Building a Working Design Example Using the Triscend A7 Evaluation Board

February 2001 **Version 1.0**

Introduction

The A7 tutorial describes how to define, create, implement, and debug a working design example using the Triscend A7 Evaluation Board. Feel free to read through this tutorial just to understand how FastChip operates. For maximum benefit, however, actually perform each step on your own computer and evaluation board.

Before Getting Started

Before starting this tutorial, make sure that you have the following items. Please refer to the ***Getting Stared with A7*** guide for information on all the items included with the Triscend A7 Starter Kit.

- FastChip 2.1.2 properly installed on your computer
- VisionCLICK properly installed on your computer
- A VisionPROBE II JTAG download/debugging cable connected between your computer's parallel port and the evaluation board
- The Diab 'C' compiler properly installed on your computer
- The Cygwin utilities properly copied to your computer
- A Triscend A7 Evaluation Board with the VisionPROBE II cable and all power supplies connected.

Scope

This tutorial introduces you to the basic operations and the design flow of the Triscend FastChip development system. Think of this tutorial as the mighty Rio Grande River; it covers a great deal of territory but not in great depth and sometimes meanders a bit. The tutorial guides you through a complete sample project using the Triscend A7 Configurable System on Chip (CSoC) device.

There are two parts to the tutorial. Part 1 describes how to use FastChip and provides a basic introduction to hardware design with CSoC devices. Part 1 also covers how to debug a CSoC design using the FastChip Device Link (FDL) utility. No software coding is required to complete Part 1.

Part 2 covers writing and compiling code for the A7's embedded ARM7TDMI RISC processor and debugging the software application using the visionCLICK debugger. Part 2 uses the A7 CSoC design created in Part 1. Software developers, however, may choose to skip Part 1 and instead use the ***MyDesignA7*** project provided on the FastChip CD-ROM.

In PART 1 of this tutorial you will learn how to …

- Invoke FastChip CSoC development system and create a new project.
- Locate and add soft modules from the FastChip library.
- Connect signals between the soft modules.
- Configure the A7's Memory Interface Unit (MIU).
- Assign I/O pins.

- Generate header and source files for the sample 'C' application. No actual coding is required to complete Part 1. A compiled example application program is provided.
- Bind your CSoC design to the hardware resources on the A7 CSoC device, which creates a CSL configuration file.
- Use the FastChip Device Link (FDL) software to combine the CSL configuration file with the compiled application code, creating a CSoC device configuration image.
- Download the CSoC configuration image to the Triscend A7 Evaluation Board via the visionPROBE JTAG download/debug cable.
- Use FDL to perform real-time debugging of your project.
- Import a design from a third-party logic design package such as schematic capture or logic synthesis.
- Bind the final design and download to Flash.

In PART 2 of this tutorial you will learn how to …

- Add some functionality to the sample 'C' application program.
- Use the Diab compiler to compile your program.
- Use the visionCLICK source-level debugger to control and monitor your application program.

In PART 3 of this tutorial you will learn how to …

- Use the FastChip command line interface to download a new CSoC device configuration image to the A7 Evaluation Board.
- Create batch files and makefiles to automate your project

Design Overview

The admittedly simple tutorial design, shown in Figure 1, introduces a number of FastChip concepts without confusing the process with an overly complex design example. When complete, the design operates on the Triscend A7 Evaluation Board.

Figure 2 shows the program flow for the example design. When the A7's dedicated Watchdog Timer times out, it generates an interrupt. The interrupt is forwarded to the ARM7TDMI processor via the interrupt controller. The processor handles the interrupt and increments a value in a register called RESULT. Don't worry, the compiled processor code is provided in Part 1 of the tutorial. Coding for the embedded ARM7TDMI processor is saved for Part 2.

The RESULT register is actually implemented in the Configurable System Logic (CSL). The processor communicates to the RESULT register via the dedicated Configurable System Interconnect (CSI) bus. In the tutorial, the RESULT register is built using a *soft module* function called a Command Register. Soft modules are pre-defined, pre-verified logic core or functions, which are provided free-of-charge with FastChip.m

In order to view the contents of the RESULT register on the A7 Evaluation Board, the RESULT register connects to a 7-segment display decoder called D1. The 7-segment display decoder is another FastChip soft module. The four-bit output from the RESULT register is displayed as a hexadecimal character by the 7-segment display decoder. The display decoder contains seven output pins that will eventually connect directly to the 7-segment LED on the evaluation board.

This example focuses on FastChip features and implementing logic functions in the Configurable System Logic (CSL) portion of the A7 Configurable System-on-Chip (CSoC) device. The Dedicated Resources are ignored until later examples.

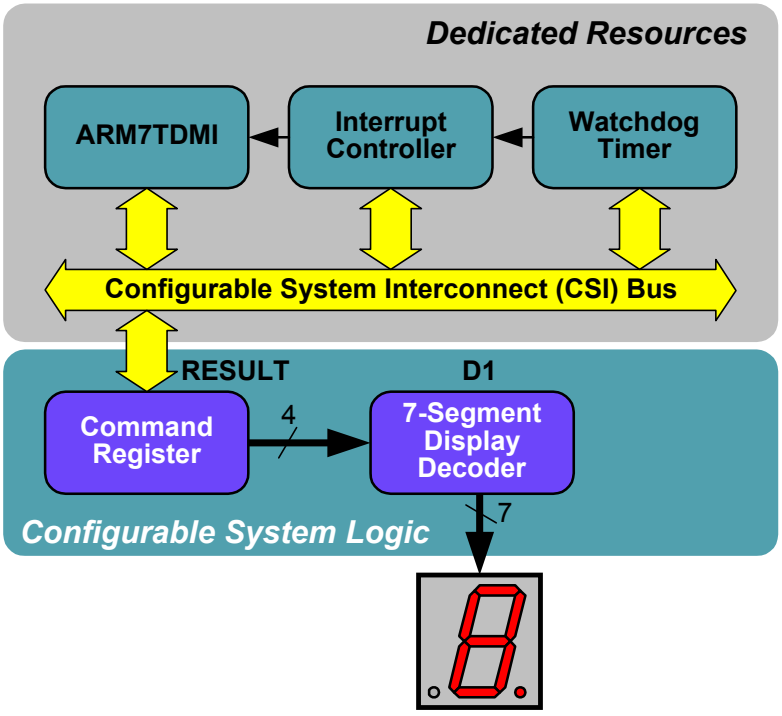


Figure 1. A block diagram of the A7 tutorial design.

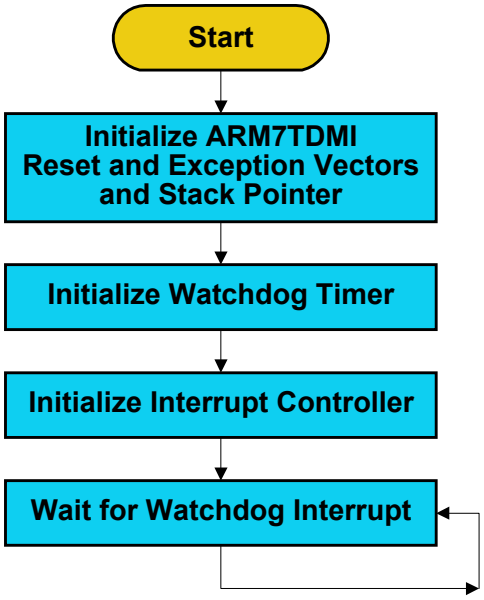


Figure 2. Tutorial Program Flow Chart

Part 1: Design with FastChip Graphical Environment

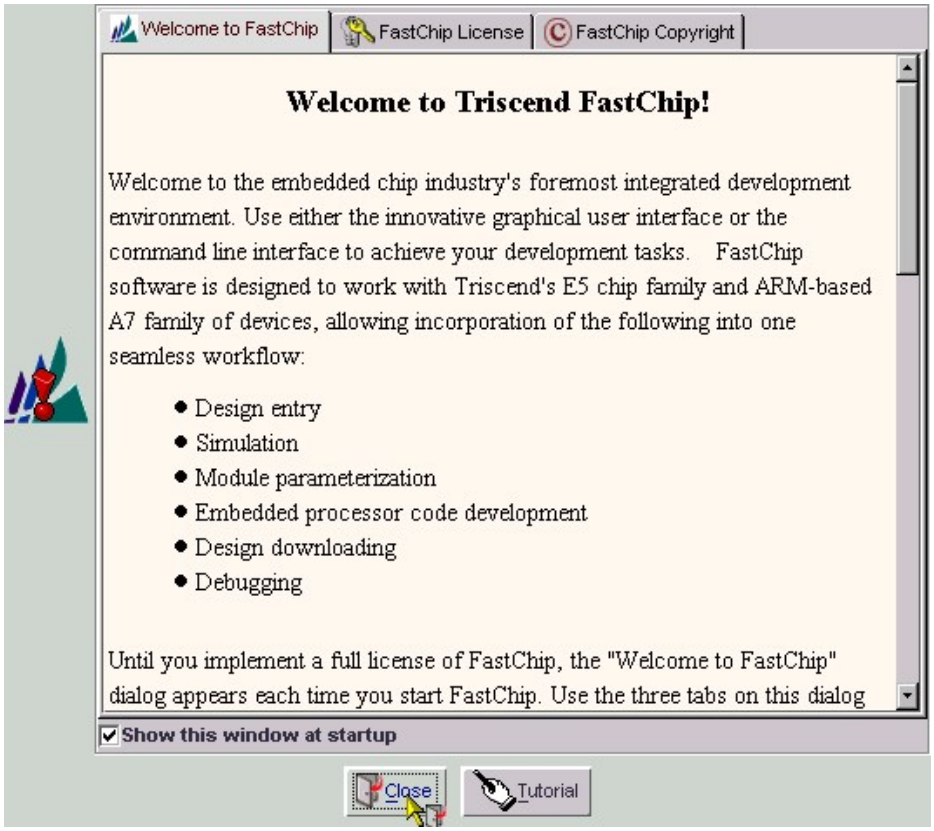
This tutorial assumes that the FastChip software, the visionClick software, and visionProbe hardware are already installed on your computer. If not, please refer to the **Getting Started with A7** guide.

Invoke FastChip

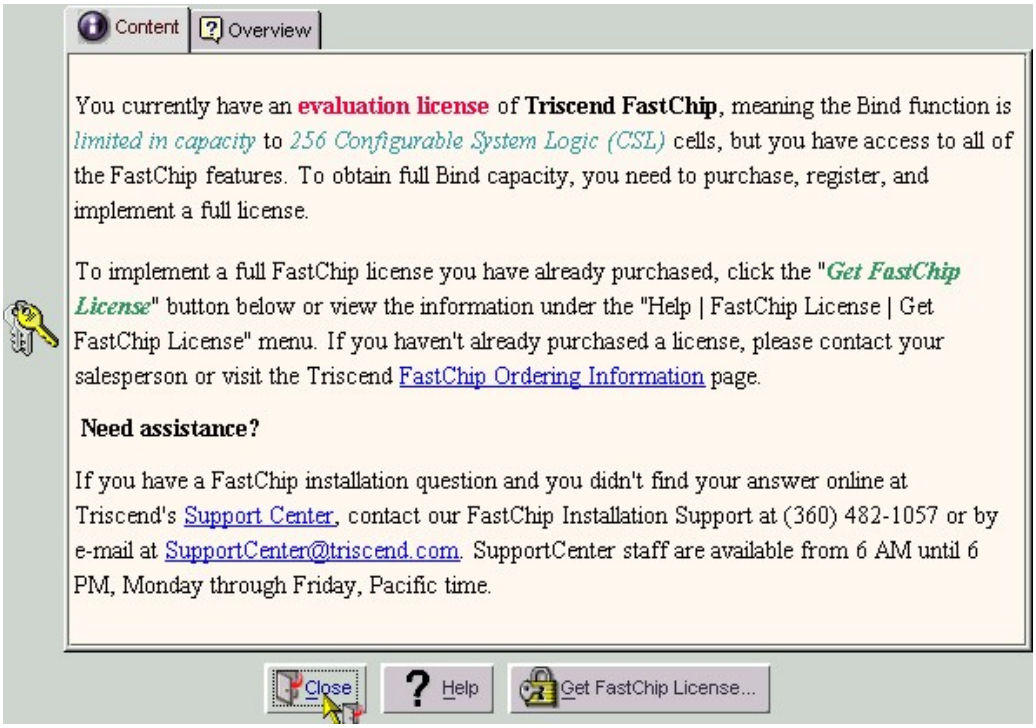
Invoke the FastChip software by double clicking the FastChip desktop icon or by choosing **Start → Programs → Triscend FastChip → Triscend FastChip 2.1.2**.

FastChip is a Java-based application and it may take a few moments for your computer to load the Java virtual machine. FastChip displays a welcome splash screen as it loads.

After the splash screen, FastChip displays a welcome dialog box describing FastChip and its capabilities. If you wish to hide this dialog box in the future, uncheck the **Show this window …** option at the bottom of the dialog box. After reading the messages in this dialog box, click **Close** to continue.

