Keywords: MAXQ2000, IAR, C Compiler, Evaluation Kit, Embedded Workbench

APPLICATION NOTE 3378

Getting Started with the IAR Compiler and the MAXQ2000 Evaluation Kit

Jun 20, 2005

Abstract: The application note describes how to create, build, and debug applications written in C and targeted for the MAXQ® platform. It demonstrates some of the features of the MAXQ2000.

Introduction

The MAXQ platform is supported by a world-class set of tools, IAR's Embedded Workbench for the MAXQ. This application note describes how to create, build, and debug applications written in C and targeted for the MAXQ platform. It will also demonstrate some features of the MAXQ2000, the first microcontroller available from the MAXQ family.

This application note was written using version 1.12B of the IAR compiler for the MAXQ platform. These instructions should hold for later versions of the product as well. These instructions are written to be used with the MAXQ2000 Evaluation Kit.

Setting up the MAXQ2000 Evaluation Kit

Before we start writing code, let's connect the MAXQ2000 evaluation kit. The kit should come with 3 boards, one with a small LCD screen. The largest board (shown in **Figure 1** with the LCD daughterboard connected) is the actual MAXQ2000 evaluation kit. We will discuss the features of this board later in this document. Take the LCD board and connect to the header on the MAXQ2000 Evaluation Kit labeled J3.
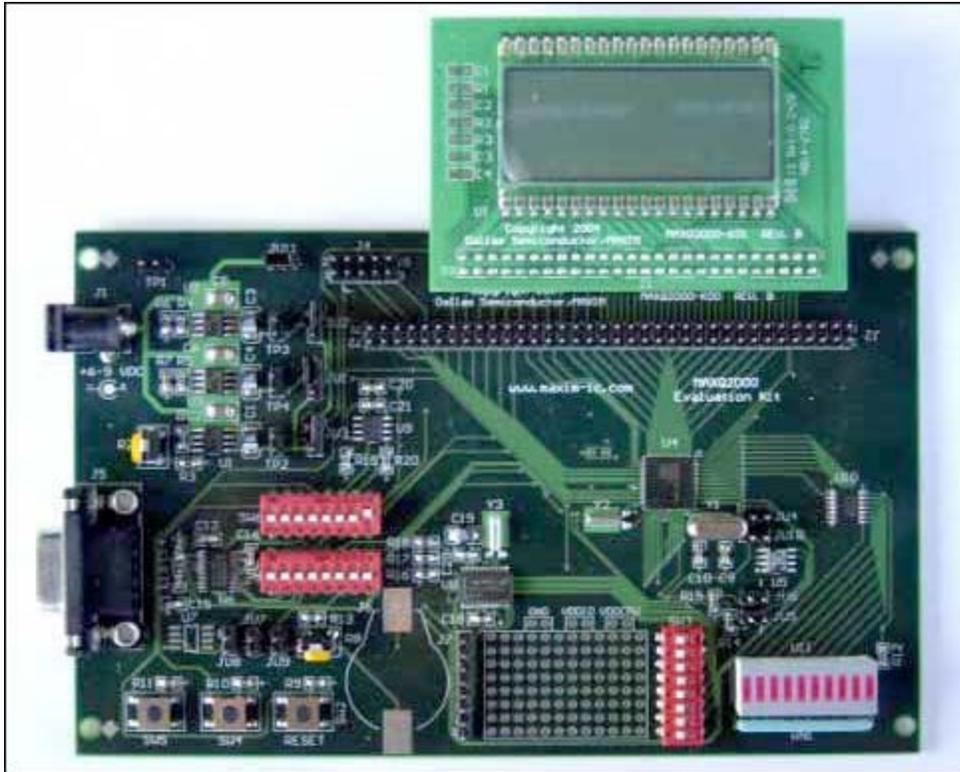
Figure 1. MAXQ2000 Evaluation Kit with LCD board attached.

The remaining board in the evaluation kit is the JTAG board. The MAXQ2000 loader and debug engine communicate using the JTAG protocol. Since virtually no general purpose, commercially available JTAG adapters exist for personal computers, Dallas Semiconductor provides a Serial-to-JTAG converter board. There should also be a small connector included with the evaluation kit. Use this connector to attach the MAXQ2000 Evaluation Kit and the JTAG board as shown in **Figure 2**. The cable connects the header labeled J4 on the MAXQ2000 Evaluation Kit to the header labeled P2 on the JTAG board. Note that the red side of the connector is on the side marked as pins "1" and "2" on both boards.
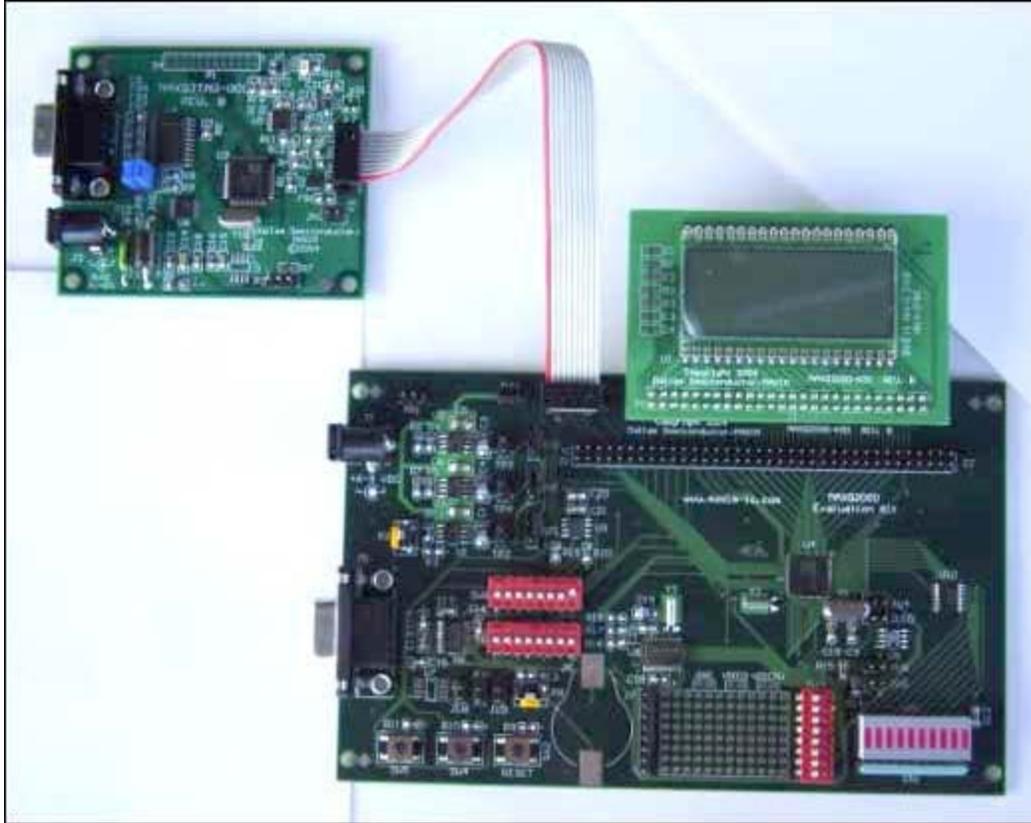
Figure 2. MAXQ2000 Evaluation Kit connected to JTAG board.

