Excalibur Solutions
Hello_world.c

October 2001, ver. 2.0 Application Note 174

Introduction

This document describes a simple **hello_world.c** program for the Altera® ARM®-based embedded processor PLDs. A **hello_world.c** program is commonly one of the first applications a software engineer writes when developing code for a new processor. A successful build and execution on the processor validates the tool chain for developing embedded software and for downloading an application to the target device. For the ARM-based embedded processor PLDs, a successful build and execution also validates the hardware configuration of the embedded stripe and the programmable logic.

The Altera **hello_world.c** program prints a message to a terminal window. The embedded software source code provided on the Altera *ARM-Based Excalibur Utilities and Resources* CD-ROM may be useful as the basis for developing additional applications.

Software and Hardware Requirements

The example hardware and software provided in this application note were developed using the following hardware and software development tools from Altera in Windows 98/NT/2000 environments:

- The Altera Quartus® II software version 1.1
- ARM Developer Suite for Altera (ADS-Lite) software version 1.1
- EPXA10 development board

Altera recommends that you work through the example in the *Excalibur EPXA10 Development Board Getting Started User Guide* before going through this design, because the example explains how to set up the Altera ByteblasterMV™ download cable and the ARM debugger (AXD) interface.

Design Overview

The hello world project includes a simple PLD design and an embedded software application. The PLD design routes outputs from dual-port RAM, DPRAM0, in the embedded stripe of the EPXA10 to I/O pins. The selected I/O pins connect to LEDs on the EPXA10 development board.

For more details about the dual-port RAM in an EPXA10 device, refer to the *ARM-Based Embedded Processor PLDs Hardware Reference Manual* and also read application note 173, *Excalibur Solutions-DPRAM Reference Design version 2.0*.

The embedded software program writes to the DPRAM0 base address 0. From the PLD side, the DPRAM0 base address outputs are continuously fed to the I/O pins. Each write to DPRAM0 address 0 is instantly visible at the LEDS. The embedded software application uses an interrupt-driven UART driver to print the corresponding LED value to a terminal window.

The ARM library includes the ANSI C **printf()** and **scanf()** functions. The simple port of these functions defaults to using the UART in the embedded stripe. The **printf()** routine is useful for displaying characters to a terminal screen. The **scanf()** routine is useful for receiving strings from a terminal keyboard. In the hello world application, the HyperTerminal application provided with the Windows operating system is used. At low level, **printf()** and **scanf()** utilize two other functions, as follows:

- **fputc**—to send a character via the EPXA10 UART
- **fgetc**—to receive a character via the UART

Installing the Design Files

During installation of the hello world design, the design files are extracted to *<Installation Directory>***example_designs\hello_world**. Table 1 lists the design files in the archive, with descriptions.

Table 1. Design Files (Part 1 of 2)

File	Description
arm_top.bdf	Block diagram file for the top-level design.
arm_top.csf	Compiler settings file for the top-level design, which stores chip definitions, device options, compilation type, etc.
arm_top.v	Verilog file for the top-level design
Debug.fsf, Release.fsf	Software build settings files, which store compiler options, software toolset, processor architecture, etc.
hello.psf	Project settings file, which stores the working directory name, relative hierarchical assignments, device assignments, etc.
hello.quartus	Project configuration file, which stores the input filenames and compiler settings files.
hello.sbd	System build descriptor file that stores information on interconnections between modules and how they should be configured.
prog_hw.bat	Batch file for programming the hardware and software image into the evaluation board flash memory.
stripe.v	Verilog instantiation of the embedded stripe
stripe.bsf	Block symbol file of the embedded stripe
software\hello_world.c	C program, which scrolls the LEDs and sends messages to a terminal window.
software\armc_startup.s	Assembly program, which initializes the stack pointers, sets up the interrupt handler, enables the cache, and jumps to the main program.
software\uartcomm.c	C program, which implements the UART I/O functions to enable the **printf** function.

Table 1. Design Files (Part 2 of 2)

File	Description
software\uart00.h, software\uartcomm.h	Header files for **uartcomm.c**
software\irq.c	C program, which initializes the interrupt controller, first-level IRQ, and FIQ handlers.
software\int_ctrl00.h	Header file for **irq.c**
software\retarget.c	C program, which implement the functions necessary to link with the ARM C libraries.

Although the project files are included in the archive, the steps for creating them are provided below, as a reference.

Creating the Design

This section describes the process for creating the hello world program. It involves the following steps:

1. Creating a Quartus II project.
2. Configuring the stripe.
3. Instantiating the stripe.

1 Creating a Quartus II Project

Follow the steps below to create a project in the Quartus II software:

1. Run the Quartus II software.
2. Choose **New Project Wizard** (File menu) and specify the following:
   - Working directory: <*Installation Directory*>**\example_designs\hello_world**
   - Project name: **hello**
   - Top-level entity name: **arm_top**
3. Click **Next**.
4. Click **Finish** to create the project.

2 Configuring the Stripe

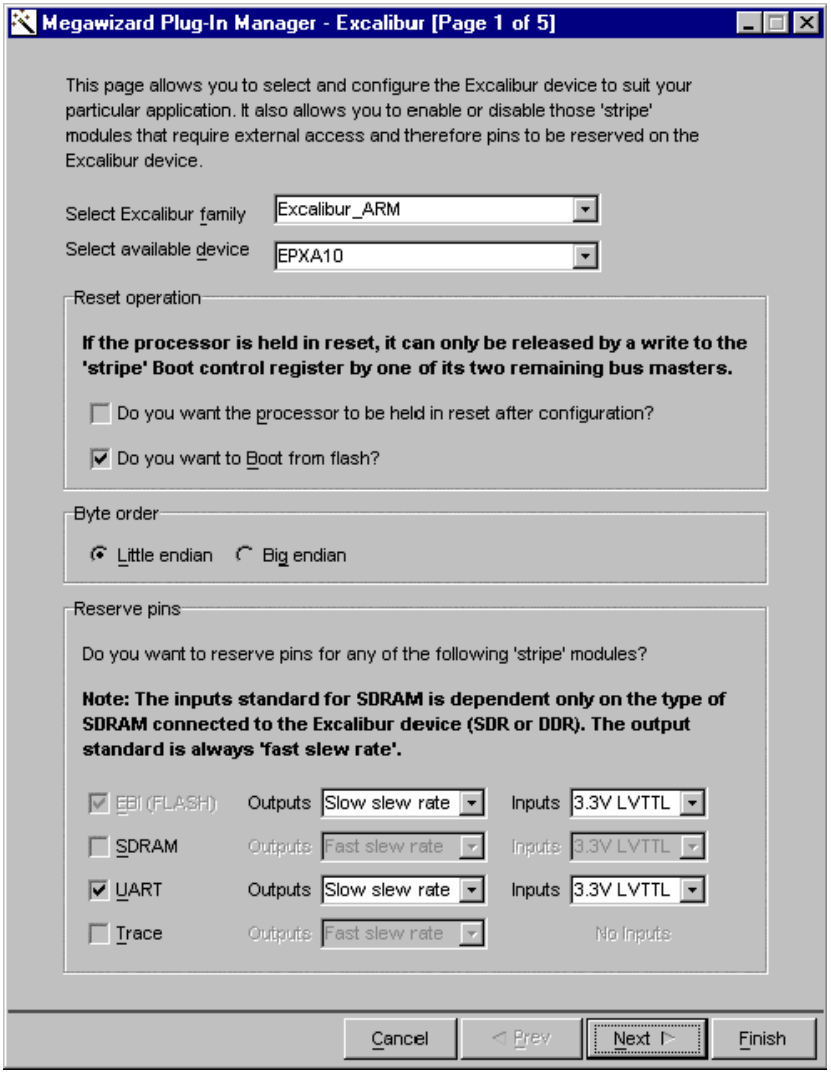
You now configure the stripe using the MegaWizard® Plug-In. At the end of this process, the wizard generates the following files:

- **hello.sbd** (system build descriptor) file, which describes the setup of the EPXA10 device (e.g., memory map, clock settings).
- **stripe.v** (Verilog HDL) files, which contain stripe instantiations.
- **stripe.h** (C header) and **stripe.s** (assembly header) files, which contain definitions of the memory map.

Follow the steps below to configure the stripe:

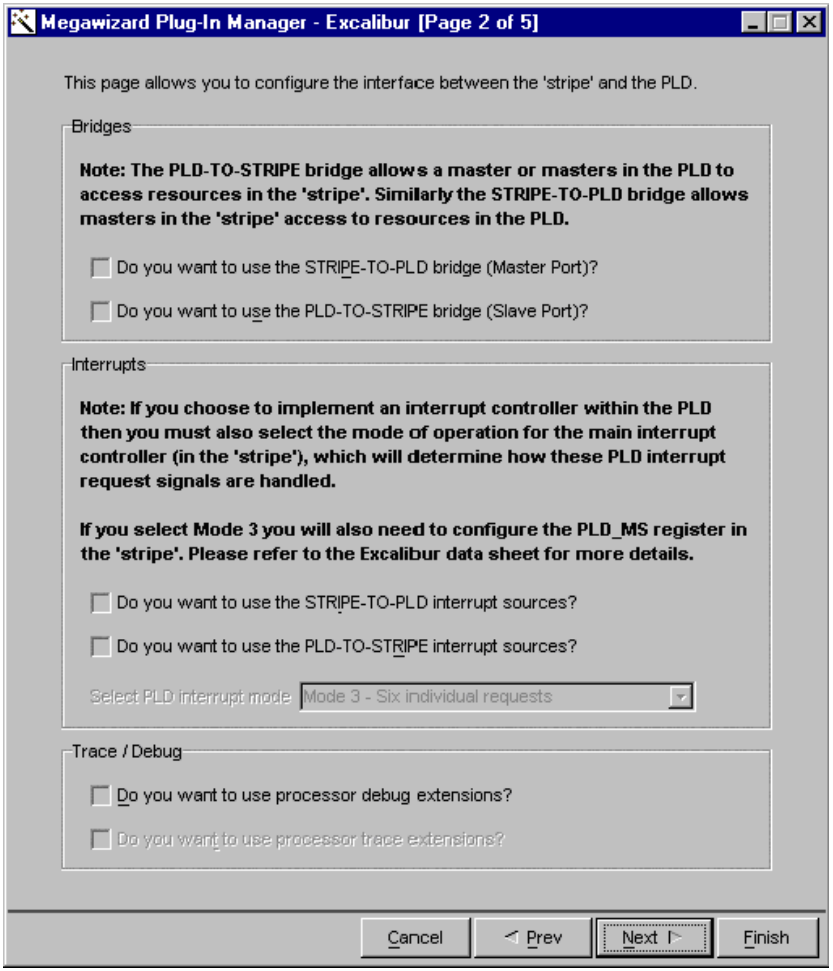
1. Choose **MegaWizard Plug-in Manager** (Tools menu).
2. Select **Create a new custom megafunction variation** and click **Next**.
3. Specify the following and click **Next**:
   - **Megafunction**: ARM-Based Excalibur (under **Installed Plug-Ins**)
   - **Output file type**: Verilog HDL
   - **Output file name**: *<Installation Directory>***\example_designs\hello_world\stripe**
4. Specify the following and click **Next** (see Figure 1 on page 5):
   - **Excalibur family**: Excalibur_ARM
   - **Available device**: EPXA10
   - **Do you want to Boot from flash?** turn on
   - **Byte order**: Little endian
   - **UART** (under **Reserve pins**): turn on.

