



EXCALIBUR™

# EPXA10 Development Kit

---

## Getting Started



101 Innovation Drive  
San Jose, CA 95134  
(408) 544-7000  
<http://www.altera.com>

**User Guide**  
**November 2002**



Copyright © 2002 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001

This user guide provides comprehensive information about getting started with the Altera® Excalibur™ EPXA10 development kit.

Table 1 shows the user guide revision history.

<b>Date</b>	<b>Description</b>
November 2002	Updates for Quartus® II version 2.2
August 2002	Changes relating to downloading files
July 2002	Rewrite to integrate the Hello World application and accommodate Quartus II version 2.1, plus Excalibur terminology changes
April 2002	Amendment to acknowledge compatibility with Quartus 2.0; and modify JSELECT text in Table 1.
January 2002	Additional step in “ <a href="#">Configuring the Development Board in Boot-from-Flash Mode</a> ” on page 15. (Now defunct.)
November 2001	Changed details in “ <a href="#">Downloading a Design to Flash Memory</a> ” on page 17. (Now defunct.)
September 2001	Removed sections duplicated in other documents.
August 2001	Minor amendments; new section on installing the Altera ByteBlasterMV™ download cable.
July 2001	Initial release.

## How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click on the binoculars icon in the top toolbar to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

## How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at <http://www.altera.com>.

For additional information about Altera products, consult the sources shown in [Table 2](#).



<b>Table 2. How to Contact Altera</b>			
<b>Information Type</b>	<b>Access</b>	<b>USA &amp; Canada</b>	<b>All Other Locations</b>
Altera Literature Services	Electronic mail	<a href="mailto:lit_req@altera.com">lit_req@altera.com</a> (1)	<a href="mailto:lit_req@altera.com">lit_req@altera.com</a> (1)
Non-technical customer service	Telephone hotline	(800) SOS-EPLD	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-7606	(408) 544-7606
Technical support	Telephone hotline	(800) 800-EPLD (6:00 a.m. to 6:00 p.m. Pacific Time)	(408) 544-7000 (1) (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-6401	(408) 544-6401 (1)
	Web site	<a href="http://www.altera.com/mysupport">http://www.altera.com/mysupport</a>	<a href="http://www.altera.com/mysupport">http://www.altera.com/mysupport</a>
	FTP site	<a href="ftp.altera.com">ftp.altera.com</a>	<a href="ftp.altera.com">ftp.altera.com</a>
General product information	Telephone	(408) 544-7104	(408) 544-7104 (1)
	Web site	<a href="http://www.altera.com">http://www.altera.com</a>	<a href="http://www.altera.com">http://www.altera.com</a>

**Note:**

(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

The *EPXA10 Development Kit Getting Started User Guide* uses the typographic conventions shown in [Table 3](#).

<i>Table 3. Conventions</i>	
Visual Cue	Meaning
<b>Bold Type with Initial Capital Letters</b>	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <b>Save As</b> dialog box.
<b>bold type</b>	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: <b>f<sub>MAX</sub></b> , <b>\QuartusII</b> directory, <b>d:</b> drive, <b>chiptrip.gdf</b> file.
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75 (High-Speed Board Design)</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>t<sub>PIA</sub></i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: < <i>file name</i> >, < <i>project name</i> >.pdf file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of Quartus II. Help topics are shown in quotation marks. Example: “Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable.”
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix <i>_n</i> , e.g., reset <sub>n</sub> .  Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\quartusII\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword SUBDESIGN), as well as logic function names (e.g., TRI) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
v	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.



*Notes:*

---

<b>About this User Guide</b> .....	iii
How to Find Information .....	iii
How to Contact Altera .....	iv
Typographic Conventions .....	v
Introduction .....	9
<b>Getting Started</b> .....	9
Software and Hardware Requirements .....	10
Excalibur Utilities .....	10
Quartus II Software .....	10
ADS-Lite .....	10
GNUPro Toolkit .....	11
EPXA10 Development Board .....	11
Design Overview .....	11
Configuring the Stripe .....	14
Compiling the Hardware Design .....	15
Compiling the Software Application .....	16
Specifying the Toolset Directory .....	16
Software Build Settings for ADS .....	17
Debug Settings for ADS .....	18
Release Settings for ADS .....	19
Software Build Settings for GNUPro .....	21
Debug Settings for GNUPro .....	21
Release Settings for GNUPro .....	24
Configuring the Development Board .....	25
Boot-from-Flash Mode .....	26
Connections and Jumper Settings .....	27
Altera Flash Programmer .....	28
Boot-from-Passive-Serial .....	28
Connections and Jumper Settings .....	29
Quartus II Programmer .....	29
Debugging the Design .....	30
ADS AXD Debugger .....	30
GNUPro Insight Debugger .....	31
<b>Appendix A—</b>	
<b>Installing the ByteBlaster Driver</b> .....	33
Introduction .....	33

Installing the ByteBlaster Driver on a Windows NT System .....33  
Installing the ByteBlaster Driver on a Windows 2000 System .....34  
Confirming the Installation .....35

### Introduction

The EPXA10 development kit includes the following components that you need for developing complete embedded systems on a programmable chip:

- EPXA10 development board with SDRAM DIMM module, featuring the EPXA10 Excalibur device
- Excalibur utilities
- Micro-ATX power supply
- Cables:
  - Two 9-pin female-to-female null modem cables
  - ByteBlasterMV™ cable
- Parallel extension cable
- Documentation



For more details about the EPXA10 device, refer to the *Excalibur Devices Hardware Reference Manual*.