There should be 3 jumpers placed on the MAXQ2000 evaluation kit. Pins 1 and 2 on headers JU1, JU2, and JU3 should be jumpered together. In addition, make sure a crystal placed at Y1 (near the microprocessor on the board). The demonstration programs in this application note assume a 13.5Mhz crystal. The JTAG board should also have 3 jumpers placed. The jumpers should be placed on headers JH1, JH2, and JH3.

The power supply required for use in this setup outputs 5V ±5% DC and 300mA, center positive. Insert the power supply into the power jack J2 on the JTAG board.

Finally we need to connect our PC serial port to the JTAG board. Use a normal, 9-pin, straight-through serial cable to connect one of the serial ports on your computer to the serial port on the JTAG board (NOTE: Do not connect to the serial port on the MAXQ2000 Evaluation Kit...see **Figure 3** for details on proper connection).
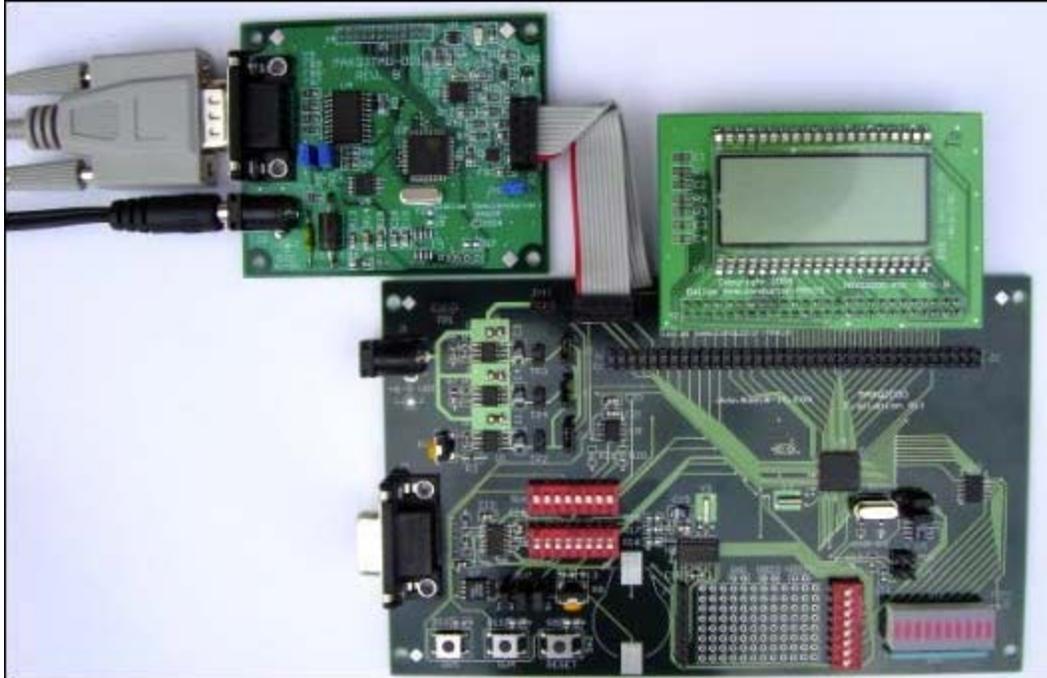
Figure 3. Correct position for serial cable, attached to JTAG board.

We are now ready to start working with the IAR tools.

Getting Started with the IAR Compiler: Hello World

IAR provides an evaluation, code-size-limited copy of its compiler for the MAXQ platform. It is available on the CD that comes with the MAXQ2000 Evaluation Kit. Additionally, you can download an evaluation copy from http://www.iar.com. Follow the instructions to install, choosing the defaults for install location and options. Note that the IAR Embedded Workbench products are only available for Windows® platforms.

Start the IAR Embedded Workbench by following the links from the Start menu: IAR Systems ➞ IAR Embedded Workbench for MAXQ ➞ IAR Embedded Workbench. Now we will create a simple application for the MAXQ2000 Evaluation Kit.

Create a new workspace. Under FILE, select NEW. A dialog box will appear that offers you a choice between "Source/Text" and "Workspace". Select WORKSPACE and click OK. A file dialog will appear to ask you to enter your name for the new workspace. Browse to the location you would like to save your new workspace and enter your workspace name. For this project, we will call the workspace "helloworld". Click SAVE once you have entered the workspace name.

Once your workspace has been opened, we need to create a project. Under the PROJECT menu, select CREATE NEW PROJECT. Another file dialog will pop-up. Make sure that under the drop-down box "Tool Chain" that MAXQ is selected. Enter a name for your project and click the CREATE button. We will call our project "helloproject".

Now that we have a project created, we need to configure it for use with the MAXQ2000 Evaluation Kit. In the project manager window, right click on the line that says "helloproject - Debug" and click on the item OPTIONS (**Figure 4**). Most of the default options are OK, but there are a couple settings that we need to change.
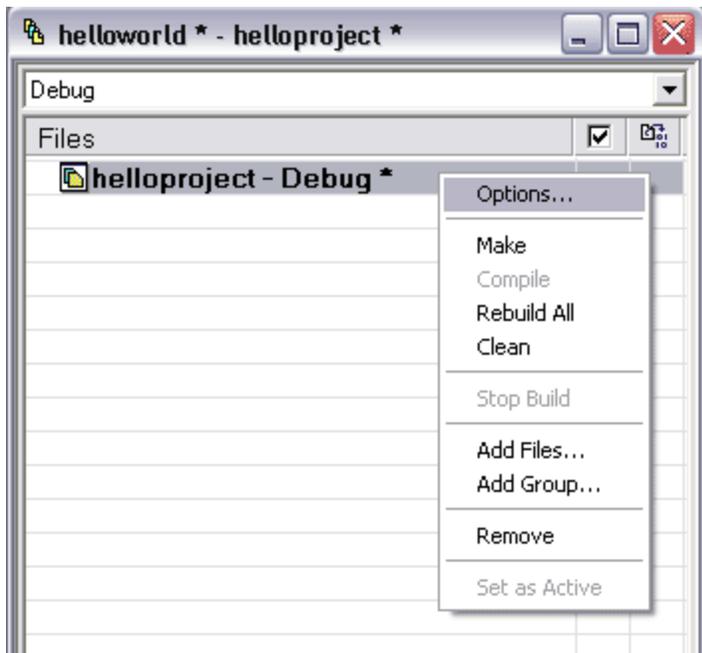
Figure 4. Right clicking on the project.