Figure 1. MegaWizard Plug-In Page 1: Select & Configure the EPXA10 Device



5. Specify the following and click **Next** (see Figure 2):
   - **Do you want to use the STRIPE-TO-PLD bridge?** turn off
   - **Do you want to use the PLD-TO-STRIPE bridge?** turn off
   - **Do you want to use the STRIPE-TO-PLD interrupt sources?** turn off
   - **Do you want to use the PLD-TO-STRIPE interrupt sources?** turn off
   - **Do you want to use processor debug extensions?** turn off

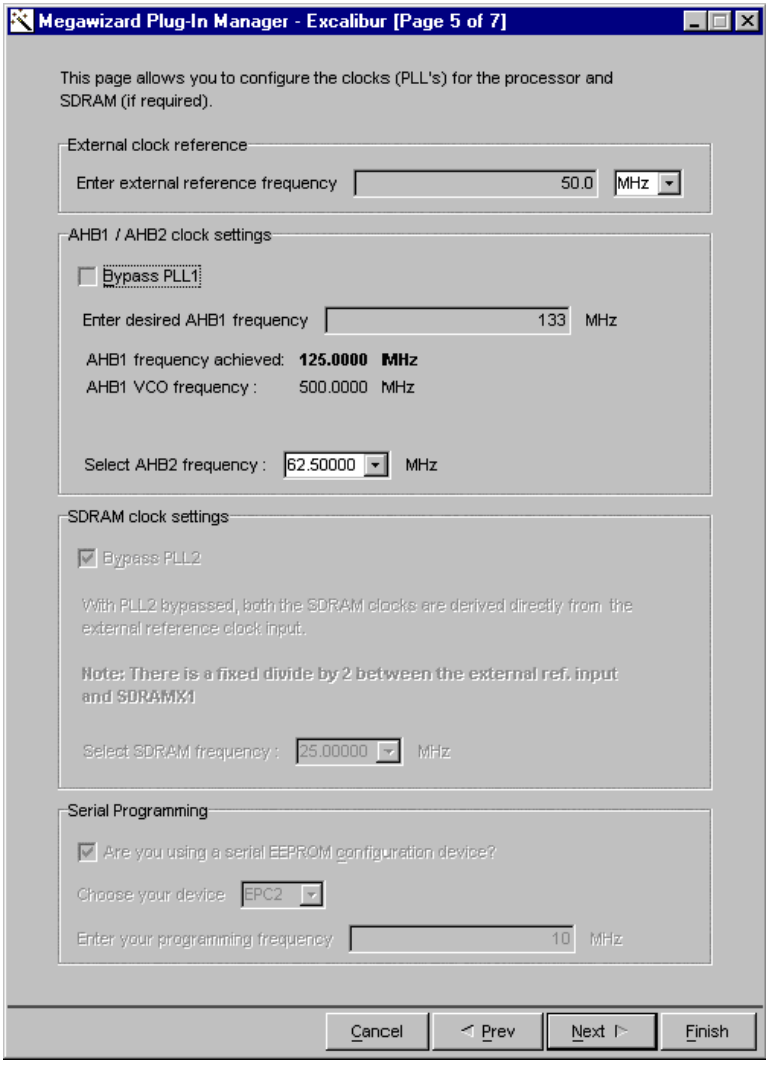
Figure 2. MegaWizard Plug-In Page 2: Select Bridges and Interrupt Sources



6. Specify the following and click **Next** (see Figure 3):
   - **External clock reference**: 50 MHz
   - **Bypass PLL1**: turn off
   - **Desired AHB1 frequency**: 133 MHz
   - **AHB2 frequency**: 62.5 MHz

   If the part number of your EPXA10 device ends in -2, the AHB1 clock runs at a maximum 166 MHz.

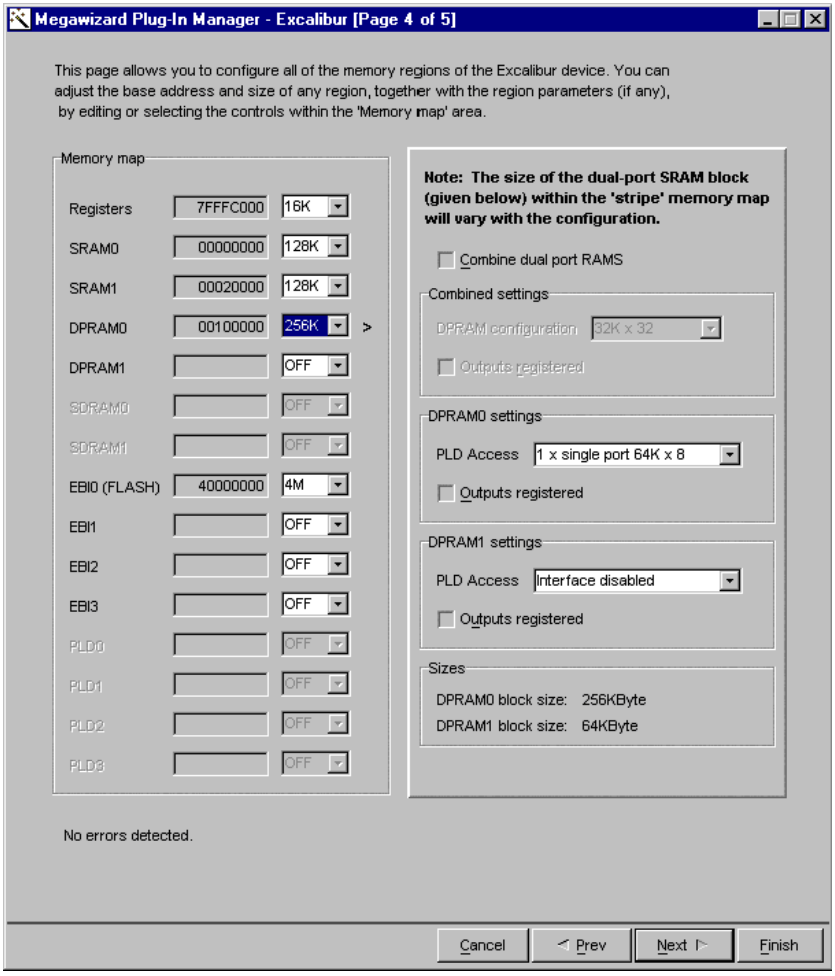
Figure 3. MegaWizard Plug-In Page 3: Configure PLLs and Clock Frequencies



7. Specify the following (see Figure 4) and click **Next**:
   - **Registers**: 7FFFC000 address, 16K size
   - **SRAM0**: 00000000 address, 128K size
   - **SRAM1**: 00020000 address, 128K size
   - **DPRAM0**: 00100000 address, 256K size

   You cannot configure DPRAM 0 as a 256K module until the PLD interface is set up as a 1 × 64K × 8 single port. To set up the PLD interface, choose **1 x single port 64K x 8** from the **PLD Access** drop-down box under **DPRAM0 settings**.

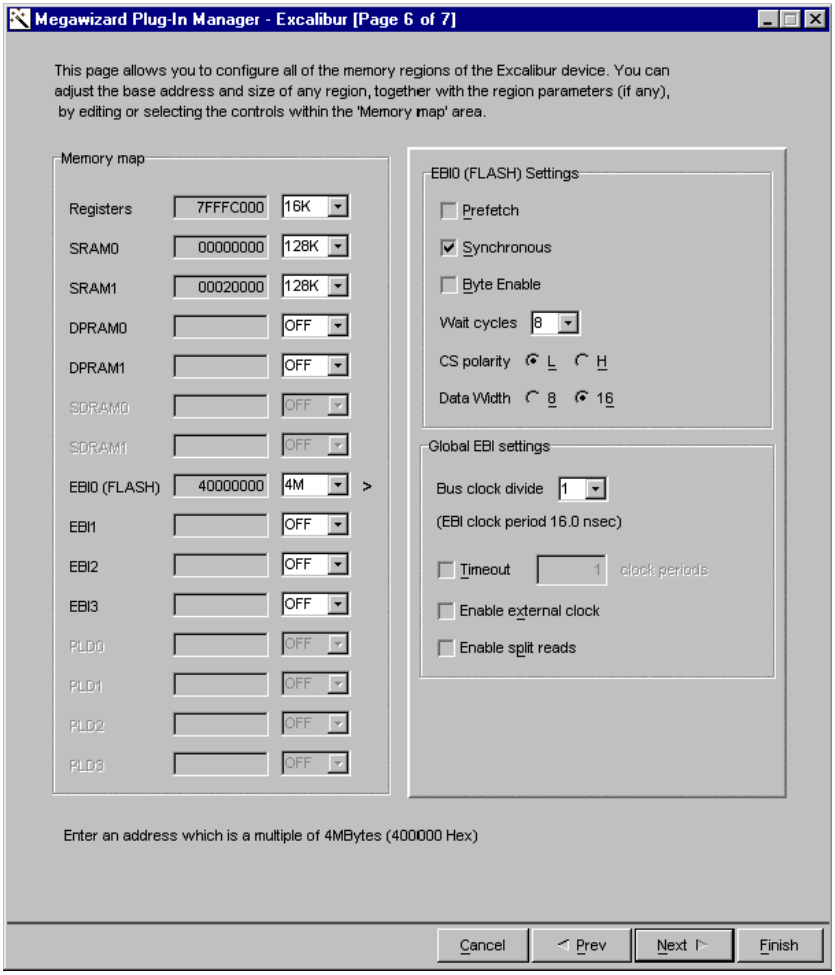
Figure 4. MegaWizard Plug-In Page 4: Specify Memory Settings and Set up the PLD Interface



– **EBI0 (FLASH)**: 40000000 address, 4M size, synchronous, 8 wait cycles, low CS polarity, 16-bit data width, 1 bus clock divide (see Figure 5)

For the EBI, further options appear when you allocate a memory region, so you can specify the EBI settings.

Figure 5. MegaWizard Plug-In Page 6: Specify EBI Settings



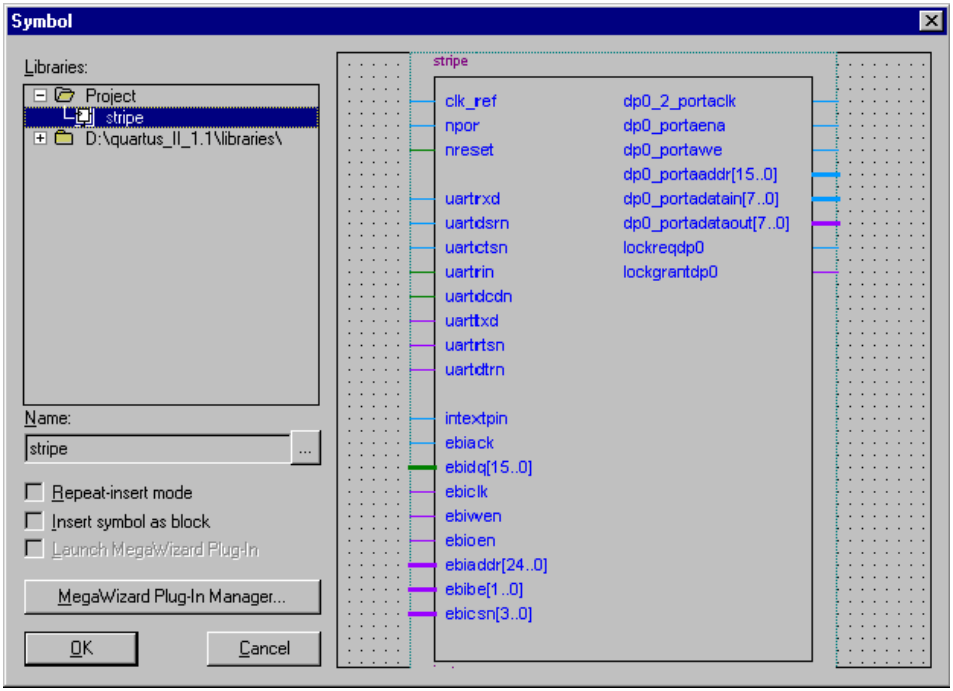
8. Click **Finish** to create the design files.

