



APPLICATION NOTE 3588

Software I²C Driver for the MAXQ2000 Microcontroller

Abstract: I²C (inter-integrated circuit) is a 2-wire interface that allows bidirectional communication between integrated circuits. This application note describes a software I²C driver for the MAXQ2000 microcontroller that permits I²C communication at 100kHz or 400kHz using any of the microcontroller's GPIO pins. Microcontrollers from the MAXQ family are well suited for such bit-banging applications because of their high-speed, flexible GPIO modules, and separate I/O supply voltage.

Introduction

An I²C (inter-integrated circuit) is a 2-wire interface that allows bidirectional communication between integrated circuits. This appnote describes the *maxqi2c* library, a software I²C driver for the [MAXQ2000 microcontroller](#) (μ C).

The *maxqi2c* library is written in C using extensions that compile using the [IAR Embedded Workbench](#) for MAXQ. It consists of two files: **maxqi2c.h** and **maxqi2c.c**. When these files are included in a MAXQ2000 firmware project, they permit flexible I²C communication at 100kHz or 400kHz using any of the μ C's GPIO pins.

Microcontrollers from the MAXQ family are well-suited for such bit-banging applications because of their high-speed, flexible GPIO modules and separate I/O supply voltage.

The files for the example project discussed in this appnote are available for [download](#) from Maxim Integrated Products.

Configuring the *maxqi2c* Library

The user should copy the *maxqi2c* library files (**maxqi2c.h** and **maxqi2c.c**) to the MAXQ2000 project directory and configure the files to create the desired I²C interface. All configurations are done by editing the following code (**Listing 1**), which can be found at the top of the **maxqi2c.h** source file:

Listing 1. maxqi2c.h customization code.

```
/* USER MUST CUSTOMIZE THE FOLLOWING DEFINE STMTS - START */
// Enter the port used for SDA and SCL
#define SDA_PORT          0
#define SCL_PORT          0

// Enter the pin used for SDA and SCL
#define SDA_PORT_BIT      0
#define SCL_PORT_BIT      1

// Uncomment one of these define statements to select I2C bus speed
#define I2C_400_KHZ
//#define I2C_100_KHZ

// Comment out the following define statement to disable clock
// stretching in i2cRecv()
```

```
#define I2C_CLOCK_STRETCHING
/* USER MUST CUSTOMIZE THE FOLLOWING DEFINE STMTS - END */
```

Note: the customizations are implemented at compile time and, therefore, are fixed at runtime.

Choosing the SCL and SDA Pins

Two GPIO pins must be selected for use as SCL and SDA. After the I/Os for SCL and SDA are chosen, the **SDA_PORT** and **SCL_PORT** define statements must be edited to reflect the desired ports for SDA and SCL. The **SDA_PORT_BIT** and **SCL_PORT_BIT** define statements must also be edited to reflect the desired pins (on the selected ports) for SDA and SCL.

The source code in Listing 1 above assigns pin 0 on I/O port 0 to act as SDA, and pin 1 on I/O port 0 to act as SCL.

Choosing the Communication Speed

Selecting a communication speed is done by commenting out one of the two statements that define **I2C_400_KHZ** and **I2C_100_KHZ**.

The source code in Listing 1 uses a 400kHz I²C bus to initialize the *maxqi2c* library into communication. The communication is actually slightly less than 400kHz (or, alternatively, the 100kHz) because the I²C interface is bit-banged. To achieve full 400kHz communication, firmware designers must study the *maxi2c* library and remove some of the source code that provides the library's inherent flexibility.

Note: delays are included in the *maxqi2c* library to satisfy the I²C specification. These delays, found at the top of the **maxqi2c.c** file, assume that the MAXQ2000 has a 20MHz system clock; the delays can be shortened if a slower clock speed is used.