In the project options, select XLINK under CATEGORY, and then select the "Include" tab. At the bottom of the dialog, check the box labeled "Override Default" and then click the small button (labeled "...") to the right of the edit line below. A file selection dialog should appear. Select the file "lnkmaxq200x.xcl"[1], and click OPEN. The project option dialog should appear as in **Figure 5**.
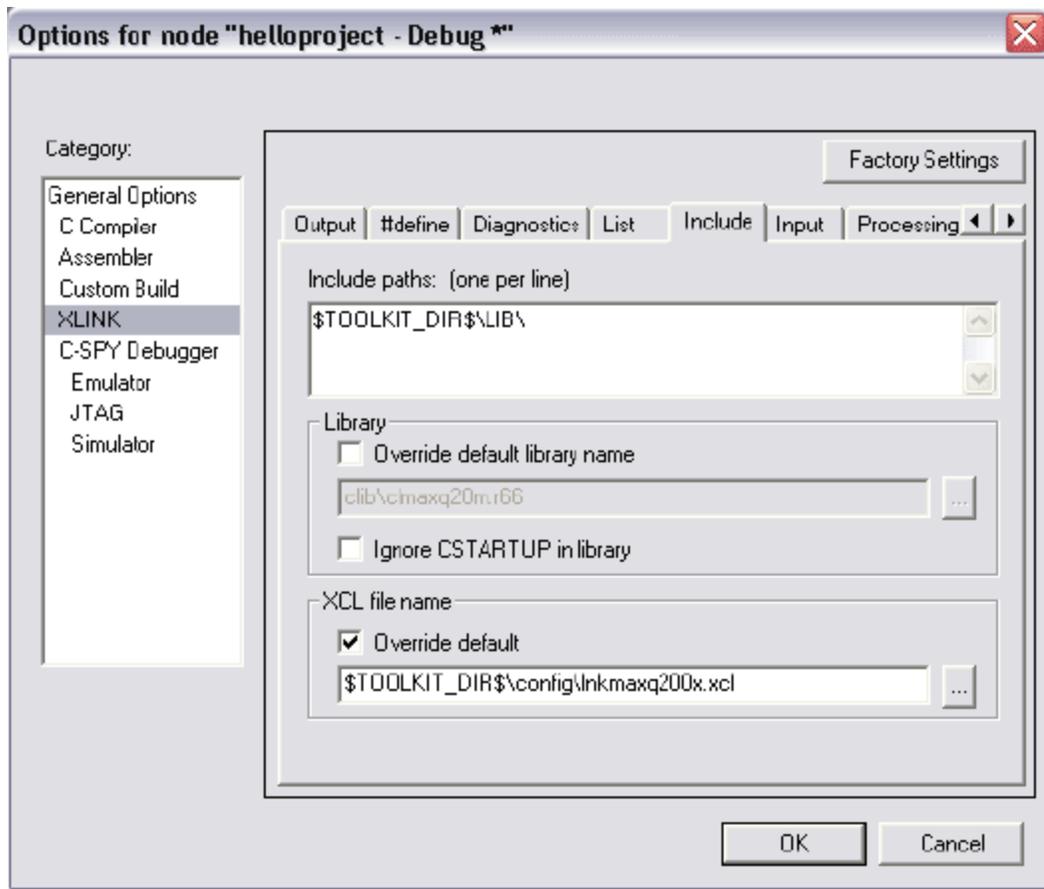
Figure 5. Setting the correct linker options for the MAXQ2000.

Now select C-SPY DEBUGGER under CATEGORY. Under the drop-down box labeled "Driver", select JTAG. This tells the IAR Embedded Workbench to debug the application on the real hardware, rather than the software simulator. Also make sure the "Device Description File" selected is '$TOOLKIT_DIR$\Config\maxq200x.ddf'. **Figure 6** shows the correct configuration for this dialog window.
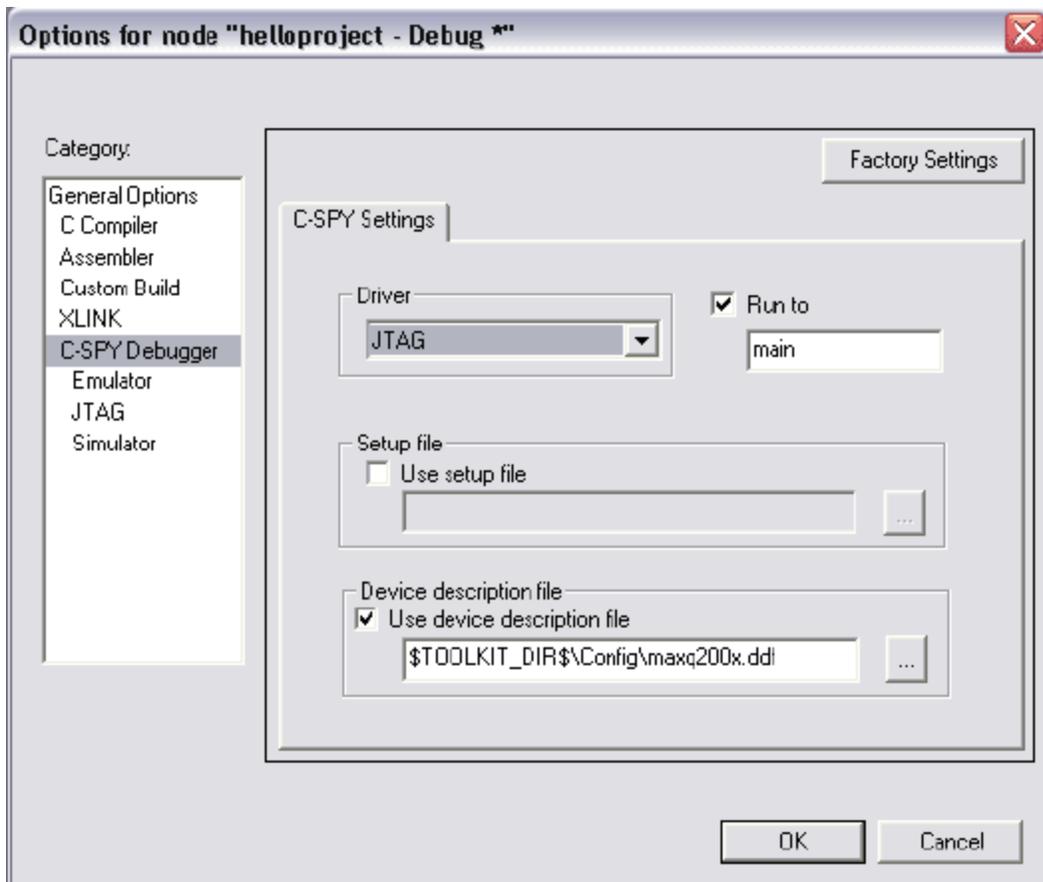
Figure 6. Debugger options for working with the MAXQ2000.