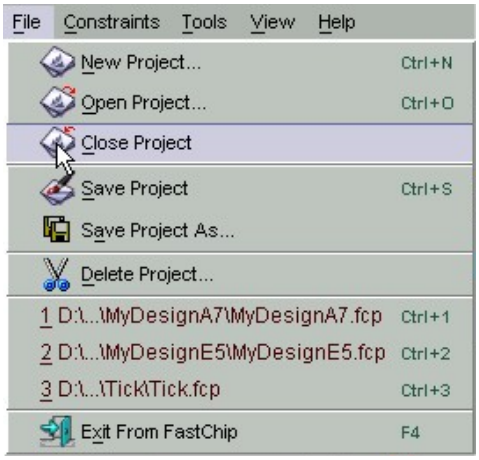


Unless you have already licensed your version of FastChip, an evaluation license window appears describing the capabilities of the evaluation—or unlicensed—version of FastChip. The evaluation version has all the same capabilities of the full, licensed version of FastChip except that it is limited to designs containing 256 or less Configurable System Logic (CSL) cells. The tutorial design functions on both the full and the evaluation versions. After reading the messages in this dialog box, click **Close** to continue.



By default, FastChip opens the last used FastChip project when you start. Before starting a new project, close the existing project by clicking **File → Close Project**.

You can change the start-up behavior of FastChip by selecting **Tools → FastChip Options** change the **Default Project Opening** settings in the **Miscellaneous Setting** panel.



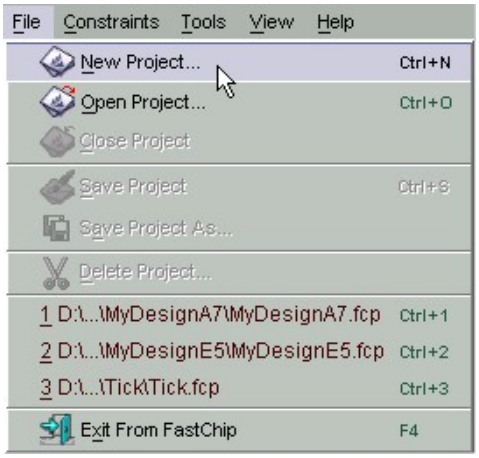
Start a New Project

You can create FastChip projects in any existing directory on your machine. By default, FastChip uses **`<FastChip Install directory>\Projects`** directory, which is the recommended location. Using the default installation settings, the project directory will be in the following location.

`<drive>:\Program Files\Triscend\FastChip\Projects`

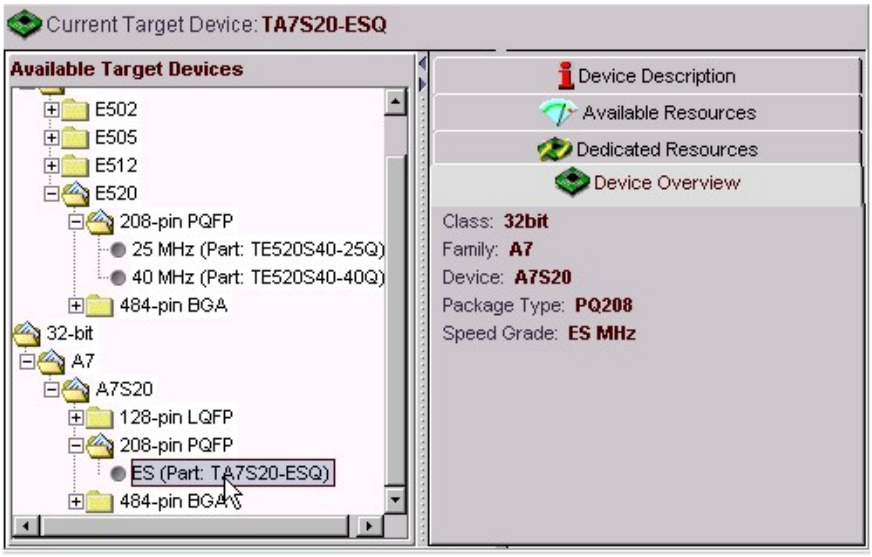
Three example projects are installed in this directory. One of the examples, MyDesignA**7**, is similar to the tutorial project you are going to create. The design you are about to create will use the compiled application program from the MyDesignA7 project.

To create a new project, select **File → New Project** from the menu bar.



Enter “MyFirstA7” as the design name in the **Project Name** field. The project will be created in the default location unless you change the file settings under Project Location. Create this project in the default location.

The default device selected by FastChip is the TE520S40-40Q. For this tutorial, select the TA7S20-ESQ from the **Available Target Devices** menu. First, click the **32-bit** folder to expand the tree below. Likewise, click **A7** to reveal the devices available in the A7 product family. Select **A7S20 → 208-pin PQFP → ES** to choose the TA7S20-ESQ device available on the A7 Evaluation Board.



The tabs on the right side of the dialog box show various characteristics of the selected device. Click the **Available Resources** tab to display the design resources inside an A7S20. The A7S20 has 2,048 Configurable System Logic (CSL) cells, 128 address Selectors, up to 154 user-defined Programmable Input/Ouput (PIO) pins, and six global buffers (GBUFs). Click **OK** to continue.

FastChip Main Window

After creating a new project and selecting the target device, FastChip displays the main window.

The figure above shows how FastChip should appear on your screen by default. If it does not appear like this figure, select **View → CSoC**.

The FastChip main window contains the following areas:

Menu Bar: Displays the various commands supported by FastChip

Toolbar: Contains shortcut icons for the most commonly used tools to build your project

Dedicated Resources: Contains icons representing the dedicated resources on the A7 CSoC, including the various peripherals. These functions are always present in a FastChip project. Click on an icon for a short summary of the features available within each peripheral.

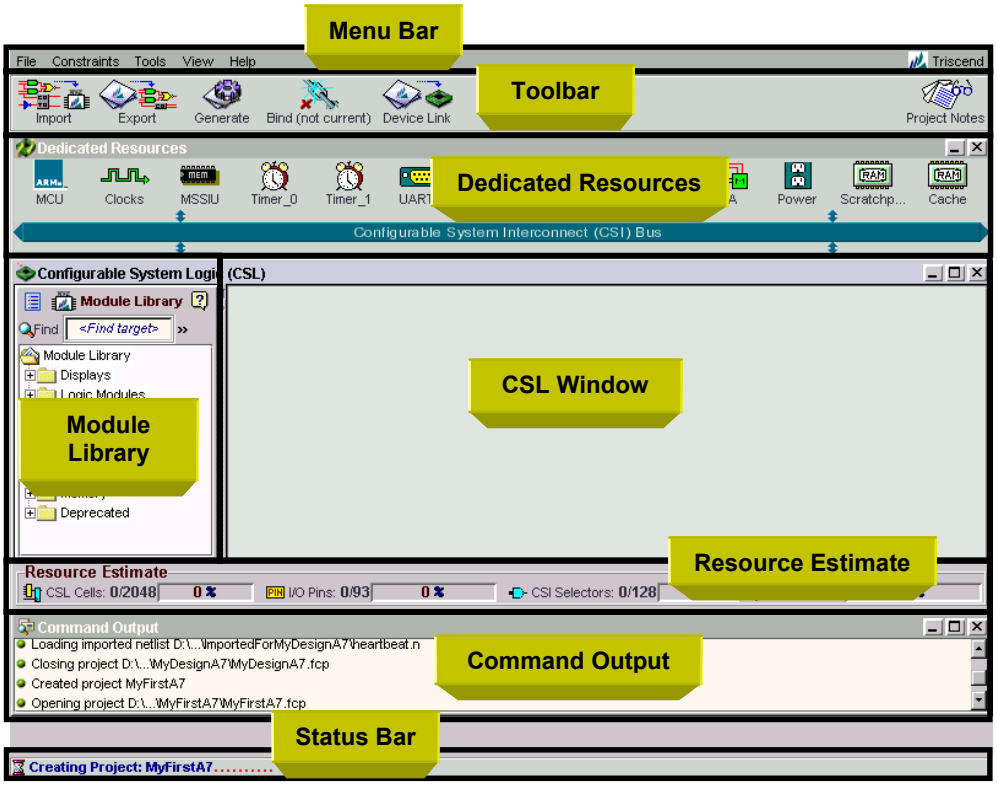
Module Library: Lists the Triscend soft modules function provided with FastChip, as well as any modules you created by importing logic designs from schematic capture or logic synthesis.

CSL Window: Contains the modules you selected and configured. These modules will be implemented using the Configurable System Logic (CSL) resources on the CSoC.

Resource Estimate: Displays the estimated resource requirements for the modules instantiated in CSL window. The actual number of resources required is determined by executing Bind and detailed in the project report.

Command Output: Shows FastChip's response to commands entered using the graphical interface.

Status Bar: Shows the status of the current operation in progress, if any.



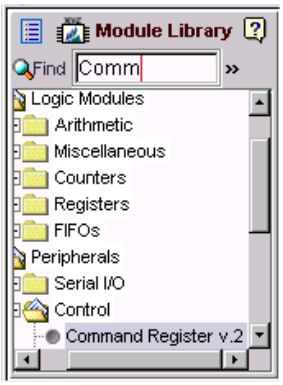
Selecting and Configuring Soft Modules

This design example uses two soft modules, selected directly from the Triscend soft module library.

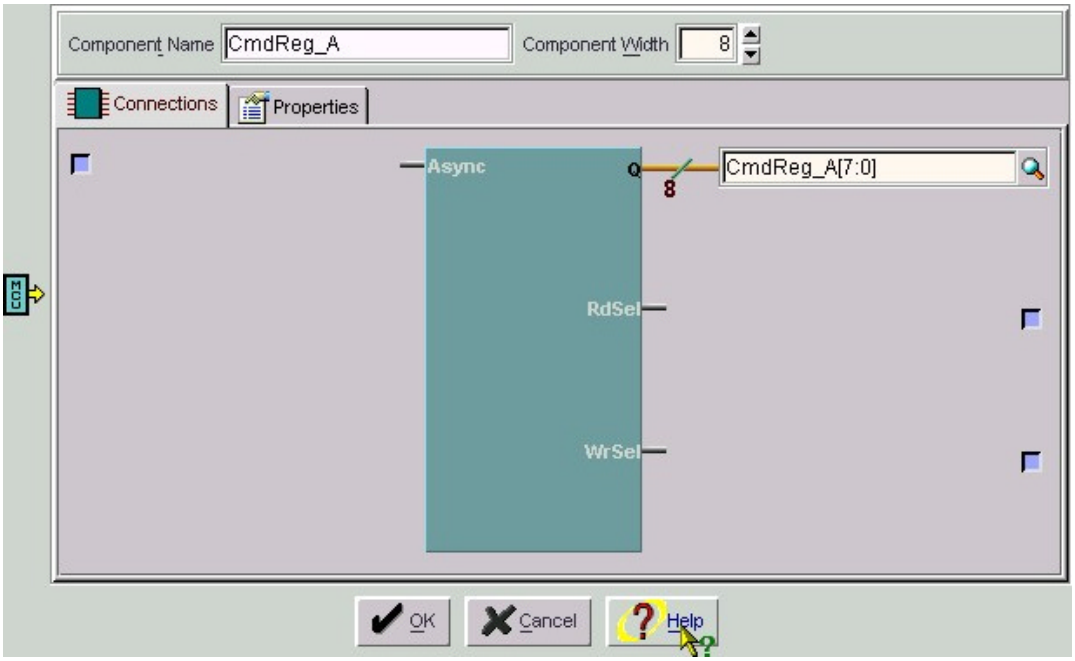
Command Register

A command register is a read/write register connected to the Configurable System Interconnect (CSI) bus. It is memory mapped, such that the processor and other bus masters can write to it or read from it. It is found in the Module Library tree under **Peripherals → Control**.

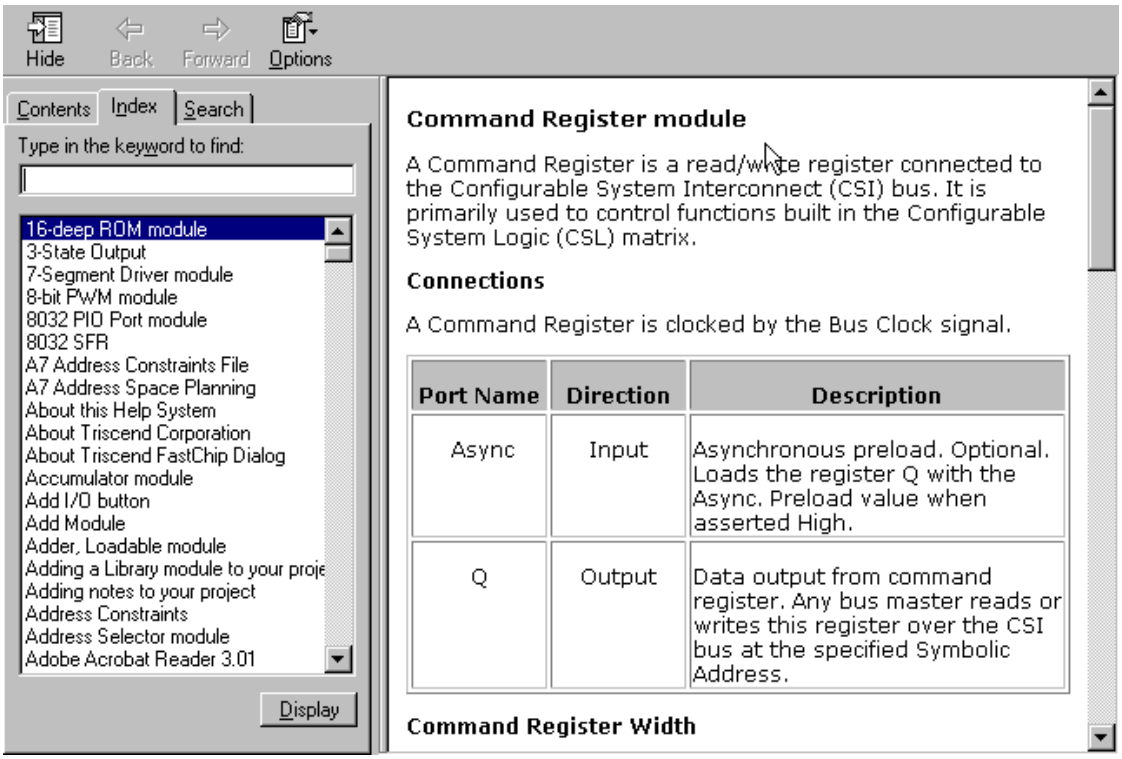
Alternatively, type "**Comm**" in the Module library text box, which highlights the Command Register module.