The EPXA10 development kit also contains example reference designs. The designs demonstrate the interfaces and applications possible for system-on-a-programmable-chip (SOPC) designs and serve as platforms for creating new designs.

The design example referenced in this guide is a simple implementation of a Hello World program that can be used as a template for your own applications. The Hello World program is commonly one of the first applications an engineer writes when developing software code for a new processor: successfully building and executing it on the processor validates the tool chain for developing embedded software. For the Excalibur devices, a successful build and execution also validates the hardware configuration of the embedded stripe and the programmable logic.

The Altera version of the Hello World program prints a message to a terminal window and scrolls the LEDs on the EPXA10 development board. The hardware and software source code is provided, which may be useful as the basis for developing additional applications.

## Software and Hardware Requirements

The following list summarizes the hardware and software development tools needed to build and run the example design in the Windows NT/2000 environment:

- The Excalibur utilities
- The Quartus® II software, version 2.2
- Either of the following:
  - Red Hat GNUPro Toolkit for ARM
  - ARM Developer Suite for Altera (ADS-Lite) software version 1.1
- EPXA10 development board

### *Excalibur Utilities*

The Excalibur utilities are applications required to start using the EPXA10 development board. The utilities included allow you to create programming files, set up the programming hardware, and download applications to flash memory. The utilities are installed automatically when you install the Quartus II software. If you do not need to modify the hardware portions of the design and do not require Quartus II, you can install the utilities separately via the standalone installer available on the SOPC Builder CD-ROM.

### *Quartus II Software*

The Quartus II development software provides a comprehensive environment for SOPC design. The Excalibur MegaWizard® Plug-In, included with the Quartus II software, enables quick and intuitive setup and customization of the embedded processor in the EPXA10 device. To create new projects or to modify embedded hardware in an existing EPXA10 project, the Quartus II development software is required.

### *GNUPro Toolkit*

GNUPro Toolkit is a complete solution for C and C++ development for the ARM processors, including a compiler, a debugger, binary utilities, libraries, and other tools. The GNUPro Toolkit is included with the Quartus II software.



Refer to the *GNUPro Toolkit User's Guide for Altera for ARM and ARM/Thumb® Development* for more information.

### *ADS-Lite*

ADS-Lite is a collection of embedded software development tools for Excalibur devices. The tools include a C/C++ compiler, assembler, linker, debugger, and various utilities. To create new embedded software applications or to modify embedded software applications for an existing EPXA10 project, the Altera ADS-Lite software (or another compiler that supports the ARM922T processor) is required. You can purchase ADS-Lite from your Altera representative.

### *EPXA10 Development Board*

The EPXA10 development board is a desktop development system. It provides a hardware platform to begin developing embedded systems. The development board provides a flexible and powerful environment for debugging your own designs.



For more details about the EPXA10 development board, refer to the [\*Excalibur EPXA10 Development Board Hardware Reference Manual\*](#).

## **Design Overview**

The Hello World project includes a simple PLD design and an embedded software application.

The PLD design consists of a memory-mapped slave peripheral that accepts 8-bit data. The data lights up corresponding LEDs on the EPXA10 development board. The embedded software program writes a data pattern to the slave peripheral to scroll the LEDs and also print the data pattern to a terminal window.

In this example, the compiler library functions **printf()** and **scanf()** interface to a terminal window by utilizing the **fputc** (send a character) and **fgetc** (receive a character) functions to use the EPXA10 stripe UART. **Printf()** is used for displaying characters to a terminal window and **scanf()** is used to receive characters from a terminal keyboard.

The Hello World project has the following directory structure:

```
\ads
  \Debug
  \Release
  \software
\gnu
  \Debug
  \Release
  \software
\common
\rtl
```

The following general points apply to the directories:

- The **ads** directory contains the Quartus project files, top-level hardware design, and software startup files that are specific to the ADS tools.
- The **gnu** directory contains the Quartus project files, top-level hardware design, and software startup files that are specific to the GNU tools.
- The **common** directory contains C programs that are used by both the ADS and GNU tools.
- The **rtl** directory contains Verilog HDL files of the slave peripheral that are used by both the ADS and GNU tools.

Table 1 lists the design files in the archive.

