Memo

To:	ECE 4175 Students
CC:	John Peatman, ECE 4175 TAs
From:	Chris Twigg
Subject:	I2C subroutines for Hi-Tech's PICC18 compiler
Sent:	November 10, 2002
Attachments:	i2c.c, i2c.h, ch17.c, ch17.h

The attached I2C files are functionally equivalent to assembly files discussed in class. In addition to the StartI2C, ReStartI2C, and StopI2C low level routines, two high level routines, SendI2C and GetI2C, have been included and are documented below:

```
char SendI2C(char I2CADD, char I2CBYTES, char *I2CBUFFER)
```

This subroutine will send data to an I2C slave device. I2CADD should contain the address of the slave device left shifted by one bit. I2CBYTES is the number of bytes to send to the slave device. I2CBUFFER is a pointer to a string containing the data to send.

Single byte example:

```
   char cTEMP = 0xEE;              // Character containing command to send

   SendI2C(0x90, 1, &cTEMP);       // Send 1 byte to slave device
```

Multi-byte example:

```
   char strTEMP[] = {0xAC, 0x0A};  // String containing commands to send

   SendI2C(0x90, 2, strTEMP);      // Send 2 bytes to slave device at
                                   //  address 0x48 (left shifted by 1,
                                   //  0x90) from strTEMP
```

In the single byte example above, notice that the variable cTEMP is preceded by an ampersand, "&", character. For the SendI2C command, we need to send a pointer to the string containing the commands to send; the ampersand does this by sending the address of cTEMP instead of its contents. Note that you do not need to use the ampersand to work with the variable otherwise (Example: cTEMP = 0x48;).

In the multi-byte example, we do not need the ampersand before strTEMP. This is due to the way C handles strings, arrays of characters. Since a string can contain any number of characters, C uses a pointer to reference the beginning of a string. In this example, strTEMP is already a pointer, so we can pass it directly to SendI2C.

```
char GetI2C(char I2CADD, char I2CCOMMAND, char I2CBYTES, char *I2CBUFFER)
```

This subroutine will receive data from an I2C slave device. I2CADD should contain the address of the slave device left shifted by one bit. I2CCOMMAND tells GetI2C whether this is a commanded get or a normal get. A commanded get is one that requires a control or internal address byte to be sent to the slave device before reading can commence. I2CCOMMAND should be 0 for a normal get and >0 for a commanded get. I2CBYTES is the number of bytes to receive, and I2CBUFFER is a pointer to a string where the bytes read will be stored. For a commanded get, the first byte in I2CBUFFER should contain the command to send.

Single byte, normal get example:

```
   char cTEMP;                  // Character to store

   GetI2C(0x70, 0, 1, &cTEMP);  // Get one byte from slave device 0x38
```

Multi-byte, commanded get example:

```
   char strTEMP[] = {0xAA, 0x00};  // String containing command

   GetI2C(0x70, 1, 2, strTEMP);    // Send command, 0xAA from strTEMP, to
                                   //  device at address 0x38, and read 2
                                   //  bytes back into strTEMP
```

Again, in the single byte example, we see the ampersand before the single byte character cTEMP. The reasoning for this is the same as for SendI2C. In the multi-byte example, there is no ampersand before strTEMP because it is already a pointer, just like in SendI2C.

For debugging purposes, you may also want to check if the subroutine has successfully transmitted any data. This can be accomplished with a simple if statement:

```
   if (!SendI2C(0x90, 1, strTEMP))
     DeviceError();
```

Notice the exclamation point before SendI2C. SendI2C and GetI2C will return a 1 if successful and a 0 if failed. In this case, we want to trap a failed transmission and call an error handler, DeviceError. You will need to write your own handler, but an example is provided in CH17.C.