Using Clock Stretching

Clock stretching for the *maxqi2c* library is only permitted at the beginning of a transfer (after the address acknowledgement if the address is being transmitted, or otherwise at the start of the transfer) during a call to the **i2cRecv()** function. Clock stretching can, therefore, be used for I²C transfers with the following format:

```
[S] [ADDR] [R] [A] [clock stretch] [DATA0] [A] ... [DATAN-1] [A]
or
[clock stretch] [DATA0] [A] ... [DATAN-1] [N] [P]
or
[clock stretch] [DATA0] [A] ... [DATAN-1] [A]
```

The description, **i2cRecv()**, in the section under **Using the *maxqi2c*** and the example code in the section titled, **Example Use of the *maxqi2c* Library**, explain how to generate I²C commands with these formats.

To enable clock stretching, the **I2C_CLOCK_STRETCHING** define statement should not be commented out. If clock stretching is not required, disable it by commenting out the **I2C_CLOCK_STRETCHING** define statement. Disabling clock stretching will slightly increase the speed of the *maxqi2c* library's **i2cRecv()** function.

The source code in Listing 1 above enables clock stretching.

Using the *maxqi2c*

Using the *maxqi2c* library to send and receive data from a software I²C driver is done with four functions: **i2cInit()**, **i2cIsAddrPresent()**, **i2cSend()**, and **i2cRecv()**. The documentation for these functions can also be found

in the **maxqi2c.h** file.

None of these functions requires formal parameters. Instead, four global variables are used to store the parameters for these functions: **i2cData** (unsigned char *), **i2cDataLen** (unsigned int), **i2cDataAddr** (unsigned char), and **i2cDataTerm** (unsigned char). This technique allows the firmware to run faster by not making copies of data during function calls. The four global variables used as parameters for the *maxqi2c* library are: **i2cData** (unsigned char *), **i2cDataLen** (unsigned int), **i2cDataAddr** (unsigned char), and **i2cDataTerm** (unsigned char).

i2cInit()

This function must be called before any of the other *maxqi2c* functions. It initializes the port pins selected in the customization code of the **maxqi2c.h** file. This function requires no parameters (local or global) and does not return a value.

i2cIsAddrPresent()

This function allows the MAXQ2000 to query the I²C bus to determine if a device with a particular address is present. The function has one parameter, the global variable **i2cDataAddr**, which must be loaded with the address of the device whose presence is being queried on the I²C bus. This function also returns a value (of type unsigned char). That value is equal to **I2C_XMIT_OK** if a device with the given address was found, or **I2C_XMIT_FAILED** if a device with the given address was not found.

To determine if a particular device is present on the I²C bus, **i2cIsAddrPresent()** transfers an I²C command with the following format:

[S] [ADDR] [W] [A] [P]

i2cSend()

This function allows the MAXQ2000 to transfer data to a device using the software I²C driver. **i2cSend()** requires the following four parameters (all global variables) to be initialized:

- **i2cData** (unsigned char *): Pointer to the first byte of an array of bytes that will be transmitted.
- **i2cDataLen** (unsigned int): Number of bytes (not including the device address) to transfer to the I²C bus.
- **i2cDataAddr** (unsigned char): The address of the device to which the data will be transferred. Note that if this variable is set to zero, the I²C data will be transferred without sending the address.
- **i2cDataTerm** (unsigned char): How the I²C transfer will be terminated. This variable can take two values when calling **i2cSend()**: **I2C_TERM_NONE** or **I2C_TERM_STOP**.

The format used to transfer data to a device on the I²C bus depends on the value of the four global variables used as parameters. **Table 1** presents the I²C command formats resulting from different values for these global variables.

Table 1. I²C Commands Sent by i2cSend()

i2cDataLen(hex)	i2cDataAddr(hex)	i2cDataTerm	I ² C Command Format
0x0002	0x7E	I2C_TERM_STOP	[S] [ADDR] [W] [A] [DATA0] [A] [DATA1] [P]
0x0002	0x7E	I2C_TERM_NONE	[S] [ADDR] [W] [A] [DATA0] [A] [DATA1] [A]
0x0002	0x00	I2C_TERM_NONE	[DATA0] [A] [DATA1] [A]
0x0002	0x00	I2C_TERM_STOP	[DATA0] [A] [DATA1] [A] [P]

Note: the last three formats of Table 1 show how **i2cSend()** can transfer data continuously to the same device

on the I²C bus.