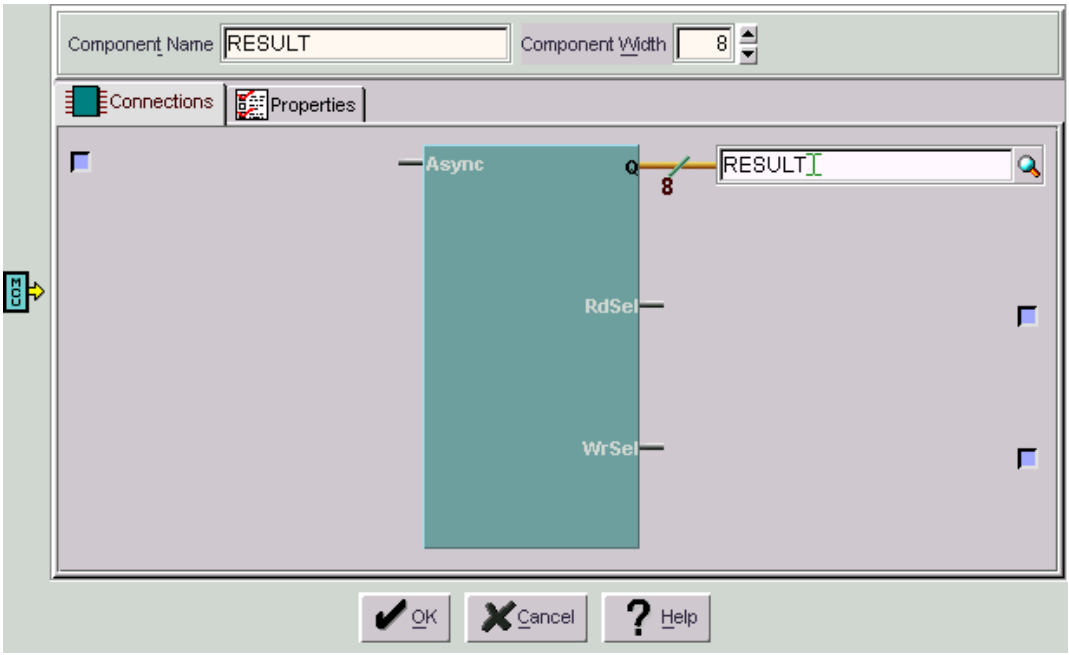
This example project uses a command register to control the 7-segment LED driver, which in turn connects to the 7-segment LED on the A7 evaluation board. Double-click the **Command Register** module or optionally, drag and drop it into the CSL window. If you drag and drop the module, click the associated icon in the CSL window. This action opens the Command Register edit dialog box.



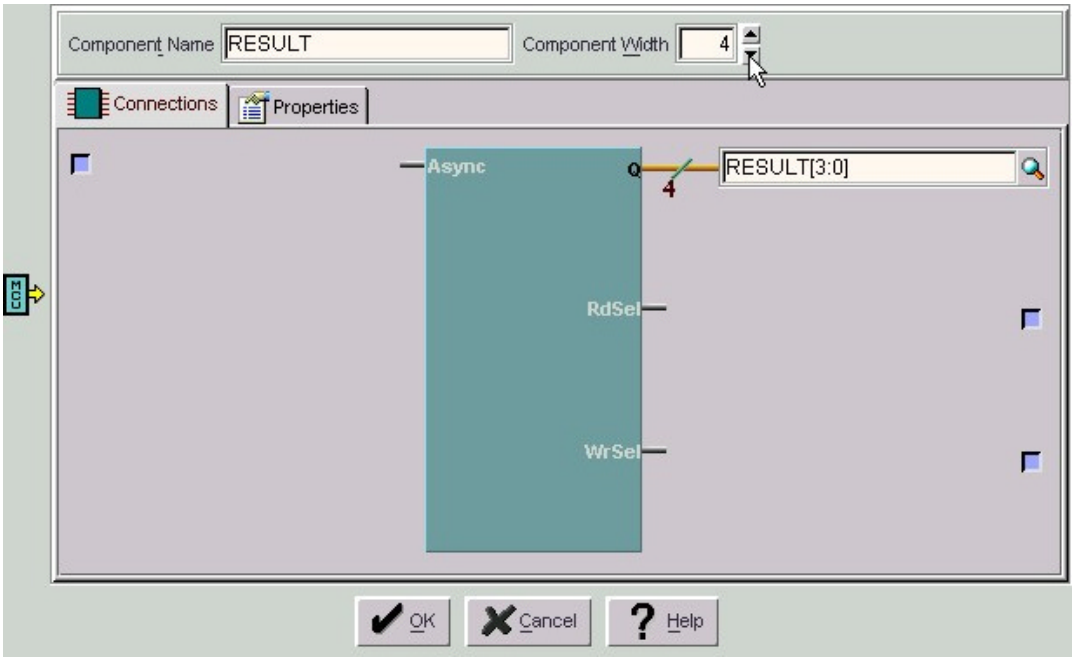
To view more information on how the Command Register module operates, click the **Help** button at the bottom of the dialog box. Doing so invokes the context-sensitive online help.



Rename the module by typing "**RESULT**" in the **Component Name** text box. Then rename the output signal bus connection for port Q to a new name by typing "**RESULT**" in the connection box to the right of the Q output port.



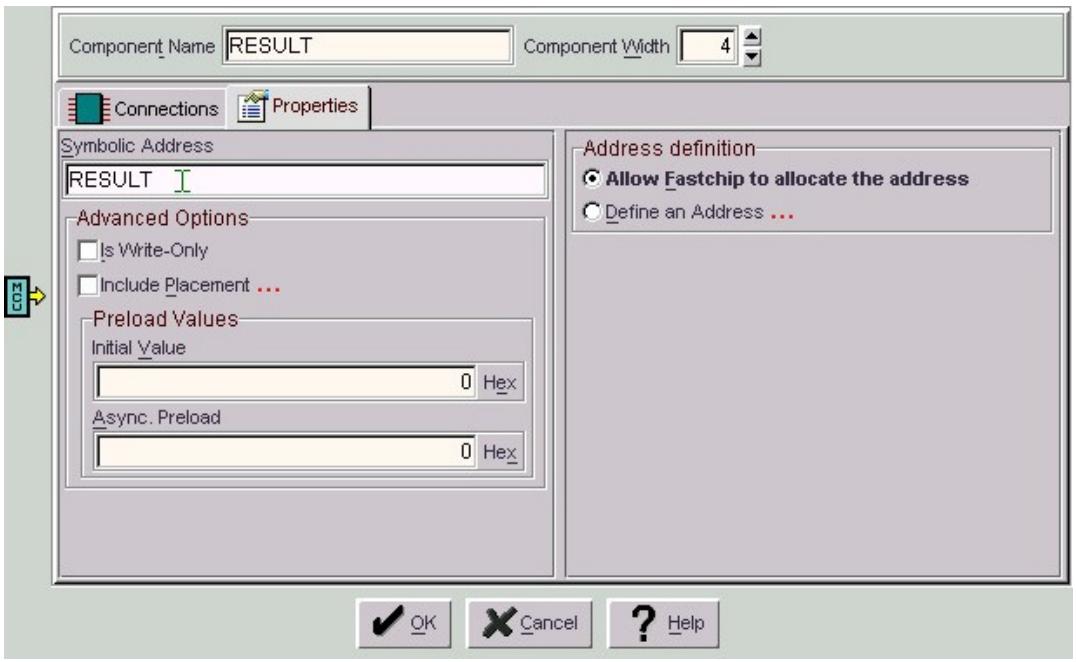
The beauty of programmable logic is that your application uses only the amount of logic required to implement your design. The tutorial design only requires a 4-bit register to hold the hexadecimal character displayed on the LED. Consequently, tailor the Command Register to use only four bits. Click the **Component Width** spinner buttons or enter a text value, ultimately setting the value to '**4**'. Note that FastChip automatically completes the name of the Q output to RESULT[3:0].



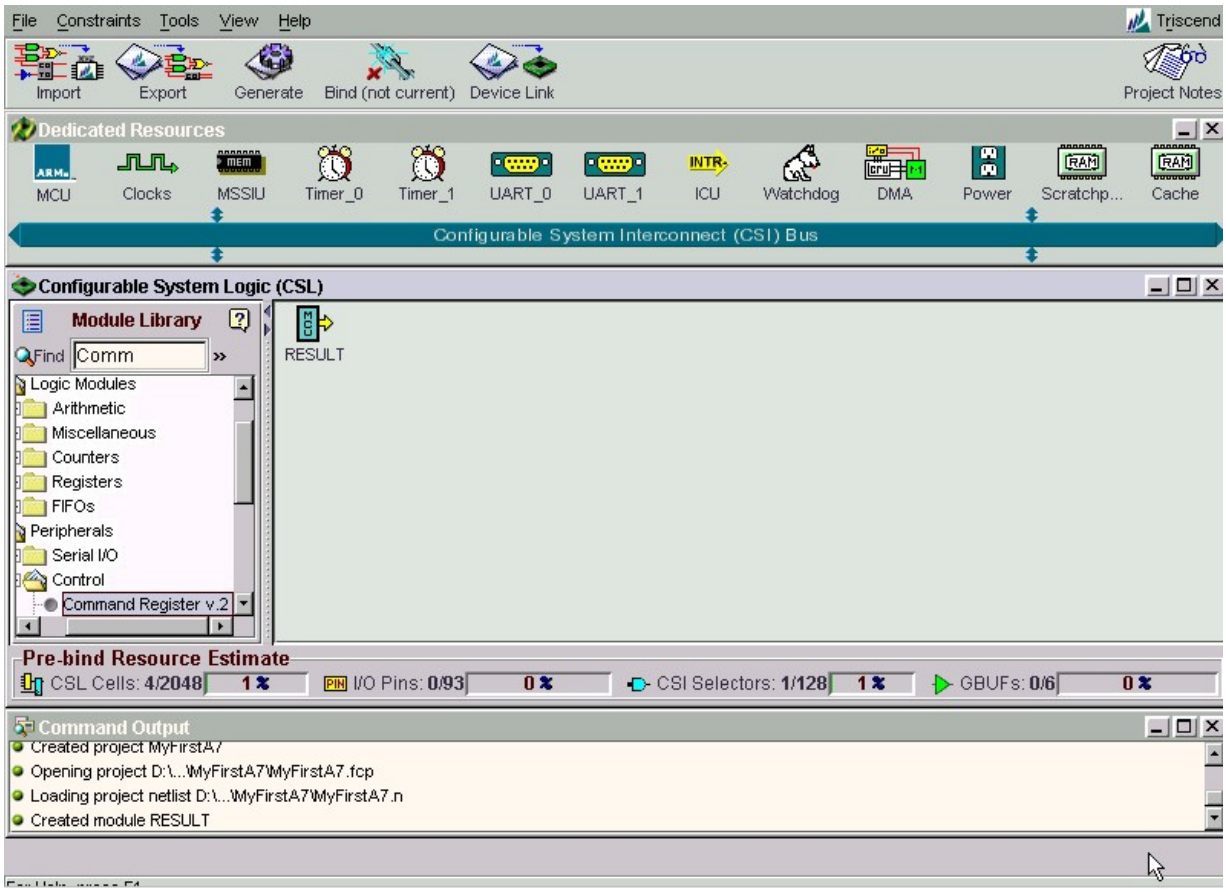
Click the **Properties** tab. Here, you can enter the *symbolic address* for the Command Register address. You can also define the *physical address* for the command register. However, FastChip is designed to support design re-use and easy integration. Consequently, FastChip can automatically

assign addresses for your application and pass the address assignments to your favorite ARM7TDMI compiler. By assigning just a symbolic name, you let FastChip handle the mundane details of creating a memory map for your application.

