ECE353 In-Class Exercise

I2C Bus - EEPROM

Problem Objectives

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

Overview

The ECE353 daughter card has two I2C peripheral devices connected to the I2C1 peripheral interface. In order to properly communicate with each device, you will need to determine what address will be used to communicate with each device. The each peripheral device expects a different data packet format, so you will examine how to send a properly formatted packet to each device.

Your task is to complete the driver for the EEPROM (MCP24LC32AT).

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 and read control word.

```
status = i2cSetSlaveAddr(i2c_base, MCP24LC32AT_DEV_ID, I2C_WRITE);
status = i2cSetSlaveAddr(i2c_base, MCP24LC32AT_DEV_ID, I2C_READ);
```

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
            );
```

Reading Data

Reading a byte of data over an I2C bus requires that we first write to the device. Each I2C device has an internal register that contains the address of the register/memory location being accessed. In order to set this register, we must write the address value over the I2C device.

After the address value has been set, the I2C master must initiate a second start, or re-start condition. The Restart condition will again send a control word that consists of the I2C address and the R/$\overline{W}$ bit set to 1. This implies that a read transaction will have two start conditions, but only a single Stop condition.

```
// Start condtion, I2C Address, Write Operation
status = i2cSetSlaveAddr(i2c_base, MCP24LC32AT_DEV_ID, I2C_WRITE);

// Send Address Byte(s)
.
.
.

// Start Conditon, I2C Address, Read Operation
status = i2cSetSlaveAddr(i2c_base, MCP24LC32AT_DEV_ID, I2C_READ);

// Read Byte(s)
.
.

// Send Stop Condition
```

1. Modify boardUtil.h

Fill in the values for the EEPROM macros at the end of the file.

2. Modify eeprom.c

a. Set the value of **MCP24LC32AT_DEV_ID** in `eeprom.c` to the correct values. For the 24LC23AT, there are no external pins for A2-A0, so you can set these bits to 0. You will need to examine the 24LC23AT data sheet to determine the correct I2C device ID.

b. Complete `eeprom_byte_write()`. Writing to a register in the 24LC32AT EEPROM requires that we send a **control byte, two address bytes, and the data written**.

c. Complete `eeprom_byte_read()`. Reading from a register in the 24LC32AT EEPROM requires that we send a **control byte, two address bytes, a second control byte, and then read the data**.

3. Observe the Serial Output

Observe the serial debug terminal to see if the EEPROM test has passed or failed.

4. What to Turn In

Turn in **eeprom.c** to the dropbox on the course website.