3 Instantiating the Stripe in the Top-Level Design File

The following steps instantiate the configured stripe in the top-level design file:

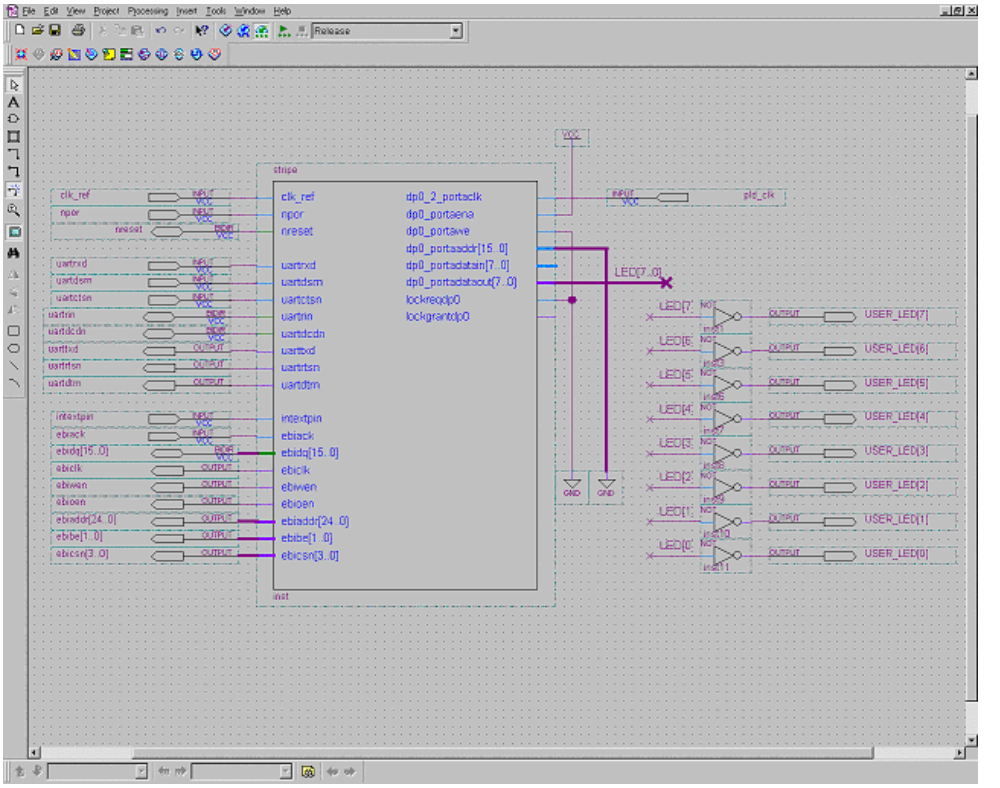
1. Choose **Open** (File menu) and select **arm_top.bdf** as the file name (**arm_top.bdf** is the block design file supplied on the Altera *ARM-Based Excalibur Utilities and Resources* CD-ROM).
2. Choose **Symbol** (Insert menu).
3. In the Symbol window, expand **Project** under **Libraries**
4. Select **stripe** from the project list (see Figure 6).

Figure 6. Stripe symbol



5. Click **OK** and click the insertion point to insert the symbol in the block diagram as shown in Figure 7 on page 11.

Figure 7. Instantiating the Stripe in the Top-Level Design



6. Choose **Save** (File menu).

Compiling the Hardware Design

This section describes the steps to compile the HDL design and fit it into an Altera device.

Specifying the Hardware Compiler Settings

The following steps select the Altera device and set the device options:

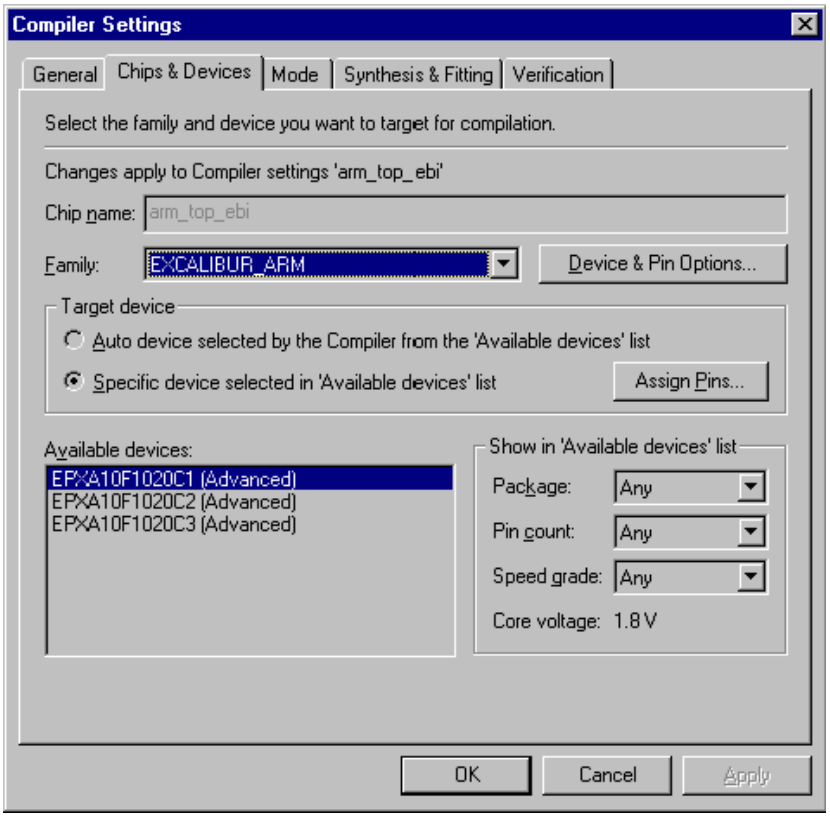
1. Choose **Compile Mode** (Processing menu).
2. Choose **Compiler Settings** (Processing menu).

3. Click the **Chips & Devices** tab and specify the following (see Figure 8):
   - **Family**: Excalibur ARM

     At this stage, if a message box appears, asking for consent to remove all location consignments before continuing, click **No**.

   - **Target device**: select **Specific device selected**
   - **Available devices**: EPXA10F1020C1

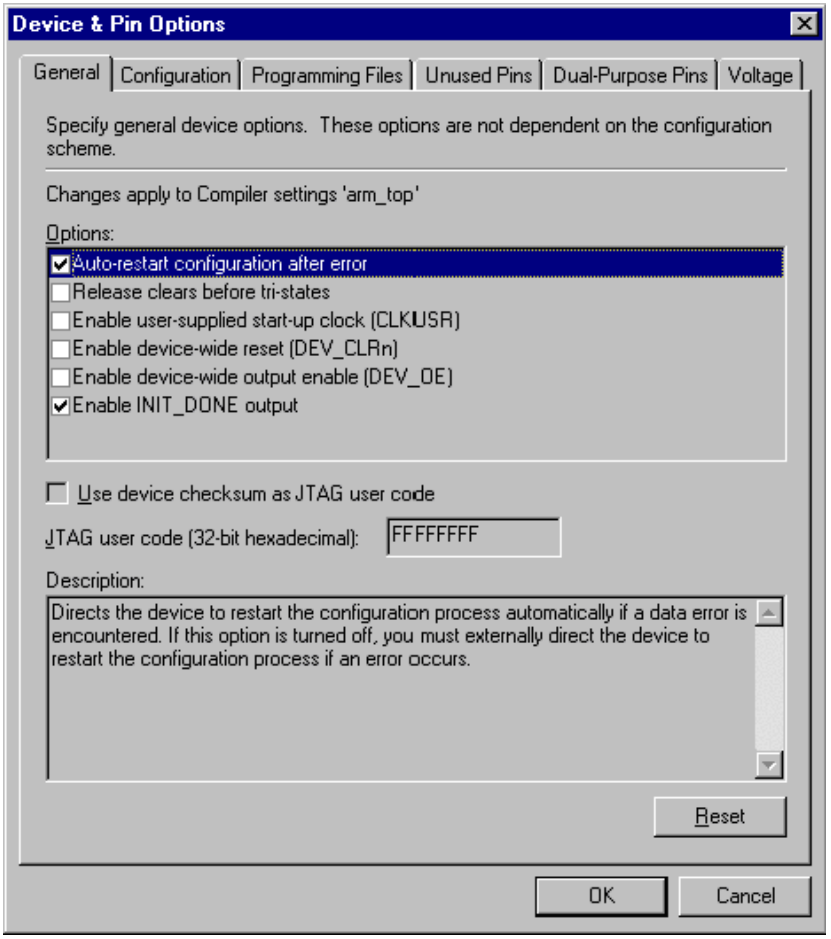
Figure 8. Compiler Settings: Chips & Devices Tab



4. Click **Device & Pin Options**

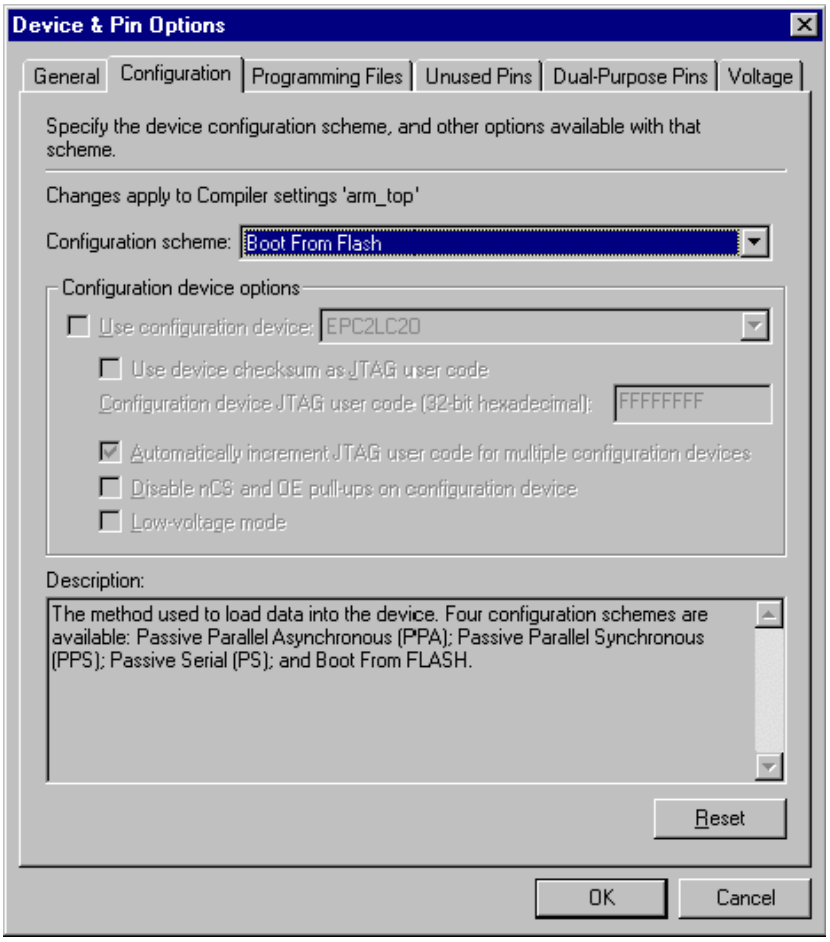
5. Click the **General** tab and specify the following (see Figure 9):
   - **Auto-restart configuration after error**: turn on
   - **Enable INIT_DONE output**: turn on