Enter "**RESULT**" in the **Symbolic Address** text box. Later in the tutorial, we will have FastChip generate a header file for the 'C' compiler. The Generate function allocates this symbol to an unused address location. Your source code can include the generated header and refer to this command register by its symbolic name, RESULT.

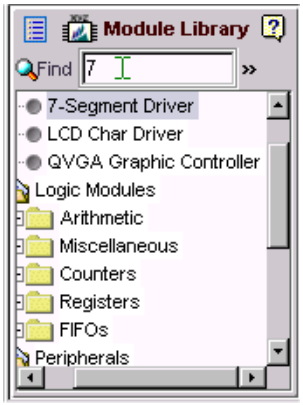


Click **OK** to complete configuration of the RESULT Command Register. FastChip creates a custom-made module called RESULT in the CSL window, based on your settings. Note that the Resource Estimate area shows that the design, so far, uses just four out of 2,048 available CSL cells and one of the 128 CSI bus address selectors.

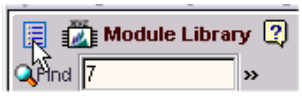


7-Segment Display Driver

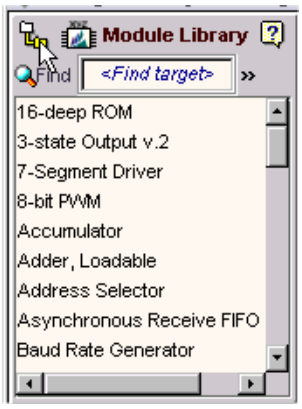
This project requires a 7-segment display driver to show the value of the RESULT register using the 7-segment LED on the A7 Evaluation Board. To locate the 7-Segment Driver, type "**7**" in the Module Library text box. FastChip finds and highlights the 7-Segment Driver module.



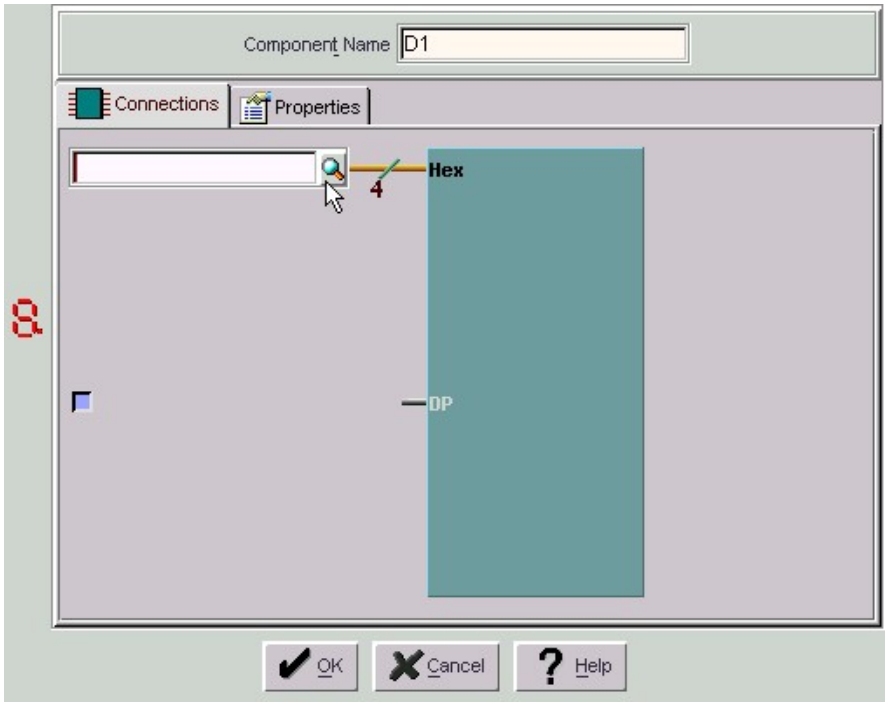
Just to demonstrate some of the other FastChip capabilities, click the list icon next the to the text box.



This action displays the module library, but this time as an alphabetical list instead of a hierarchical tree. Click the hierarchy icon to switch back to the tree view.



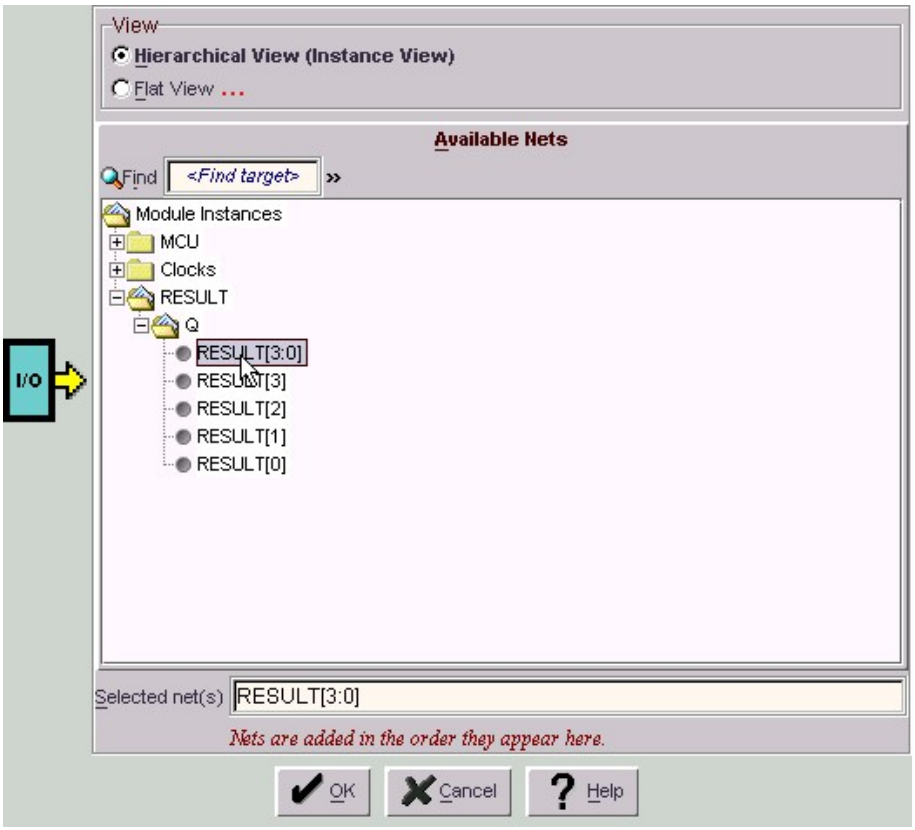
Double-click the **7-Segment Driver** in the Module Library. Rename the **Component Name** to **D1**.



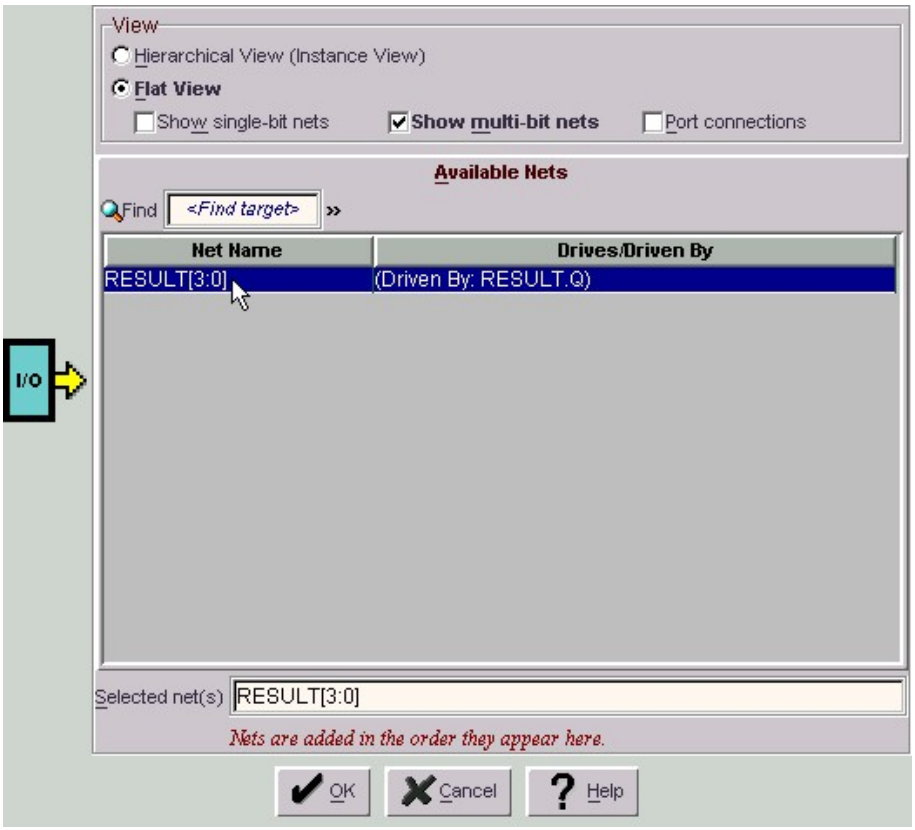
The input to the 7-segment module is a four-bit value. The 7-segment module converts this four-bit digital value into the appropriate hexadecimal display on the 7-segment LED. In this design, connect the output of the RESULT register to the Hex port. There are a few ways to complete the connection. The most obvious is simply to type the name "RESULT[3:0]" in the Hex input port text box. However, you may not always remember the name of the desired connection.

Another approach is to use FastChip's connection browser. From within the 7-segment driver dialog box, click the magnifying glass icon next to the **Hex** input port text box. There are two views of the connections within the design; the Hierarchical View shows all connections as a tree while the Flat View shows the connections alphabetically.

To see the signals connected to the RESULT register, click the **RESULT** folder icon. Then choose the **Q** port, revealing the signals connected to the output port. The signals are listed as a four-bit bit bus called RESULT[3:0] and as individual signals RESULT[3] through RESULT[0]. You could either select **RESULT[3:0]** or hold down the shift key and select the four individual nets.



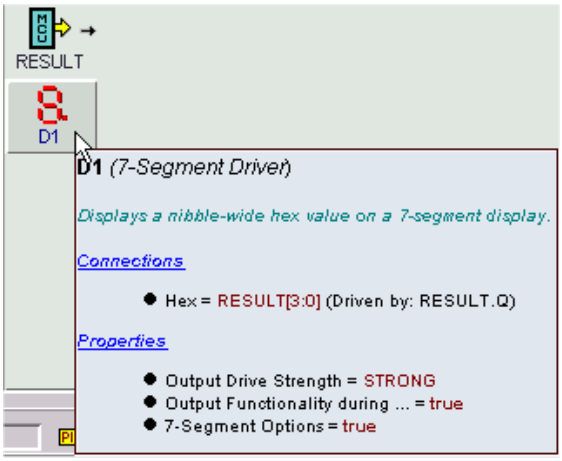
Now choose the **Flat View** option. Because this input port expects a multi-bit value, FastChip only displays multi-bit nets by default. Play around a bit by clicking on the **Show single-bit nets** option and the **Port connections** option. Also, try the **Find** function. Finally, select the bus called **RESULT[3:0]** and click **OK** when finished.



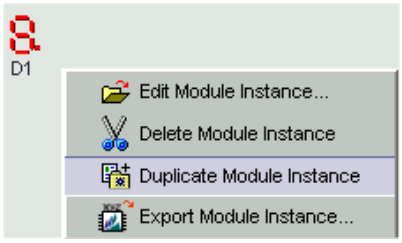
Click **OK** to return to the CSL window, and notice that **D1** has been added to the CSL window and that seven I/O pins are now used.

Various FastChip Features

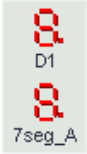
Let the cursor linger over the D1 module icon. FastChip displays a pop-up tool tip that shows how the D1 module is configured and connected. This allows you to quickly review the properties of a module without clicking on it. Also, note the small black arrow to the right of the RESULT module. This indicates that RESULT drives a signal into D1. This information is also shown in the tool tip. Under the Connections sub-heading, note that the Hex port connects to a bus called "RESULT[3:0] (Driven by RESULT.Q)", meaning that the 'Q' output port of the module named "RESULT" drives a bus called "RESULT[3:0]".



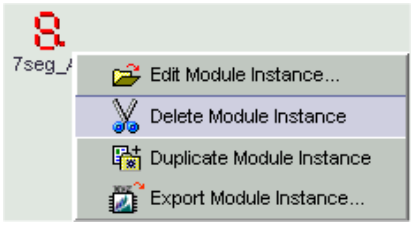
To demonstrate a few other FastChip capabilities, click your right mouse button while the cursor is over the D1 module to reveal the module menu. Select **Duplicate Module Instance**.



This duplicates the D1 module under a new component name 7seg_A, including all module settings and connections. If you were to use this module, you probably want to edit it and modify some of the connections.



The tutorial does not require this new component. To delete the module, place the cursor over the **7seg_A** module and click the right mouse button to reveal the module menu. This time, however, select **Delete Module Instance**. FastChip then displays a confirmation dialog box asking if you really want to delete it. Click **Yes**. Once deleted, there is no undo command in FastChip.