<b>Table 1. Hello World Design Files</b>	
<b>File</b>	<b>Description</b>
<ads or gnu>\arm_top.bdf	Block diagram file for top-level design.
<ads or gnu>\hello.csf	Compiler settings file for top-level design that stores chips definitions, device options, compilation type, etc.
<ads or gnu>\hello.psf	Project settings file that stores working directory name, relative hierarchical assignments, device assignments, etc.
<ads or gnu>\hello.quartus	Project configuration file that stores input filenames and compiler settings files.
<ads or gnu>\hello.sbd	System build descriptor file that stores information on interconnections between modules and how they should be configured.
<ads or gnu>\prog_hw.bat	Batch file for programming hardware and software image into the flash of EPXA10 evaluation board.
<ads or gnu>\stripe.v	Verilog HDL instantiation of embedded stripe.
<ads or gnu>\stripe.bsf	Block symbol of embedded stripe.
ads\Debug.fsf	Software build settings file for ADS that stores compiler options with debug information and no code optimization.
ads\Release.fsf	Software build settings file for ADS that stores compiler options without debug information and with code optimization.
gnu\Debug.fsf	Software build settings file for GNU that stores compiler options with debug information and no code optimization.
gnu\Release.fsf	Software build settings file for GNU that stores compiler options without debug information and with code optimization.
ads\software\armc_startup.s	Assembly file for ADS which initializes the stack pointers, sets up the interrupt handlers, enables the instruction and data caches, sets up the MMU, and finally jumps to the main program.
ads\software\retarget.c	C file that implements functions necessary to link with the ARM C libraries.
gnu\software\crt0.s	Assembly file for GNU which initializes the stack pointers, sets up the interrupt handlers, enables the instruction and data caches, sets up the MMU, and finally jumps to the main program.
gnu\software\epxa10.c	C file for GNU which provides the functions necessary to run the C runtime environment.
gnu\software\armelf.x	Linker script file for GNU.
common\main.c	C file that scrolls the LEDs and sends messages to a terminal window.
common\uartcomm.c	C file that implements the UART I/O functions to enable the printf function.
common\irq.c	C file that initializes the interrupt controller, first-level IRQ and FIQ handlers.
common\exceptions.c	C file that handles exceptions.
rtl\single_transaction_slave.v	Verilog HDL file of slave peripheral.
rtl\regfile	Verilog HDL register file.

## Configuring the Stripe

The example design has been pre-built. To become familiar with the process, or to modify it for your own designs, follow the steps below for configuring the stripe:

1. Run the Quartus II software.
2. Choose **Open Project** (File menu) and select the **hello.quartus** project file from either the **ads** or **gnu** directory, depending on the compiler you are using.
3. Choose **Open** (File menu) and select **arm\_top.bdf** to open the block diagram of the top-level design.
4. Double-click on the **stripe** module to invoke the MegaWizard® Plug-In.
5. On page 3 of the wizard, specify the following settings and click **Next**:
  - **Excalibur family**: Excalibur\_ARM
  - **Available device**: EPXA10
  - **Byte order**: Little endian
  - **UART** (under **Reserve pins**): turn on



The EPXA10 UART I/O pins are enabled, because the design sends a message to a terminal window through this peripheral.

- Under **Reset Operation**, do one of the following, depending on your desired boot mode:
  - To boot from flash memory, turn on **Do you want to Boot from flash?**
  - To boot in passive-serial mode, turn on **Do you want the processor to be held in reset after configuration?**
- 6. On page 4 of the wizard, specify the following settings and click **Next**:
  - **Do you want to use the STRIPE-TO-PLD bridge (Master Port)?** turn on



The stripe-to-PLD bridge signals are enabled to allow the processor to access the slave peripheral in the PLD.

- Under **Interrupts**: turn off both options
- Under **Trace/Debug**: turn off both options

7. On page 5 of the wizard, specify the following settings and click **Next**:

- **External clock reference**: 50 MHz
- **Bypass PLL1**: turn off
- **Desired AHB1 frequency**: 166 MHz
- **AHB2 frequency**: 81.25 MHz



The EPXA10 board uses a 50-MHz oscillator for the reference clock. The AHB1 clock setting is for a –2 speed grade of the EPXA10 device.

8. On page 6 of the wizard, specify the following settings and click **Next**:

- **Registers**: 7FFFC000 address, 16K size
- **SRAM0**: 00000000 address, 128K size
- **SRAM1**: 00020000 address, 128K size
- **EBIO (FLASH)**: 40000000 address, 4M size, 8 Wait cycles, Low CS polarity, 16-bit Data Width, 1 Bus clock divide
- **PLD0**: 80000000 address, 16K size



When you enter the EBIO(FLASH) address, the remaining EBIO(FLASH) settings appear.

