



APPLICATION NOTE 3698

Getting Started with Rowley CrossWorks and the MAXQ2000 Evaluation Kit

Abstract: The MAXQ2000 is a powerful, low-cost, low-power microcontroller with considerable peripheral support for many applications. With the support of Rowley Associates' powerful CrossWorks tools for the MAXQ development environment, complex applications can be written in C and debugged. The application note shows how to set up the MAXQ2000 Evaluation Kit and get started using the CrossWorks tools. As an application, a simple running counter on an LCD illustrates the functions of both the MAXQ2000 Evaluation Kit and CrossWorks.

Overview

The MAXQ microcontroller development platform is supported by Rowley Associates' CrossWorks programming tools. This application note describes how to use CrossWorks v.1.0 and the [MAXQ2000 Evaluation Kit](#) to create, build, and debug applications written in C and targeted for the MAXQ platform. These instructions should remain applicable for later versions of CrossWorks. Features of the MAXQ2000 microcontroller are demonstrated as part of the Setting Up description below.

Setting Up the MAXQ2000 Evaluation Kit

Before we start writing code, we must connect the MAXQ2000 evaluation kit.

The kit comes with three boards, one of which has a small LCD screen. **Figure 1** shows the largest board with the LCD daughterboard connected. This is the actual MAXQ2000 EV kit, which we will discuss later. Take the LCD board and connect it to the header on the MAXQ2000 Evaluation Kit board (Figure 1) labeled J3.

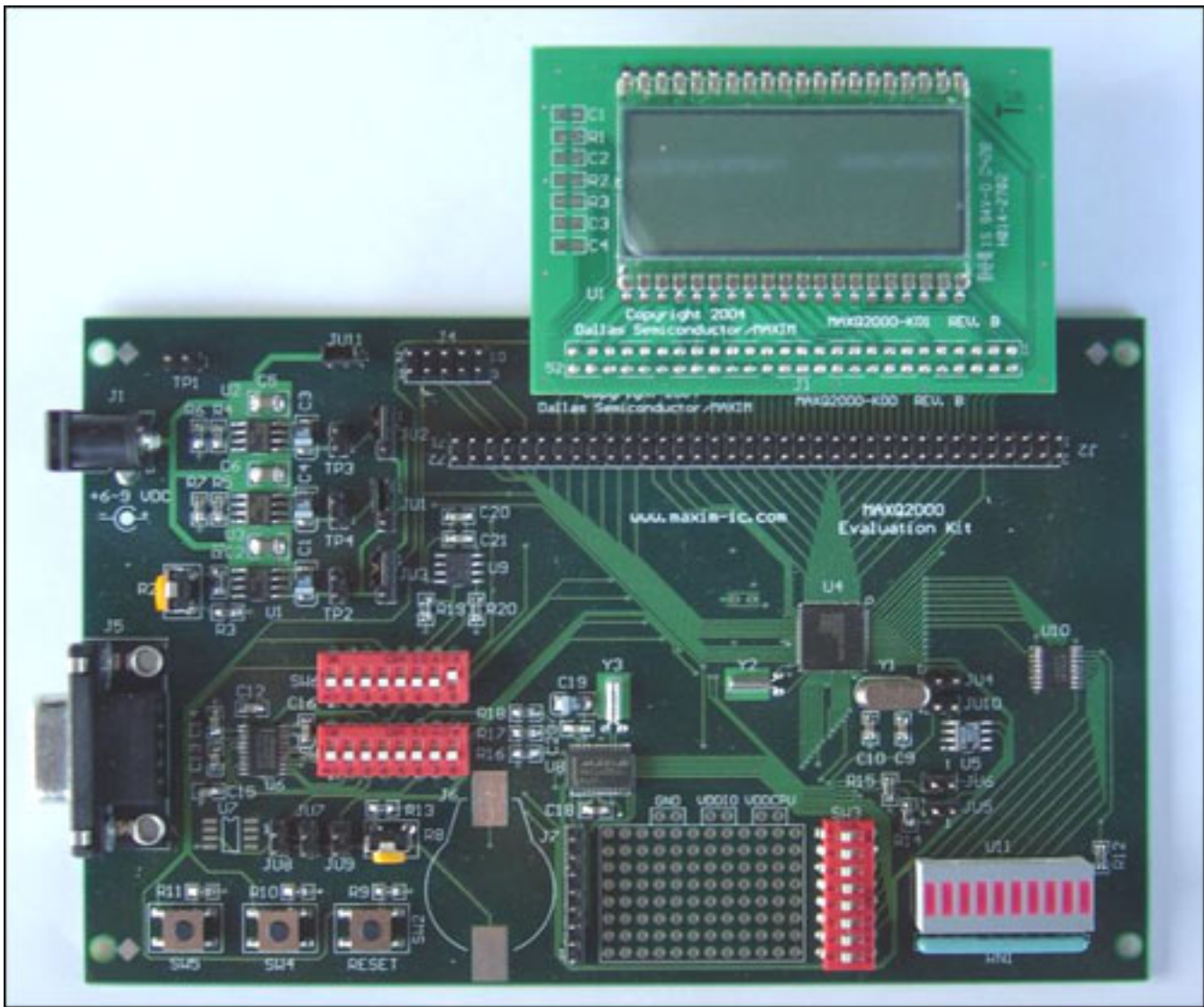


Figure 1. MAXQ2000 Evaluation Kit with LCD board attached.

The MAXQ2000 loader and debug engine communicate using the JTAG protocol. Because virtually no general-purpose, commercially available JTAG adapters exist for personal computers, Dallas Semiconductor provides a Serial-to-JTAG converter board, the third board in the evaluation kit. Use the small 10-pin cable included with the evaluation kit to attach the header labeled J4 on the MAXQ2000 Evaluation Kit to the header labeled P2 on JTAG board. See **Figure 2**. Note that the red side of the connector is on the side marked as pins 1 and 2 on both boards.