At this point, the tutorial design has just two modules, resulting in a fairly sparse and straightforward CSL window display. However, a design with many modules can become confusing. FastChip allows you to reorder the modules in the display window so that you can group associated functions together. To relocate a module, follow the steps shown below.

Click and hold the left mouse button over the module.

Drag and drop the module to the desired location.

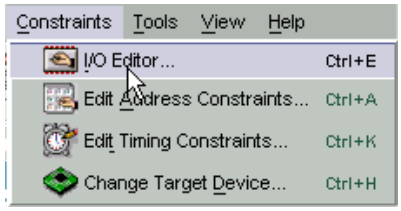
Release the mouse button.

At this point, the custom logic design for this simple design example is complete. Select **File → Save Project** to save your design to disk.

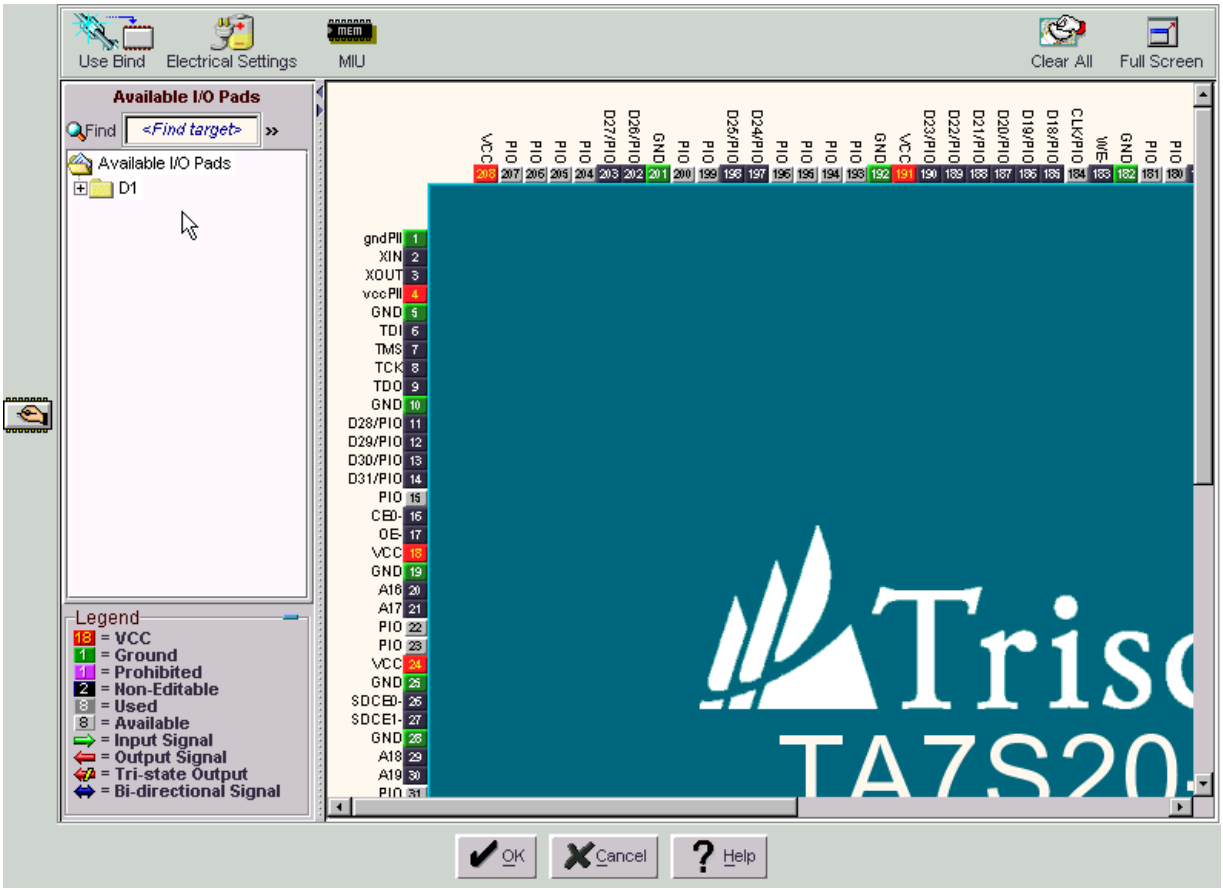
Assign I/O Locations, Configuring the Memory Interface Unit

For most designs, it is best to let FastChip make the first I/O assignment. That way, FastChip can optimize the I/O assignment according to the CSL logic design. However, not all designs have this luxury. For example, the tutorial design example operates on the Triscend A7 Evaluation Board, which already has a pre-defined pin assignment.

In order for the example design to correctly light up the LEDs on the evaluation board correctly, the output pads must be assigned to specific package pins. The FastChip I/O Editor provides an intuitive, graphical interface to assign I/O pads to specific package pins. Choose **Constraints → I/O Editor** from the menu.



FastChip displays the I/O Editor window. All unassigned I/O pads are displayed in the "Available I/O Pads" area on the left. The display on the right shows a graphical view of the selected package, with all pins numbered, labeled, and color-coded. The legend in the lower left-hand shows the meaning of the color-coding. Black pins are dedicated functions like JTAG, clocks, etc. Ground connections are green; power connections are red.

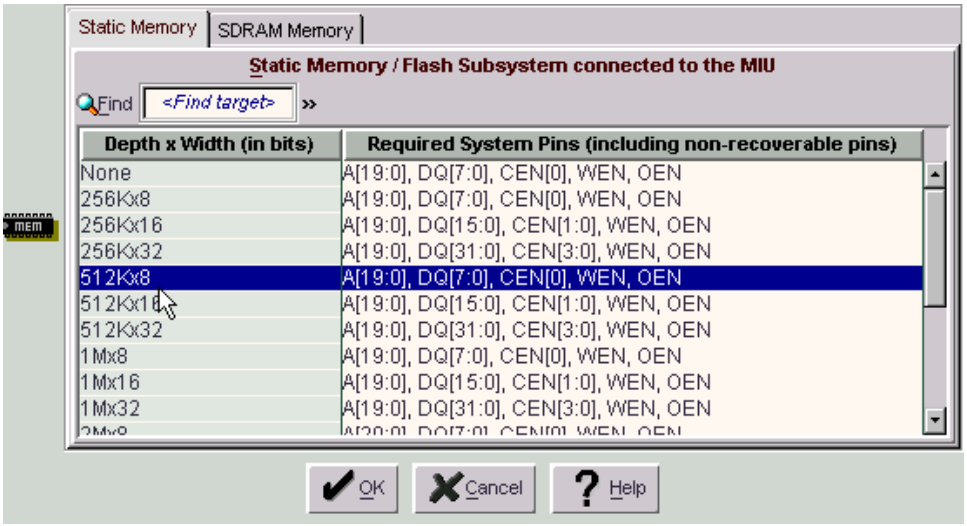


Defining the Memory Interface Settings

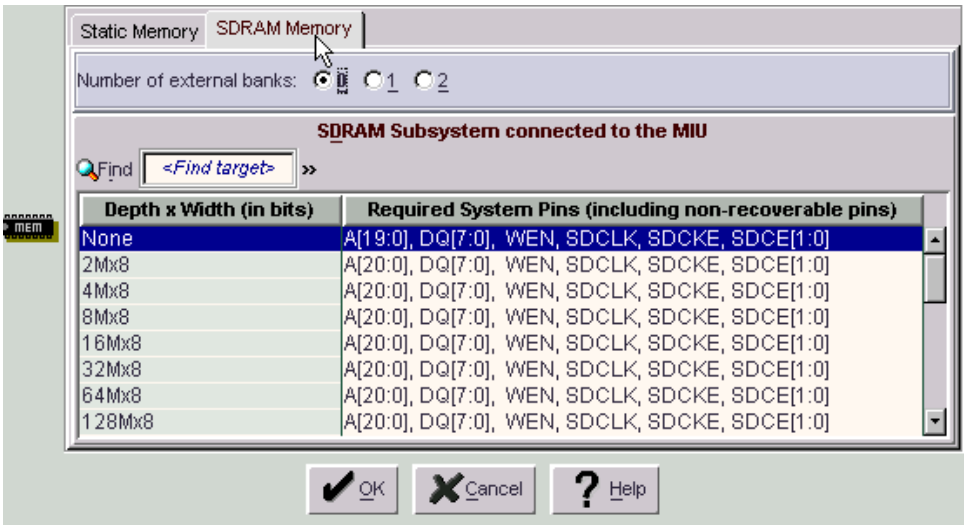
The Triscend A7 CSoC has a Memory Interface Unit (MIU) that supports both static and dynamic memories. The static memory interface typically connects to an external Flash device containing the personality for the CSL programmable logic resources portion of the CSoC device and the compiled binary code for your design. However, the static memory interface optionally connects to other types of static memory such as SRAM or ROM. The static memory interface supports up to 16M-bytes of external memory in x8, x16, or x32 configurations.

The optional dynamic memory interface connects to one or two banks of external SDRAM. The MIU supports up to 256M-bytes of SDRAM in x8, x16, or x32 configurations.

The MIU settings reserve the required pins in the I/O Editor. If a MIU pin is not required for your application, the pin is released back to your design as a Programmable Input/Output (PIO) pin. Start by defining the static memory settings for this example application. The A7 Evaluation Board has a 512Kx8 Flash memory installed in the Flash socket label U11. Accordingly, select **512Kx8** from the dialog box. This action reserves the required I/O pins in the I/O Editor. When finished, click the **SDRAM Memory** tab.

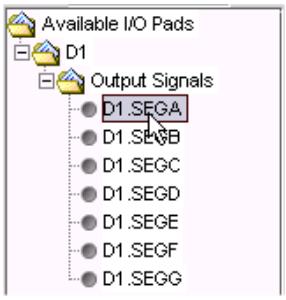


Though supplied on the A7 Evaluation Board, this example design does not use SDRAM. Consequently, select **0** as the **Number of external banks** and **None** for the memory size. Click **OK** when finished.

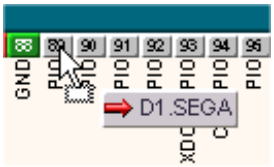


Assigning I/O Locations

Now that the MIU pins have been reserved, start assigning the 7-segment LED driver outputs to the appropriate device pins. To assign a pad to a package pin, first expand the **Available I/O Pads** tree and find the desired I/O.



Then, click and hold the left mouse button to drag the I/O pad to the desired pin location.

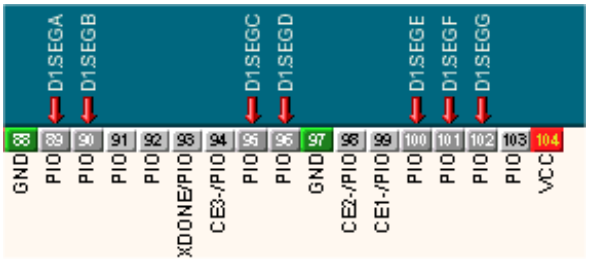


Once over the desired location, release the left mouse button to drop the pad. Note that the assigned pin disappears from the **Available I/O Pads** tree.

If you make an error, you can move the pad by dragging and dropping it onto another pin. Likewise, you can drag an assigned pin back into the **Available I/O Pads** area. Assign all pads to their respective package pins according to the following table.

Module	Pad Name	Pin Number
D1	D1.SEGA	89
	D1.SEGB	90
	D1.SEGC	95
	D1.SEGD	96
	D1.SEGE	100
	D1.SEGF	101
	D1.SEGG	102

When finished, your assignments should appear as shown below. Note that the Available I/O Pads area should be empty.



Click **OK** after you finished assigning pads. The I/O assignments are stored in a file called **`<project_name>.ioc`**, located in the project directory. You may also edit the contents of this file using your favorite text editor.

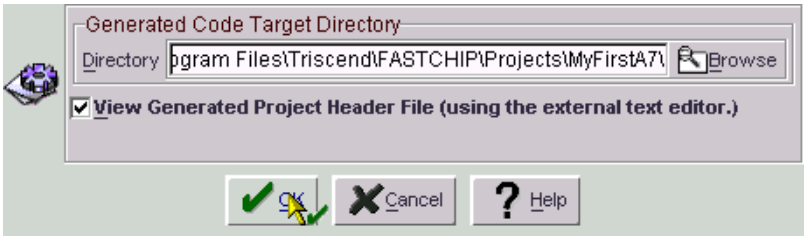
Generate a Header File for Your Compiler

Earlier in the design example, the Command Register was given the symbolic address RESULT. In order to pass the symbol and address of this register to your compiler, FastChip generates a 'C'-language header file with declarations for the addressable registers in the configurable logic.

FastChip does not generate initialization code for the A7's dedicated resources. Instead, the device drivers are provided in source form for easy integration.