The last project option required is in item JTAG, a sub-item of C-SPY DEBUGGER under the CATEGORY list. In the edit box labeled "COM Port:", enter the COM port on your PC that you will use to communicate with the MAXQ2000 Evaluation Kit. Typically this will be COM1 or COM2, although many users with USB Serial Port adapters will use COM4 or higher.

Now we have the IAR tools configured properly to work with the MAXQ2000 Evaluation Kit. Click the OK button on the project options window. Next, we will need to create some source code for our project. First, we'll start with a simple HelloWorld-style application that just toggles an LED.

Click on the new file button on the toolbar (the blank page at the very left side of the toolbar) or select NEW from the FILE menu. Select SOURCE/TEXT and click the OK button. A new window labeled "Untitled1" will appear. Type the following code into the new window:

```
#include <iomaxq200x.h>

void main()
{
    unsigned int counter1;
    unsigned int counter2;
    PD0 = 0xff;
    while (1)
    {
      for (counter1 = 0; counter1 < 0xffff; counter1++)
      {
        for (counter2 = 0; counter2 < 0x10; counter2++)
        {
```

```
      }
    }
    PO0 = PO0 ^ 0xff;
  }
}
```

Save this file by clicking the Save button (the disk icon on the toolbar) or selecting FILE→SAVE from the menu. In the file dialog that appears, type "demo.c" for the filename and click save. Note that the file "demo.c" does not appear in the project window. Before we can build the project, we need to add this source file to the project. Right click on the line "helloproject - Debug" in the project window, and select "Add Files..." from the menu. When the file dialog appears, select the file "demo.c" and click OPEN. Now the source has been added and we are ready to build.

Under the PROJECT menu, select the option REBUILD ALL. The message window at the bottom of the screen should output a large amount of data, ending with:

```
Total number of errors: 0
Total number of warnings: 0
```

If any errors or warning appeared, make sure that you entered the code correctly and followed all of the instructions for project set-up. One common warning that might appear at this point is "last line of file ends without a new line". If you receive this error, go to the last line of your source code, and hit ENTER a couple times to create some new lines after the last closing bracket ('}').

Once we successfully build the project, we are ready to run it. Make sure you have the JTAG board and MAXQ2000 board connected and powered as described in the first section of this document. Also, for this first demonstration, we will need to turn on one switch that will enable the LED segment to light up. Find the switches labeled SW6, and turn switch number 8 to the on (upper) position. To run the project, select the PROJECT menu, and then the DEBUG option. A window should appear to tell you the application is downloading.

The project will start by hitting its first breakpoint at the first line of executable code, "PD0 = 0xff;". We'll explore the debugging options available in the IAR Embedded Workbench later. For now, just hit the GO button (three blue arrows) in the toolbar or select GO under the DEBUG menu. You should see most of the LED segments blinking on and off about once a second.

We have just compiled, loaded, and run our first application for the MAXQ2000 Evaluation Kit using the IAR Embedded Workbench. Next, we will make our sample application a little more complex. After that, we will discuss some of the features of the MAXQ2000 Evaluation Kit, and then we will walk through some of the debugging features available in the IAR Embedded Workbench.

A Simple Application: Displaying a Counter on the LCD

Now that we have successfully toggled an LED, let's move on to something a little more complicated-using the LCD. The MAXQ2000 microcontroller has an integrated 132-segment LCD controller and an on-chip resistor divider for contrast control. It is also capable of supplying power directly to an LCD screen. The MAXQ2000 Evaluation Kit comes with a simple, static LCD screen that has four digits along with some simple punctuation (a couple colons and periods, see **Figure 7**). For starters, we'll write an application that just implements a counter on the LCD. Later, we'll talk more about the code that controls the LCD and show a more complicated example.

Create a new workspace and project using the steps described previously (remember that you will need to re-enter all of the XLINK, C-SPY and JTAG options for this new project). We'll call this workspace LCDDemo and the project SimpleLCD. Download the source code for this example. Place the file lcdcounter.c into the same directory where you just created the LCDDemo project. In the IAR project

window, right click on the line that says "SimpleLCD - Debug" and select "Add Files...". Select the file lcdcounter.c that we just added and click OPEN.

Before we run the project, let's look at a couple important segments of the code. First, scroll all the way down to the bottom of the file, to the line that reads void main(). This is the main entry point for the application. You can see that we initialize the LCD (the function initLCD) and then enter an infinite loop while (1). The main loop calls a function show which displays a number on the LCD screen. Every time through the loop, we increment the value count by 1, rolling over if we exceed 19999, the maximum value our LCD screen can display.

Let's go ahead and run the application. Click the DEBUG button (far right on the toolbar) or select PROJECT → DEBUG from the menu. Note that the IAR Embedded Workbench automatically builds the application before loading it onto the evaluation kit. Again we see the IAR display change to debug mode, and execution stops on the first line of code in void main(). Press the GO button (three blue arrows) and look at the display screen. It should be rapidly counting upwards. If you watch long enough, it will wrap around when it gets close to 20000.

Now is a good time to show one of the more useful features of the MAXQ2000 Evaluation Kit-the RESET button. On the low left side of the Evaluation Kit board is a switch labeled SW2 and RESET. Press the button and watch the LCD screen. It should instantly start over from 0 again. This button is tied to the reset pin of the microcontroller-if you ever need to restart your application, just press this button.

Features of The MAXQ2000 Evaluation Kit

Now that we have built an application using the LCD daughterboard of the MAXQ2000 Evaluation Kit, let's take a closer look at all the toys that come with the Evaluation Kit. We won't be covering all of the components of the evaluation kit here, but you can find more information in MAXQ2000 Evaluation Kit schematics and "MAXQ2000 Evaluation Kit Getting Started Guide", both available on the CD that comes with the evaluation kit.

