

Laboratory Worksheet #6

Code for Data Transfer on I²C Bus

This worksheet will help you start writing the code for lab 3, part 2. You have to include this worksheet in your lab notebook.

Refer to the variable declarations. The SMB0CN special function register is bit addressable. Write down the value for each variable that will allow you to access each individual bit independently.

```
/*Individual bits of the SMB0CN 0xC0 register */
sbit BUSY   = _____; /*SMBUS0 BUSY*/
sbit ENSMB  = _____; /*SMBUS0 ENABLE*/
sbit STA    = _____; /*SMBUS0 START FLAG*/
sbit STO    = _____; /*SMBUS0 STOP FLAG*/
sbit SI     = _____; /*SMBUS0 SIERRUPT PENDING FLAG*/
sbit AA     = _____; /*SMBUS0 ASSERT/ACKNOWLEDGE FLAG*/
sbit SMBFTE = _____; /*SMBUS0 FREE TIMER ENABLE*/
sbit SMBTOE = _____; /*SMBUS0 TIMEOUT ENABLE*/
(Note: The lines above are to show help you keep track of the sbits
used with the I2C bus. They are already defined in C0851F020.h and
shouldn't be part of you code.)
```

The following functions perform simple operations on the I²C bus. Make sure you understand how each one works. They will be used together to send and receive data over the bus.

```
//-----
// Routine:    i2c_read
// Inputs:     none
// Outputs:    input byte
// Purpose:    Reads data from the I2C bus
//-----
unsigned char i2c_read (void)
{
    unsigned char input_data;

    while (!SI);           // Wait until we have data to read
    input_data = SMB0DAT;   // Read the data
    SI = 0;                // Clear SI
    return input_data;     // Return the read data
}
```

Embedded Control Laboratory Worksheet

```
//-----  
// Routine:    i2c_read_and_stop  
// Inputs:     none  
// Outputs:    input byte  
// Purpose:    Sends I2C Stop Transfer  
//-----  
unsigned char i2c_stop_and_read (void)  
{  
    unsigned char input_data;  
  
    while (!SI);           // Wait until we have data to read  
    input_data = SMB0DAT;  // Read the data  
    SI = 0;                // Clear SI  
    STO = TRUE;           // Perform I2C stop  
    while (!SI);         // wait for stop  
    SI = 0;  
    return input_data;    //Return the read data  
}  
  
//-----  
// Routine:    i2c_start  
// Inputs:     none  
// Outputs:    none  
// Purpose:    Sends I2C Start Transfer  
//-----  
void i2c_start (void)  
{  
    while (BUSY);         // Wait until the SMBus0 is free  
    STA = TRUE;           // Set Start bit  
    while (!SI);         // Wait until start sent (look at SI)  
    STA = FALSE;         // Clear Start bit  
    SI = 0;               // Clear SI  
}  
  
//-----  
// Routine:    i2c_write  
// Inputs:     output byte  
// Outputs:    none  
// Purpose:    Writes data over the I2C bus  
//-----  
void i2c_write (unsigned char output_data)  
{  
    SMB0DAT = output_data; // Store data in SMB0DAT register  
    while (!SI);         // Wait until we are done with send  
    SI = 0;               // Clear SI  
}
```

Embedded Control Laboratory Worksheet

```
//-----  
// Routine:    i2c_stop_and_write  
// Inputs:    output byte  
// Outputs:    none  
// Purpose:    Sends I2C Stop Transfer  
//-----  
void i2c_write_and_stop (unsigned char output_data)  
{  
    SMB0DAT = output_data; // Store data in SMB0DAT register  
    STO = TRUE;           // Set Stop bit  
    while (!SI);          // Wait until we are done with send  
    SI = 0;                // Clear SI  
}
```

Now, write the following functions using the given pseudocode. They will call the functions seen above. Make sure you understand how each of the above functions work within these two functions. When writing these functions, use the same prototypes (i.e. function name, parameter order, etc.) as seen here.

Note: When writing these functions, be sure to process the data in the buffer in the same order it is processed on the bus. For example, in the first function, be sure to write `buffer[0]` first, then `buffer[1]`, up to `buffer[num_bytes - 1]`. In the second, be sure to save the first byte read to `buffer[0]`, the second to `buffer[1]`, etc.

```
// Write data to the I2C bus  
  
// Parameters:  
// unsigned char addr - address of the device that will be written to  
// unsigned char start_reg - first register that will be written  
// unsigned char* buffer - array of data to be written  
// unsigned char num_bytes - number of elements in the array  
  
void i2c_write_data(unsigned char addr, unsigned char start_reg,  
unsigned char *buffer, unsigned char num_bytes)  
{  
    Start I2C transfer  
  
    Write address of device that will be written to  
  
    Write the start register  
  
    *** You will need to use a loop for the following line ***  
    Write all bytes in buffer except for the last one  
  
    Write the last byte and stop  
}
```

Embedded Control Laboratory Worksheet

```
// Read data from the I2C bus

// Parameters:
//  unsigned char addr - the address of the device to read from
//  unsigned char start_reg - the first register to read from
//  unsigned char* buffer - array where the read data will be stored
//  unsigned char num_bytes - number of bytes to be read from device

void i2c_read_data(unsigned char addr, unsigned char start_reg,
unsigned char *buffer, unsigned char num_bytes)
{
    Start I2C transfer

    Write the address of the device to read from

    Stop and write the first register to be read

    Start I2C transfer

    Write address again, this time indicating a read operation

    For (0 to num_bytes - 2)
        Set acknowledge bit

        Read data, save it in the buffer

    Clear acknowledge bit

    Read the last byte and stop, save it in the buffer
}
```

When writing the for-loops for these two functions, remember that you do not want to read/write `buffer[num_bytes - 1]` inside the loop (This is the last byte in the buffer). The final byte will be written/read in the `write_and_stop()` or `read_and_stop()` call at the end of the functions.

You will be given code to test your functions in the next lab.