The `i2cSend()` function returns a value (of type unsigned char) equal to **I2C_XMIT_OK** if the addressed device acknowledged every byte, or **I2C_XMIT_FAILED** if the addressed device did not acknowledge a byte. The function will return immediately when a single byte is not acknowledged.

i2cRecv()

This function allows the MAXQ2000 to receive data from a device using the software I²C driver. The **i2cRecv()** function requires the following four parameters (all global variables) to be initialized:

- **i2cData** (unsigned char *): Pointer to the first byte of an array of bytes where the received values will be stored.
- **i2cDataLen** (unsigned int): Number of bytes (not including the device address) that will be received from the I²C bus.
- **i2cDataAddr** (unsigned char): The address of the device from which the data will be received. Note that if this variable is set to zero, the I²C data will be received without sending the address.
- **i2cDataTerm** (unsigned char): How the I²C transfer will be terminated. This variable can take three values when calling **i2cRecv()**: **I2C_TERM_NONE**, **I2C_TERM_ACK**, or **I2C_TERM_NACK_AND_STOP**.

The format used to receive data from a device on the I²C bus depends on the value of the four global variables used as parameters. **Table 2** presents the I²C command formats resulting from different values for these global variables.

Table 2. I²C Commands Sent by i2cRecv() with Clock stretching Disabled

i2cDataLen(hex)	i2cDataAddr(hex)	i2cDataTerm	I ² C Command Format
0x0002	0x7E	I2C_TERM_NACK_AND_STOP	[S] [ADDR] [R] [A] [DATA0] [A] [DATA1] [N] [P]
0x0002	0x7E	I2C_TERM_ACK	[S] [ADDR] [R] [A] [DATA0] [A] [DATA1] [A]
0x0002	0x00	I2C_TERM_ACK	[DATA0] [A] [DATA1] [A]
0x0002	0x00	I2C_TERM_NACK_AND_STOP	[DATA0] [A] [DATA1] [N] [P]

Note: the last three formats of Table 2 show how **i2cRecv()** can receive data continuously from the same device on the I²C bus.

The **i2cRecv()** function returns a value (of type unsigned char) equal to **I2C_XMIT_FAILED** if the address was sent as part of the I²C command and was not acknowledged. Otherwise, **I2C_XMIT_OK** is returned.

Example Use of the *maxqi2c* Library with Clock Stretching

The following example shows how the *maxqi2c* library can be used to receive 16-bit samples from the MAX1169 ADC and to transfer them to a PC using the MAXQ's RS-232 port.

Schematic

This example was implemented using the [MAX1169 ADC evaluation kit](#) and the [MAXQ2000 evaluation kit](#) (Rev B). **Figure 1** shows how both evaluation kits are connected. The pin-0 and pin-1 of I/O port 0 on the MAXQ2000 (which are available on J2-30 and J2-28 respectively) serve as the master SDA and SCL lines on the I²C bus.