LCD Daughterboard

We've already seen the LCD board in action a little bit. Segment and common signal generation is controlled by several registers on the MAXQ microcontroller. The LCD daughterboard has been wired so that it is easy to write a numerical digit to any one of the four locations on the LCD screen. Since the mapping for each 7-segment LCD digit is the same, we include the following table in our code to help us write the proper LCD register values:

```
#define  LCD_PATTERN_0     0x03F
#define  LCD_PATTERN_1     0x006
#define  LCD_PATTERN_2     0x05B
#define  LCD_PATTERN_3     0x04F
#define  LCD_PATTERN_4     0x066
#define  LCD_PATTERN_5     0x06D
#define  LCD_PATTERN_6     0x07D
#define  LCD_PATTERN_7     0x007
#define  LCD_PATTERN_8     0x07F
#define  LCD_PATTERN_9     0x067

int PATTERNS[] = { LCD_PATTERN_0, LCD_PATTERN_1, LCD_PATTERN_2, LCD_PATTERN_3,
                   LCD_PATTERN_4, LCD_PATTERN_5, LCD_PATTERN_6, LCD_PATTERN_7,
                   LCD_PATTERN_8, LCD_PATTERN_9 };

int getLCDDigit(int digit)
{
    return PATTERNS[digit];
}
```

Using the getLCDDigit function, we can use the following code to write values to the controller screen:

```
/* write the value 612 to the LCD screen */
LCD2 = getLCDDigit(6);
LCD1 = getLCDDigit(1);
LCD0 = getLCDDigit(2);
```

This LCD board can also display periods and colons, making it ideal for clocks, temperature readouts, and any other simple numerical display. Figure 7 shows a complete map of the segments on the LCD daughterboard, along with the LCD register bits that are used to enable each segment.
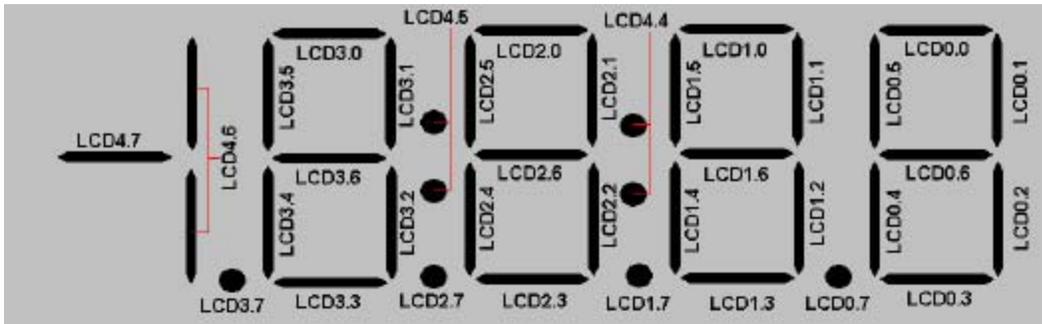

Figure 7. Mapping LCD segments to MAXQ2000 register bits.

Push Buttons

There are three push-buttons on the MAXQ2000 Evaluation Kit, two that can be tied to external interrupts and one that is tied to the reset signal on the microcontroller. The reset button can be used to restart execution of your application, as we demonstrated earlier with the LCD counter.

The other two buttons can be tied to external interrupts if enabled by the switches. Each button has a pair of external interrupts to which it can be connected. Push button 1, labeled SW4 on the evaluation kit board, can be tied to either port 5 pin 2 (external interrupt #10) or port 5 pin 3 (external interrupt #11). Push button 2, labeled SW5 on the evaluation kit board, can be tied to either port 7 pin 0 (external interrupt #14) or port7 pin 1 (external interrupt #15). The following table describes the switches you can use to connect the push buttons to the external interrupt pins.

Switch (set to ON)	Connect push button	To port pin...	External Interrupt #
SW6, switch 2	Push Button 1 (SW4)	Port 5 Pin 2	10
SW6, switch 3	Push Button 1 (SW4)	Port 5 Pin 3	11
SW6, switch 4	Push Button 2 (SW5)	Port 7 Pin 0	14
SW6, switch 5	Push Button 2 (SW5)	Port 7 Pin 1	15

Note that these pins have alternate functions, which is why we have a choice of which external interrupt we can connect to the push buttons. For instance, port 7 pins 0 and 1 are used for the serial port transmit and receive lines for UART 0. If you tie the push buttons to one of these, you will have a hard time using the UART 0 functionality. Remember that there are two UARTs on the MAXQ2000, so you have some options when you decide how you are going to configure your MAXQ2000 Evaluation Kit.

Serial Connector

The MAXQ2000 Evaluation Kit contains a 9-pin serial connector and an RS-232 level converter. This means that you can take a normal, straight-though serial cable and connect it from the Evaluation Kit to your PC. This is identical to the serial cable you are using to connect to the smaller Serial-to-JTAG

board.

In order to read incoming data over this serial port, switch number 3 on SW1 must be in the ON position. Note that this could create a conflict if you are using pin P7.1 to generate external interrupts (if SW6.5 is ON). To write data over this serial port, switch number 7 on SW1 must be in the ON position. Again, this could create a conflict if you are using P7.0 to generate external interrupts (if SW6.4 is ON).

LED Panel

We've already used our LED panel in our simple, HelloWorld-style application. Let's take a closer look at what exactly we see on that panel. **Figure 8** shows the LED panel with the individual lights associated with their output pins.
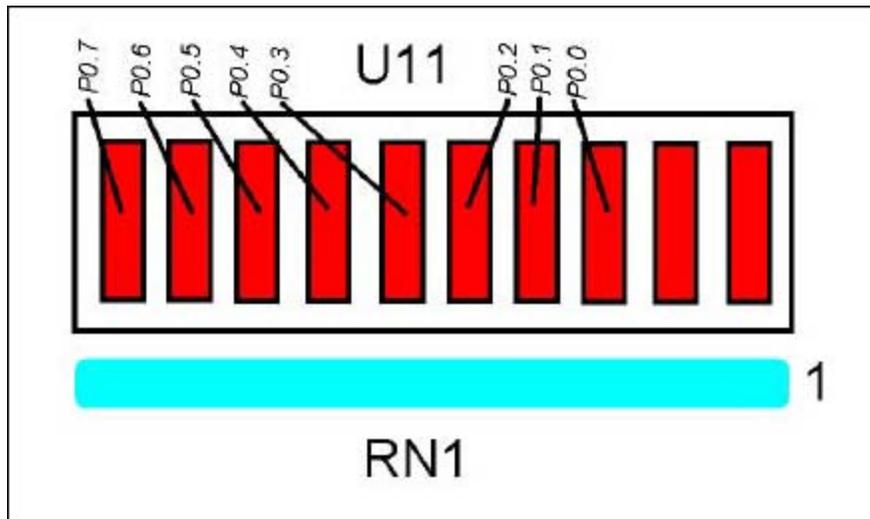

Figure 8. Mapping LEDs to MAXQ2000 register bits.