Figure 9. Device & Pin Options: General Tab



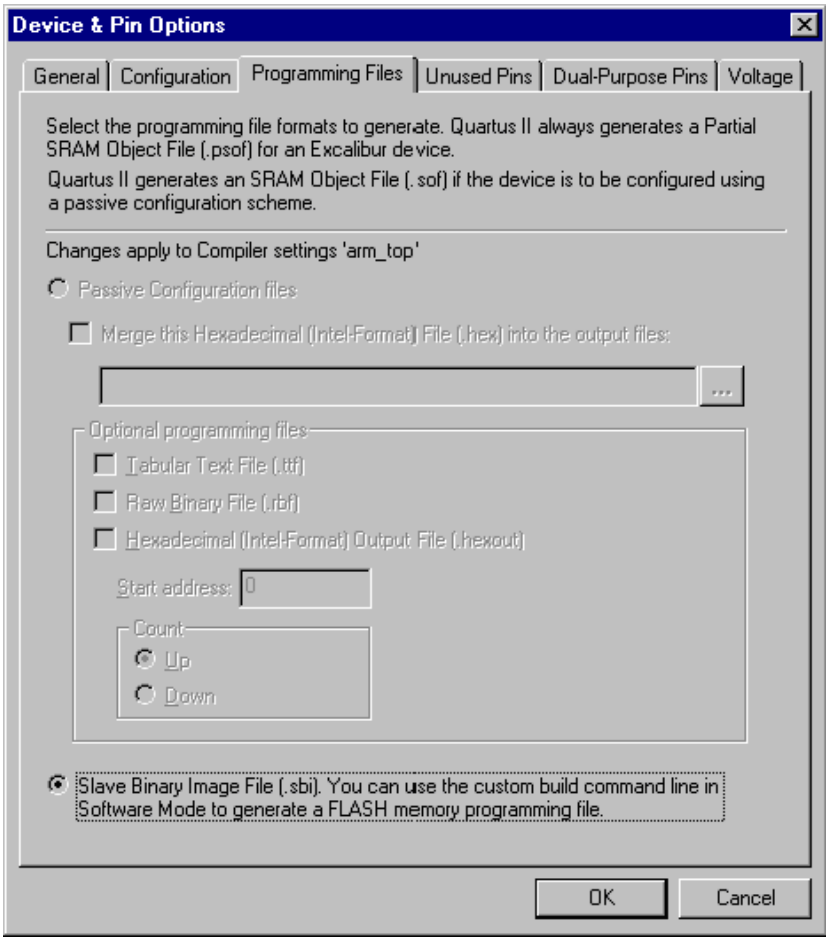
6. Click the **Configuration** tab and choose **Boot From Flash** from the **Configuration scheme** drop-down list (see Figure 10).

Figure 10. Configuration Options: Configuration Tab



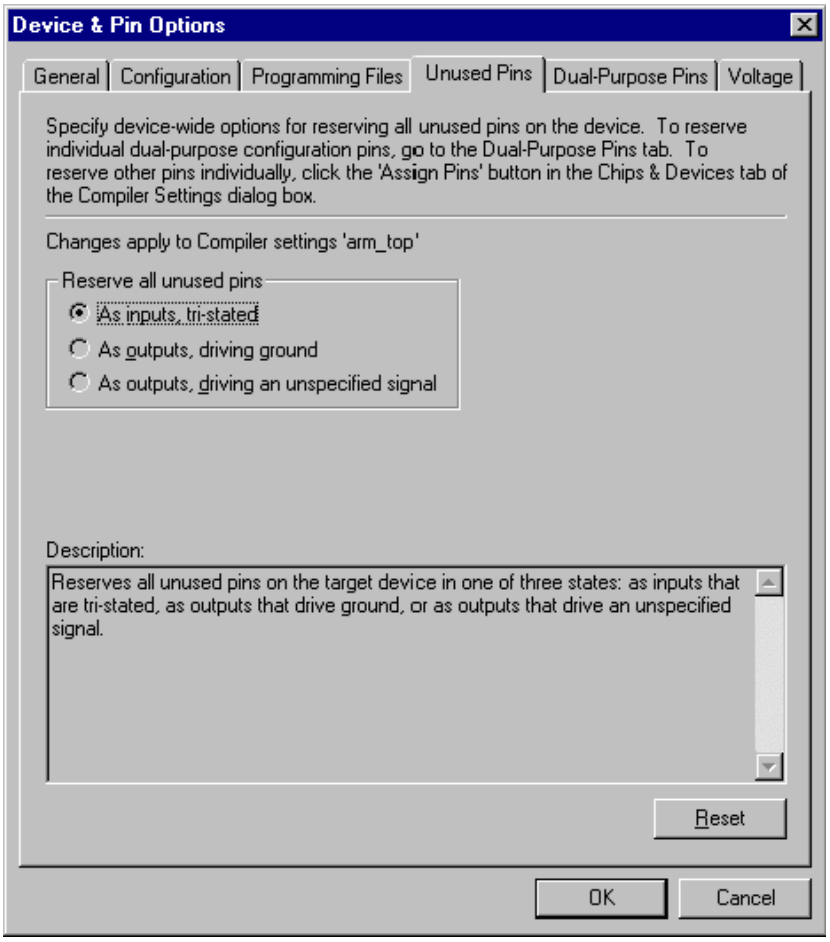
7. Click the **Programming Files** tab and select **Slave Binary Image File** (see Figure 11).

Figure 11. Device & Pin Options: Programming Files



8. Click the **Unused Pins** tab and select **As inputs, tri-stated** under **Reserve all unused pins** (see Figure 12).

Figure 12. Device & Pin Options: Unused Pins



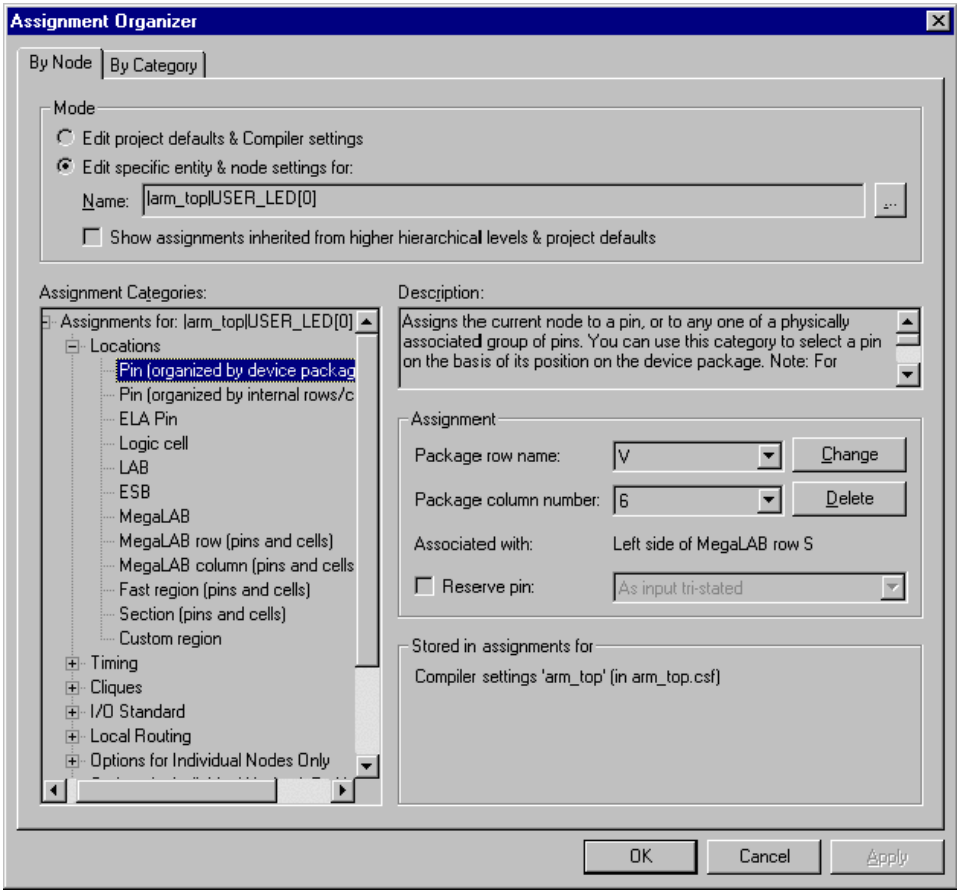
9. Click **OK** twice.

Assigning Nodes to Pins

The following steps assign nodes to pins on the EPXA10 development board:

1. Click on the `USER_LED[0]` output node in **arm_top.bdf** and choose **Assignment Organizer** (Tools menu) (see Figure 13 on page 18).
2. Click the **By Node** tab.
3. In **Assignment Categories**, expand the **Locations** list under **Assignments for**.
4. In the **Locations** list, click on **Pin (organized by device package)** and specify the following under **Assignment**:
   - Package row name: **V**
   - Package column number: **6**
5. Click **OK**.

Figure 13. Assignment Organizer

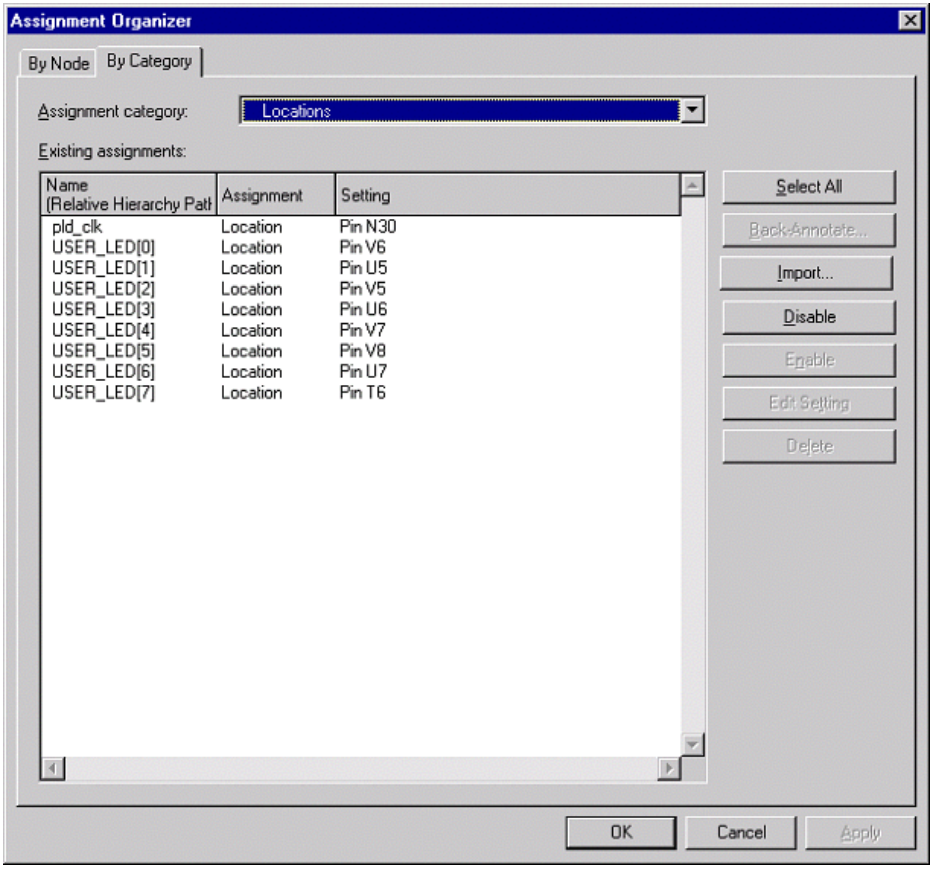


6. Repeat steps 1 to 5 for the following signals:

`USER_LED[1]` - U5
`USER_LED[2]` - V5
`USER_LED[3]` - U6
`USER_LED[4]` - V7
`USER_LED[5]` - V8
`USER_LED[6]` - U7
`USER_LED[7]` - T6
`Pld_clk` - N30

7. Verify that your pin assignments match those shown in Figure 14. To display your pin assignments, in the **Assignment Organizer** window, click the **By Category** tab, and specify **Locations** in the **Assignment Category** drop-down list.