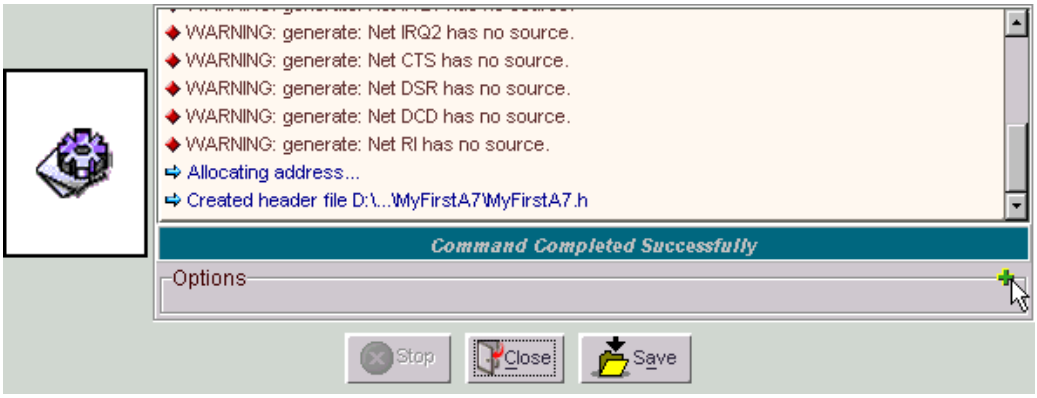
Click the **Generate** button to invoke the **Generate Code** dialog box.

Save the generated file in the default location. You can also choose the target directory where the generated files will be stored. These files will not actually be used during this part of the tutorial. If the **View Generated Project Header Files …** option is checked, FastChip displays the contents of header files using the selected text editor (usually Windows Notepad). Click **OK** when finished.

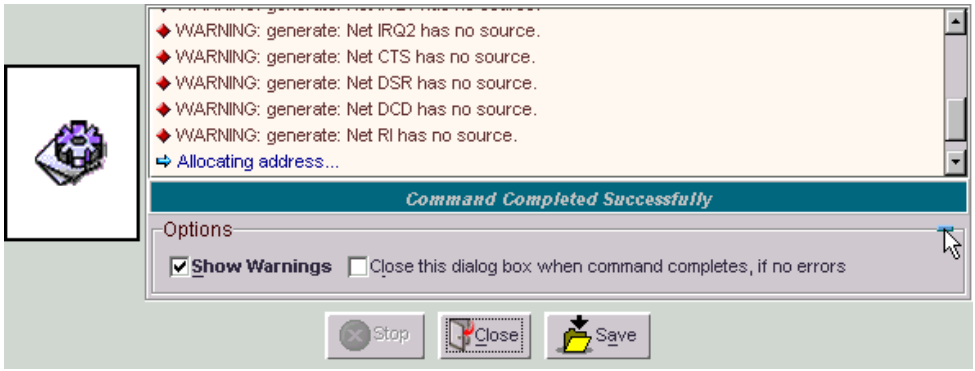


While creating the design, FastChip will issue a few warnings. These warnings are caused by some of the unconnected *sideband signals*. These sideband signals connect to the various dedicated resources on the A7 CSoC device like the embedded UARTs, processor, etc. In the tutorial design, the dedicated resources do not connect to the outside world and consequently, these particular warnings can be ignored.

You can also tell FastChip not to display these warnings in the future, if you desire. Click the green plus sign to the far right of **Options**.



Clicking the plus sign expands the Options pane, exposing the available options beneath. The warning messages only appears if the **Show Warnings** option is checked. By default, FastChip displays all messages, asking you to confirm by clicking OK. You can also have FastChip skip the confirmation step, if there are no errors, by checking the **Close this dialog box …** option. Click the blue minus sign to minimize the options pane.



As a convenience, the Save button saves the contents of this dialog box as an HTML file that you can send to the Triscend SupportCenter should you run into a problem. This does not save your header file in HTML format.

Click **Close** to generate the header file for this example design. Unless you specified another name, FastChip will save a copy at **`MyFirstA7.h`**.

If the **View Generated Project Header Files …** option was checked earlier, FastChip will invoke your default text editor and display the contents of the header file. Scroll down until you find the section marked “BEGIN SOFT MODULE REGISTER DEFINITION”. Once there, the header shows that the RESULT register is assigned to address 0x1000ffffc.

```
/* ========== BEGIN SOFT MODULE REGISTER DEFINITION ======= */
/* =  You can use the following register definition       = */
/* =  with the access macros defined above.  You can      = */
/* =  also type cast the register definition yourself     = */
/* =    eg. to access a 4 bytes register 'MyReg'          = */
/* =        as unsigned int -> TWORD(MyReg)=102;          = */
/* =        as string -> sprintf((char *)MyReg, "abc");   = */
/* ========================================================= */

#define RESULT 0x100ffffc /* size = 4 bytes */

/* ========== END SOFT MODULE REGISTER DEFINITION ======= */
```

The FastChip tutorial example does not actually use this header file. However, it will be used in later design examples.

Bind the Project

In a process called Bind, FastChip compiles the design shown in the CSL window into to CSL logic within the CSoC device. This process is analogous to the compile-link-load process when compiling a software program or the map-place-route process for creating a gate array or FPGA.

Bind is a computational-intensive process, much more so than a simple compiler for a processor. Many complex computing algorithms optimize your design to fit it into the CSL logic. The Bind process for this simple project may take five to ten minutes on computers that only meet the minimum system requirement. The status bar on the wait dialog box shows you how Bind is progressing.

The ultimate result from Bind is an initialization file called **`<project name>.csl`**. This file is later combined with your software application image to download to the CSoC.

FastChip indicates the current Bind status for the project. If Bind must be executed before completing the project, the Bind icon indicates, "Bind (not current)". If the Bind step has already been completed, the icon indicates just "Bind". To process the tutorial design, click the **Bind** icon.

Because Bind is compute-intensive, FastChip provides various **Effort Levels** allowing you to trade off compute runtime versus Bind quality of results. The Maximum setting provides the best overall results but at significantly longer run times. There is also a timing-driven mode where you can specify the timing requirements for your design. Bind then attempts to compile the design and meet your requirements. Again, this adds to the overall run time.

For this design, accept the default Bind effort level, **Minimum**, for minimum bind effort and click **OK**.

As Bind progresses, you will see various messages in the output dialog box.

If you have not already licensed FastChip, you will see a message describing that you are currently running Bind in Evaluation mode. The message also describes the limitation of the evaluation mode and how to obtain a FastChip license. This tutorial design functions with either the full or evaluation modes.

Bind then performs a design-rule check (DRC) to find any potential problems. For this design example, Bind reports that the sideband signals, like IRQ0, IRQ1, etc., are not connected. These specific warnings may be ignored, as this example does not use any of the sideband signals.

Bind then performs automatic address allocation; similar to that when you generated the header file for your 'C' compiler. This step finds all of the addressable items in your CSL design—essentially anything using one of the CSoC's address selectors. Bind then assigns a physical address to every selector based on the design requirements such as address size and any relative address assignments specifying the address relationship between two or more selectors.

The mapping step decomposes your CSL logic design and then compacts the logic into the resources inside the CSL cells, the selectors, and the CSI bus resources.

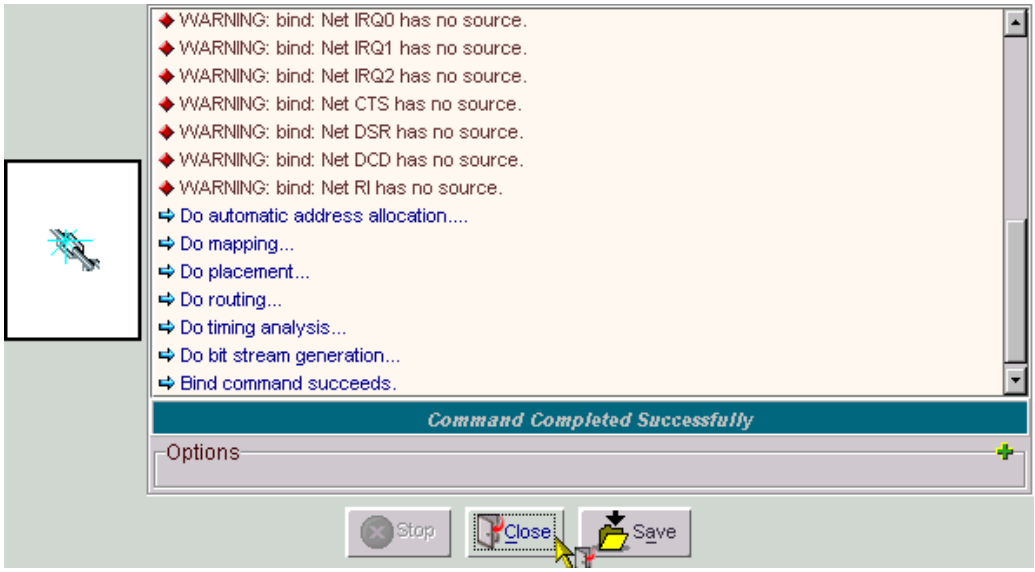
The placement step determines the best location for each used CSL, selector, and bus resource out of the available resources on the selected target CSoC device. This is analogous to the placement step of a printed circuit board place-and-route package.

The routing connections step decides how best to interconnect the CSL, selector, and bus resources given their current placement on the targeted CSoC device.

After completing the mapping, placement, and routing steps, FastChip performs a static timing analysis of all of the logic paths within the device. These paths are described in detail when you generate a project report.

Bitstream generation creates the binary programming file for the compiled CSL design. This file, called <project name>.csl, is used later to physically program the target CSoC device.

Finally, Bind reports that it successfully completed your design.



Again, the Save button saves the Bind output, primarily for customer support or to review any errors or warnings reported by Bind.

Click **Close** when finished. The Bind result is saved in the **`<project_name>.csl`** file.

Select **File → Save Project** to save the current state of the project.

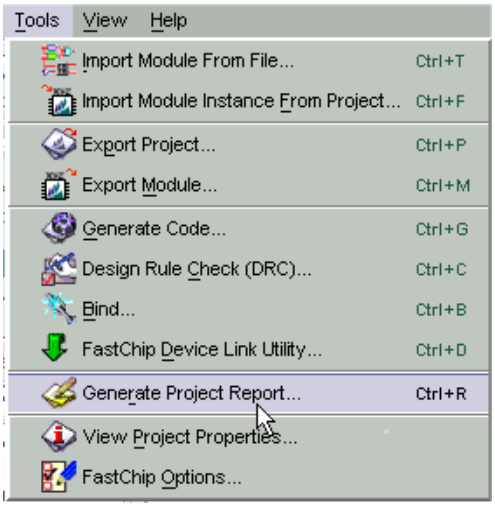
Note that the Bind button no longer indicates "(not current)". Also note that FastChip updates the resource estimates area with the exact resources as determined by Bind. Before running Bind. FastChip quickly estimates CSL usage based on assumptions about the design. After running Bind, FastChip knows exactly the CSL resources required. Usually, only the CSL cell count will change.

Before Bind …

After Bind …

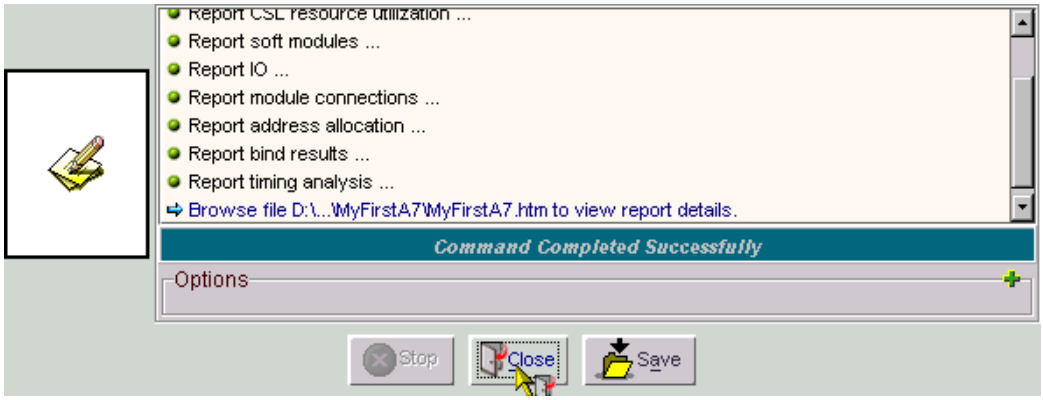
View the Project Report

To investigate what Bind has done for your project, select **Generate Project Report** from the **Tools** menu to display the project report dialog box.