In order to write values to the I/O ports, we first need to set the direction of the port 0 pins to be output. Each pin has an independent input/output direction control. In our first application, we use the following line of code to set all of the port 0 pins to be output:

```
PD0 = 0xff;
```

When the appropriate I/O port is set to output, the LED for that pin will be lit when the appropriate bit in the PO0 register is set to 1. Note that switch number 8 on switch bank SW6 must be in the ON position to power the LED array.

MAX1407

The MAXQ2000 Evaluation Kit comes with a MAX1407, which has a 4-channel 16-bit analog-to-digital converter, and two 10-bit digital-to-analog converters. The MAXQ2000 communicates with it through its on-chip SPI™ master. This makes the MAX1407 the perfect companion device for interfacing the MAXQ2000 with real world signals. There is a sample application online[2] that uses a thermistor along with the MAXQ2000 Evaluation Kit to display the current temperature. View the source code in this sample application for information on the hardware connections and configuration you will need to perform to run that application.

See: MAX1407 QuickView data sheet.

Using the IAR Compiler to Debug Applications

Now let's move onto a new application that will introduce us to the IAR debugging tools. Create a new workspace and project as we have done before. We will call our workspace "lcdtime" and our project "rtc_demo". Now add the source file lcd_rtc.c from the source download for this application note³. Build this application and load it onto your MAXQ2000 Evaluation Kit by hitting the DEBUG button.

This application is actually an extension of the LCD counter application we wrote earlier. When you start running it, it looks the same-displaying a quickly incrementing count on the LCD screen. However, press the SW4 button and the application will display the current value of the real time clock. At first this may not seem too usable, because the time is almost certainly incorrect. Press the SW5 button a few times and you will see the minutes increment. You may soon get tired of pressing this button if you need to increment the hours as well. In this case, you can hold down the SW5 button and press the SW4 button to increment the hours. If you press the SW4 button on its own again, you will switch back to the counter display.

Go ahead and run this application by hitting the GO button (three blue arrows). Play around with it a little to get an understanding of what the application does. Once you have a grasp on the application, we will use it to introduce some of IAR's debugging tools.

Debugging with IAR: Pausing an Application

With the application running, press the button with the red hand on the left hand side of the toolbar. You can also go to the DEBUG menu and select BREAK. This will cause the processor to stop, and IAR will display your current location in the application.

With the application paused, you have several options. If you hold your mouse over some of the other buttons on the debugging toolbar (the other buttons with blue arrows) you will see a hint window pop up and tell you the function of the button. Some of the more useful buttons are:

- Step Over: Steps over the next instruction. If the next instruction is a function call, pressing this button will not take you into the function for debugging.
- Step Into: Steps into the next instruction. If the next instruction is a function call, pressing this button will take you inside the called function.
- Run to Cursor: Runs the application until execution reaches the current position of your cursor in the code. This can be a little more convenient than using the breakpoint functionality to run to specific point in the code.
- Go: Allow the application to execute normally. The application can be stopped by pressing the pause button or by reaching a breakpoint.

Debugging with IAR: Breakpoints

Breakpoints allow us into stop the application when it reaches a specific location. Let's show this with an example. Stop the application if you still have it running. Find the function pushButtonInterrupt in the code, and find the line:

```
IF (EIF1 & 0x04)
```

Right click on this line and select the option TOGGLE BREAKPOINT from the popup menu (not the option 'Toggle Bookmark'!). A red 'X' should appear in the margin to the left of this line. Now hit the DEBUG button again and run the application by pressing the GO button. You should see the LCD counter executing.

Press the SW5 button. You'll notice that the counter application seems to freeze. If you look back at the IAR embedded workbench, you will see that the application has stopped, and the line we added a breakpoint to has been highlighted, as in **Figure 9**.

```c
}


/**************************************************
 * The interrupt handler for module 1, including my
 * Must be careful in here to make sure I react onl
 * want, since some pins are dual purpose.
 */
#pragma vector = 1
__interrupt void pushButtonInterrupt()
{
    int temp;
    unsigned long time;
    temp = PI7;
    pbcount++;                              // increm
    // check for interrupt 10 (0x04) or interrupt 15
    if (EIF1 & 0x04)
    {
        // interrupt 10
        if (display_mode == DISPLAY_MODE_CLOCK)
        {
            // first check to see if the other is hel
            // if so, increment the hours.
            if ((temp & 0x02) == 0)
            {
                // then the other button is being hel
                while (RCNT & 0x08) {}
                RCNT = 0x8001;
                while (RCNT & 0x08) {}
                RCNT = 0x8000;
```

Figure 9. Hitting a breakpoint in the IAR Embedded Workbench.

Breakpoints are very useful tools in debugging an application. A developer often wants to know if a code path has been followed or is being missed. For instance, if our push button interrupt routine was not working, we might have to consider two alternatives—1) the push button interrupt routine code is coded incorrectly so we are not seeing what we expect, and 2) the push button interrupt code is not being executed at all. Using breakpoints, we could figure out if number 2 was the case, which would help us determine where to concentrate our debugging efforts.

Debugging with IAR: Local Variables

With the application still at the breakpoint, open the LOCALS window (if it is not already open). You can open this window by selecting the LOCALS option under the VIEW menu. If you are still paused on the line we set our breakpoint on earlier, the locals window should look something like this:
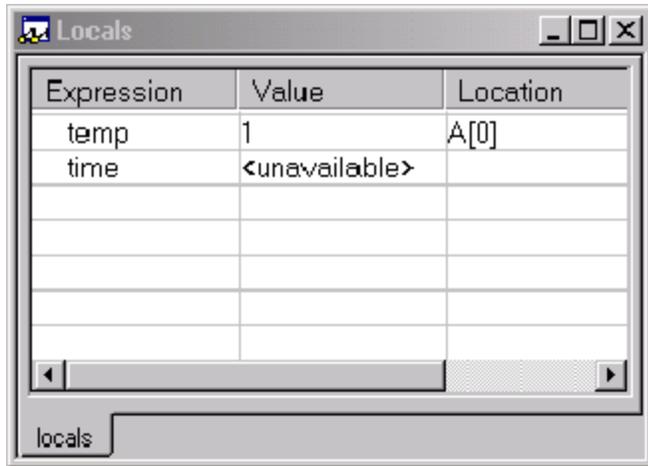
Figure 10. Local variable window in the IAR Embedded Workbench.