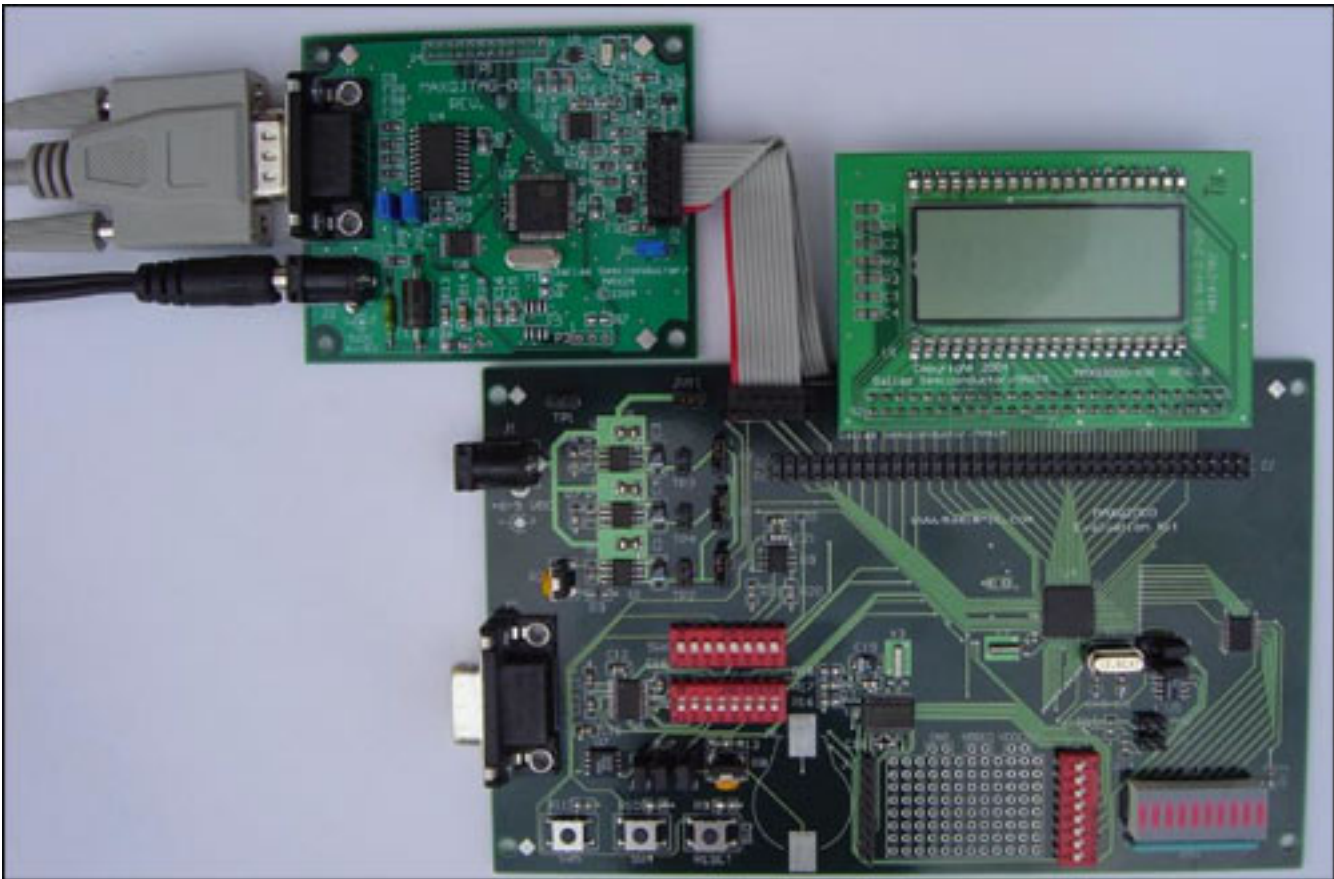


Figure 3. Correct position for the serial cable attached to JTAG board.

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

Creating a MAXQ2000 Project in CrossWorks

Rowley Associates provides a fully-functional version of CrossWorks for MAXQ with a 30-day evaluation license which can be downloaded from their website at: www.rowley.co.uk/maxq/index.htm. Follow the instructions during installation, choosing the defaults for installation location and other options. Note that Rowley Associates' CrossWorks for MAXQ is currently only available for Windows platforms. You will need to obtain a 30-day product activation key from Rowley Associates by email before proceeding; follow the instructions on Rowley Associates' website under Support: Evaluating CrossWorks.

After installation is complete, start CrossWorks for MAXQ by selecting the following links from the Start Menu: **Rowley Associates Limited**, then **CrossWorks MAXQ 1.0**, then **CrossStudio**. The main window will open with an overview of the features available in CrossWorks.

Create a new project by selecting **File**, then **New**, then **New Project** from the menu. In the dialog box which appears, select **C Executable** from the templates, enter a name and location for your new project, and click **OK** (See **Figure 4**).