9. Click **Finish** to create the software header files and the Verilog HDL instantiation of the stripe.

## Compiling the Hardware Design

The example design has already been pre-built. To become familiar with the process, or to modify it for your own designs, follow the steps below for compiling the hardware design:

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Compiler Settings and choose **Device**.
3. Choose the following settings:
  - Family: **Excalibur ARM**
  - Target device: **Specific device selected**
  - Available devices: **EPXA10F1020C2**
4. Click **Device & Pin Options**.

5. Click the **General** tab and choose the following:
  - **Enable INIT\_DONE output:** turn on



The `INIT_DONE` external pin needs to be enabled, because this pin gates `CLK0` (the clock source for the PLD logic in the Hello World design).

6. Click the **Configuration** tab and choose one of the following from the **Configuration scheme** drop-down list, depending on your desired boot mode:
  - **Boot from Flash** to boot from flash memory
  - **Boot from Passive Serial (can use Configuration Device)** to hold the processor in reset after configuring the PLD



Refer to the *Configuration Logic* section of the *Excalibur Devices Hardware Reference Manual* for more information on configuring the PLD.

7. Click the **Unused Pins** tab and select **As inputs, tri-stated** under **Reserve all unused pins**.

This sets all the critical control lines for the interfaces on the EPXA10 board to be pulled to their inactive state.

8. Click **OK** twice.
9. Choose **Start Compilation** (Processing menu) to compile (i.e. synthesize, place-and-route) the hardware design.

## Compiling the Software Application

This section describes how to specify the software build settings for the Quartus II development tools to compile the software application. The location of the software toolset needs to be specified before you can compile your software application.

The example design has the software build settings already set. To become familiar with the process, or to modify it for your own designs, follow the procedure to set the various compiler, assembler, and linker options.



Before you perform a software build using GNUPro on UNIX workstations (e.g., Linux, Solaris, or HP-UX), make three additional variable settings. The variable settings for Solaris 2.5 are:

```
ALTERA_ARM9GP_HOST = H-sparc-sun-solaris2.5
ALTERA_ARM9GP_ROOT = /edatools/solaris/arm_gnupro
ALTERA_ARM9GP_VER = arm9-020528
```

The variable settings assume the path to the installed GNUPro toolset is:

```
/edatools/solaris/arm_gnupro/arm9-020528/H-sparc-sun-solaris2.5/bin
```

## Specifying the Toolset Directory

The following steps specify the directory for the software toolset:

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Files & Directories and choose **Toolset Directories**.
3. Perform one of the following actions, depending on the software toolset you are using:
  - Click on **GNUPro for ARM** under **Software toolset**, and browse to the GNUPro executable directory *<GNUPro installation directory>\bin*.
  - Click on **ADS Standard Tools** under **Software toolset** and browse to the ADS executable directory *<ADS installation directory>\bin*.
4. Click **OK**.

## Software Build Settings for GNUPro

For the example design you will produce two versions of the software build settings for GNUPro: debug settings and release settings. For the debug settings version you will not apply optimization, but include debug information to facilitate using the GNU Insight debugger. You will apply optimization to the release settings version, but debug information is unnecessary because the release settings version will only be used when the software code has been debugged and is known to be working. The release settings version of the software build produces a software image that is faster than the software image produced with debug settings.

The following steps add the software files to the project:

1. Choose **Settings** (Assignments menu).
2. Under **Category**, expand **Files & Directories** and choose **Add/Remove**.
3. Browse to and add the following files:
  - ..\common\main.c
  - ..\common\uartcomm.c
  - ..\common\irq.c
  - ..\common\exceptions.c
  - software\crt0.s
  - software\epxa10.c
4. Click **OK**.

### *Debug Settings for GNUPro*

Follow the steps below to specify software build settings that generate the hello world software program with extra debug information.

1. Choose **Settings** (Assignments menu).
2. Under **Category**, expand **Software Build Settings** and choose **General**.
3. Choose **Debug** from the **Current Software Build Settings** drop-down list.
4. Choose **CPU** from the **Software Build Settings** list and specify the following:
  - **Processor architecture:** ARM922T
  - **Software toolset:** GNUPro for ARM
  - **Byte order:** Little endian
  - **Output file format:** Hexadecimal File
  - **Output file name:** Debug\hello.hex
  - Do one of the following, depending on your desired boot mode:
    - To boot from flash memory, select **Flash memory configuration** under **Programming File Generation** and browse to **hello.sbi**

- For passive-serial boot mode, select **Passive configuration** under **Programming File Generation** and browse to **hello.psof**
5. Choose **C/C++ Compiler** from the Software Build Settings list and specify the following:
    - **Level:** Zero
    - **Preprocessor definitions:** type `DEBUG`
    - **Additional include directories:** type `., .. \common`
    - **Generate debug information:** turn on
  6. Choose **Assembler** from the Software Build Settings list and specify the following:
    - **Additional include directories:** type `.`
    - **Generate debug information:** turn on
    - **Keep local symbols in symbol table:** turn on
    - **Use C preprocessor:** turn on

7. Choose **Linker** from the Software Build Settings list and specify the following:
  - Object/library modules: type  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtbegin.o, ↓`  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtend.o`
  - **Additional library directories:** type  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%, ↓`  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/arm-elf/lib`
  - Under **Link type:** select **Custom link script** and browse to `software\armelf.x`
  - **Command-line options:** add `-lc -lgcc -lc`



The `crtbegin` and `crtend` files set up the environment and clean up for the C run-time libraries.