Figure 14. Pin Assignments for pld_clk and User LEDs



8. Click **OK**.

9. Choose **Start Compilation** (Processing menu) to compile the hardware design.

Compiling the Hello World Software Application

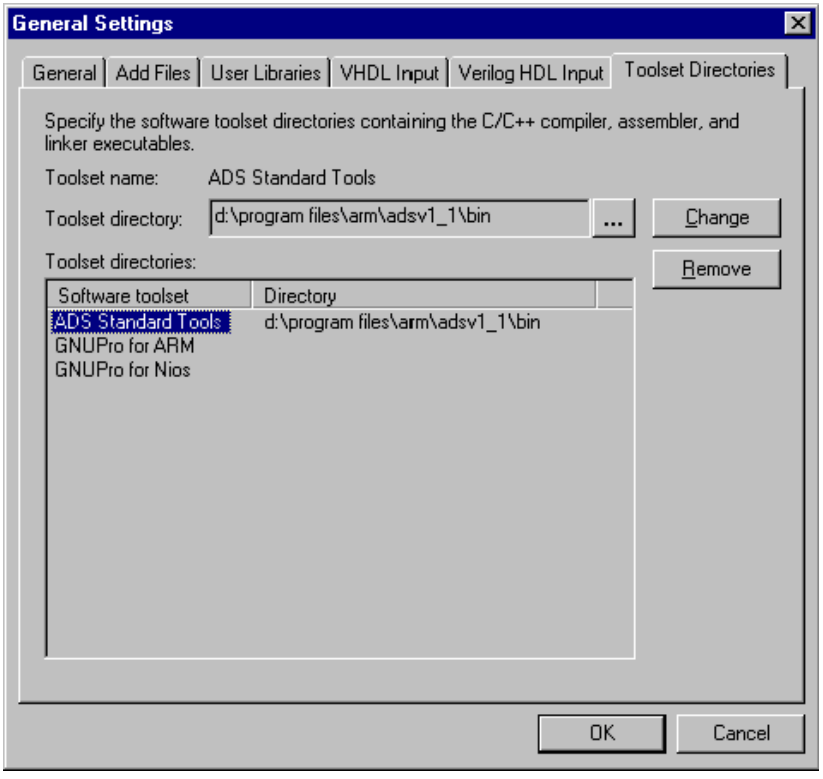
This section describes how to specify the software build settings for the Quartus II development tools to compile the hello world software application. First, you specify the location of the ADS-Lite software. Then you set the various assembler, compiler, assembler, and linker options to produce the embedded software image and the flash memory programming file containing both the hardware and software image.

Specifying the Toolset Directory

The following steps specify the directory for the ADS toolset executables:

1. Choose **General Settings** (Project menu).
2. Click the **Toolset Directories** tab.
3. Click on **ADS Standard Tools** under **Software toolset**, browse to the directory where your ADS toolset is installed and click **Open** (see Figure 15).

Figure 15. General Settings: Toolset Directories Tab



4. Click **OK**.

Specifying the Software Build Settings

For the hello world design you will produce two versions of the software build settings: debug settings and release settings. For the debug settings version you will not apply optimization, but will include debug information to facilitate using the ARM AXD debugger. For the release settings version you will apply optimization, but will not include debug information. The release settings version will only be used after the software code has been debugged and is known to be working. The release settings version of the software build typically produces a software image that is faster than the software image produced with debug settings.

For either version of the software build settings, the following steps add the software files to the project and specify the software options to produce a RAM image:

1. Choose **Software Mode** (Processing menu).

2. Choose **General Settings** (Project menu).

3. Click the **Add Files** tab, browse to the **software** directory, and click **Open**.

4. Control-click to select **armc_startup.s**, **irq.c**, **uartcomm.c**, **hello_world.c**, **retarget.c**, and click **Open**.

5. Click **OK**.

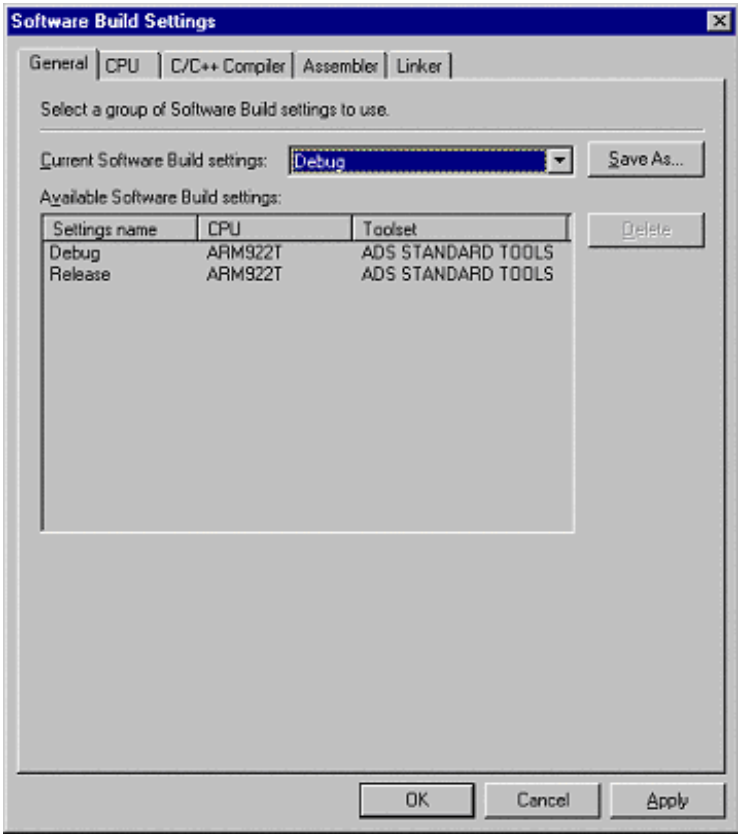
Now you can produce the two versions of the software build settings.

Specifying Debug Settings

Follow the steps below to specify software build settings that generate the hello world software program with extra information to facilitate code tracing and fixing.

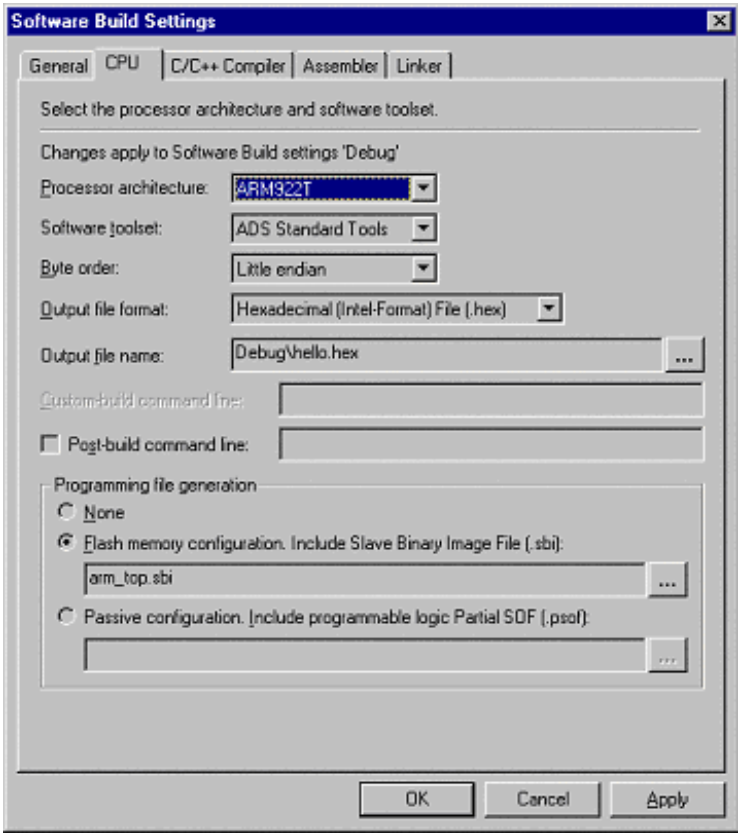
1. Choose **Software Build Settings** (Processing menu).
2. Click the **General** tab.
3. Select **Debug** from the **Current Software Build settings** drop-down list (see Figure 16).

Figure 16. Software Build Settings: General Tab



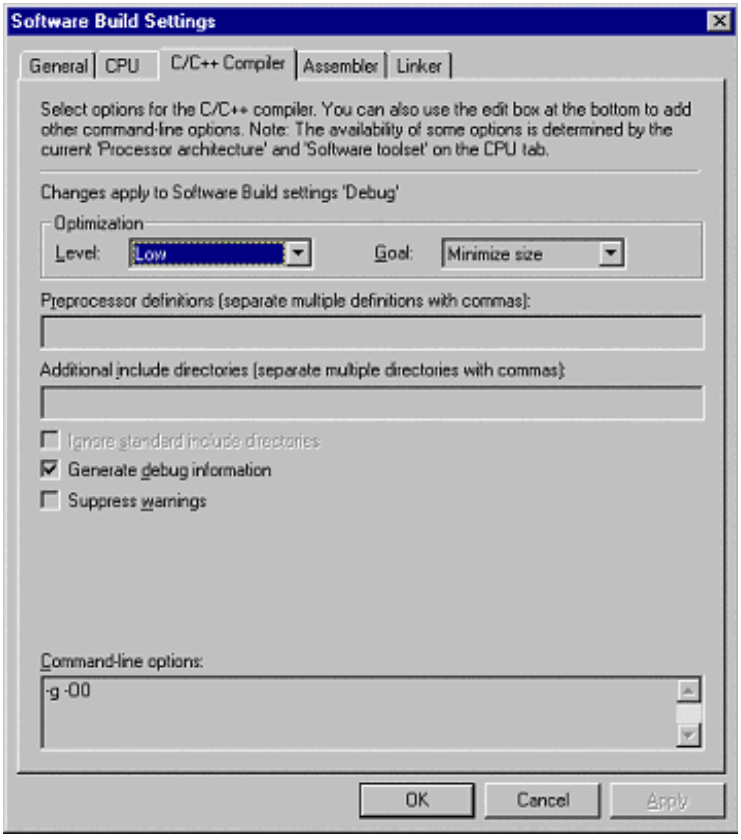
4. Click the **CPU** tab and specify the following (see Figure 17):
   - **Embedded processor architecture**: ARM922T
   - **Software toolset**: ADS Standard Tools
   - **Byte order**: Little endian
   - **Output file format**: Hexadecimal File
   - **Output file name**: Debug\hello.hex
   - **Programming file generation**: select **Flash memory configuration** and browse to **arm_top.sbi**

Figure 17. Software Build Settings: CPU Tab



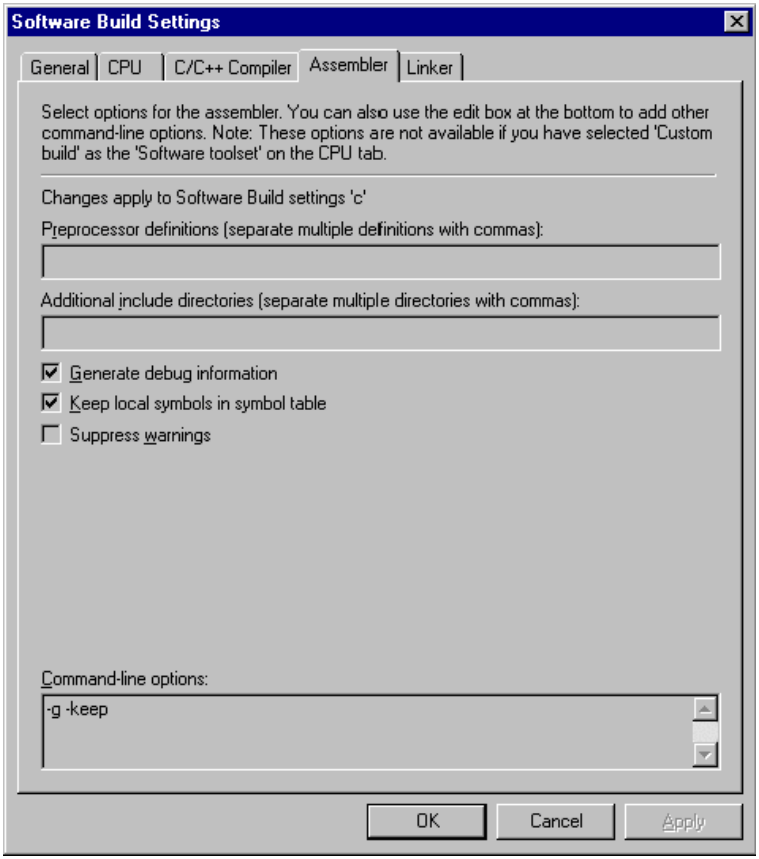
5. Click the **C/C++ Compiler** tab and specify the following (see Figure 18):
   - **Level**: Low
   - **Goal**: Minimize size
   - **Generate debug information**: turn on