Note the two entries temp and time. The temp variable is used to display the value of the PI7 register. Note the other entry time says <unavailable>. This means that the variable is not currently in scope-it either has no value or its value will not be used by the function again.

The LOCALS window shows the variables declared and used in the function where the application has paused. This is a powerful tool for telling us what the current state of our application is-we can check for unexpected values in our variables or determine where the application will go next based on the values of the variables.

But that is not all the LOCALS window can do-you can change the values of your variables during program execution. Click on the value of temp (probably a 1), and enter a new integer value. You have just changed the state of your executing application. This allows developers to see how their applications would react to different inputs without rebuilding and reloading the application.

Debugging with IAR: Watch Windows

With the application still paused, open up the WATCH window (VIEW → WATCH). Under expression, enter pbcount. The value 1 should appear in the WATCH window (unless you have pressed more buttons and let the application run again). The watch window allows us to enter any expression we want, and will show us the evaluated value of that expression. In this case, we have entered the name of a global variable that tells us how many times the push button interrupt code has been entered. The watch window is a little more powerful than this, however-click on pbcount again in the WATCH window, and instead type pbcount ^ 0x05. The IAR Embedded Workbench will evaluate this expression (exclusive-or of the value of pbcount with a 5), and shows the result (4).
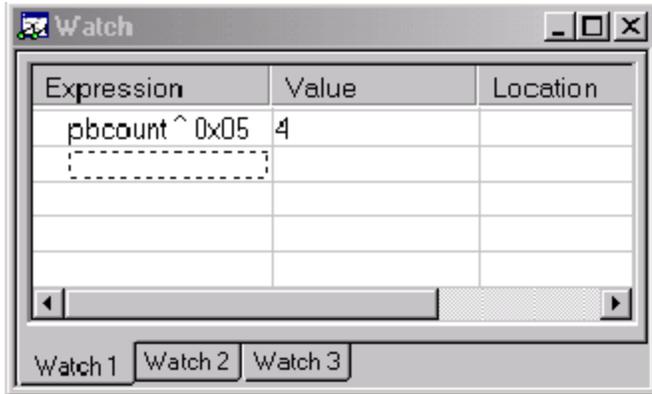
Figure 11. Watch window for expressions in the IAR Embedded Workbench.

Debugging with IAR: Call Stack

Open up the call stack (VIEW → CALL STACK). The Call Stack shows you the functions that have been called to get to where the code is currently executing. Right now, since we are servicing an interrupt, only pushButtonInterrupt() is shown (along with some debug information). Let's step through the application a little bit to see this window in action. We are heading down to the line showTime() under the code segment labeled interrupt 15. You can either press STEP OVER several times, or right-click on the line and select RUN TO CURSOR. You should now have the line showTime() highlighted. Press the STEP INTO button and we should see the first line of code in the showTime() function highlighted. Look at the Call Stack again-now the showTime() function is the first line listed, followed by pushButtonInterrupt(). Since it is a stack, the function we are currently in is listed first, followed by the function that called us, followed by the function that called that one, etc.
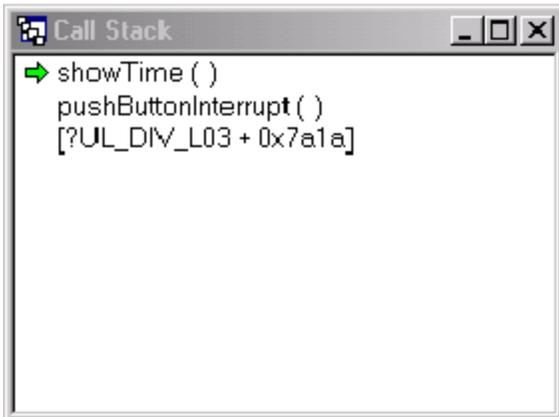

Figure 12. Call stack in the IAR Embedded Workbench.

Debugging with IAR: Memory Window

The IAR Embedded Workbench also allows us to look at the memory of the device running our application. Select VIEW → MEMORY to see the Memory Window. The first time this window opens, it is probably showing the memory contents of the code segment. Click on the drop-down box and you can see the options for the memory types we can view. Select the Data option to view the contents of the MAXQ2000's on-chip RAM.
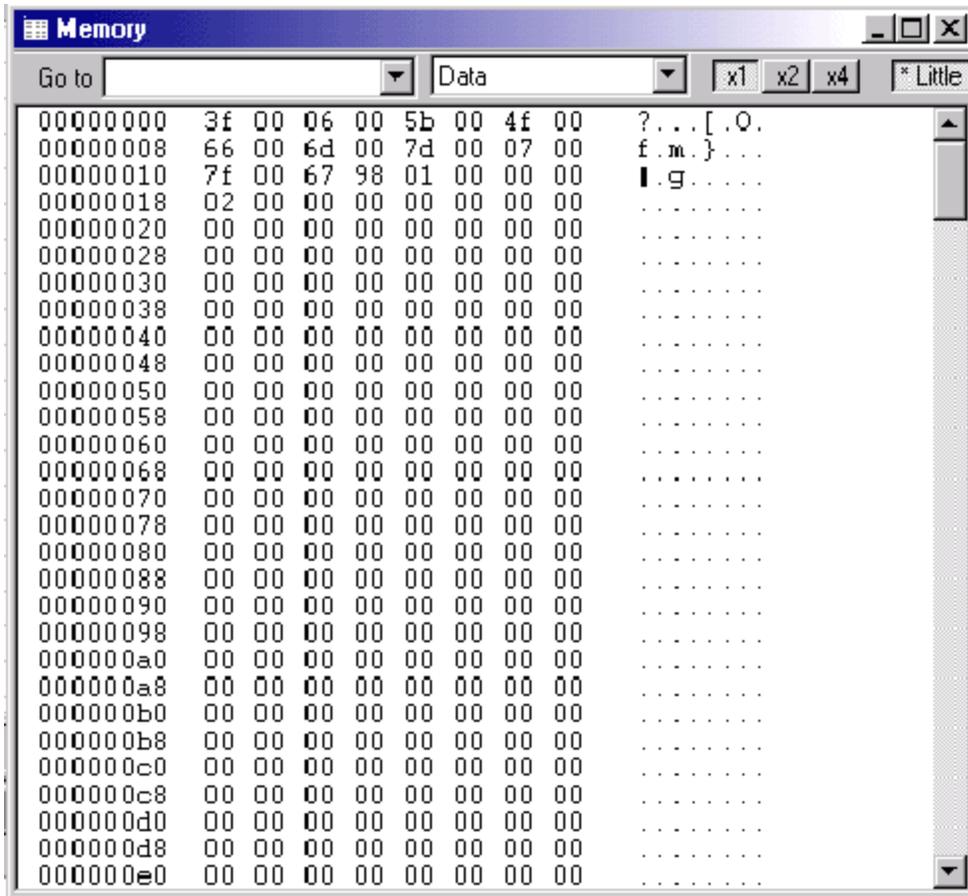
Figure 13. Memory contents display in the IAR Embedded Workbench.