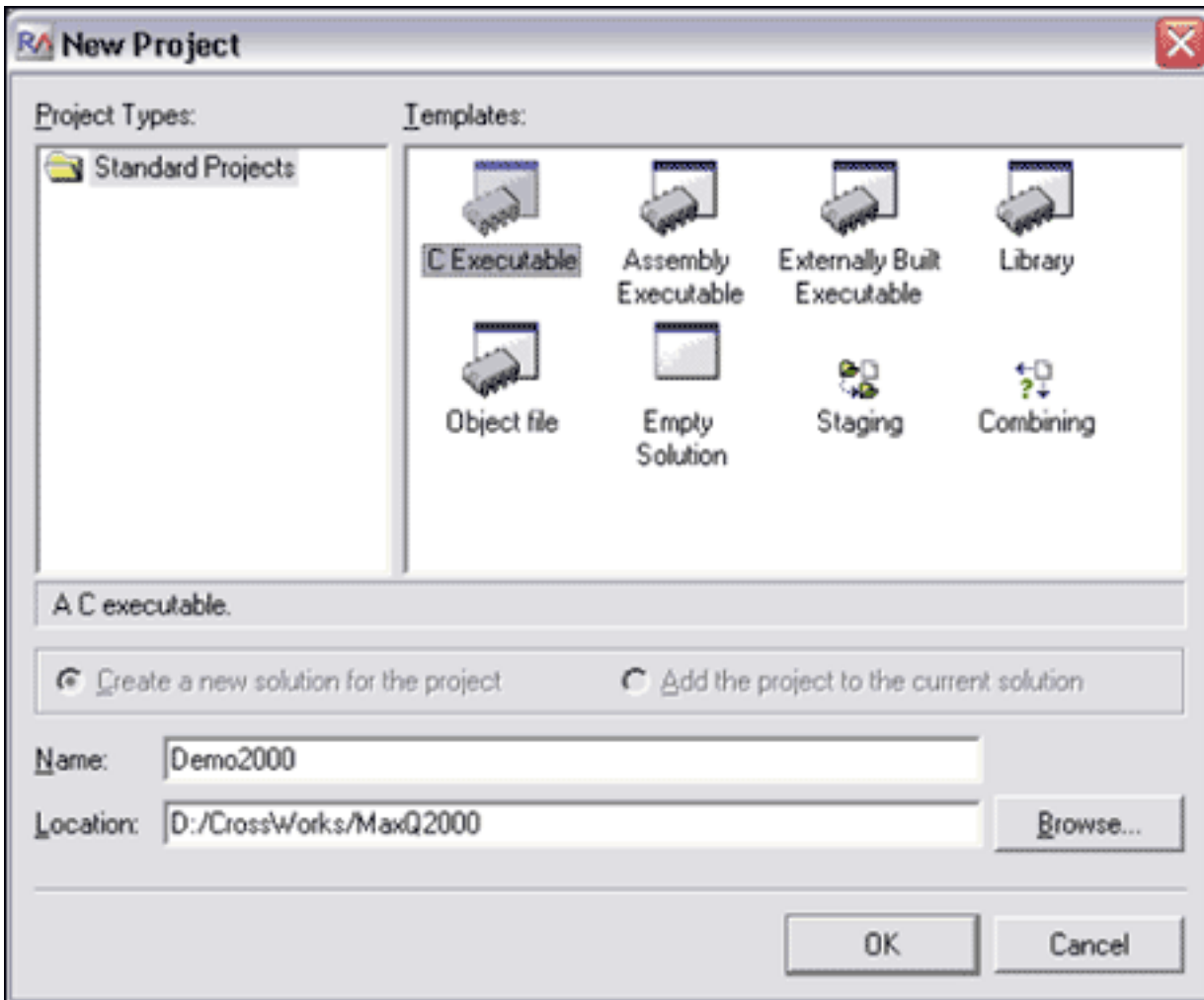


Figure 4. New Project options dialog window.

In the Project Setup dialog which follows (**Figure 5**), verify that the MAXQ2000 is selected for the **Target Processor** option. The rest of the settings can be left at their default values. Click **Finish** to generate the new MAXQ2000 project.

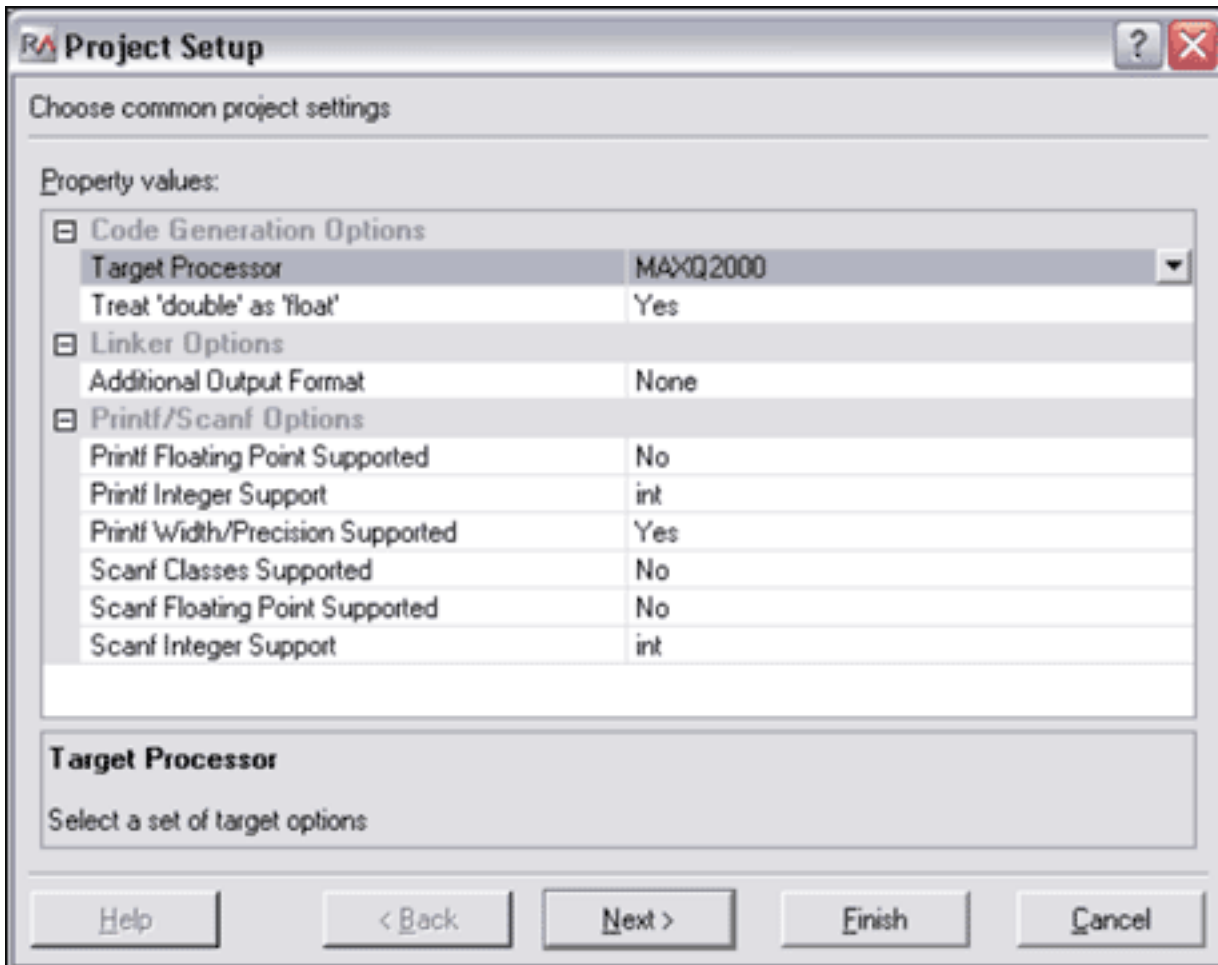


Figure 5. New Project Setup dialog window.