Figure 18. Software Build Settings: C/C++ Compiler Tab



6. Click the **Assembler** tab and specify the following (see Figure 19):
   - **Generate debug information**: turn on
   - **Keep local symbols in symbol table**: turn on

Figure 19. Software Build Settings: Assembler Tab

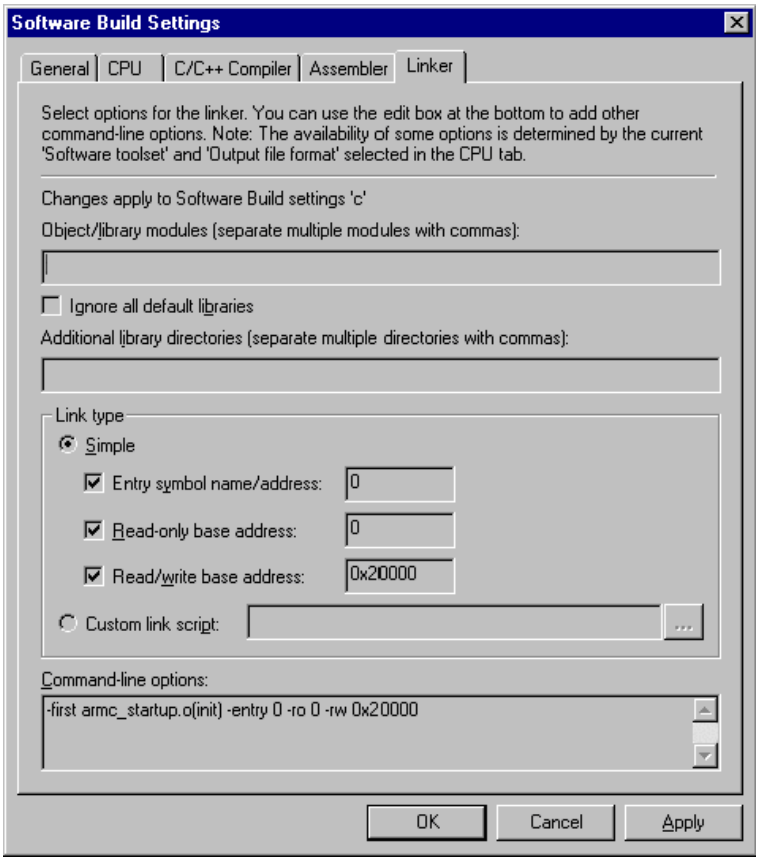


7. Click the **Linker** tab and specify the following (see Figure 20):
   - Under **Link type**: select **Simple**
   - **Entry symbol name/address**: turn on and type: `0x0`
   - **Read-only base address**: turn on and type: `0x0`
   - **Read/write base address**: turn on and type: `0x20000`
   - **Command-line options**: insert at the beginning:

     ```
     -first armc_startup.o(init)
     ```

     The section **init** in the **armc_startup.s** assembly file must be the beginning of the user's software program.

Figure 20. Software Build Settings: Linker Tab



8. Click **Apply**.

9. Click **OK**.

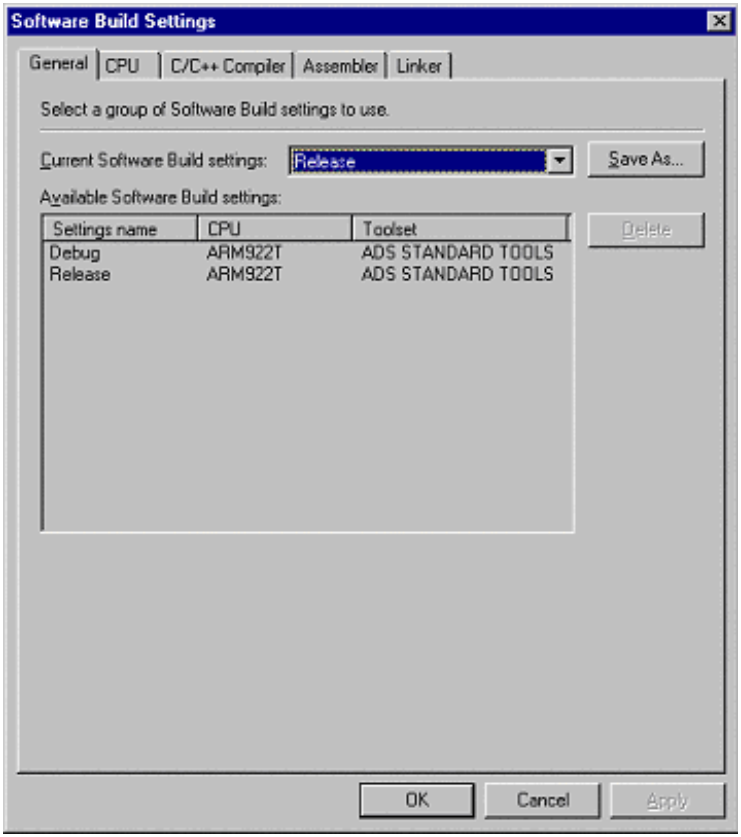
10. Click on **Start Software Build** (Processing menu) to build the software program.

Specifying Release Settings

Follow the steps below to specify software build settings that will generate the hello world software program with optimization for speed.

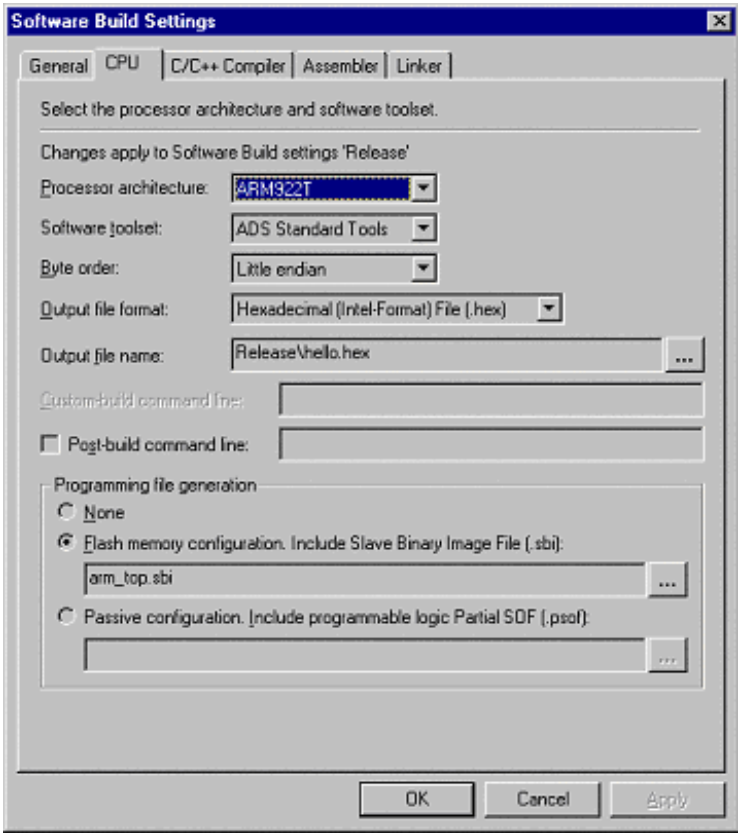
1. Choose **Software Build Settings** (Processing menu).
2. Click the **General** tab.
3. Select **Release** from the **Current Software Build settings** drop-down list (see Figure 21).

Figure 21. Software Build Settings: General Tab



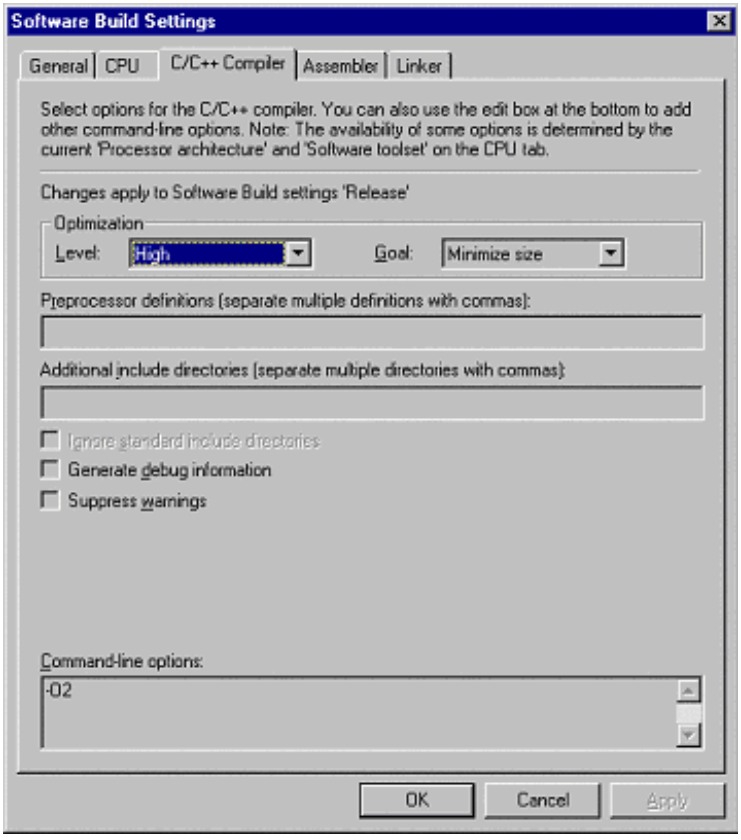
4. Click the **CPU** tab and specify the following (see Figure 22):
   - **Embedded processor architecture**: ARM922T
   - **Software toolset**: ADS Standard Tools
   - **Byte order**: Little endian
   - **Output file format**: Hexadecimal File
   - **Output file name**: Release\hello.hex
   - **Programming file generation**: select **Flash memory configuration** and browse to **arm_top.sbi**

Figure 22. Software Build Settings: CPU Tab



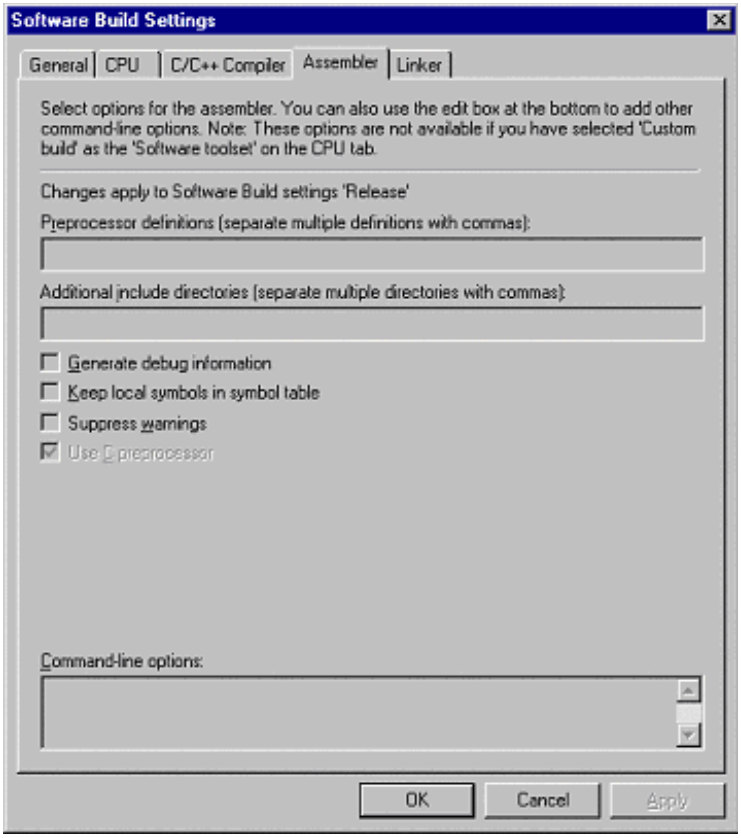
5. Click the **C/C++ Compiler** tab and specify the following (see Figure 23):
   - **Level**: High
   - **Goal**: Minimize size
   - **Generate debug information**: turn off