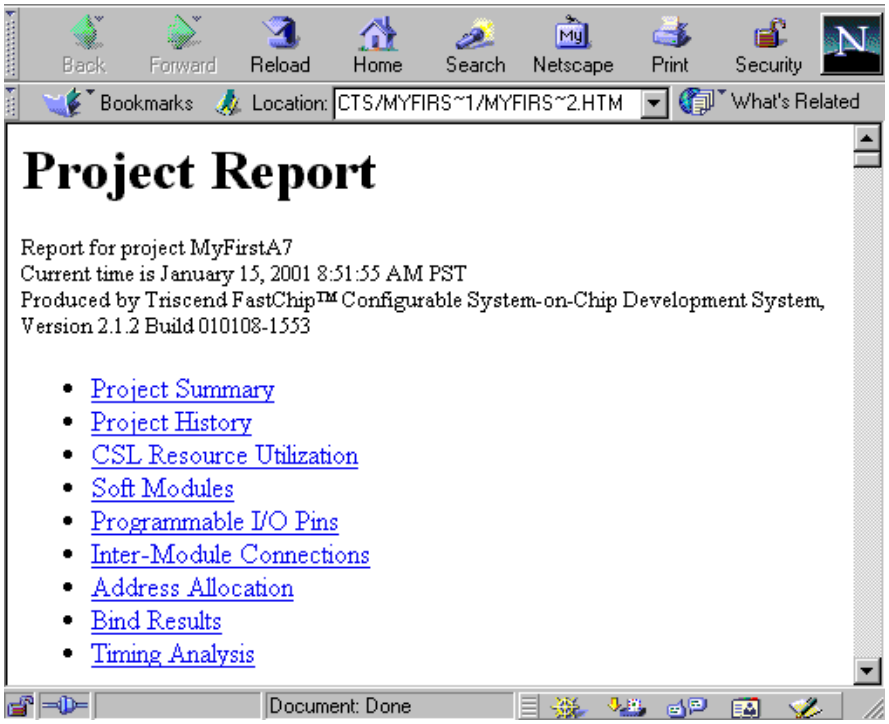


When you click **OK**, FastChip creates the project report.

Click **Close** to invoke your default HTML browser to display the report file.



Browse through the report to look at the result of binding the project. When you are finished, leave the browser window opened. This report will be used later when debugging the project.



Setup the A7 Evaluation Board

Before attempting to download the design, be sure to perform the following steps. These key items must be completed before you can reliably communication with the A7 Evaluation Board.

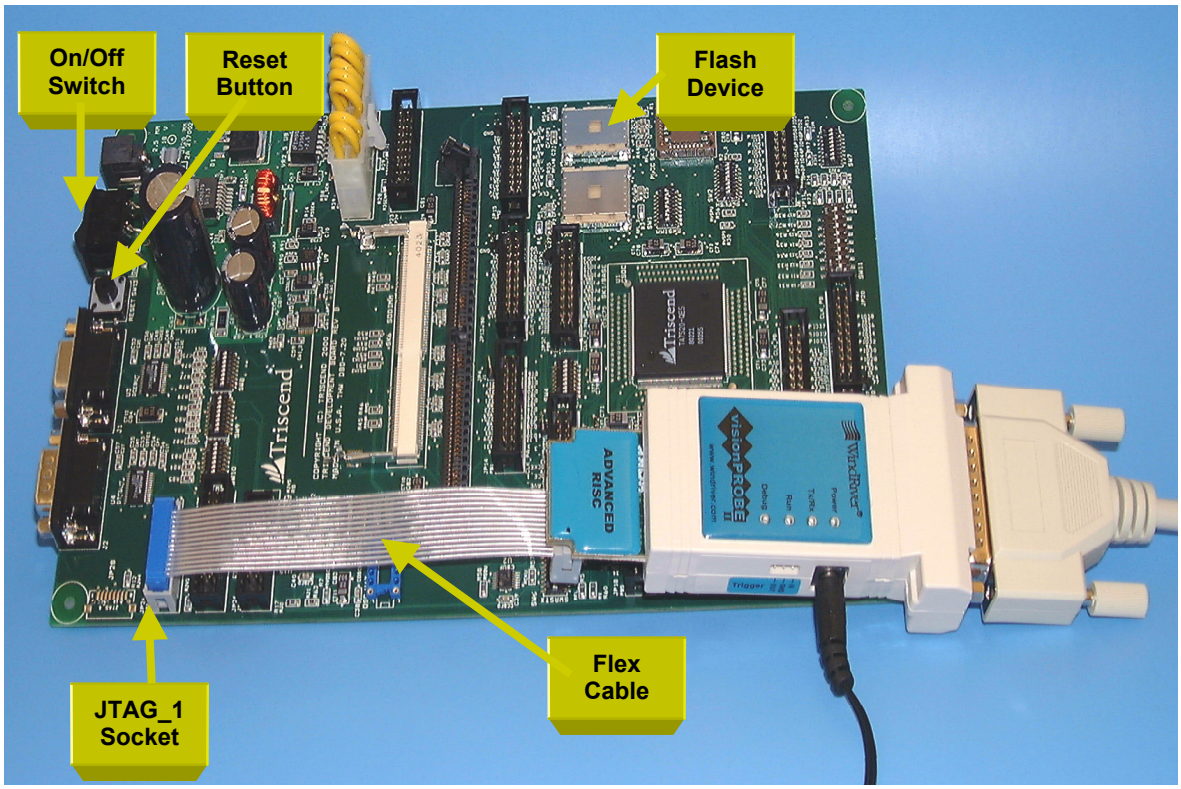
- Configure the parallel port for visionPROBE II
- Apply power to the visionPROBE II cable
- Plug the visionPROBE II cable into the board
- Apply power to the board and switch it on
- Apply reset to the board by pressing the reset button

Configure the parallel port for visionPROBE II

For a detailed description of how to configure the parallel port and connect the visionPROBE II cable to the board, refer to the pamphlet by WindRiver called *visionPROBE II Hardware Installation Guide*. There are three very important considerations for configuring the visionPROBE II cable.

1. The parallel printer port on your computer must support and be configured as an Extended Capabilities Port (ECP). On some computers, this is accomplished as a Bios setting. Other computers can be configured via the operating system.
2. Make sure that the allocated I/O address and interrupt request (IRQ) settings in the file **`<install directory>:\ESTII\comdll.cfg`** match those for your parallel printer port.
3. There is a utility to test your visionPROBE cable and configuration called **`VPTest.exe`**, located in the **`<install directory>:\ESTII\`** directory.

Additional details on the A7 evaluation board are available in the **`\Docs`** subdirectory of the FastChip installation directory in an Adobe Acrobat file titled *Triscend_A7_Evaluation_Board.pdf*.



Apply power to the visionPROBE II cable

Plug in the 9 VDC 300 mA power supply and apply power to the cable with the side-mounted barrel jack.

The polarity of the connector on the visionPROBE II and board is important, especially if you purchase your supply from a third-party. For the visionPROBE II cable, the polarity is: Ring = Positive, Tip = negative, and for the board it is: Ring = negative, Tip = positive.

Plug the visionPROBE II cable into the board

The cable must be connected to the board using the flexible extender cable. To align the couplings correctly, match the tiny faint downward-sloping triangles ▼ that appear in the plastic molding of both the female end of the extender cable and the JTAG_1 male pin socket on the board.

Apply power to the board and switch it on

Plug in the 15 VDC power supply. Connect the power cable to the board with the barrel jack, and set the power rocker switch to the On position, so that the green LED lights up.

Reset the board

To assure adequate reset signal to the A7 device, press the Reset button on the board.

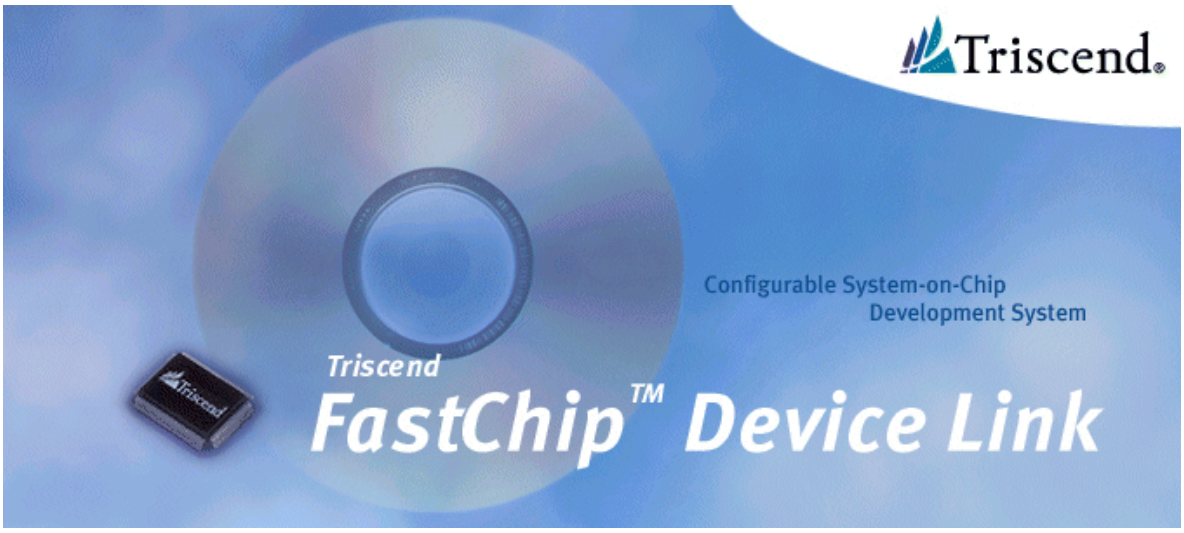
Invoke FastChip Device Link Utility

Launching FDL from FastChip

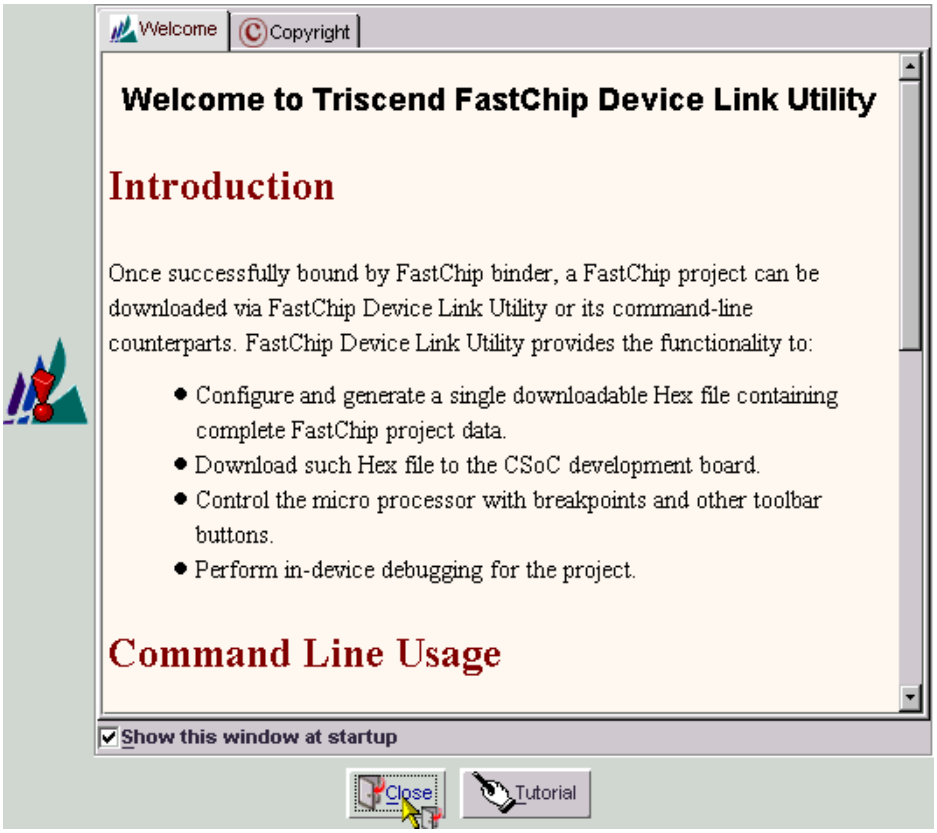
Use FastChip Device Link (FDL) to configure, program, and debug the CSoC. With the MyFirstA7 project still opened, click the **Device Link** button to invoke the FastChip Device Link software.

FDL is designed to be a stand-alone program, separate from FastChip. FDL can be installed as a stand-alone application on a computer in your development lab or on the manufacturing floor. Likewise, if your customer also has a visionPROBE II cable, you can provide FDL to your end customer to download a configuration file to your prototype system. That way you can provide just a data file to your end customer without revealing the internal details of your FastChip project.

Once FDL loads, it displays a splash screen.

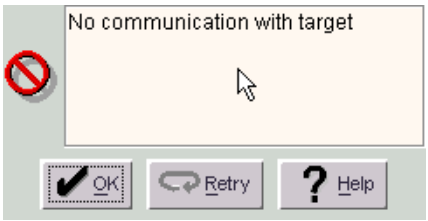


A welcome screen also appears describing the capabilities of FDL and how to execute FDL from a command line, minus the graphic interface. If you wish, uncheck the **Show this window …** option box at the bottom of the dialog box to prevent this dialog from appearing the next time that you run FDL. Click **Close** to continue.



The main FDL window should appear. At this point, you may encounter an error dialog box indicating that FDL has **No communication with the target**. You can continue to create a configuration file but you must correct the communication problem before you can download and debug your application.