Next, we need to enter enough code to make the MAXQ2000 Evaluation Kit do something interesting. If the Project Explorer window is not open, open it by selecting **View** and then **Project Explorer**. Now open the `main.c` file from the Project Explorer window by double-clicking on it. Enter the following code (deleting all the original code in `main.c` first):

```
#include <MAXQ2000.h>

void main(void)
{
    int i = 0;
    int j = 0;
    int k = 1;

    LCRA = 0x03E0;    // Set LCD configuration
    LCFG = 0xF3;     // Set up all segments as outputs, normal operation
                    // mode, and enable display.
    while (1) {
        for (i = 0; i < 500; i++) {
            for (j = 0; j < 500; j++) {
                // delay loop
            }
        }
        k = (k << 1);
        if (k == 64) {
            k = 1;
        }
        LCD0 = k;
        LCD1 = k;
    }
}
```

```
LCD2 = k;  
LCD3 = k;  
}  
}
```

After this code has been entered, make sure that the MAXQ2000 Evaluation Kit board and Serial-to-JTAG board are powered up and connected as described earlier. The status bar at the bottom of the CrossWorks window should have an indicator reading MAXQ Serial to JTAG with a yellow light next to it. If, however, there is a Disconnected indicator with a grey light next to it, connect to the Serial-to-JTAG board by selecting **Target**, and then **Connect MAXQ Serial to JTAG** from the menu.

Once the Serial-to-JTAG board is connected and ready, build and execute the project code by selecting **Build**, then **Build and Run** from the menu. A series of messages ending with Verify Completed should appear in the Output window; the LCD segments on the MAXQ2000 Evaluation Kit display should begin animating as the code runs.

The code shown above demonstrates several features of CrossWorks for MAXQ. First, all the registers for the MAXQ2000 are predefined in the **MAXQ2000.h** include file. This file is shipped with CrossWorks and is located in the directory **%Program Files%\Rowley Associates Limited\CrossWorks MAXQ 1.0\include**. This directory is searched automatically when projects compile, so there is no need to copy the **MAXQ2000.h** file into your project directory.

Including the **MAXQ2000.h** file in the project allows code to reference all the MAXQ2000's internal registers directly from C, as shown above, with the registers LCRA, LCFG, LCD0, LCD1, LCD2, and LCD3. For the full list of system and peripheral registers supported by the MAXQ2000, refer to the [MAXQ2000 User's Guide Supplement](#).

Note: Registers whose names include embedded brackets, such as A[0] and DP[0], must be referenced in CrossWorks with underscores, as A_0 and DP_0. Additionally, individual register bits may not be set directly from C code, so IMR.0 = 0 is not allowed.

Features of the MAXQ2000 Evaluation Kit

Before moving on to the CrossWorks debugger, we will set up a more in-depth application which demonstrates some of the features of the MAXQ2000 evaluation kit board. The code for this example is available for [download](#).

This application displays a running counter value on the LCD which constantly increments or decrements like a stopwatch timer. The counter value rolls over at 19999, the maximum value which the LCD can display. Two pushbuttons on the MAXQ2000 Evaluation Kit board are programmed to act as control switches for this application:

- Pressing SW5 causes the counter value to reset to 0000.
- Pressing SW4 causes the counter to reverse direction. If it was previously counting upward, it will begin counting downward, and vice versa. If the counter drops below 0, it rolls under to 19999.

To change the application code, right-click on **main.c** in the Project Explorer window and select **Remove** from the pop-up menu. Next, right-click on **Source Files** and select **Add Existing File**. Select the **demo2000.c** file you downloaded from the above link. After the file has been added to the project, select **Build**, and then **Build and Run** from the menu. The new application should compile, load, and execute on the MAXQ2000 Evaluation Kit. After the application is running, check that all the DIP switches on the board are turned off except for SW6.2 and SW6.5, which should be turned on.

Now is a good time to demonstrate 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 RESET button is tied to the reset pin of the MAXQ2000. If you ever need to restart your application, just press this button.

Initializing the MAXQ2000 LCD Controller

The MAXQ2000 provides a hardware controller module for liquid crystal displays (LCDs) that run in full-bias, 1/2-bias, or 1/3-bias modes with static, x2, x3, or x4 multiplexing. This means that the 37 lines that can be dedicated to LCD drive functionality on the MAXQ2000-RAX (COM0 through COM3 and SEG0 through SEG32) can drive up to 132 LCD segments (33 segments X 4 commons) on a x4 multiplexed display.

Each segment on an active LCD display requires a continuous pattern of voltage to be driven across the segment and common lines between which the segment is connected. This voltage pattern keeps the segment either turned on or off as desired without developing a DC voltage bias which could damage the LCD glass. The MAXQ2000 LCD controller automatically generates these patterns in the background. This means that the LCD controller registers only need to be modified when the segments displayed on the LCD are changing.

As in the previous example, the first step is to initialize the LCD controller registers for the display being used.

```
void initLCD(void)
{
    LCRA = 0x03E0;    // Set LCD configuration
    LCFG = 0xF3;     // Set up all segments as outputs, normal operation
                  // mode, and enable display.
}
```

The LCRA register controls three important functions: the display type, which is static in this instance; the adjustable resistance between VADJ and ground, which can be used to modify the contrast of the display; and the frequency of the patterns used to drive the LCD segments. The LCFG register turns the LCD controller on and off, and controls which dual-purpose pins are to be used as port pins and which are to be used as LCD segments.

Writing Values to the Display

Regardless of the type of LCD used and the multiplexing mode, each LCD segment is controlled by a single bit in one of the LCD controller display registers. Setting this bit causes the segment to turn on (dark); clearing the bit causes the segment to turn off (transparent). The LCD segments that we are using are memory-mapped for the MAXQ2000 Evaluation Kit board, as shown in **Figure 6**.