Figure 23. Software Build Settings: C/C++ Compiler Tab



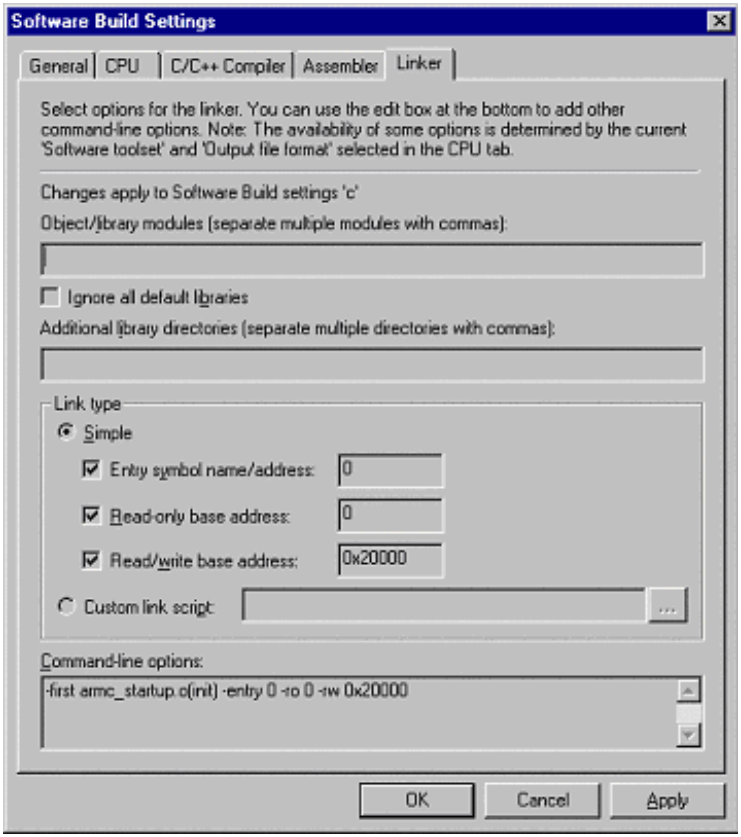
6. Click the **Assembler** tab and specify the following (see Figure 24):
   - **Generate debug information**: turn off
   - **Keep local symbols in symbol table**: turn off

Figure 24. Software Build Settings: Assembler Tab



7. Click the **Linker** tab and specify the following (see Figure 25):
   - Under **Link type**: select **Simple**
   - **Entry symbol name/address**: turn on and type: `0`
   - **Read-only base address**: turn on and type: `0`
   - **Read/write base address**: turn on and type: `0x20000`
   - **Command-line options**: insert at the beginning:
     `-first armc_startup.o(init)`

Figure 25. Software Build Settings: Linker Tab



8. Click **Apply**.

9. Click **OK**.

10. Choose on **Start Software Build** (Processing menu) to build the software program.

Configuring the Development Board Using Boot from Flash

The EPXA10 device can be configured by a variety of different methods. The configuration process involves setting up the embedded stripe registers and the on-chip SRAM, and initializing the PLD array. The **hello.hex** file created in the previous section configures the ARM-based embedded processor PLD by loading the software and PLD images from an external flash device.

This section covers the steps necessary to set up the board connections for configuring the EPXA10 device from flash memory. Altera provides a bootloader utility on the *ARM-Based Excalibur Utilities and Resources* CD-ROM to facilitate booting from external flash memory. The utility performs the following functions:

1. Initializes the device registers and sets up the memory map according to the system build descriptor (.**sbd**) file produced by the MegaWizard Plug-In.
2. Loads the software into the RAM on the device.
3. Resets the watchdog timer and sets the embedded processor endianness.
4. Passes control to the user's code.

Setting up the Development Board

This section describes the hardware connections needed to run the hello world design.

Refer to the section "Jumpers" in the *Excalibur EPXA10 Development Board User Guide version 1.0* for more information.

1. Connect a ByteBlasterMV download cable from your PC to the 10-pin header labeled M/B BLASTER on the EPXA10 development board.
2. Connect one end of a null modem cable to a serial port on your PC and connect the other end to the serial port connector labeled P2 on the EPXA10 development board.
3. Connect an ATX power supply unit to ATX_CONNECTOR on the EPXA10 development board.
4. Ensure that the ATX power supply unit is set to the correct voltage (115 V or 230 V) for the mains supply before plugging it in.
5. Set the jumper settings shown in Table 2 on page 33.

Table 2. Jumper Settings

Jumper	Setting *(1)*	Description
JSELECT	1-2	Uses the ARM processor's PROC_nTRST signal to reset the JTAG controller to use the AXD debugger.
BOOT_FLASH	2-3	Sets the EPXA10 device to boot from flash memory.
DEBUG_EN	2-3	Disables the hardware trigger in the watchdog timer to prevent the EPXA10 device from resetting when using the AXD debugger.
MSEL1, MSEL0	1-2	Boot from 16-bit flash memory.

Note:
(1) Pin 1 for each jumper is indicated on the board.

Refer to the *EPXA10 Development Board User Guide* for the remaining jumper settings.

Configuring the EPXA10 Device with the Flash Image

When you select the boot-from-flash option in the Excalibur MegaWizard Plug-In, the Quartus II software automatically produces a **hello_flash.hex** file when you run a compilation in software mode. This file is used to configure the EPXA10 device from flash memory. The Quartus II software performs the following individual tasks:

- Merges the **hello.hex** software image, **hello.sbd**, and **arm_top.sbi** (PLD slave binary image) files into an object file using the **MakeProgFile** utility.
- Links the above object file with the boot library, **LIBBOOT_XA_ADS.A**, to produce a **hello_flash.elf** file using **Armlink**, which is an ARM utility provided with the ADS toolset on the Altera *ARM-Based Excalibur Utilities and Resources* CD-ROM.
- Creates a **hello_flash.hex** programming file using the **FromElf** utility.

To configure the EPXA10 device with the flash programming file, proceed as follows:

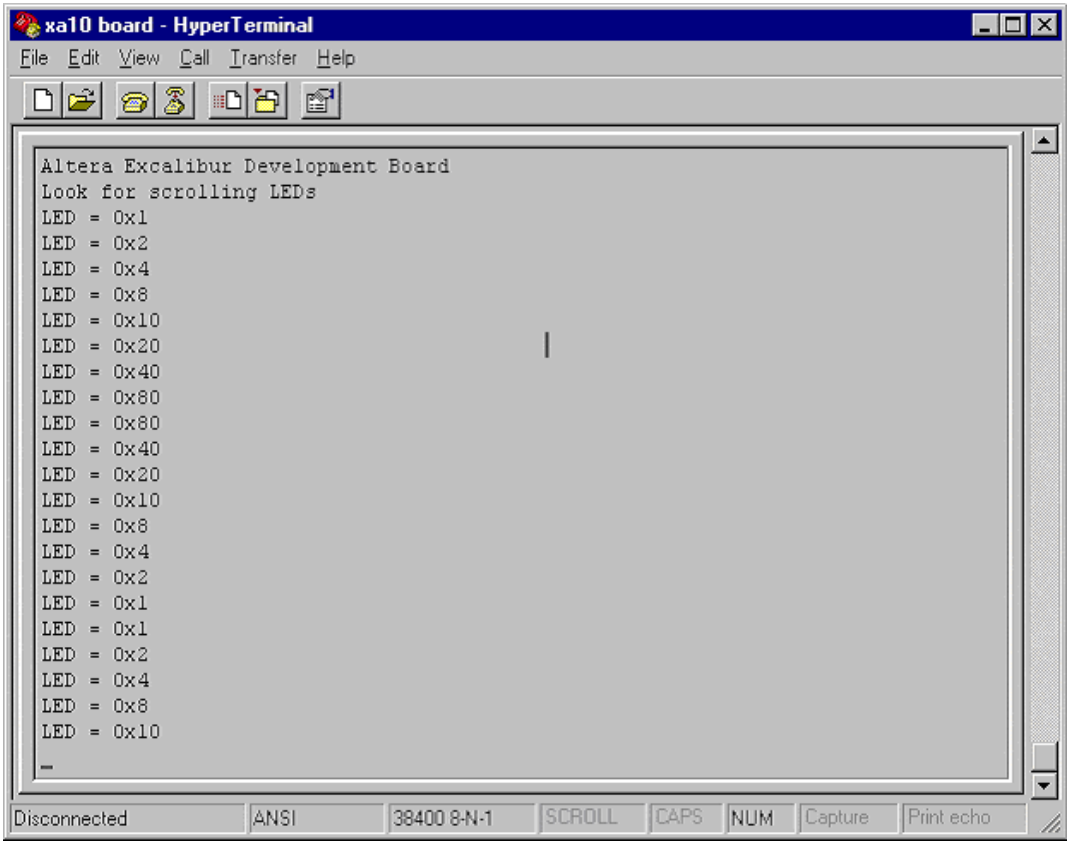
1. Open a Command Prompt window and change to the <*Installation Directory*>**\example_designs\hello_world** directory.

2. Type the command:
   `exc_flash_programmer hello_flash.hex -g`
   This command configures the EPXA10 development board from flash memory.

3. Start up HyperTerminal with the settings: **38400 baud rate**, **8 data bits**, **1 stop bit**, **no parity**, and **no flow-control**. You should see the LEDs light up alternately and the corresponding value displayed on the terminal window.

   Figure 26 shows a typical HyperTerminal window.

Figure 26. HyperTerminal



Launching the AXD Debugger

This section explains how to launch the AXD debugger, download **hello.elf** into the EPXA10 device, and run the application software. Figure 27 on page 36 shows a typical AXD debugger window.

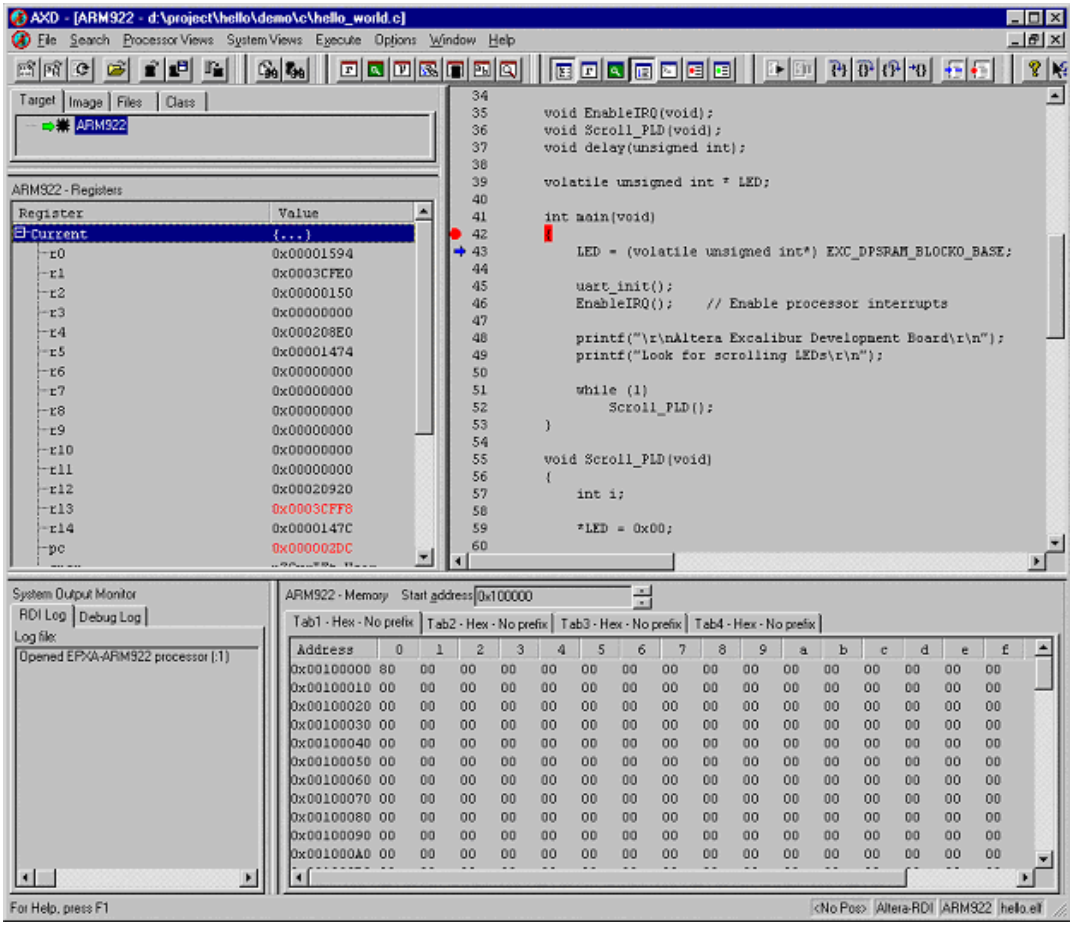
Refer to the *ARM Developer Suite Debuggers Guide* for more information on the debug commands.