Just as with the LOCALS window, we can change the values of the RAM directly here. Highlight one of the entries and type in the new hex value you would like to see. Just as with the LOCALS window, this can be a useful tool for developers to change the state of their application without rebuilding and reloading the application.

Debugging with IAR: Register Window

Similar to the Memory Window is the Register Window (VIEW ➞ REGISTER). This window shows the register map of the MAXQ2000. The first registers that appear are the core registers of the MAXQ platform, such as the accumulators, data pointers, and loop counters. Click on the drop down box and you can see several of your other options. For an example, select the Port I/O registers. You can now see and edit all of the registers associated with the input/output ports on the MAXQ2000.
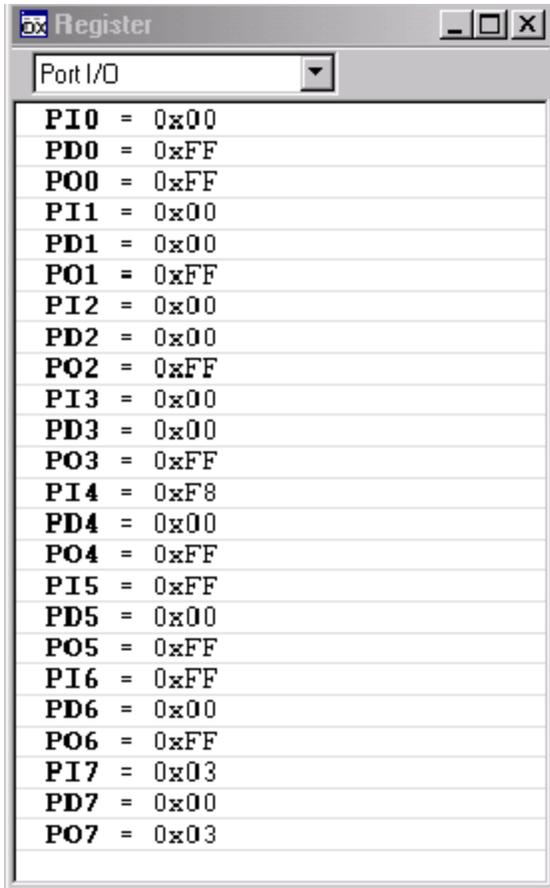
```
5X Register                          _|□|x|
Port I/O                         ▼
   PI0  =  0x00
   PD0  =  0xFF
   PO0  =  0xFF
   PI1  =  0x00
   PD1  =  0x00
   PO1  =  0xFF
   PI2  =  0x00
   PD2  =  0x00
   PO2  =  0xFF
   PI3  =  0x00
   PD3  =  0x00
   PO3  =  0xFF
   PI4  =  0xF8
   PD4  =  0x00
   PO4  =  0xFF
   PI5  =  0xFF
   PD5  =  0x00
   PO5  =  0xFF
   PI6  =  0xFF
   PD6  =  0x00
   PO6  =  0xFF
   PI7  =  0x03
   PD7  =  0x00
   PO7  =  0x03
```

Figure 14. Register contents display in the IAR Embedded Workbench.

Support Options

There are several options available for support for the MAXQ platform. One is an online discussion forum monitored by Dallas Semiconductor developers to provide answers to questions posted by users. It will also serve as a news outlet for developers, containing information on the latest tools available and other issues of interest. Go to the main page to register and log into your account:

Dallas Semiconductor Discussion Forum

For questions that are not appropriate for a public forum, please contact us through our Maxim Support Center.

For general news and information, and as your starting point for information on the MAXQ2000, the MAXQ platform, and future MAXQ devices, see the MAXQ homepage.

Troubleshooting

As with starting out with any new device, there are typically a few problems that people run into when first trying to communicate. Many problems can be solved by verifying that all the instructions in the first part of this document were followed (such as board hookup and project configuration). Below are some more common problems and solutions:

Problem	Possible Solution
When I hit debug, IAR did not	

complain at all but my MAXQ2000 hardware is not doing anything.	Make sure that you have selected the JTAG driver under the C-SPY Debugger project options.
I get an error complaining about a corrupt *.d66 file.	Make sure under the XLINK → Include options, you have selected to override the default XCL file with the lnkmaxq200x.xcl file.
I have configured everything correctly, but cannot load the application when I press the DEBUG button.	Make sure no other software is using the COM port you have selected. Often, PDA software will own the serial port from the time you boot your computer. You can either choose a different COM port, or turn off your PDA software.
IAR tries to start loading the application, but it never passes. All my cables and configurations are right.	Try restarting the IAR Embedded Workbench.

For these and other problems with loading applications onto the MAXQ2000, it may be useful to open the TOOL OUTPUT tab on the message window at the bottom of the screen. The messages listed here might give you a clue about why the load failed.

Conclusion

The MAXQ2000, Dallas Semiconductor's first device introduced from the MAXQ platform, is a powerful, low-cost, low-power microcontroller with plenty of peripheral support for consumer applications. With the support of IAR's Embedded Workbench, complex applications can be written in C and debugged with the assistance of powerful tools, yielding fast time-to-market and high quality products.

Relevant Links
- MAXQ Platform Home Page
- Dallas Semiconductor Discussion Forum for Microcontrollers

Notes
[1] This file should be in the directory that automatically opens up. If it is not, browse to "Program Files/IAR Systems/Embedded Workbench 3.2/MAXQ/config" and select the file.
[2] Download thermistor sample application
[3] Download source code for this application note

MAXQ is a registered trademark of Maxim Integrated Products, Inc.
Windows is a registered trademark and registered service mark of Microsoft Corporation.

Related Parts		
MAXQ2000	Low-Power LCD Microcontroller	Free Samples
MAXQ2000-KIT	Evaluation Kit for the MAXQ2000	

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples