1 Instructions

You may work in pairs with a partner on this assignment if you wish or you may work alone. If you work with a partner, only submit one project file with both of your names in it; you will each earn the same number of points. Your project files must be uploaded to Blackboard by the assignment deadline. Section 4 describes what to submit and by when; read it now.

2 Lab Project Objectives

1. Configure and use the DTIM module to busy delay for a specific time period.
2. Configure and use the GPIO module to configure pins for digital I/O and primary/secondary/tertiary functions.
3. Configure and use the I2C module to communicate with the Wii Nunchuk using the I^2C serial interface standard.
4. Configure and use the PIT module to generate periodic interrupts at a specific frequency.

3 Lab Project

[Ref: [1,2,3]] Many embedded applications need to read data from sensors. It is common that an interface driver should be built as a part of execution environment to support the applications. For this project your task is to design and implement a driver to read data from a Wii Nunchuck.

The Wii Nunchuk controller consists of a 10-bit 3-axis accelerometer[4], 2-axis (X and Y) analog joystick, and two push buttons labeled C and Z. To retrieve state information from the Nunchuk, the I^2C-bus Standard Mode (100 Kbps) protocol is used. The Nunchuk connector contains six pins. Looking into the Wii Nunchuk connector, the pins are,

```
+---------+
| 1  2  3 |
| 4  5  6 |
|___---___|
```

Pin	Function
1	I^2C Serial Data (SDA)
2	Not connected
3	+3 V
4	Ground
5	Not connected
6	I^2C Serial Clock (SCL)

The Nunchuk reports it state information as six data bytes, in this format,

Byte	Description
0	Joystick x-axis (left/right) position. Center is nominally 128, full left is 0x00, full right is 255.
1	Joystick y-axis (up/down) position. Center is nominally 128, full up is 255, full down is 0[5].
2	Bits 9:2 of the 10-bit x-axis acceleration value
3	Bits 9:2 of the 10-bit y-axis acceleration value
4	Bits 9:2 of the 10-bit z-axis acceleration value
5	Button state and acceleration low order bits

The format of the sixth byte (number 5) is,

---

1 http://www.robotshop.com/ca/content/PDF/inex-zx-nunchuck-datasheet.pdf

2 http://www.musclera.com/wii-nunchuk-demonstration/

3 http://wiibrew.org/wiki/Wiimote/Extension_Controllers/Nunchuck

4 Part number: STMicroelectronics LIS3L02AE – Mems Inertial Sensor. I could not find the data sheet for the AE; the closest variant seems to be the LIS3L02AL and the data sheet is on the course website.

5 There is going to be some variation among different Nunchuks and among different Nunchuk manufacturers. Two of the Nunchuks I have tested reported these values: Nunchuk 1: center left/right = 135, center up/down = 129, full left = 1, full right = 254, full up = 254, full down = 0; Nunchuk 2: center left/right = 133, center up/down = 129, full left = 6, full right = 239, full up = 232, full down = 0.

Bit	Description
7:6	Bits 1:0 of the 10-bit z-axis acceleration value
5:4	Bits 1:0 of the 10-bit y-axis acceleration value
3:2	Bits 1:0 of the 10-bit x-axis acceleration value
1	Button C state (0 = pressed, 1 = not pressed)
0	Button Z state (0 = pressed, 1 = not pressed)

The acceleration values returned in bytes 2-4 are 8-bit values, and are the most-significant 8-bits of the actual 10-bit acceleration values returned by the inertial sensor. Bits 7:2 of byte 5 contains the least-significant 2-bits of the actual 10-bit acceleration values. Hence, to retrieve the complete 10-bit acceleration values we can write,

```
int x_accel = (int)(data[2] << 2 | (data[5] & 0x0C) >> 2);
int y_accel = (int)(data[3] << 2 | (data[5] & 0x30) >> 4);
int z_accel = (int)(data[4] << 2 | (data[5] & 0xC0) >> 6);
```

