CS97 Lab 5: i2c Communication

Due by 11:59pm Thursday, Nov 8th, 2012

This lab assignment is designed to get you familiar with interfacing digital sensors which use i2c communcation. In particular we will be developing a driver for the InvenSense MPU-6050 6-axis gyro and accelerometer sensor. First, run **update97**. This will create the **cs97/labs/05** directory. In this directory is a tar file **lab5.tar.gz**. Untar the lab file.

```
$ update97
$ cd cs97/labs/05
$ tar xvf lab5.tar.gz
```

The program **handin97** will only submit files in this labs directory, so make sure your programs are in this directory! Be sure to run a 'make clean' before you submit, so you are submitting only your source code and not any binary or object files.

Wiring the Sensor

For this lab we'll be using the Tmote Sky. The motes have 5 colored wires already soldered to them. Connect them to the mpu-6050 as follows:

```
Red    to VDD
Black  to GND
Yellow to SCL
Green  to SDA
Blue   to VIO
```

i2c Communication

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions.

To write the internal MPU-6050 registers, the master transmits the start condition (S), followed by the i2c address and the write bit (0). At the 9th clock cycle (when the clock is high), the MPU-60X0 acknowledges the transfer. Then the master puts the register address (RA) on the bus. After the MPU-6050 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). To write multiple bytes after the last ACK signal, the master can continue outputting data rather than transmitting a stop signal. In this case, the MPU-6050 automatically increments the register address and loads the data to the appropriate register. The

following figures show single and two-byte write sequences.

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

To read the internal MPU-6050 registers, the master sends a start condition, followed by the i2c address and a write bit, and then the register address that is going to be read. Upon receiving the ACK signal from the MPU-6050, the master transmits a start signal followed by the slave address and read bit. As a result, the MPU-6050 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9th clock cycle. The following figures show single and two-byte read sequences.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Running the Driver Code

We have provided the beginning of a simple mpu-6050 driver. To run the code go to the **contiki-2-6/apps/mpu-6050** directory. The makefile provided has build and installation targets.

```
cd contiki-2-6/apps/mpu-6050
make
make mpu-6050.upload
```

The first make command builds the binary and the second make command uploads the binary to the mote. You can ignore any "warning: internal error: unsupported relocation error" messages you get while compiling the program.

To see any output from your program you can login into the mote.

```
make login
```

You communicate with i2c devices by reading and writing registers on the device. The address of the registers are provided by the device datasheet. The datasheet for the mpu-6050 can be found **here**.

The sample code we have given you provides one driver function **i2c_read_byte()** which will read one byte from the specified device and register. The i2c address of our device is **0x68**. This is specified in **mpu-6050.h**. As you develop your driver, use **#define** for all addresses and put them in this header file. The program simply reads the contents of the **WHO_AM_I** register. This register contains the device id, hich is printed to the output terminal.

Completing the Driver Code

The goal of this assignment is to write a simple driver that reads and displays information from the accelerometer and gyro. To complete the driver, you will need the following functions:

```
void i2c_write_byte(uint8_t slave_addr, uint8_t register_addr, uint8_t data)
void i2c_read_bytes((uint8_t slave_addr, uint8_t register_addr, uint8_t length, uint8_t *data)
```

The first function writes the byte specified in **data** to device and register specified. The second function reads multiple bytes specified by **length** and places them in the buffer **data** (which must be large enough to hold all the data. Use the above diagrams and the code in **i2c_read_byte()** as a guide to writing those functions. Once you have those functions working, you can use them to start getting data from the sensor. The first step is turning on the sensor, as by default, the sensor is in the off state. As documented on page 41 of the datasheet, register 0x6b controls the sleep state of the device. To turn the sensor on, you need to set Bit6 to 0 (note, the LSB of the byte is Bit0 and the MSB of the byte is Bit7). To do this, first read the register to a variable of type uint8_t, change bit 6 of the variable to a 1, and then write that variable back out to the register. Now the sensor is on, you can start reading from it. The register that holds the accelerometer data is 0x3b, which is shown on page 30 of the datasheet. Here you want to use the i2c_read_bytes() function you wrote to read the 6 bytes of accelerometer data. To get accelerometer and gyro readings, simply read 14 bytes starting same register. Why do we need to read 14 bytes? Because between the accelerometer and gyro registers are two bytes, that represents the temperature. To complete this assignment, write the above functions and the code to print the accelerometer, temperature, and gyro values at the rate of 10 Hz to the terminal.

Handing in Your Assignment

Once you are satisfied with your programs, hand them in by typing **handin97** at the Linux prompt.
Important: Before handing in your assignment, run **make clean** in each of the program directories. This will remove any object code and tempory files before you submit.

You may run **handin97** as many times as you like, and only the most recent submission will be recorded. This is useful if you realize, after handing in some programs, that you'd like to make a few more changes to them.

Home Page

Last updated: Sunday, November 04, 2012 at 09:46:06 PM