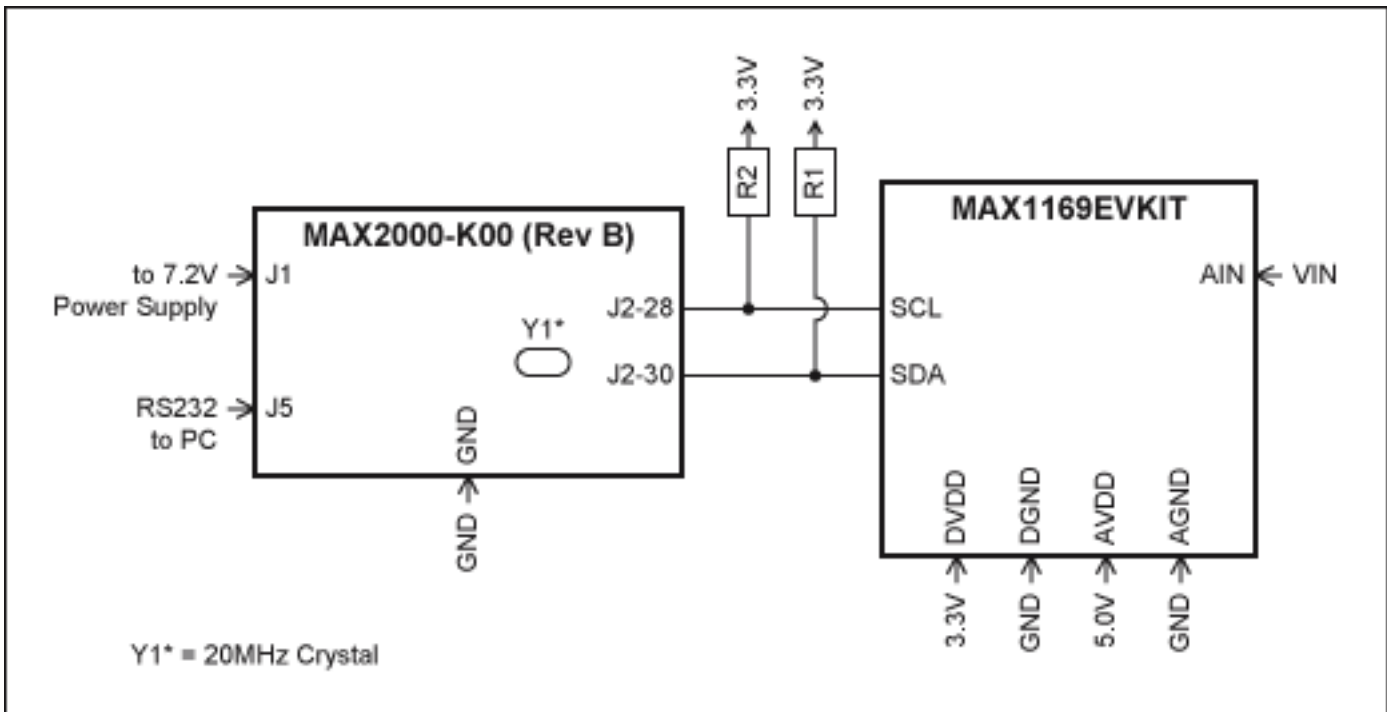


Figure 1. Schematic shows the MAX1169 evaluation kit and the MAXQ2000 evaluation kit (Rev B) connected and ready to be used by the maxqi2c library.

Note: the MAXQ2000 high-frequency crystal (Y1) on the MAXQ2000 evaluation kit has been replaced with a 20MHz crystal. The jumper settings for the MAX1169 evaluation kit and the switch settings for the MAXQ2000 evaluation kit must be set as indicated in **Table 3** and **Table 4**:

Table 3. Jumper Settings for the MAX1169 Evaluation Kit

Jumper	Shunt Position
JU1	Shunt on pins 1 and 2
JU2	Shunt on pins 1 and 2
JU3	Shunt on pins 1 and 2
JU4	No shunt
JU5	No shunt

Table 4. Switch Settings for the MAXQ2000 Evaluation Kit (Rev B)

Switch	Position
SW1-1	OFF
SW1-2	OFF
SW1-3	OFF
SW1-4	ON
SW1-5	OFF
SW1-6	OFF
SW1-7	ON
SW1-8	OFF
SW6-1	OFF
SW6-2	OFF
SW6-3	OFF
SW6-4	OFF
SW6-5	OFF
SW6-6	OFF
SW6-7	OFF
SW6-8	ON

Firmware

The firmware file for this example (**max1169.c**) is given in **Appendix A**. The complete project can be downloaded from the [Maxim MAXQ2000 webpage](#) and can be compiled using the IAR Embedded Workbench for the MAXQ. In the example here, the code customization for the *maxqi2c* library (found at the top of the **maxqi2c.h** file) is exactly the same as the source code shown in Listing 1.

The **max1169.c** file includes two header files, **iomaxq200x.h** and **maxqi2c.h**. Note that the **iomaxq200x.h** file in the example will override the **iomaxq200x.h** file in the IAR Embedded Workbench for MAXQ include path. The **iomaxq200x.h** file creates definitions for every pin on every port that are required for the *maxqi2c* library. The **maxqi2c.h** file is included to allow the firmware to call functions in the *maxqi2c* library.