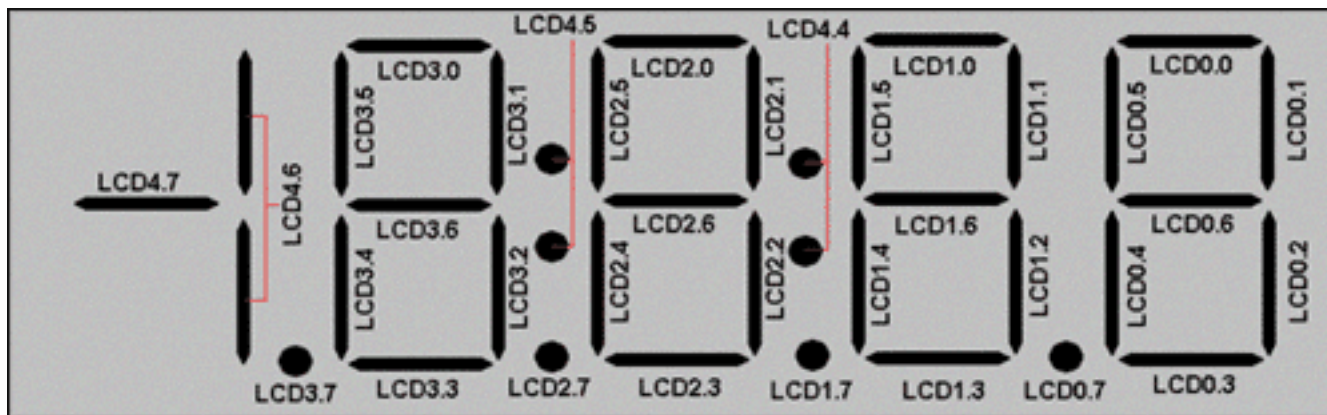


Figure 6. Memory mapping of LCD segments to LCD display-memory register bits.

Each digit on the LCD occupies one LCD display memory register, and the patterns of bits are the same for each digit. Consequently, we can write numeric values to the LCD by using a lookup table for the segment pattern for each digit from 0 to 9.

```
#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
};

/*****
 * Returns the value that will need to be placed in one of LCD0-LCD3 to display
 * a digit 0-9. No bounds checking is done here. If you ask for a digit other
 * than 0-9, you will get a bogus display.
 */
int getLCDDigit(int digit)
{
    return PATTERNS[digit];
}

```

With these routines in place, we can write our counter value (4 digits, with a special case '1' for the fifth digit) to the display as follows:

```

int show(int value)
{
    if (value >= 10000)
        LCD4 = 0x40;
    else
        LCD4 = 0;
    LCD3 = getLCDDigit((value / 1000) % 10);
    LCD2 = getLCDDigit((value / 100) % 10);
    LCD1 = getLCDDigit((value / 10) % 10);
    LCD0 = getLCDDigit((value) % 10);
    return 0;
}

```

Debouncing Pushbutton Inputs

The other interactive elements in our application are the two pushbuttons, SW4 and SW5, which are connected through the DIP switch block, SW6, to port pins P5.2 and P7.1, respectively. (The RESET switch requires no programming support, as it is hardwired directly to the active-low reset line on the MAXQ2000.)

The power-on default mode of all the MAXQ2000 port pins (except for P4.0 through P4.3, which comprise the JTAG interface) is input mode, with a weak internal pullup from the port pin to VDDIO. Since the SW4 and SW5 switches are wired to pull the port pins down to ground when depressed, the port pins are already in the configuration that we need. The state of the port pin can be read quite simply by examining the port-pin input bit (PI5.2 for SW4 and PI7.1 for SW5); the switch is depressed when the bit is zero and released when the bit is one.

Because these are mechanical switches, however, a single press can result in a number of rapid transitions from 0 to 1. To prevent this, we will perform some simple debouncing by using the existing main loop and delay as building blocks.

```
while(1)
```

```

{
for (i = 0; i < 32000; i++)
{
// just a delay loop
}
show(counter);

if (((PI5 & 0x04) == 0) && (debounce1 == 0)) {
inc *= -1;
debounce1 = 20;
}

if (((PI7 & 0x02) == 0) && (debounce2 == 0)) {
counter = 0;
debounce2 = 20;
}

counter += inc;
if (counter > 19999) { counter = 0; }
if (counter < 0) { counter = 19999; }

if ((debounce1 > 0) && ((PI5 & 0x04) == 0x04)) { debounce1--; }
if ((debounce2 > 0) && ((PI7 & 0x02) == 0x02)) { debounce2--; }
}

```

When a switch transitions from high to low, a debounce counter is set. The switch must remain in the high state for 20 clicks of the main counter before another high-to-low transition will be accepted. Besides preventing switch bounce, the debounce counter ensures that holding a switch down will not cause the function to repeat; the switch must be released and pressed again.

Using the CrossWorks Debugger

Now that this application is running on the MAXQ2000, we can examine the features of the CrossWorks debugging system. The same JTAG interface used to load code into the MAXQ2000 also supports a number of debugging features in hardware. Some examples include:

- Step-by-step single-instruction execution
- Breakpointing by execution address (up to four simultaneously active breakpoints available)
- Direct register reads and writes
- Code and stack memory dumping
- Data memory dumping and direct editing

These debugging features are supported by the MAXQ2000 (and many other MAXQ microcontrollers) at a very low level, allowing in-system debugging with almost no impact on the processor resources available to the application. The use of the dedicated JTAG port for this purpose means that no other interfaces (such as the UART ports on the MAXQ2000) need to be used to communicate with the host system. Additionally, the breakpointing mechanism operates entirely in the background; whenever the MAXQ2000 is not halted by the debugger, it runs at full speed. It is even possible to add, remove, or modify breakpoints while the MAXQ2000 is running, entirely in the background.

CrossWorks utilizes these hardware debugging features to provide C- and assembly-level source code debugging, with a full-featured set of debugging functions including breakpointing, variable and register watches, and memory dump and editing modes. To begin debugging the application, select **Build**, followed by **Build and Debug** from the menu. The application will compile, load into the MAXQ2000 Evaluation Kit, and halt execution just inside the main() function (**Figure 7**).