The environment variables `ALTERA_ARM9GP_ROOT`, `ALTERA_ARM9GP_VER`, and `ALTERA_ARM9GP_HOST` are automatically set by the GNUPro installer.

The linker script file implements the functions required for the standard library calls this program uses.

8. Click **Apply** and **OK**.
9. Click on **Start Software Build** (Processing menu) to build the software application and create a flash programming file.

As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make debug` at a DOS-prompt in the `gnu` directory.

### *Release Settings for GNUPro*

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

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Software Build Settings and choose **General**.
3. Choose **Release** from the **Current Software Build settings** drop-down list.
4. Choose **CPU** tab folder and specify the following:
  - **Processor architecture:** ARM922T
  - **Software toolset:** GNUPro for ARM
  - **Byte order:** Little endian
  - **Output file format:** Hexadecimal File
  - **Output file name:** Release\hello.hex
  
  - Do one of the following, depending on your desired boot mode:
    - To boot from flash memory, select **Flash memory configuration** under **Programming File Generation** and browse to **hello.sbi**
    - For passive-serial boot mode, select **Passive configuration** under **Programming File Generation** and browse to **hello.psof**
5. Choose **C/C++ Compiler** from the Software Build Settings list and specify the following:
  - **Optimization:** High
  - **Additional include directories:** type `., .\common`
  - **Generate debug information:** turn off
6. Choose **Assembler** from the Software Build Settings list and specify the following:
  - **Additional include directories:** type `.`
  - **Generate debug information:** turn off
  - **Keep local symbols in symbol table:** turn off
  - **Use C preprocessor:** turn on

7. Choose **Linker** from the Software Build Settings list and specify the following:
  - **Object/library modules:** type  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtbegin.o, ↓`  
  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtend.o`
  - **Additional library directories:** type  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%, ↓`  
  
`%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/arm-elf/lib`
  - Under **Link type:** select **Custom link script** and browse to **software\armelf.x**
  - **Command-line options:** type `-lc -lgcc -lc`
8. Click **Apply** and **OK**.
9. Click on **Start Software Build** (Processing menu) to build the software application and create a flash programming file.



As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make release` at a DOS-prompt in the **gnu** directory.

## Software Build Settings for ADS

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


The following steps add the software files to the project:

1. Choose **Settings** (Assignments menu).


2. Under Category, expand Files & Directories and choose **Add/Remove**.
3. Browse to and add the following files:
  - ..\common\main.c
  - ..\common\uartcomm.c
  - ..\common\irq.c
  - ..\common\exceptions.c
  - software\armc\_startup.s
  - software\retarget.c
4. Click **OK**.

### *Debug Settings for ADS*


Follow the steps below to specify software build settings that generate the software program with extra debug information.

1. Choose **Settings** (Assignments menu).
  2. Under Category, expand Software Build Settings and choose **General**.
  3. Choose **Debug** from the **Current Software Build Settings** drop-down list.
  4. Choose **CPU** from the Software Build Settings list and specify the following settings:
    - **Processor architecture:** ARM922T
    - **Software toolset:** ADS Standard Tools
    - **Byte order:** Little endian
    - **Output file format:** Hexadecimal File
    - **Output file name:** Debug\hello.hex (hexadecimal file)
    - Do one of the following, depending on your desired boot mode:
      - To boot from flash memory, select **Flash memory configuration** under **Programming File Generation** and browse to **hello.sbi**
-  A **.sbi** file is a slave binary image file.
- For passive-serial boot mode, select **Passive configuration** under **Programming File Generation** and browse to **hello.psof**


 A `.psof` is a partial SRAM object file.

 To generate a programming file, but not include the hardware image, select either **Flash memory configuration** or **Passive configuration** to correspond with your desired boot scheme, but leave the filename blank.

5. Choose **C/C++ Compiler** from the Software Build Settings list and specify the following settings:
  - **Level:** Low
  - **Goal:** Minimize size
  - **Preprocessor definitions:** type `DEBUG`
  - **Additional include directories:** type `., . . . \common`
  - **Generate debug information:** turn on
6. Choose **Assembler** from the Software Build Settings list and specify the following settings:
  - **Additional include directories:** type `.`
  - **Generate debug information:** turn on
  - **Keep local symbols in symbol table:** turn on
7. Choose **Linker** from the Software Build Settings list and specify the following settings:
  - 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:** add  
`-first armc_startup.o(init)`

 This command-line option sets section `init` in `armc_startup.s` to the reset vector address `0`.

