```
//*****************************************************************************
//
// Bradley Manzo
// Thomas Ke
// EEC 172 SQ23
// Lab 3 Code
//
//*****************************************************************************

// Standard includes
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>

// Driverlib includes
#include "hw_types.h"
#include "gpio.h"
#include "hw_apps_rcm.h"
#include "hw_common_reg.h"
#include "hw_ints.h"
#include "hw_memmap.h"
#include "hw_nvic.h"
#include "interrupt.h"
#include "prcm.h"
#include "rom.h"
#include "rom_map.h"
#include "prcm.h"
#include "spi.h"
#include "systick.h"
#include "uart.h"
#include "utils.h"

// Common interface includes
#include "uart_if.h"

// Pin configurations
#include "Adafruit_GFX.h"
#include "Adafruit_SSD1351.h"
#include "glcdfont.h"
#include "pin_mux_config.h"


//*****************************************************************************
```

```
//                GLOBAL VARIABLES -- Start
//*****************************************************************************

// some helpful macros for systick

// the cc3200's fixed clock frequency of 80 MHz
// note the use of ULL to indicate an unsigned long long constant
#define SYSCLKFREQ 80000000ULL

// macro to convert ticks to microseconds
#define TICKS_TO_US(ticks) \
    ((((ticks) / SYSCLKFREQ) * 1000000ULL) + \
    ((((ticks) % SYSCLKFREQ) * 1000000ULL) / SYSCLKFREQ))\

// macro to convert microseconds to ticks
#define US_TO_TICKS(us) ((SYSCLKFREQ / 1000000ULL) * (us))

// systick reload value set to 40ms period
// (PERIOD_SEC) * (SYSCLKFREQ) = PERIOD_TICKS
#define SYSTICK_RELOAD_VAL 3200000UL

#define MASTER_MODE      1

#define SPI_IF_BIT_RATE  100000
#define TR_BUFF_SIZE     100

#define BLACK          0x0000
#define BLUE           0x001F
#define GREEN          0x07E0
#define CYAN           0x07FF
#define RED            0xF800
#define MAGENTA        0xF81F
#define YELLOW         0xFFE0
#define WHITE          0xFFFF

#define CONSOLE        UARTA1_BASE
#define CONSOLE_PERIPH  PRCM_UARTA1
#define UartGetChar()       MAP_UARTCharGet(CONSOLE)
#define UartPutChar(c)      MAP_UARTCharPut(CONSOLE,c)
#define MAX_STRING_LENGTH    80

// track systick counter periods elapsed
// if it is not 0, we know the transmission ended
volatile int systick_cnt = 1;
```

```
extern void (* const g_pfnVectors[])(void);
volatile unsigned char P59_intstatus;
volatile unsigned long P59_intcount;
volatile unsigned char P2_intstatus;
volatile unsigned long P2_intcount;

unsigned long start_int;
unsigned long end_int;

char TextRx[MAX_STRING_LENGTH+1];
int TextRxLength = 0;
char TextTx[MAX_STRING_LENGTH+1];
int TextTxLength = 0;
int i = 0;

uint64_t delta = 0;
uint64_t delta_us = 0;

// Int to accumulate bits onto to form message
uint32_t message;
uint32_t prev_message;

// Variables to maintain repetition logic
char prev_char;
int repetitions = 0;
char character;
// Array to maintain font color
int colors[7] = {BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, WHITE};
int font_count = 0;
// Array to store characters corresponding to repeated button presses
char letters3[6][3] = {{'A', 'B', 'C'},
                       {'D', 'E', 'F'},
                       {'G', 'H', 'I'},
                       {'J', 'K', 'L'},
                       {'M', 'N', 'O'},
                       {'T', 'U', 'V'}};

char letters4[2][4] = {{'P', 'Q', 'R', 'S'},
                       {'W', 'X', 'Y', 'Z'}};


//*****************************************************************************
//                      GLOBAL VARIABLES -- End
```

```
//*****************************************************************************

//*****************************************************************************
//                    LOCAL FUNCTION PROTOTYPES
//*****************************************************************************
static void BoardInit(void);

//*****************************************************************************
//                    LOCAL FUNCTION DEFINITIONS
//*****************************************************************************

/**
 * Reset SysTick Counter
 */
static inline void SysTickReset(void) {
    // any write to the ST_CURRENT register clears it
    // after clearing it automatically gets reset without
    // triggering exception logic
    // see reference manual section 3.2.1
    HWREG(NVIC_ST_CURRENT) = 1;

    // clear the global count variable
    systick_cnt = 1;
}

/**
 * SysTick Interrupt Handler
 *
 * Keep track of whether the systick counter wrapped
 */
static void SysTickHandler(void) {