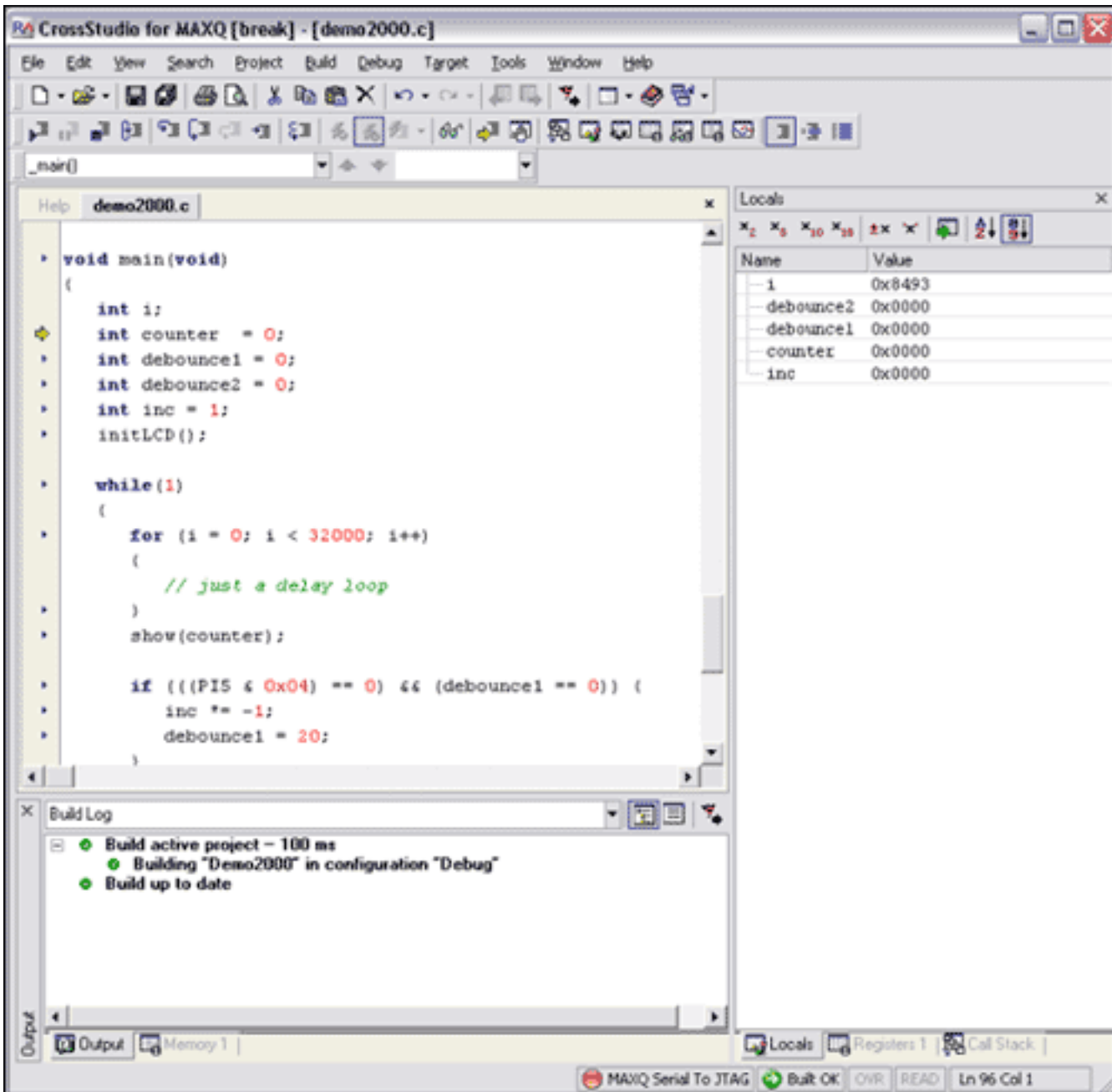


Figure 7. CrossWorks debugging mode.

Several CrossWorks debugger features can be seen in Figure 7.

- The current execution line is marked with a yellow arrow in the left margin. All executable lines (which excludes comments, whitespace, and some braces) are marked with blue triangles also on the left.
- Local variables and their values are shown in the section to the right of the application code display. These values can be edited by clicking the value and typing in a new one.

The red light next to the MAXIQ Serial To JTAG label at the very bottom of the screen indicates that the application is paused. From this point, there are a number of options for continuing execution.

- **Step Into** (F11) advances execution by a single line, allowing you to step through the code. If the line consists of a function call, the function will be stepped into, moving the execution point to the start of the function.
- **Step Over** (F10) advances execution by a single line as well, unless that line consists of a function call. In the latter case, execution will run through the function without stopping (the function is executed, not skipped) and pause on the line following the function call.

- **Step Out** (Shift+F11) can only be used from inside a function (not the top-level function main()). It advances execution through the rest of the function, and stops at the line following whichever one called the function.
- **Run to Cursor** (Control+F10) advances execution until it reaches the line on which the cursor is placed. The cursor must be on an executable code line for this to work.

The application can also be run at normal speed by selecting **Go** (F5). This releases the application from debug control and lets it run normally. While the application is running, the light next to the MAXQ Serial to JTAG label at the bottom blinks green to indicate that the application is free-running but that the debugger will break in and pause execution if necessary. There are two principal methods to cause a running application to pause and re-enter debug control.

- The **Break** command (Control+ ".") will cause the application to halt immediately at whatever location it is currently.
- Breakpoints may be set anywhere in the application by left-clicking the blue triangle that marks an executable code line. This changes the blue triangle to a red circle. If the application reaches a breakpoint while it is running, it will halt and re-enter debug mode at the breakpointed line. Up to four breakpoints can be set or cleared while the application is paused or free-running. (Note that since the various stepping functions occupy a breakpoint, only three breakpoints may be set if the step-by-step mode is used.)

Finally, selecting **Stop** (Shift+F5) will terminate the debug session and release the application to free-run.

There are three different modes for viewing applications while in debug mode:

- Source Mode (Control+T, S) shows the application as C code only.
- Assembly Mode (Control+T, A) shows the application in assembly. Step-by-step and breakpointing functions may be used on individual assembly code lines.
- Interleaved Mode (Control+T, I) (**Figure 8**) shows both C and assembly code in the same window.