8. Click **Apply** and **OK**.
9. Choose **Start Software Build** (Processing menu) to build the software program and create a flash programming file.

 As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make debug` at a DOS prompt in the `ads` directory.

### *Release Settings for ADS*

Follow the steps below to specify software build settings that will generate the **hello world** software program with optimization turned on.

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Software Build Settings and choose **General**.
3. Choose **Release** from the **Current Software Build settings** drop-down list.
4. Choose **CPU** from the Software Build Settings list and specify the following:
  - **Processor architecture:** ARM922T
  - **Software toolset:** ADS Standard Tools
  - **Byte order:** Little endian
  - **Output file format:** Hexadecimal File
  - **Output file name:** Release\hello.hex
  
  - Do one of the following, depending on your desired boot mode:
    - To boot from flash memory, select **Flash memory configuration** under **Programming File Generation** and browse to **hello.sbi**
    - For passive-serial boot mode, select **Passive configuration** under **Programming File Generation** and browse to **hello.psof**
5. Choose the **C/C++ Compiler** from the Software Build Settings list and specify the following:
  - **Optimization:** High
  - **Goal:** Minimize size
  - **Additional include directories:** type `.,..\common`
  - **Generate debug information:** turn off
6. Choose the **Assembler** from the Software Build Settings list and specify the following:
  - **Additional include directories:** type `.`
  - **Generate debug information:** turn off
  - **Keep local symbols in symbol table:** turn off
7. Choose the **Linker** from the Software Build Settings list and specify the following:
  - 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:** add  
`-first armc_startup.o(init)`

8. Click **Apply** and **OK**.
9. Choose **Start Software Build** (Processing menu) to build the software program and create a flash programming file.



As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make release` at a DOS prompt in the **ads** directory.

## Configuring the Development Board

After compiling the hardware design and software application, you can now configure the EPXA10 board with the example design. 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 in order to boot, in addition to initializing the PLD array. The processor boots from address 0H; there are two ways to make code available at this address: boot-from-flash or boot-from-passive-serial.

## Boot-from-Flash Mode

Setting the `BOOT_FLASH` external pin to high puts the device in boot-from-flash mode upon power up. In this mode, the processor accesses the bootcode from flash memory connected to EBIO, where the bottom 32 Kbytes are mapped from address 0H. The stripe registers are mapped to their default addresses, based at 7FFFC000H and the remaining peripherals are not mapped.

Altera provides a bootcode program with the Excalibur utilities to facilitate booting from external flash memory. The bootcode 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 RAM whether it is on- or off-chip memory.
3. Resets the watchdog timer and sets the embedded processor's endianness.
4. Loads the PLD configuration data into the device.
5. Passes control to the user's code.

The `hello_flash.hex` flash programming file created in [“Compiling the Software Application” on page 16](#) is used to configure the embedded processor PLD by loading the software and hardware images from an external flash device. The software image is the `hello.hex` (Hexadecimal) file created in [“Compiling the Software Application” on page 16](#) and the hardware image is the `hello.sbi` created in the [“Compiling the Hardware Design” on page 15](#).

### Connections and Jumper Settings

To set up the EPXA10 development board in boot-from-flash mode, perform the following steps:

1. Connect one end of the 25-pin parallel cable to the parallel port of the computer and the other end to the ByteBlasterMV download cable.
2. Connect the ByteBlasterMV download cable to the 10-pin JTAG header on the edge of the development board.



If you have not previously used a ByteBlasterMV download cable with your PC, you must install the ByteBlaster driver before continuing. See [“Appendix A—Installing the ByteBlaster Driver”](#) on page 33.

3. Set the jumpers according to [Table 2](#) below.

Jumper	Setting	Description
BOOT_FLASH	2-3	Sets the EPXA10 device to boot from a 16-bit flash memory
MSEL0, MSEL1	1-2	Configuration mode is serial JTAG
JSELECT	1-2	Connects both the PLD TAP controller and the processor TAP controller to the ByteBlaster header
DEBUG_EN	2-3	Puts the embedded processor into DEBUG mode, which prevents the watchdog timer from affecting the operation of the processor
JP40	1-2	Selects the 32.768-MHz external oscillator for PLD CLK0
JP54	2-3	Enables PLD CLK0
JP51	2-3	Enable the reference clock for the EPXA10 stripe



Refer to the section “Jumpers” in the [Excalibur EPXA10 Development Board Hardware Reference Manual](#) for more information.

4. Ensure that the voltage setting on the ATX power supply is appropriate to your AC power outlet supply. The voltage switch is located near the power receptacle on the ATX power supply.
5. Connect the ATX power supply to the ATX power connector on the development board and switch on the power.



On power up, the LEDs near the ATX connector illuminate, which indicates that the board is fully powered.

### *Altera Flash Programmer*

Altera provides a flash programming utility, which initially explores the development board, providing a listing of the setup and the JTAG configuration chain. It then downloads a small application into the EPXA10 device and probes the flash device to determine the required programming algorithm. After programming completes, the message `Flash programmed successfully` appears in the command prompt window. The EPXA10 device resets and boots from the data programmed into the flash memory.