Since the acceleration values are 10-bits, the full range will be [0, 1023]. The LIS3L02AE can detect ±2 g of acceleation. If you are holding the Nunchuk upright and facing forward in your hand, the x-axis acceleration decreases as you rotate your hand to the left, and increases as you rotate your hand to the right. For the Nunchuk I have been using, the range of x-axis acceleration values for a stationary Nunchuk is: (rotated left) $388 \leq x\_accel \leq 853$ (rotated right). The y-axis acceleration values decrease as you point the front of the Nunchuk up and increase as you point it down. For my Nunchuk, I saw a stationary range of: (pointing up) $480 \leq y\_accel \leq 860$ (pointing down). The z-axis acceleration values decrease as the Nunchuk is rotated upside-down and increase as it is rotate right-side up. For my Nunchuk, I saw a stationary range of: (upside-down) $304 \leq z\_accel \leq 816$ (right-side up). These values will increase more (up to 1023) and decrease more (down to 0) as the movement of the Nunchuk is accelerated.

Internally the Nunchuk uses some sort of undocumented processor which is controlled by writing command bytes to registers. To retrieve the Nunchuk state information, we use the following protocol,

Initialize Nunchuk (Perform Once)

1. Configure MCF52259 to be master-transmitter. Sends the start bit.
2. Send the Nunchuk I^{2}C address (1010010_2 = 0x52) with R/$\overline{\text{W}}$ = write (the transmitted byte is 0xA4).
3. Transmit command 0x55 to Nunchuk register 0xF0 (send 0xF0 followed by 0x55).
4. Configure MCF52259 to be slave-receiver. Sends the stop bit.
5. Configure MCF52259 to be master-transmitter. Sends the start bit.
6. Send the Nunchuk I^{2}C address (1010010_2 = 0x52) with R/$\overline{\text{W}}$ = write (the transmitted byte is 0xA4).
7. Transmit command 0x00 to Nunchuk register 0xFB (send 0xFB followed by 0x00).
8. Configure MCF52259 to be slave-receiver. Sends the stop bit.

Read from Nunchuk

1. Configure MCF52259 to be master-transmitter. Sends the start bit.
2. Send the Nunchuk I^{2}C address (1010010_2 = 0x52) with R/$\overline{\text{W}}$ = write (the transmitted byte is 0xA4).
3. Transmit command 0x00 (send 0x00).
4. Configure MCF52259 to be slave-receiver. Sends the stop bit.
5. Configure MCF52259 to be master-transmitter. Sends the start bit.
6. Send the Nunchuk I^{2}C address (1010010_2 = 0x52) with R/$\overline{\text{W}}$ = read (the transmitted byte is 0xA5).
7. Receive the six data bytes from the Nunchuk. ACK the first five, and NACK the last one.
8. Configure MCF52259 to be slave-receiver. Sends the stop bit.

Software Requirements

1. The program shall display the state of the Wii Nunchuk on the console in the format shown below. For the buttons, 1 shall be displayed if the button is pressed and 0 if the button is not pressed.

   x-value y-value x-accel-value y-accel-value z-accel-value z-button c-button

2. The display shall be updated at 1 Hz.

3.1 Software Design

I am not going to dictate a detailed software design but I am going to require that you partition your code into modules with each module implementing common, specific functionality. The required modules are listed below.

Module	Source Files	Section	Remarks
dtim	dtim.c, dtim.h	§3.3.1	Reused
gpio	gpio.c, gpio.inc, gpio.h	§3.3.2	Reused
i2c	i2c.c, i2c.h	§3.3.3	New (implement the pseudocode in the Lecture Notes)
int	int.h, int.inc, int.s	§3.3.4	Reused (modify int.s per the BB announcement)
main	global.inc, main.c	§3.3.5	Provided
oct_nunchuk	oct_nunchuk.c, oct_nunchuk.h	§3.3.6	New (implement the pseudocode in the Lecture Notes)
pit	pit.c, pit.h	§3.3.7	Reused

3.3.1 dtim module — *dtim.c, dtim.h*

Reuse the code from Lab Project 5. Use *dtim_busy_delay_us*() to implement the required I^2C delays as described in the I2C pseudocode.