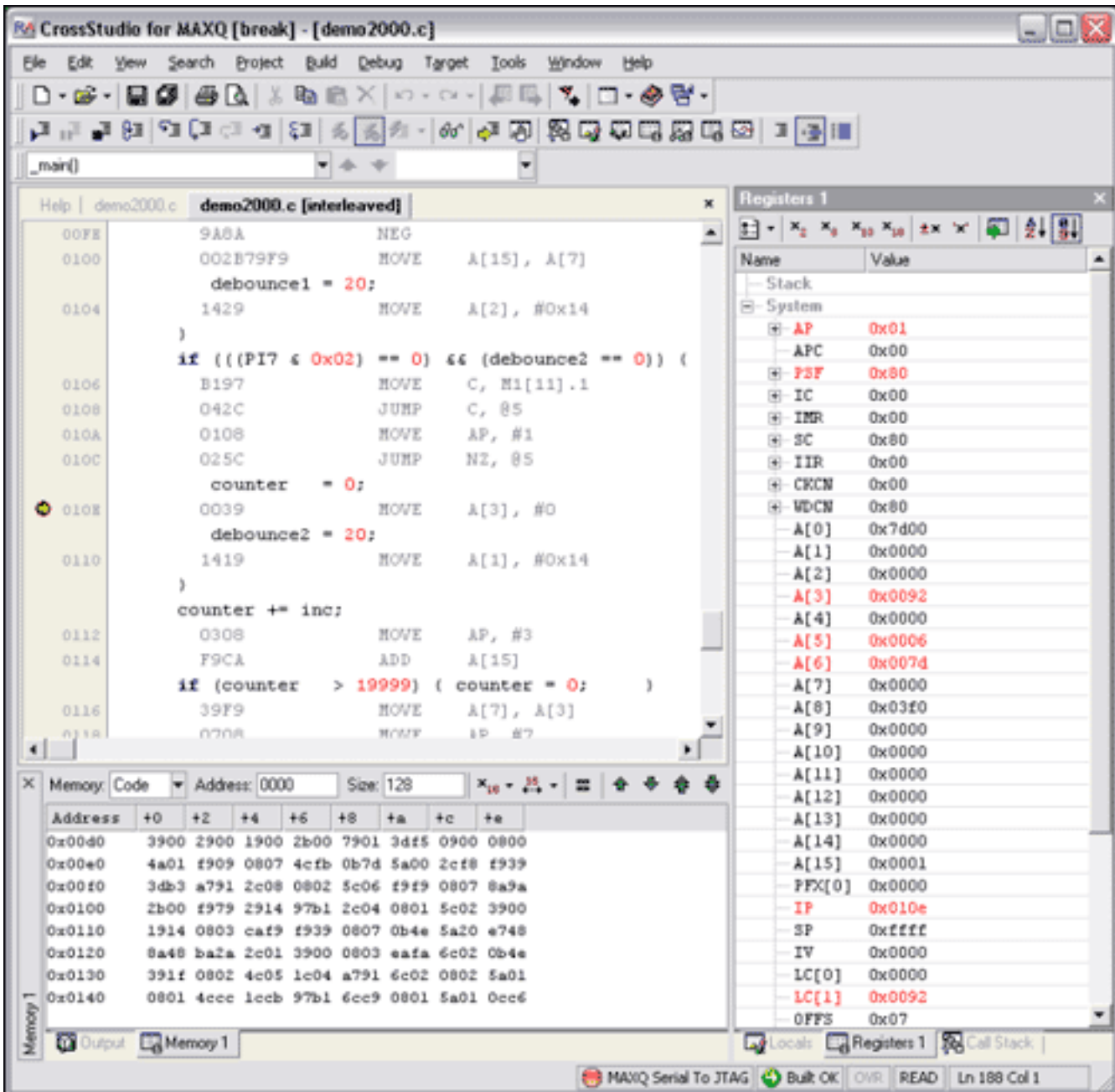


Figure 8. Debugging in interleaved mode.

Using the Register Window

While an application is paused in debug mode, it is possible to view and edit the registers of the MAXQ2000 directly (Figure 9). To open this display, select **Debug**, then **Debug Windows**, and **Registers (1,2,3,4)** from the menu. There are four different register windows available for quick display; each of these windows may be configured to show a different group of registers by clicking the Groups icon on the upper left.

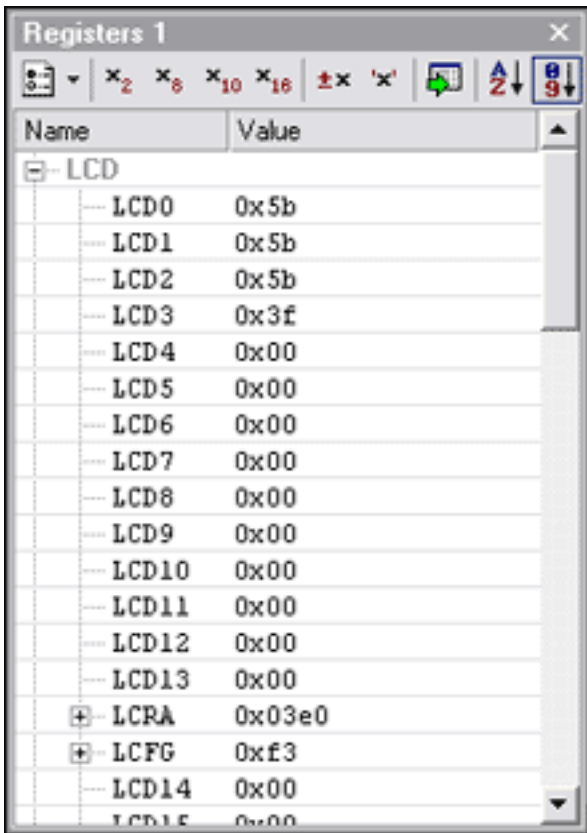


Figure 9. Register window.

Using the Call Stack Window

Another debugging window shows the current call stack of the application (**Figure 10**). This display lists all the functions called to reach the point at which the code is currently executing. The function currently being executed is shown at the end of the list; the prior function that called the current function appears on the line above. This display pattern continues until `main()` is shown on the first line. To open this window, select **Debug**, followed by **Debug Windows**, and **Call Stack**, or hit **Control+Alt+S**.

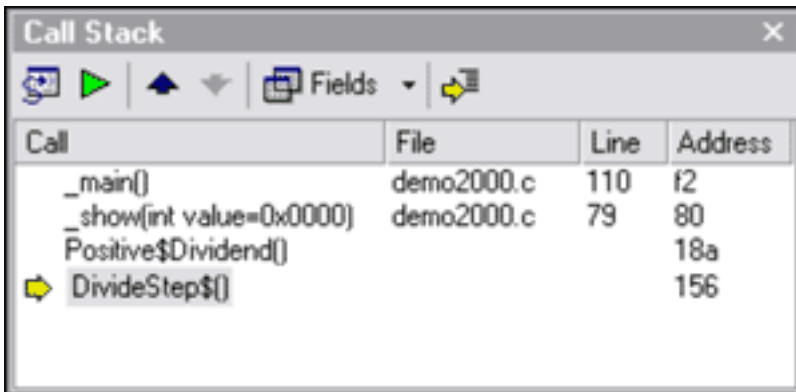


Figure 10. Call Stack window.