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



Debugger software that interfaces with the EPXA10 PLD via JTAG must not be running when downloading a design into flash memory.

1. Open a Command Prompt window and change to the **hello\ads** or **hello\gnu** directory.
2. Run the batch file **prog\_hw.bat**.
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 scrolling and the corresponding value displayed on the terminal window.

### **Boot-from-Passive-Serial**

Setting the `BOOT_FLASH` external pin to low selects passive-serial mode: the embedded processor is held in reset while the PLD and stripe are configured. The configuration logic master configures the memory map, the appropriate stripe memory, and the PLD. The configuration bitstream contains register writes to enable writable memory, such as on-chip SRAM, at address 0, and to write code into it. The register writes are followed by PLD configuration data. When the data transfer is complete, the bitstream writes to the boot control register to release the embedded processor from reset.

### Connections and Jumper Settings

To set up the EPXA10 development board in boot-from-passive-serial mode, follow steps 1 to 5 in “Connections and Jumper Settings” on page 28, but set the jumpers as shown in Table 3.

<b>Jumper</b>	<b>Setting</b>	<b>Description</b>
BOOT_FLASH	1-2	Sets the EPXA10 device to boot-from-serial mode
MSEL0, MSEL1	1-2	Configuration mode is serial JTAG
JSELECT	1-2	Connects both the PLD TAP controller and the processor TAP controller to the ByteBlaster header
DEBUG_EN	2-3	Puts the embedded processor into debug mode, which prevents the watchdog timer from affecting the operation of the processor
JP40	1-2	Selects the 32.768-MHz external oscillator for PLD CLK0
JP54	2-3	Enables PLD CLK0
JP51	2-3	Enables the reference clock for the EPXA10 stripe

### Quartus II Programmer

Follow the steps below to configure the device using the Quartus II programmer:

1. Choose **Open Programmer** (Processing menu) in the Quartus II software.
2. Click **Setup** and choose **ByteBlasterMV** from the list under **Hardware Type**. Click **Close**.
3. Choose **JTAG** from the **Mode** drop-down.
4. Click **Auto Detect** to scan the JTAG chain. You should see EPXA10F1020 and EPXA-ARM922 listed under the **Device** column.
5. Right-click on **EPXA10F1020** under **Device**; choose **Change File**, and specify **hello.sof**.
6. Turn on the **Program/Configure** check box.
7. Click **Start** to configure the EPXA10 device with the specified programming file.

## Debugging the Design

After configuring the the EPXA10 board with the example design, you can now debug the design. The procedure involves setting up the ADS or GNU debugger to interface with the EPXA10 development board, and downloading the debug symbols of your software application. Follow the instructions below for the appropriate ADS or GNU debugger.

### ADS AXD Debugger

The ADS tools includes the AXD debugger to pause the embedded processor and observe the detailed operation of the design.



For more information on the AXD debugger, refer to the *ARM Developer Suite - Debuggers Guide*.

The Excalibur utilities include the RDI (**Altera\_RDI.dll**). The RDI is the standard application interface between ARM processors and ARM-supported debuggers. If your debugger supports the RDI from ARM and the ARM922T processor, you can use your existing debugger with an EPXA10 device, using a ByteBlaster™ cable.

To set up AXD and run the application software, perform the following steps.

1. Start AXD by selecting **Start > Programs > ARM Developer Suite > AXD Debugger** (Start menu).
2. Select **Configure Target** (Options menu) in the AXD debugger window.
3. If Altera-RDI is listed as a target, click it to highlight it, then click **OK** to connect to the embedded processor. If Altera-RDI is not listed as a target, you must add it, by performing the following steps:
  - a. Click **Add** in the Choose Target window.
  - b. Browse to the directory in which the Quartus II software is installed.
  - c. Navigate to the `<Quartus installation directory>\bin` directory.
  - d. Select **Altera-RDI.dll**. Click **Open**.
  - e. Click on Altera-RDI in the Choose Target window; click **OK** to connect to the embedded processor.
4. Click on **Load Debug Symbols** (File menu) in the AXD debugger window and browse to `ads\debug\hello.elf`. Click **Open**.

5. Click on **Step** (Execute menu) in the AXD debugger window to execute an assembly instruction.
6. Click on **Registers** (Processor Views menu) to display the processor registers.
7. Right-click on an instruction's line number and choose **Toggle Breakpoint** to set or clear a breakpoint.
8. Click on **Memory** (Processor Views menu) and type an address to display the memory contents.
9. Click on **Go** (Execute menu) in the AXD debugger window to run the application software.

After successfully debugging the design using the debug settings version of the software, you can run the release settings version.