3.3.2 gpio module — *gpio.c, gpio.inc, gpio.h*

You should be able to reuse your code from previous projects and augment or modify as necessary.

3.3.3 i2c module — *i2c.c, i2c.h*

This module contains functions which uses the I^2C serial interface standard to communicate with an I^2C device. Suggested functionality,

1. Implements *i2c_acquire_bus*() which busy-waits until the I^2C bus is idle.
2. Implements *i2c_init*() which initializes the ColdFire I2C module per the pseudocode.
3. Implements *i2c_reset*() which enables the I2C module, makes the MCF52259 a slave-receiver, disables I2C interrupts, and disables the generation of repeated start bits.
4. Implements *i2c_rx*() which will receive n data bytes from an I^2C peripheral per the pseudocode.
5. Implements *i2c_rx_byte*() which will receive one byte from an I^2C peripheral per the pseudocode.
6. Implements *i2c_send_stop*() to send a stop bit.
7. Implements *i2c_tx*() which will transmit n data bytes to an I^2C peripheral.
8. Implements *i2c_tx_addr*() which will transmit the slave address and the read/write bit to an I^2C peripheral.
9. Implements *i2c_tx_byte*() to transmit one byte to an I^2C peripheral.
10. Implements *i2c_tx_complete*() which returns true if a transfer has completed and false if it has not.

Remarks

I have given you the pseudocode for this module in the lecture notes.

3.3.4 int module — *int.h, int.inc, int.s*

Reuse the code from Lab Project 5. Modify *int.s* to delete the statements on lines 126-128 that write 1 to IMRL [MASKALL].

3.3.5 main module — *main.c, global.inc*

Here is my *main.c*,

```c
//**************************************************************************************************************
// FILE: main.c
//**************************************************************************************************************
#include <stdio.h>         // For printf()
#include "i2c.h"           // For enumerated type i2c_mod
#include "oct_nunchuk.h"
#include "pit.h"

//==============================================================================================================
// Private Preprocessor Macros
//==============================================================================================================
#define forever while (1)

//==============================================================================================================
// Static Function Definitions
//==============================================================================================================
static void hw_init();
static void console_update();
static void sw_init();

//==============================================================================================================
// Private Global Variables
//==============================================================================================================
static volatile int g_console_update;

//==============================================================================================================
// Function Definitions
//==============================================================================================================

//--------------------------------------------------------------------------------------------------------------
// FUNCTION: g_console_update()
//
// DESCRIPTION
// Called by PIT 0 ISR at 1 Hz to update the console with the Wii Nunchuk state information.
//--------------------------------------------------------------------------------------------------------------
static void console_update()
{
    g_console_update = 1;
}

//--------------------------------------------------------------------------------------------------------------
// FUNCTION: hw_init()
//
// DESCRIPTION
// Initializes the MCF52259 hardware peripherals.
//--------------------------------------------------------------------------------------------------------------
static void hw_init()
{
    int_inhibit_all();
    oct_nunchuk_init(i2c_mod_1);
    pit_init(pit_timer_0, pit_freq_1_hz, console_update);
    int_uninhibit_all();
}

//--------------------------------------------------------------------------------------------------------------
// FUNCTION: main()
//--------------------------------------------------------------------------------------------------------------
int main()
{
    hw_init();
    sw_init();
    forever {
        printf("%d %d %d %d %d %d %d\n",
            oct_nunchuk_pos_x(),
            oct_nunchuk_pos_y(),
            oct_nunchuk_accel_x(),
            oct_nunchuk_accel_y(),
            oct_nunchuk_accel_z(),
            oct_nunchuk_button_c(),
            oct_nunchuk_button_z()
        );
        g_console_update = 0;
    }
}
```

```
//------------------------------------------------------------------------------------------------------------------------------------
// FUNCTION: sw_init()
//
// DESCRIPTION
// Software initialization.
//------------------------------------------------------------------------------------------------------------------------------------
static void sw_init()
{
    g_console_update = 1;
}
```

