ECE353 In-Class Exercise

I2C Bus

Problem Objectives

- Determine the I2C address of external devices
- Determine how to send data transactions to different devices.

Overview

The ECE353 daughter card has a MCP23017 IO expander connected to the I2C1 peripheral interface. In order to properly communicate with the device, you will need to determine what its I2C bus address from the ECE353 carrier card schematics.

You have been provided with an I2C library that allows you to send and receive data. Your task is to complete driver for the IO expander (MCP23017). You will provide the correct bit masks for the Master/Slave Control register to initiate data transfers for both devices.

Driver Information

Setting the Slave Address

In order to send data packets to the correct device, we must set the Master Slave Address register to the I2C address of the external device. Once the MSA register is set, all data transactions will be start with a command word that includes the address of the specified slave device.

The following function has been provided to set the slave address in the MSA register.

```
i2c_status_t i2cSetSlaveAddr(
  uint32_t baseAddr,
  uint8_t slaveAddr,
  i2c_read_write_t readWrite
)
```

The following examples show how to initiate a write control word.

```
status = i2cSetSlaveAddr(i2c_base, MCP23017_DEV_ID, I2C_WRITE);
```

Transferring Data

Data is transferred by setting the RUN bit in the Master Control/Status Register (MCS). The MCS also contains status flags to determine if the I2C address and data bytes were acknowledged by the slave device. The final action the MCS takes is to determine when to send a Start Condition, control word (I2C Slave Address + R/$\overline{W}$ Bit), and Stop Condition.

Writing Data

Sending the Start Condition, Slave Address, and 1st data byte

The first byte of data transferred must be written to the Master Data Register (MDR). In order to start an I2C transaction, a Start Conditon must be generated, followed by the control word containing the I2C address and R/$\overline{W}$ bit set to 0, followed by the byte of data written to the MDR.

The following function call would begin a write operation where the 1st byte transmitted to the slave device is 0x00.

```
i2cSendByte(
                i2c_base,
                0x00 ,
                I2C_MCS_START | I2C_MCS_RUN
            );
```

Sending the Last Byte of Data

In order to terminate a transaction, a Stop condition must follow the last byte of data being transferred. This can be accomplished using the following function call.

```
 i2cSendByte(
                i2cBase,
                last_byte,
                I2C_MCS_RUN | I2C_MCS_STOP
              );
```

Sending a Byte that is NOT the first or last Byte

If the byte of data that is being transferred is not the first or last byte of the data transfer, the following function call would be used.

```
i2cSendByte(
                i2cBase,
                last_byte,
                I2C_MCS_RUN
            );
```

1. Determine I2C Addresses

Set the value of **`MCP23017_DEV_ID`** in `io_expander.c` the correct value. Examine the schematic to determine the values of A2-A0 for the MCP23017.

2. Modify `io_expander.c`

Writing to a register in the MCP23017 IO Expander requires that we send a **control byte, one address byte, and the data written**.

a. Determine the addresses of registers `IODIRA, IODIRB, GPIOA, and GPIOB` of the MCP23017 by examining page 5 of the data sheet. #defines for these registers can be found at the top of `io_expander.c`
b. For all the instances of `i2cSetSlaveAddr` and `i2cSendByte`, replace FIXME with the correct data and MCS bit masks to initiate the data transfer. See page 2 above for examples of the correct combination of bit masks to send for the MCS value.

3. Observe Led Matrix

If you have implemented the data transfers to the IO expander correctly, you should see the right most column o LEDs light up.

4. What to Turn In

Turn in **`io_expander.c`** to the dropbox on the course website.