## GNUPro Insight Debugger

The GNUPro Toolkit from Red Hat incorporates the Insight graphical debugger. The debugger interfaces with the GNU debugger stub provided by Altera in the Quartus II software, version 2.1 or later. The Altera-supplied stub runs on a host PC and debugs the code running on the Excalibur device using the JTAG debug module.



Refer to the GNUPro Insight debugger manual for more information on the debug commands.

The following steps explain how to set up the GNUPro Insight debugger and run the application software:

1. In a Command Prompt window, start up the GNU debugger stub by typing:  

```
<Quartus installation directory>\bin\gdbstub.␣
```
2. Execute `<GNUPro installation directory>\bin\arm-elf-gdb.exe` to start Insight.
3. Click on **Open** (File menu) in the Insight debugger and browse to `gnu\debug\hello.elf`.

4. Choose **Connect To Target** (Run menu) in the Insight Debugger window and specify the following in the dialog box:
  - Target: **Remote/TCP**
  - Port: **9999**
  - Ensure that **Download Program** under **More Options** is not selected.
5. Set a breakpoint by right-clicking on the line number and selecting **Set Breakpoint**.
6. Click on **Run** (Run menu) in the Insight debugger to run the application software.
7. Click on **Registers** (View menu) to display the processor registers.
8. Right-click on an instruction's line number and choose **Set Breakpoint** to set a breakpoint.
9. Click on **Memory** (View menu) to display the memory contents.

After successfully debugging the design using the debug settings version of the software, you can run the release settings version.



## Introduction

If you are using either Windows NT 4.0 or Windows 2000, the Altera ByteBlaster driver must be installed before you can configure the EPXA10 development board using the flash programmer. This section explains how to install the ByteBlaster driver from the **qutilities** directory, either for Windows NT 4.0 or Windows 2000, depending on which operating system you are using.

## Installing the ByteBlaster Driver on a Windows NT System

Follow the steps below to install the Altera ByteBlaster driver on a Windows NT 4.0 system:

1. Open the control panel.
2. Select **Multimedia**.
3. Select the **Devices** tab.
4. Select **Add**.
5. Select **Unlisted or Updated Driver** from the **List of Drivers** list box and choose **OK**.
6. If the Quartus II software is installed, type or browse to \*path to Quartus II*\drivers in the text box and choose **OK**.



If the Quartus II software is not installed, but the **qutilities** package is installed, type or browse to \*path to qutilities*\drivers in the text box and choose **OK**.



If neither the Quartus II software nor the **qutilities** package is installed, **utilities** must be installed before installing the ByteBlaster driver.

7. Choose **OK** again in the **Install Driver** window.
8. Select **Altera ByteBlaster** in the **Add Unlisted or Updated Driver** window and choose **OK**.
9. Restart the PC.

## Installing the ByteBlaster Driver on a Windows 2000 System

Follow the steps below to install the Altera ByteBlaster driver on a Windows 2000 system:

1. Open the control panel.
2. Select **Add/Remove Hardware** to start the Add/Remove Hardware Wizard and click **Next** to continue.
3. In the Choose a Hardware Task panel, select **Add/Troubleshoot a device** and click **Next** to continue. Windows 2000 searches for new plug and play hardware (**New Hardware Detection** window).
4. In the Choose a Hardware Device window, select **Add a new device** and click **Next** to continue.
5. In the Find New Hardware window, select **No, I want to select the hardware from a list** and click **Next** to continue.
6. In the Hardware Type window, select **Sound, video and game controllers** and click **Next** to continue.
7. In the Select a Device Driver window, select **Have Disk ...**
8. If the Quartus II software is installed, type or Browse to `\<path to Quartus II>\drivers\win2000` and click **OK**.



If the Quartus II software is not installed, but the **qutilities** package is installed, type or Browse to `\<path to qutilities>\drivers\win2000` and click **OK**.



If neither the Quartus II software nor the **qutilities** package is installed, **qutilities** must be installed before installing the ByteBlaster driver.

9. In the Digital Signature Not Found warning dialog box, click **Yes** to continue the installation.
10. In the Select a Device Driver window, select the hardware to install and click **Next** to continue.
11. The Start Hardware Installation window displays the hardware being installed. Click **Next** to continue.
12. In the Digital Signature Not Found warning dialog box, click **Yes** to continue the installation.

13. In the Completing the Add/Remove Hardware Wizard window, click **Finish**. A system dialog appears prompting a reboot so that the new settings can take effect.

## Confirming the Installation

After the system is rebooted, follow the steps below to ensure the ByteBlaster driver is accessible:

1. Connect the ByteBlasterMV cable to the PC's parallel port LPT1.
2. Open a command prompt window.
3. Type `jtagconfig ↵` at the command prompt.
4. If the PC cannot connect to the ByteBlasterMV cable, the following message appears:

```
Unable to lock chain (Hardware not attached)
```

5. Type `jtagconfig --add ByteBlaster LPT1 ↵`.

You can now use the ByteBlasterMV cable to configure the EPXA10 development board.



*Notes:*