3.3.6 oct_nunchuk module — *oct_nunchuk.c, oct_nunchuk.h*

This module contains functions to interface to the Wii Nunchuk via Octopus Project Board 3. Suggested functionality,

1. Implements *oct_nunchuk_accel_x*() to return the x-axis acceleration value.
2. Implements *oct_nunchuk_accel_y*() to return the y-axis acceleration value.
3. Implements *oct_nunchuk_accel_z*() to return the z-axis acceleration value.
4. Implements *oct_nunchuk_button_c*() to return the state of button C.
5. Implements *oct_nunchuk_button_z*() to return the state of button Z.
6. Implements *oct_nunchuk_init*() to initialize the Octopus Nunchuk module and the I2C module for communication with the Wii Nunchuk. Configures PIT 1 to generate periodic interrupts at 4 Hz (every 250 ms).
7. Implements *oct_nunchuk_on_button_c*() which will save a callback function pointer for the user's function which is to be called when Nunchuk button C is pressed.
8. Implements *oct_nunchuk_on_button_z*() which will save a callback function pointer for the user's function which is to be called when Nunchuk button Z is pressed.
9. Implements *oct_nunchuk_on_stick_down*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the down position.
10. Implements *oct_nunchuk_on_stick_left*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the left position.
11. Implements *oct_nunchuk_on_stick_right*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the right position.
12. Implements *oct_nunchuk_on_stick_up*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the up position.
13. Implements *oct_nunchuk_pos_x*() to return the joystick x-axis (left/right) position.
14. Implements *oct_nunchuk_pos_y*() to return the joystick y-axis (up/down) position.
15. Implements *oct_nunchuk_read*() which is called every 250 ms by the PIT 1 interrupt service routine. This function reads the current state of the Nunchuk, stores the state information in global variables, and calls the user's callback functions as appropriate.
16. Implements *oct_nunchuk_reset*() which resets all of the callback function pointers to null.
17. Implements *oct_nunchuk_tx_cmd*() which transmits a command to the Nunchuk.

Remarks

I have given you the pseudocode for this module in the lecture notes.

3.3.7 pit module — *pit.c, pit.h*

You should be able to reuse your code from previous projects and augment or modify as necessary. Use PIT 1 to generate a periodic interrupt at 4 Hz to read the state of the Wii Nunchuk buttons and joystick. Use PIT 0 to generate a periodic interrupt at 1 Hz to update the console with the Nunchuk state information.

3.2 Build Notes

Since you will be using the *printf*() function from the C Standard Library, you must enable the library by navigating to *Your Project* > Properties > C/C++ Build > Settings > Librarian and checking Enable Automatic Library Configurations and then to *Your Project* > Properties > C/C++ Build > Settings > ColdFire Linker > General and unchecking No Standard Library.

Assembly language include files—those with a .inc file name extension—are placed in the Project_Headers folder of your project. However, the assembler needs to be instructed to look in that folder for the include files. Navigate to *Your Project* > Properties > C/C++ Build > Settings > ColdFire Assembler > Input. In the User Path area you will see an icon with a green + sign. Click on that icon to open the Add Directory Path dialog. In the dialog, click on Workspace. In the Folder Selection dialog, expand your project folder and select Project_Headers. Click OK, OK, and OK to close the dialogs.

4 What to Submit for Grading and the Assignment Deadline

For each source code file, put a header comment block at the top of the source code file that contains: (1) the name of the source code file; (2) the lab project number; (3) your name (and your partner's name); (4) your email address (and your partner's email address); (5) the course number and name, CSE325 Embedded Microprocessor Systems; and (6) the semester, Fall 2013.

Make arrangements with the TA to demo your working program in the BYENG 217 lab for grading. After the demo, copy your .h header and .c source code files to a directory named *proj06*. Zip the *proj06* directory and upload the zip archive to Blackboard using the project submission link by the deadline which is 4:00am Mon 9 Dec 2013. Consult the online syllabus for the late and academic integrity policies.