current row frequency

row=i; //update row number

}

}

// find the maximum power in the column frequencies and the column number

max\_power=0; //initialize max\_power=0

for(i=4;i<7;i++) { //loop 3 times from 4>7 (the indecies of the columns)

if (power\_all[i] > max\_power) { //if power of the current column frequency > max\_power

max\_power=power\_all[i]; //set max\_power as the0 current column frequency

col=i; //update column number

}

}

if(power\_all[col]<=10 && power\_all[row]<=10) //instead 0 in the original example, set a threshold to avoid noise in the lab

new\_digit = 1;

if((power\_all[col]>70 && power\_all[row]>70) && (new\_digit == 1)) { // check if maximum powers of row & column exceed certain threshold

new\_digit = 0;

col -= 4;

return buttonList[row][col];

}

return 'x';

}

static int

rowCheck(char text) {

int message;

if(text == '0') { message = 0; maxLength = 0;}

else if(text == '1') { message = 1; maxLength = 2;}

else if(text == '2') { message = 2; maxLength = 2;}

else if(text == '3') { message = 3; maxLength = 2;}

else if(text == '4') { message = 4; maxLength = 2;}

else if(text == '5') { message = 5; maxLength = 2;}

else if(text == '6') { message = 6; maxLength = 2;}

else if(text == '7') { message = 7; maxLength = 3;}

else if(text == '8') { message = 8; maxLength = 2;}

else if(text == '9') { message = 9; maxLength = 3;}

else if(text == '#') { message = 10; maxLength = 0;}

else if(text == '\*') { message = 11; maxLength = 0;}

else{message = 12;}

return message;

}

static void Char2OLED(signed char newChar, signed char oldChar)

{

int i = rowCheck(newChar);

int n;

char input;

if(newChar == '\*'){

Str4UART[k] = '\0';

k--;

drawChar(x, y, ' ', WHITE, BLACK, 1);

x -= 8;

if (x == 0 && y >= 8) {

x = 119;

y -= 8;

}

if (x <= 3 && y <= 3) {

x = 0;

y = 0;

}

}

else if(newChar == '#'){

fillRect(0, 0, 127, 63, BLACK);

Str4UART[k] = '\0';

printf("K ---> %d\n", k);

printf("WTF %s\n", Str4UART);

for(n = 0; n <= strlen(Str4UART); n++){

printf("%c", Str4UART[n]);

MAP\_UARTCharPut(UARTA1\_BASE, Str4UART[n]);

}

printf("\n\n");

x = 0;

y = 0;

k = -1;

}

if(newChar != 'x' && newChar != '#' && newChar != '\*'){

if(newChar == oldChar && j < maxLength){

j++;

}

drawChar(x, y, checkList[i][j], WHITE, BLACK, 1);

input = checkList[i][j];

Str4UART[k] = input;

printf("#%d: input => %c\n", k, Str4UART[k]);

//printf("last => %c\n", Str4UART[k-1]);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! Main function

//!

//! \param none

//!

//!

//! \return None.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main() {

// Initializations

BoardInit();

PinMuxConfig();

SPI\_Init();

Timer\_Init();

UART\_Setup();

Adafruit\_Init();

fillScreen(BLACK);

drawLine(0, 64, 127, 64, WHITE);

InitTerm();

ClearTerm();

Message("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

Message("\t\t\t IR Remote TX/RX Application Start \n\r");

Message("\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

Message("\n\n\n\r");

// variable setups

unsigned long ulStatus;

char newChar = 'x';

char oldChar = 'x';

int i;

j = 0;

k = -1;

x = 0;

y = 0;

Rx = 0;

Ry = 0;

isNew = 1;

isSampling = 0;

isNewChar = 0;

isProcessing = 0;

sample\_num = 0;

ulStatus = MAP\_UARTIntStatus(TIMERA1\_BASE, true);

MAP\_UARTIntClear(TIMERA1\_BASE, ulStatus);

MAP\_UARTIntEnable(UARTA1\_BASE,UART\_INT\_RX|UART\_INT\_RT);

MAP\_UARTEnable(UARTA1\_BASE);

TimerIntEnable(TIMERA0\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerEnable(TIMERA0\_BASE, TIMER\_A);

//?

MAP\_TimerLoadSet(TIMERA1\_BASE, TIMER\_A, A1TICK);

MAP\_TimerIntEnable(TIMERA1\_BASE, TIMER\_A);

MAP\_TimerEnable(TIMERA1\_BASE, TIMER\_TIMA\_TIMEOUT);

while (1) {

if (isSampling == 1) {

isSampling = 0;

// 1024\*V\_bias/V\_ref = 1024\*1.2/3.3 = 372

buf[sample\_num-1] = (int) readExternalADC();

}

if (isProcessing == 1) {

// disable sampling timer

MAP\_TimerDisable(TIMERA0\_BASE, TIMER\_A);

MAP\_TimerIntDisable(TIMERA0\_BASE, TIMER\_TIMA\_TIMEOUT);

unsigned long ulStatus;

ulStatus = MAP\_TimerIntStatus(TIMERA0\_BASE, false);

MAP\_TimerIntClear(TIMERA0\_BASE, ulStatus);

// reset state variables

isProcessing = 0;

sample\_num = 0;

for (i = 0; i < 7; i++)

power\_all[i] = goertzel(coeff\_array[i]) >> 13;

newChar = decode();

//print the pressed button on console

if(newChar != 'x'){

printf("### %c\n", newChar);

printf("------------------------------\n");

}

if (newChar > 0 && newChar != 'x') {

MAP\_TimerDisable(TIMERA1\_BASE, TIMER\_A);

MAP\_TimerIntDisable(TIMERA1\_BASE, TIMER\_TIMA\_TIMEOUT);

// disable

if((newChar != oldChar || isNewChar == 1) && newChar != '\*'){

k++;

j = 0;

oldChar = 'x';

x+=8;

if(x > 127){

x = 0;

y+=8;

if(y > 127)

y = 0;

}

}

//printf("check NEW %c \n", newChar);

Char2OLED(newChar, oldChar);

isNewChar = 0;

oldChar = newChar;

//printf("check OLD %c \n\n", oldChar);

MAP\_TimerLoadSet(TIMERA1\_BASE, TIMER\_A, A1TICK);

MAP\_TimerIntEnable(TIMERA1\_BASE, TIMER\_A);

MAP\_TimerEnable(TIMERA1\_BASE, TIMER\_TIMA\_TIMEOUT);

}

// Re-enable sampling timer

MAP\_TimerLoadSet(TIMERA0\_BASE, TIMER\_A, A0TICK);

MAP\_TimerIntEnable(TIMERA0\_BASE, TIMER\_A);

MAP\_TimerEnable(TIMERA0\_BASE, TIMER\_TIMA\_TIMEOUT);

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Close the Doxygen group.

//! @}

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pinmux.c

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// pinmux.c

//

// configure the device pins for different peripheral signals

//

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

//

//

// Redistribution and use in source and binary forms, with or without

// modification, are permitted provided that the following conditions

// are met:

//

// Redistributions of source code must retain the above copyright

// notice, this list of conditions and the following disclaimer.

//

// 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.

//

// Neither the name of Texas Instruments Incorporated 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

// OWNER 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.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This file was automatically generated on 7/21/2014 at 3:06:20 PM

// by TI PinMux version 3.0.334

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "pinmux.h"

#include "hw\_types.h"

#include "hw\_memmap.h"

#include "hw\_gpio.h"

#include "pin.h"

#include "rom.h"

#include "rom\_map.h"

#include "gpio.h"

#include "prcm.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void

PinMuxConfig(void)

{

//

// Set unused pins to PIN\_MODE\_0 with the exception of JTAG pins 16,17,19,20

//

PinModeSet(PIN\_01, PIN\_MODE\_0);

PinModeSet(PIN\_02, PIN\_MODE\_0);

PinModeSet(PIN\_04, PIN\_MODE\_0);

PinModeSet(PIN\_15, PIN\_MODE\_0);

PinModeSet(PIN\_21, PIN\_MODE\_0);

PinModeSet(PIN\_52, PIN\_MODE\_0);

PinModeSet(PIN\_53, PIN\_MODE\_0);

PinModeSet(PIN\_58, PIN\_MODE\_0);

PinModeSet(PIN\_59, PIN\_MODE\_0);

PinModeSet(PIN\_60, PIN\_MODE\_0);

PinModeSet(PIN\_61, PIN\_MODE\_0);

PinModeSet(PIN\_62, PIN\_MODE\_0);

PinModeSet(PIN\_63, PIN\_MODE\_0);

PinModeSet(PIN\_64, PIN\_MODE\_0);

//

// Enable Peripheral Clocks

//

MAP\_PRCMPeripheralClkEnable(PRCM\_GPIOA1, PRCM\_RUN\_MODE\_CLK);

MAP\_PRCMPeripheralClkEnable(PRCM\_GPIOA2, PRCM\_RUN\_MODE\_CLK);

MAP\_PRCMPeripheralClkEnable(PRCM\_GPIOA3, PRCM\_RUN\_MODE\_CLK);

MAP\_PRCMPeripheralClkEnable(PRCM\_GSPI, PRCM\_RUN\_MODE\_CLK);

MAP\_PRCMPeripheralClkEnable(PRCM\_UARTA0, PRCM\_RUN\_MODE\_CLK);

MAP\_PRCMPeripheralClkEnable(PRCM\_UARTA1, PRCM\_RUN\_MODE\_CLK);

//

// Configure PIN\_03 for GPIO Output (DC)

//

MAP\_PinTypeGPIO(PIN\_03, PIN\_MODE\_0, false);

MAP\_GPIODirModeSet(GPIOA1\_BASE, 0x10, GPIO\_DIR\_MODE\_OUT);

//

// Configure PIN\_08 for GPIO Output (R)

//

MAP\_PinTypeGPIO(PIN\_08, PIN\_MODE\_0, false);

MAP\_GPIODirModeSet(GPIOA2\_BASE, 0x2, GPIO\_DIR\_MODE\_OUT);

//

// Configure PIN\_18 for GPIO Output (OC)

//

MAP\_PinTypeGPIO(PIN\_18, PIN\_MODE\_0, false);

MAP\_GPIODirModeSet(GPIOA3\_BASE, 0x10, GPIO\_DIR\_MODE\_OUT);

//

// Configure PIN\_05 for GPIOOutput (ADC - CS)

//

MAP\_PinTypeGPIO(PIN\_05, PIN\_MODE\_0, false);

MAP\_GPIODirModeSet(GPIOA1\_BASE, 0x40, GPIO\_DIR\_MODE\_OUT);

//

// Configure PIN\_50 for SPI0 GSPI\_CS

//

MAP\_PinTypeSPI(PIN\_50, PIN\_MODE\_9);

//

// Configure PIN\_45 for SPI0 GSPI\_CLK (CL)

//

MAP\_PinTypeSPI(PIN\_45, PIN\_MODE\_7);

//

// Configure PIN\_07 for SPI0 GSPI\_MISO (SL)

//

MAP\_PinModeSet(PIN\_06, PIN\_MODE\_7);

//

// Configure PIN\_07 for SPI0 GSPI\_MOSI (SL)

//

MAP\_PinTypeSPI(PIN\_07, PIN\_MODE\_7);

//

// Configure PIN\_55 for UART0 UART0\_TX

//

MAP\_PinTypeUART(PIN\_55, PIN\_MODE\_3);

//

// Configure PIN\_57 for UART0 UART0\_RX

//

MAP\_PinTypeUART(PIN\_57, PIN\_MODE\_3);

//

// Configure PIN\_01 for UART1 UART1\_TX

//

MAP\_PinTypeUART(PIN\_01, PIN\_MODE\_7);

//

// Configure PIN\_02 for UART1 UART1\_RX

//

MAP\_PinTypeUART(PIN\_02, PIN\_MODE\_7);

}

Procedure:

We just followed the procedure written on the lab manual: we first built our circuit and then followed the github example to build our own DTMF detection and decoding logics, which was pretty similar to what we did in the previous lab. Then, we used about the same logic from lab 3 to build the circuit for transferring data between two boards.

Notice:

Since in this lab, we need to connect much more things with each other comparing with Lab 3. Thus, I used the blue line on the bread board as GND and red line as VCC and then connect them with the OLED, the microphone and the LDO. Also, using GND from on GPIO on the board can give more stable signals and reduce the noise.

Besides, when we connected the two boards together using UART1, we also need to pay attention that we want the GND on two boards have the same sampling rate so in our design, we use one GND from one of the boards to build the circuit in order to avoid any possible noise and intervention that using multi GND may have.

As for our design, we used to have trouble in successfully sending characters from one board to the other by using UART. The reason for that was in our case design, we initialize the counting flag k to 0 but we do k++ immediately so that we cannot going to the case for displaying the received message on OLED. After we found the bug, we turned flag k to a signed integer and initialized it to -1 at the very beginning.