The firmware is divided into five steps labeled with comments in the **max1169.c** file (see Appendix A).

Step 1 initializes UART0 to communicate asynchronously at 19200bps. Note that if the MAXQ2000 system clock is not 20MHz, the assignment to register PRO will have to be changed to obtain the desired baud rate.

Step 2 calls the **i2cInit()** function responsible for initializing the pins used on the MAXQ2000 for the I²C bus.

Step 3 initializes the parameters and calls the **i2cRecv()** function. The parameters are initialized to transfer an I²C command with the following format:

```
[S] [ADDR] [R] [A] [clock stretch] [DATA0] [A] [DATA1] [A (termination)]
```

Step 4 sets the address parameters to zero. This forces the **i2cRecv()** function to transfer an I²C command with the following format:

```
[clock stretch] [DATA0] [A] [DATA1] [A (termination)]
```

Step 5 is a loop that repeats indefinitely. The loop calls **i2cRecv()** (with the format defined in step 4), which receives a 16-bit sample from the MAX1169. This 16-bit sample is transferred (MSB first) to a PC using UART0.

Since the termination parameter, **i2cDataTerm**, is always equal to **I2C_TERM_ACK**, the loop repeats indefinitely and the MAX1169 never sees a stop condition.

APPENDIX A: max1169.c

```
/*
 * DEMO of maxqi2c Software I2C Driver
 * (uses evkits for the MAX1169 and MAXQ2000)
 *
 * by: Paul Holden - MAXIM INTEGRATED PRODUCTS
 *
 *
 * DESC: Test program for the maxqi2c.c/maxqi2c.h I2C
 * driver for the MAXQ2000. The program reads
 * 16-bit samples from the MAX1169 (running in
 * continuous conversion mode) and transmits them
 * using the UART0 port.
 *
 * NOTE - THE FOLLOWING CODE ASSUMES THE MAXQ2000 HAS
 * A Fsysclk=20MHz.
 */

#include "iomaxq200x.h"
#include "maxqi2c.h"

void main()
{
    unsigned char data[2];

    // 1. Init UART0
    PD7_bit.bit0 = 1; // Set TX0 pin as output
    SCON0 = 0x42;
    SMD0 = 0x02;
    PR0 = 0x07DD; // 19200bps

    // 2. Init bit-banged I2C port
    i2cInit();

    // 3. Send initial I2C request
    // [S] [ADDR+R] [A] [clock_stretch] [DATA0] [A] [DATA1] [A (termination)]
    i2cData = (unsigned char *)&data; // cast needed!
    i2cDataAddr = 0x7E;
    i2cDataLen = 0x0002;
    i2cDataTerm = I2C_TERM_ACK;
    i2cRecv();

    // 4. Init continuous conversion
    // [clock_stretch] [DATA0] [A] [DATA1] [A (termination)]
    i2cDataAddr = 0x00;

    // 5. Receive a 16-bit sample and transfer it to the UART0 port
    // one byte at a time. Repeat forever...
    while (1)
    {
        i2cRecv();

        while(!SCON0_bit.TI); // Wait for UART0 Buffer to be empty
        SCON0_bit.TI = 0; // Reset TI flag
        SBUF0 = data[0]; // Send data byte 0
        while(!SCON0_bit.TI); // Wait for UART0 Buffer to be empty
    }
}
```

```
        SCON0_bit.TI = 0;          // reset TI flag
        SBUF0         = data[1]; // Send data byte 1
    }
}
```

Application Note 3588: www.maxim-ic.com/an3588

More Information

For technical support: www.maxim-ic.com/support

For samples: www.maxim-ic.com/samples

Other questions and comments: www.maxim-ic.com/contact

Automatic Updates

Would you like to be automatically notified when new application notes are published in your areas of interest?
[Sign up for EE-Mail™.](#)

Related Parts

MAX1169: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

MAXQ2000: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN3588, AN 3588, APP3588, Appnote3588, Appnote 3588

Copyright © by Maxim Integrated Products

Additional legal notices: www.maxim-ic.com/legal