To start up the AXD debugger, follow the steps below:

1. Run the Quartus II software, if it is not already running.
2. Click on **Launch Debugger** (Processing menu) in the Quartus II software window. The AXD debugger appears, with the program counter pointing at the first line of **armc_startup.s**.

   If the full version of ADS is used, the AXD could open pre-loaded with **armulator.dll** as the target instead of **Altera-RDI.dll.** See the *EPXA10 Development Board Getting Started User Guide* for instructions on changing the target environment.
3. Click on **Step** (Execute menu) in the AXD debugger to execute the first instruction only.
4. Click on **Go** (**Execute** menu) in the AXD debugger to run the application software continuously until it hits the breakpoint automatically set at the main function.
5. Click on **Registers** (Processor Views menu) to display the processor registers.
6. Click on an instruction and Choose **Toggle Breakpoint** (Execute menu) to set or clear a breakpoint.
7. Click on **Memory** (Processor Views menu) to display the memory contents.
8. After successfully debugging the design using the debug settings version of the software, you can run the release settings version.

Figure 27. AXD Debugger



Appendix A
Hello_world.c

This C code prints user-specified text by means of the UART in the ARM-based embedded processor device, which must be attached to a serial terminal.

Hello_world.c is provided on the Altera *ARM-Based Excalibur Utilities and Resources* CD-ROM.

```
/*
*   C Code for a simple application
*
*       This code prints a user specified message using the embedded uart
*       as stdin / stdout.
*       When combined with the appropriate pld files, this software is
*       programmed into flash using the JTAG and flash interface.
*
*       It tests the UART by outputting information
*       over the UART at a baud rate of 38400 with 8 bits per
*       character no parity and one stop bit, with no flow control.
*       You must connect a serial terminal to the board to control this.
*
*       It tests the address and data lines of the flash memory
*       For this simple application, no logic exists in the PLD.
*
*       Using the associated make_hello.bat file validates the
*       embedded software tool chain for compiling, linking and creating
*       flash downloadable images.
*
*   Copyright (c) Altera Corporation 2001.
*   All rights reserved.
*
*/

#include <stdio.h>
#include "uartcomm.h"
#include "..\stripe.h"

void EnableIRQ(void);
void Scroll_PLD(void);
void delay(unsigned int);

volatile unsigned int * LED;

int main(void)
{
        LED = (volatile unsigned int*) EXC_DPSRAM_BLOCK0_BASE;
```

```
        uart_init();
        EnableIRQ();// Enable processor interrupts

        printf("\r\nAltera Excalibur Development Board\r\n");
        printf("Look for scrolling LEDs\r\n");
        while (1)
            Scroll_PLD();
}
void Scroll_PLD(void)
{
        int i;

        *LED = 0x00;

        for(i = 0; i < 8; i++)
        {
            *LED = 0x01 << i;
            delay(5000000);
            printf("LED = 0x%x\r\n", *LED);
        }

        *LED = 0x00;

        for(i = 0; i < 8; i++)
        {
            *LED = 0x80 >> i;
            delay(5000000);
            printf("LED = 0x%x\r\n", *LED);
        }

        *LED = 0x00;
}
void delay(unsigned int time)
{
        volatile int i;
        for(i = 0; i < time; i++);
}
void CAbtHandler(void)
{
        printf("Data abort\r\n");
}

void CPabtHandler(void)
{
        printf("Error prefetch abort\r\n");
}

void CDabtHandler(void)
{
        printf("Error data abort\r\n");
}

void CSwiHandler(void)
{
        printf("Error swi\r\n");
}
```

```
void CUdefHandler(void)
{
        printf("Error undefined instruction\r\n");
}
}
```

Notes:

Appendix B
armc_startup.s

This assembly program initializes the stack pointers, sets up the interrupt handler, enables the cache, and jumps to the main program.

armc_startup.s is provided on the Altera *ARM-Based Excalibur Utilities and Resources* CD-ROM.

```
;/*****************************************************************************
;*
;*       Code to startup the ARM C environment
;*       =====================================
;*
;*       Copyright (c) Altera Corporation 2000-2001.
;*       All rights reserved.
;*
;*****************************************************************************/

        IMPORT CIrqHandler
        IMPORT CFiqHandler
        IMPORT CPabtHandler
        IMPORT CDabtHandler
        IMPORT CSwiHandler
        IMPORT CUdefHandler

        GET ..\stripe.s

        ; --- Standard definitions of mode bits and interrupt (I & F) flags in PSRs

Mode_USR EQU  0x10
Mode_FIQ EQU  0x11
Mode_IRQ EQU  0x12
Mode_SVC EQU  0x13
Mode_ABT EQU  0x17
Mode_UNDEFEQU0x1B
Mode_SYS EQU  0x1F ; available on ARM Arch 4 and later

I_Bit        EQU      0x80 ; when I bit is set, IRQ is disabled
F_Bit        EQU      0x40 ; when F bit is set, FIQ is disabled

; --- System memory locations
```

```
; We have mapped 256k of SRAM at address 0x100000 I'm using the top of
; this memory as a stack
RAM_LimitEQU EXC_SPSRAM_BLOCK1_BASE + EXC_SPSRAM_BLOCK1_SIZE
SVC_StackEQU RAM_Limit           ; 8K SVC stack at top of memory
IRQ_StackEQU RAM_Limit-8192      ; followed by  1K IRQ stack
ABT_StackEQU IRQ_Stack-1024      ; followed by  1K ABT stack
FIQ_StackEQU ABT_Stack-1024; followed by  1K FIQ stack
UNDEF_StackEQUFIQ_Stack-1024; followed by  1K UNDEF stack
USR_StackEQU UNDEF_Stack-1024    ; followed by  USR stack


;
; If booting from flash the entry point Start is not arrived at immediately after
; reset the quartus project file is doing a few things under your feet that you need
; to be aware of.
;
; The Excalibur Megawizard generated a file (.sbd) which contains the information
; about the setup you requested for your memory map, IO settings, SDRAM settings etc.
;
; This file, along with your hex file and PLD image is converted to an object file,
; and compressed in the process.
;
; This object file is linked with Altera's boot code. Altera's boot code then
; configures excalibur's registers to give the setup you requested via the MegaWizard,
; it uncompresses the PLD image and the hex file and loads them.
;
; So at this point your memory map should be setup and contain the memory you
; initially requested.
;
; For more information on this flow please see the document
; Toolflow for ARM-Based Embedded Processor PLDs
;
        AREA init, CODE, READONLY
        b   Start
        b   UdefHnd
        b   SwiHnd
        b   PabtHnd
        b   DabtHnd
        b   Unexpected
        b   IrqHnd
        b   FiqHnd

Unexpected
        b   Unexpected

UdefHnd
        stmdbsp!,{r0-r12,lr}
```

```
        bl  CUdefHandler
        ldmiasp!,{r0-r12,lr}
        subspc,lr,#4

SwiHnd
        stmdbsp!,{r0-r12,lr}
        ; So far all we're using SWI for is enabling and disabling interrupts
        ; when in User mode
        subs r0, lr, #4
        ldr r0,[r0]
        mov r1, #0xffffff
        and r0,r0,r1
        cmp r0, #1
        beq Enable_IRQ_SWI
        b   EndOfSwi

Enable_IRQ_SWI
        ; Enable interrupts for User Mode
        mrs r1,spsr
        mvn r0,#0xc0
        and r1,r1,r0
        msr spsr_c,r1
        b   EndOfSwi

EndOfSwi
        ldmiasp!,{r0-r12,lr}
        movspc, lr

IrqHnd
        stmdbsp!,{r0-r12,lr}
        bl  CIrqHandler
        ldmiasp!,{r0-r12,lr}
        subspc,lr,#4

PabtHnd
        stmdbsp!,{r0-r12,lr}
        bl  CPabtHandler
        ldmiasp!,{r0-r12,lr}
        subspc,lr,#4

DabtHnd
        stmdbsp!,{r0-r12,lr}
        bl  CDabtHandler
        ldmiasp!,{r0-r12,lr}
        subspc,lr,#4

FiqHnd
```

```
        stmdbsp!,{r0-r7,lr}
        bl  CFiqHandler
        ldmiasp!,{r0-r7,lr}
        subspc,lr,#4

Start
        ;/* Turn on the instruction cache */ ;
        MRC p15,0,r0,c1,c0,0
        LDR r1,=0x80001078
        ORRSr0,r0,r1
        MCR p15,0,r0,c1,c0,0

        ; --- Initialise stack pointer registers
; Enter SVC mode and set up the SVC stack pointer
MSR     CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit ; No interrupts
LDR     SP, =SVC_Stack

; Enter IRQ mode and set up the IRQ stack pointer
MSR     CPSR_c, #Mode_IRQ:OR:I_Bit:OR:F_Bit ; No interrupts
LDR     SP, =IRQ_Stack

; Enter FIQ mode and set up the FIQ stack pointer
MSR     CPSR_c, #Mode_FIQ:OR:I_Bit:OR:F_Bit ; No interrupts
LDR     SP, =FIQ_Stack

; Enter UNDEF mode and set up the UNDEF stack pointer
MSR     CPSR_c, #Mode_UNDEF:OR:I_Bit:OR:F_Bit ; No interrupts
LDR     SP, =UNDEF_Stack

; Enter ABT mode and set up the ABT stack pointer
MSR     CPSR_c, #Mode_ABT:OR:I_Bit:OR:F_Bit ; No interrupts
LDR     SP, =ABT_Stack


; --- Initialise memory system
        ; ...

; --- Initialise critical IO devices
        ; ...

; --- Initialise interrupt system variables here
        ; ...

; --- Now change to User mode and set up User mode stack.
MSR     CPSR_c, #Mode_USR:OR:I_Bit:OR:F_Bit ; No interrupts
LDR     SP, =USR_Stack
```

```
IMPORT   __main

; --- Now enter the C code
B __main ; note use B not BL, because an application will never return this way

;*********************************************************************************
;*           Useful admin functions
;*********************************************************************************/

        EXPORT EnableIRQ

        AREA |C$$Code|,CODE,READONLY

;
;       The ARM C runtime code puts us into User mode, to enable interrupts
;       we need to be in supervisor mode, so use a SWI to do this
;
EnableIRQ
        Swi 1
        mov pc, lr

        END

        AREA |Stacks|, DATA, NOINIT
        %   8192
SvcStackTop
        %   1024
IrqStackTop
        %   1024
FiqStackTop
        %   1024
AbtStackTop
        %   1024
UndefStackTop

        END
```

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com