Using the Variable and Watch Windows

Other debug windows are available from the Debug and Debug Windows menu. Locals and Globals windows display the values of local (to the current function or scope) and global variables respectively. The local-variable display window is shown in Figure 7 above. An additional window, the Watch window, can be used to show not only the values of variables but also the values of arbitrary C expressions (**Figure 11**). MAXQ2000 registers, however, cannot be used in these expressions. Up to four separate Watch windows may be defined, each with a

separate list of variables and expressions.

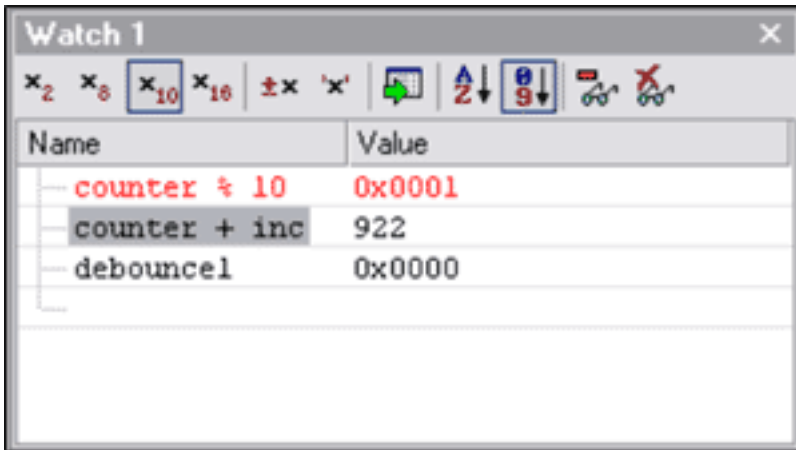


Figure 11. Watch window.

Viewing Code and Data Memory

The Memory windows (reached from **Debug**, then **Debug Windows**, and finally **Memory (1,2,3,4)**) can be configured to show the current values of any segment of code or data memory. These values will be updated following each execution step or pause at a breakpoint (**Figure 12**). In addition, the data memory values can be edited directly by clicking and typing in new values.

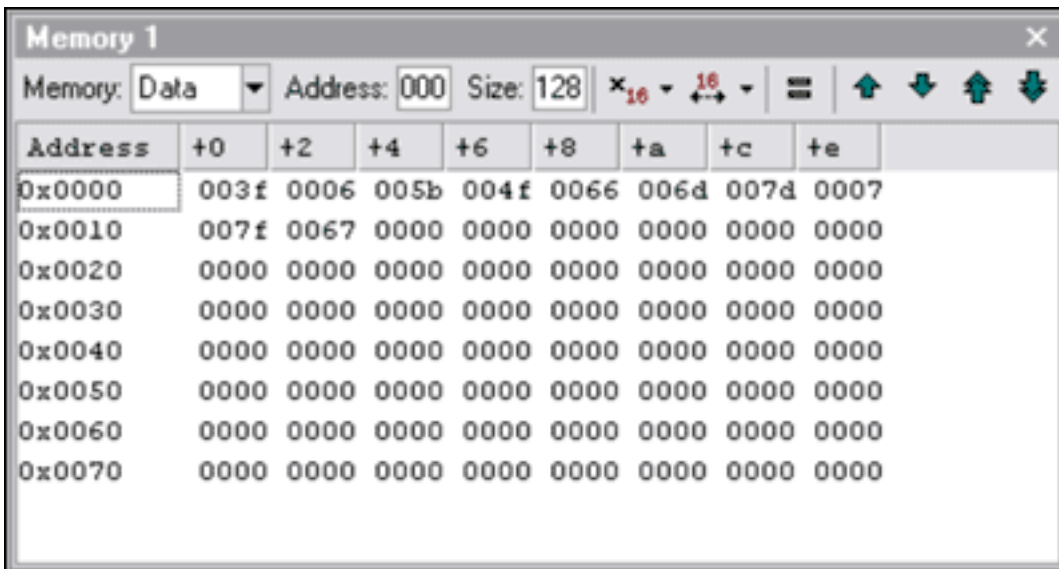


Figure 12. Data Memory window.

Support Options

There are several support options available for the MAXQ platform. An [online discussion forum](#) monitored by Dallas Semiconductor developers answers questions posted by users. It also serves as a news outlet for developers, as it contains information on the latest tools available and other topics of interest.

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

When starting with any new device, there are a few typical problems that people encounter when trying to communicate. Many problems can be solved by verifying that all the instructions presented in the first part of this document were followed (such as board hookup and project configuration). Additional common problems and solutions are shown below.

Problem	Possible Solution
When I compile my application, I get an "undeclared identifier" error each place I use a MAXQ2000 register.	Make sure you have the line <code>#include <MAXQ2000.h></code> at the start of your application.
When I select Connect MAXQ Serial to JTAG , I get a "Device is not responding" error.	Make sure: that both boards are connected and powered up; that the JTAG cable connects with the red wire going to pin 1 on both sides; and that jumpers P2 (on the Serial-to-JTAG board) and JU11 (on the MAXQ2000 board) are both closed.
When I select Connect MAXQ Serial to JTAG , I get a "Cannot open serial link" error.	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.
The LCD segments are scrambled when I run the demo application.	Make sure that the LCD daughterboard is hanging off the top side of the MAXQ2000 Evaluation kit, not in the position where it hangs downward over the center of the board.

Conclusion

The MAXQ2000 is a powerful, low-cost, low-power microcontroller with plenty of peripheral support for many applications. With the support of Rowley Associates' CrossWorks for the MAXQ development environment, complex applications can be written in C and debugged with the assistance of powerful tools. This effort results in fast time-to-market and high-quality products.

Application note 3698: www.maxim-ic.com/an3698

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

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

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

AN3698, AN 3698, APP3698, Appnote3698, Appnote 3698

Copyright © by Maxim Integrated Products

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