    // increment every time the systick handler fires
    systick_cnt++;
}

//*****************************************************************************
//
//! 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();
}

// Register Interrupt Handler
// P59 handler wired to IR receiver
static void GPIOA0IntHandler(void)
{
    unsigned long ulStatus;
    // Records interrupt status of IR receiver from GPIO
    ulStatus = MAP_GPIOIntStatus(GPIOA0_BASE, true);
    // Clears interrupt status from GPIO
    MAP_GPIOIntClear(GPIOA0_BASE, ulStatus);
    // Records the current time and calculates duration since last
    delta = systick_cnt*SYSTICK_RELOAD_VAL - SysTickValueGet();
    // Resets SysTick count and repetitions
    SysTickReset();
    // Converts clock cycles to milliseconds
    delta_us = TICKS_TO_US(delta);// clear interrupts on GPIOA0
    // Sets IR Int received flag high
    P59_intstatus = 1;
    P59_intcount++;
}
static void UARTA1IntHandler(void)
{
    unsigned long ulStatus;
```

```
    // Records interrupt status of UART
    ulStatus = MAP_UARTIntStatus(CONSOLE, true);
    // Clears interrupt status from UART
    MAP_UARTIntClear(CONSOLE, ulStatus);
    // As long as there are chars to receive, build string
    while(UARTCharsAvail(CONSOLE))
    {
        TextRx[TextRxLength] = UARTCharGetNonBlocking(CONSOLE);
        TextRxLength++;
    }
    // Sets UART Int received flag high
    P2_intstatus = 1;
    P2_intcount++;
}

static void SysTickInit(void) {

    // configure the reset value for the systick countdown register
    MAP_SysTickPeriodSet(SYSTICK_RELOAD_VAL);

    // register interrupts on the systick module
    MAP_SysTickIntRegister(SysTickHandler);

    // enable interrupts on systick
    // (trigger SysTickHandler when countdown reaches 0)
    MAP_SysTickIntEnable();

    // enable the systick module itself
    MAP_SysTickEnable();
}

//*****************************************************************************
//
//! Main function
//!
//! \param none
//!
//!
//! \return None.
//
//*****************************************************************************

//bool signal_detector = 0;
```

```
int main() {
    unsigned long ulStatus;

    BoardInit();

    PinMuxConfig();

    //
    // 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 to OLED
    //
    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_ACTIVELOW |
                    SPI_WL_8));

    //
    // Enable SPI for communication to OLED
    //
    MAP_SPIEnable(GSPI_BASE);

    Adafruit_Init();

    // Enable SysTick
    SysTickInit();
```

```
    // Configure UART to A1BASE

MAP_UARTConfigSetExpClk(CONSOLE,MAP_PRCMPeripheralClockGet(CONSOLE_PERIPH
),
                        UART_BAUD_RATE, (UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));

    // Configure UART FIFO queue
    UARTFIFODisable(CONSOLE);

    MAP_UARTIntRegister(CONSOLE, UARTA1IntHandler);

    UARTFIFOLevelSet(CONSOLE, UART_FIFO_TX1_8, UART_FIFO_RX1_8);

    // Initialize UART interrupts
    ulStatus = MAP_UARTIntStatus(CONSOLE, false);

    MAP_UARTIntClear(CONSOLE, ulStatus);

    MAP_UARTIntEnable(CONSOLE,UART_INT_RX);

    // Initialize UART Terminal
    InitTerm();

    // Clear UART Terminal
    ClearTerm();

    // Register Interrupt Handler
    // (Port, pointer to handler function)
    MAP_GPIOIntRegister(GPIOA0_BASE, GPIOA0IntHandler);

    // Configure Falling Edge
    // (Port, bit-packed pin select, interrupt trigger mechanism)
    MAP_GPIOIntTypeSet(GPIOA0_BASE, 0x10, GPIO_FALLING_EDGE);

    // Interrupt Status
    // (Port, True: masked interupt status, false: raw interrupt status)
    // Returns the current interupt status enumerated as a bit field
    // of the values described in GPIOIntEnable()
    ulStatus = MAP_GPIOIntStatus(GPIOA0_BASE, false);

    // Clear Interrupt
    // (Port, with field returned from status above)
```

```
MAP_GPIOIntClear(GPIOA0_BASE, ulStatus);

// clear global variables
P59_intstatus = 0;
P59_intcount = 0;

// Enable Interrupt
// (Port, Flags)
MAP_GPIOIntEnable(GPIOA0_BASE, 0x10);

SysTickReset();

// Position in pixels
// Text to Transmit Position
int xTx = 0;
int yTx = 64;
// Text to Receive Position
int xRx = 0;
int yRx = 0;
setCursor(xTx, yTx);
setTextSize(1);
setTextColor(WHITE, BLACK);
fillScreen(BLACK);
memset(TextTx, 0, sizeof TextTx);
memset(TextRx, 0, sizeof TextRx);

while (1) {
   // Waits until a UART or GPIO interrupt is received
   while ((P59_intstatus==0) && (P2_intstatus==0)) {;}
   // If GPIO Interrupt (IR) Recevied
   if(P59_intstatus)
   {
      setCursor(xTx, yTx);
      // clear flag
      P59_intstatus=0;
      // If longer than standard repeat, stop remembering past input
      if(delta_us > 150000)
      {
         repetitions = 0;
         prev_message = 0;
         prev_char = 0;
      }
      // If larger than "1" and not garbage data, decode the message
      if((delta_us > 2500) && (delta_us < 150000) && (message != 0))
```

```
        {
            // If message message is new, and previous char wasn't a debug character, increment
the x position
            if(message != prev_message && prev_char != '!' && prev_char != '1' && prev_char !=
'2' && prev_char != '3')
            {
                // Append character to Transmitting Text
                TextTx[TextTxLength] = character;
                TextTxLength++;
                // If at edge of screen, go down to beginning of new line (\n\r)
                if(xTx >= 120)
                {
                    xTx = 0;
                    if(yTx < 120)
                        yTx += 8;
                    else
                        yTx = 64;
                }
                // Otherwise increment by width of character
                else
                {
                    xTx += 6;
                }
            }
            // If message is new and previous character was a backspace, decrement the x
position
            if(message != prev_message && prev_char == '3')
            {
                xTx -= 6;
            }
            // If last remembered word is the same: increment repetitions
            // otherwise, message is done repeating and should print


/*=====================================================================*/
            //      Infrared Decoding

/*=====================================================================*/
            // 0 Button (Space)
            if(message == 0b00000010111111010000000011111111)
            {
                character = ' ';
                prev_message = message;
                prev_char = character;
```

```
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);

}
// 1 Button pressed (Font Color Change)
else if(message == 0b00000010111111011000000001111111)
{
    if(font_count < 6)
        font_count++;
    else
        font_count = 0;
    character = '1';
    prev_message = message;
    prev_char = character;
}
// 2 button pressed
else if(message == 0b00000010111111010100000010111111)
{

    if(prev_message == message)
    {
        repetitions++;
    }
    else {
        repetitions = 0;
    }
    //letters = {'A', 'B', 'C'};
    if (repetitions > (sizeof(letters3[0]) - 1))
        repetitions = repetitions - (sizeof(letters3[0]) - 1);
    character = letters3[0][repetitions];
    prev_message = message;
    prev_char = character;
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);
}
// 3 button pressed
else if(message == 0b00000010111111011100000000111111)
{
    if(prev_message == message)
    {
        repetitions++;
    }
    else {
        repetitions = 0;
    }
    //letters = {'D', 'E', 'F'};
```

```
    if (repetitions > (sizeof(letters3[0]) - 1))
        repetitions = repetitions - (sizeof(letters3[0]) - 1);
    character = letters3[1][repetitions];
    prev_message = message;
    prev_char = character;
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);
}
// 4 button pressed
else if(message == 0b00000010111111010010000011011111)
{
    if(prev_message == message)
    {
        repetitions++;
    }
    else {
        repetitions = 0;
    }
    //letters = {'G', 'H', 'I'};
    if (repetitions > (sizeof(letters3[0]) - 1))
        repetitions = repetitions - (sizeof(letters3[0]) - 1);
    character = letters3[2][repetitions];
    prev_message = message;
    prev_char = character;
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);

}
// 5 button pressed
else if(message == 0b00000010111111011010000001011111)
{
    if(prev_message == message)
    {
        repetitions++;
    }
    else {
        repetitions = 0;
    }
    //letters = {'J', 'K', 'L'};
    if (repetitions > (sizeof(letters3[0]) - 1))
        repetitions = repetitions - (sizeof(letters3[0]) - 1);
    character = letters3[3][repetitions];
    prev_message = message;
    prev_char = character;
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);
```

```
}
// 6 button pressed
else if(message == 0b00000010111111010110000010011111)
{
   if(prev_message == message)
   {
      repetitions++;
   }
   else {
      repetitions = 0;
   }
   //letters = {'M', 'N', 'O'};
   if (repetitions > (sizeof(letters3[0]) - 1))
      repetitions = repetitions - (sizeof(letters3[0]) - 1);
   character = letters3[4][repetitions];
   prev_message = message;
   prev_char = character;
   drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);

}
// 7 button pressed
else if(message == 0b00000010111111011110000000011111)
{
   if(prev_message == message)
   {
      repetitions++;
   }
   else {
      repetitions = 0;
   }
   //letters = {'P', 'Q', 'R', 'S'};
   if (repetitions > (sizeof(letters4[0]) - 1))
      repetitions = repetitions - (sizeof(letters4[0]) - 1);
   character = letters4[0][repetitions];
   prev_message = message;
   prev_char = character;
   drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);

}
// 8 button pressed
else if(message == 0b00000010111111010001000011101111)
{
   if(prev_message == message)
   {
```

```
        repetitions++;
    }
    else {
        repetitions = 0;
    }
    //letters = {'T', 'U', 'V'};
    if (repetitions > (sizeof(letters3[0]) - 1))
        repetitions = repetitions - (sizeof(letters3[0]) - 1);
    character = letters3[5][repetitions];
    prev_message = message;
    prev_char = character;
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);

}
// 9 button pressed
else if(message == 0b00000010111111011001000001101111)
{
    if(prev_message == message)
    {
        repetitions++;
    }
    else {
        repetitions = 0;
    }
    //letters = {'W', 'X', 'Y', 'Z'};
    if (repetitions > (sizeof(letters4[0]) - 1))
        repetitions = repetitions - (sizeof(letters4[0]) - 1);
    character = letters4[1][repetitions];
    prev_message = message;
    prev_char = character;
    drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);

}
// Enter button pressed (MUTE)
else if(message == 0b00000010111111010000100011110111)
{
    prev_message = message;
    character = '2';
    prev_char = character;
}
// Delete button pressed (LAST)
else if(message == 0b00000010111111010000001011111101)
{
    //if(xTx >= 6)
```

```
        TextTx[TextTxLength] = '/0';
        // By removing from scope
        TextTxLength--;
        character = '3';
        prev_message = message;
        prev_char = character;
        xTx -= 6;
        fillRect(xTx,yTx,6, 8,BLACK);
    }
    else
    {
    // Otherwise, debugging character
            character = '!';
            prev_char = character;
            prev_message = message;
    }
    // If not a debugging or a function character, print the character to the screen
    if(character != '!' && character != '1' && character != '2' && character != '3')
    {
        drawChar(xTx, yTx, character, colors[font_count], BLACK, 1);
    }
    // If Enter button is pressed, transmit the text
    if(character == '2' && TextTxLength !=0)
    {
        for(i = 0; i < (TextTxLength + 1); i++)
        {
            UARTCharPut(CONSOLE,TextTx[i]);
        }
        TextTxLength = 0;
        memset(TextTx, 0, sizeof TextTx);
        setCursor(0, 64);
        xTx = 0;
        yTx = 64;
        fillRect(0,64,128,128,BLACK);
    }
    // Resets repetitions
    message = 0;
}
// If time is between 1300 and 2500 ms, accumulate a "1"
else if(delta_us < 2500 && delta_us > 1300)
{
    message = message << 1;
    message = message + 1;
}
```

```
    // If time is less than 1300 ms, accumulate a "0"
    else //if(delta_us > 0 && delta_us < 1300)
    {
        message = message << 1;
    }
    start_int = 0;
}
// Otherwise, receiving a UART Interrupt
else // P2_intstatus == 1
{
    // clear flag
    P2_intstatus=0;
    // Refresh top of screen
    fillRect(0,0,128,64,BLACK);
    xRx = 0;
    yRx = 0;
    setCursor(xRx, yRx);
    for(i = 0; i < TextRxLength + 1; i++)
    {
        // If receiving character is in the alphabet, print to OLED
        if(TextRx[i] >= 65 && TextRx[i] <= 90 || TextRx[i] == ' ')
        {
            drawChar(xRx, yRx, TextRx[i], colors[font_count], BLACK, 1);
            xRx += 6;
        }
        // If '1' Button (Font Color Change)
        if(TextRx[i] == '1')
        {
            if(font_count < 6)
                font_count++;
            else
                font_count = 0;
        }
        // MUTE Button (New Line)
        if(TextRx[i] == '2')
        {
            if(yRx <= 56)
            {
                yRx += 8;
                xRx = 0;
            }
            else
            {
                yRx = 0;
```

```
                }
            }
        }
        xRx = 0;
        yRx = 0;
        // Flush Transmitting text buffer
        memset(TextTx, 0, sizeof TextTx);
    }
  }
}

//*****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//*****************************************************************************
```