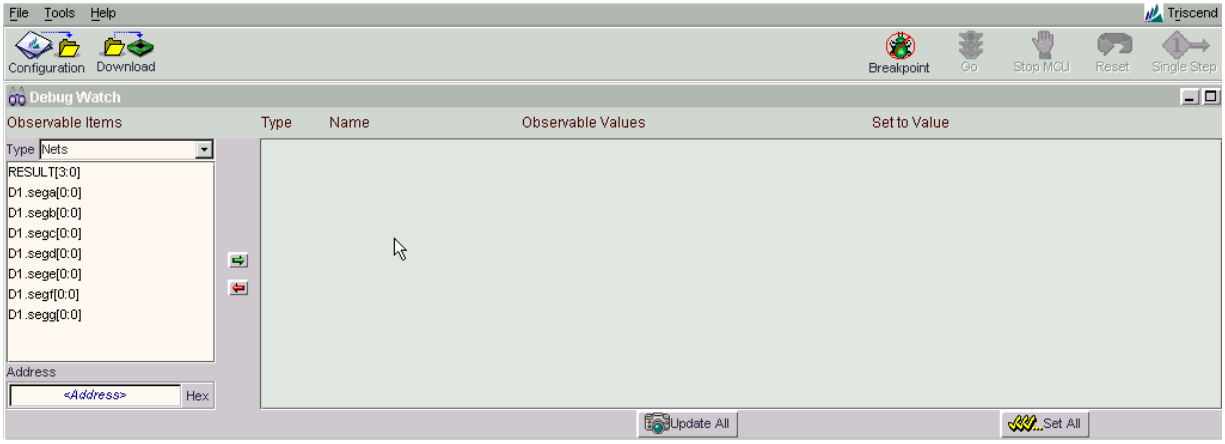
If you wish to connect to the evaluation board, follow the steps described in the *Setup the A7 Evaluation Board* section and click **Retry**. Optionally, click **OK** to build a configuration file. However, if FDL cannot communicate with the visionPROBE II cable, you will not be able to download or debug an application.



The most likely causes for the "No communication with target" message are …

- Power is not applied to the target board or the visionPROBE II cable.
- The visionPROBE II cable is not connected to the target board or to the computer parallel printer port.
- The computer parallel port is not properly configured for the visionPROBE II cable. Please refer to the WindRiver ***visionPROBE II Hardware Installation*** guide, available within the A7 Starter Kit or on the visionCLICK CD-ROM.

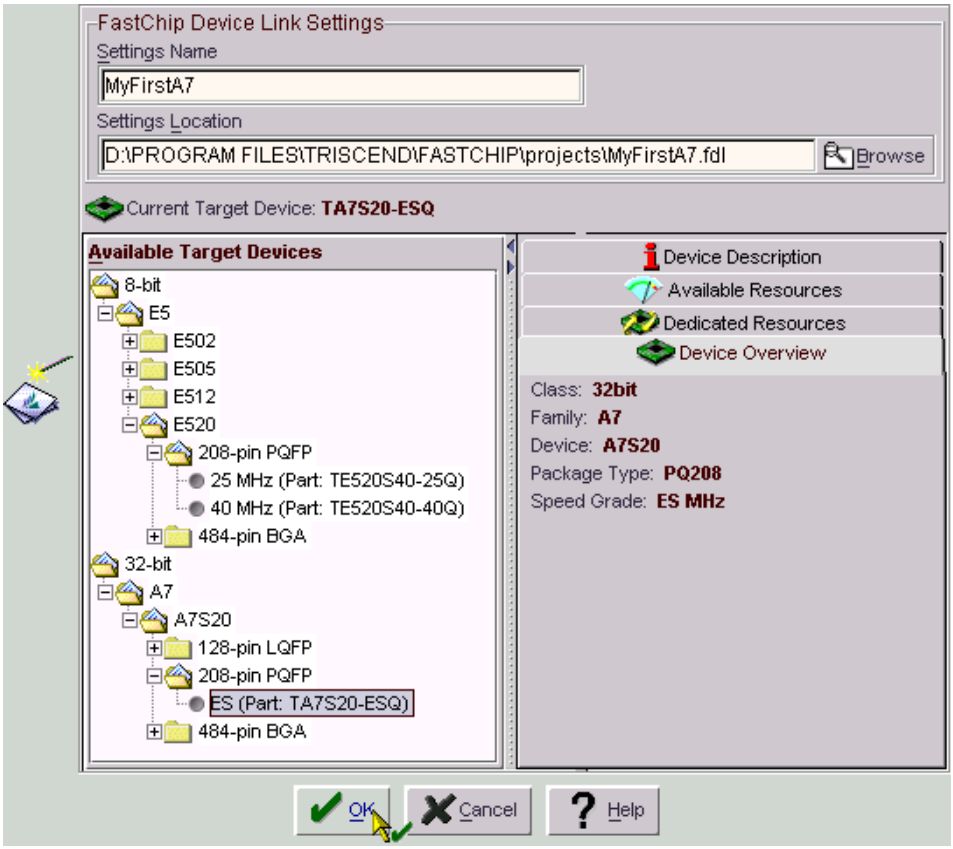
The primary FDL window shows a list of observable items along the left edge of the screen and the currently monitored list along the right.



Launching FDL from a Desktop Icon or the Start Menu

Optionally, you can also launch FastChip Device Link (FDL) utility from the desktop icon—if you chose to this option during installation—or from the Windows Start menu. However, because it is not launched from an associated FastChip project, it starts blank with "(No Settings)" displayed on the title bar, and the File menu active. Select **New FastChip Device Link Settings …** from the **File** menu so that you can associate various memory configurations and clock sources without re-Binding the design.

Type "**MyFirstA7**" as the **Settings Name**. Select the **TA7S20-ESQ** from the **Available Target Devices**, just as you did earlier when you created your FastChip project. Use the Browse button to set the directory **Settings Location** to the MyFirstA7 project directory. Once you are satisfied, click OK to save the settings, **MyFirstA7.fdl**, to your project directory.



At this point, the FastChip Device Link Utility window will look exactly as if you have launched it from the FastChip toolbar directly.

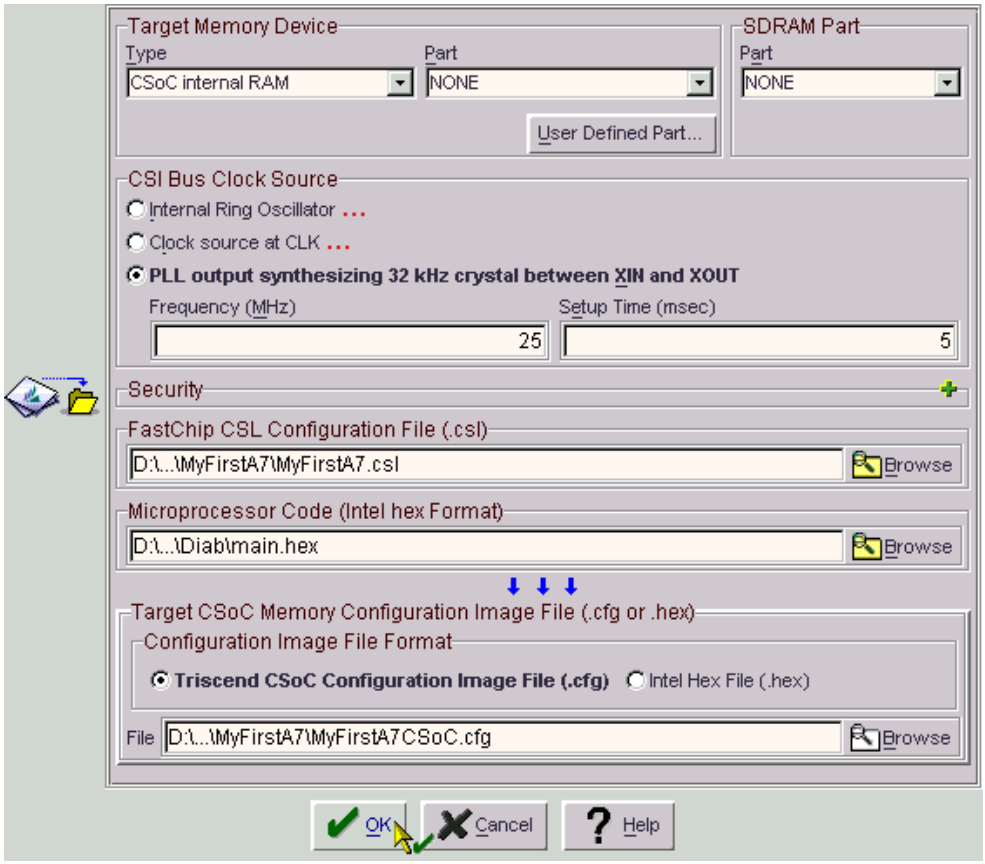
If you start FDL from the FastChip toolbar, the device link settings **`<project_name>.fdl`** is automatically created in the project directory—eliminating the need to perform the **New Device Link Settings** step mentioned above.

Create CSoC Configuration Image

Before actually downloading the design to the CSoC device, the Configuration step combines the CSL data created by Bind with software application image created by your ARM7TDMI compiler to create the final configuration file. This is also the point where you specify where the configuration file will be stored and configure the clock source for the application.

To create a configuration file, click the Configuration icon from the FDL toolbar.

Set the configuration options as follows.



- **Target Memory Device:** Use the default setting **CSoC Internal RAM** to directly program the CSL, and place the microprocessor code into internal RAM. Other options allow you to program an external Flash device over JTAG and to create a Hex for programming an external Flash or EPROM device using a dedicated device programmer.
- **CSI Bus Clock Source:** Select **PLL output …** to synthesize a clock frequency from the external 32 kHz watch crystal connected to the A7 CSoC device on the evaluation board.. Type "**25**" in the **Frequency (MHz)** field and set the **Setup Time (msec)** to "**5**".

Enter the following file locations to specify the source data files of the design.

- **FastChip CSL Configuration File (.csl):** By default, this field points to the *.csl file created during the Bind process. You can also select a specific *.csl file by clicking Browse.
- **Microprocessor Code (Intel Hex Format):** This FastChip tutorial purposely skips over the steps to create and compile an ARM7TDMI application program. Later examples cover coding a great detail. In order to make this tutorial operate, select a pre-compiled program that increments the RESULT register whenever the Watchdog Timer expires. Click **Browse** and select the following file.

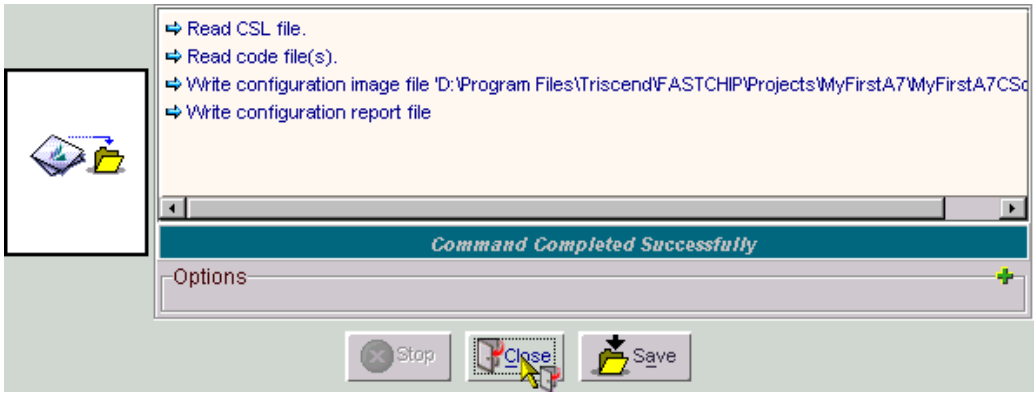
  `<FastChip installation directory>\Projects\MyDesign\3rdParty\Diab\main.hex`

Enter the following file location to specify the output data file.

- **Triscend CSoC Configuration Image (.cfg)**: FastChip combines the CSL configuration file and the processor code to create a final configuration image. This file contains all the data necessary to configure the CSoC device on the evaluation board. By default, FDL saves the configuration image in the current project directory. The configuration image can be saved as either a Triscend *.cfg file or as an Intel MCS-86 *.hex file, which is compatible with third-party device programmers for programming Flash and EPROM devices.

There is also an optional Security field where you can read-protect, write-protect, or both read- and write-protect the contents of the CSoC device. Read-protecting the device prevents monitoring of the device via JTAG. Write-protecting the device prevents the contents from being re-written. The security settings remain valid until power is completely removed from the device.

After defining the configuration options, click **OK** to continue.

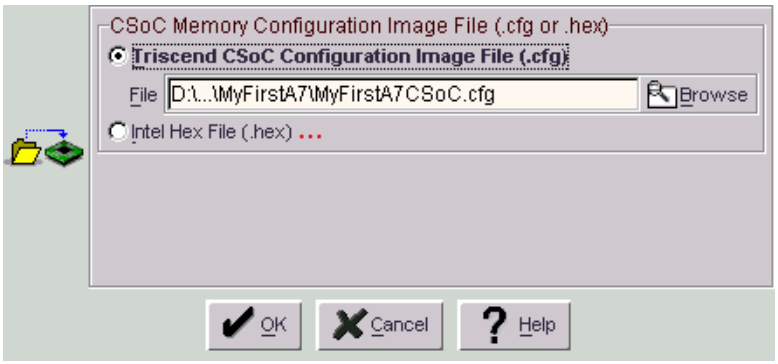


The Configuration utility creates a configuration report file, **`<project_name>.cfr`**, which contains the register settings for the Memory Interface Unit (MIU) and clock configuration settings.

Download the CSoC Configuration Image

After creating the configuration image, click the **Download** icon to invoke the Download dialog box. You must have the visionPROBE II cable connected between the A7 evaluation board and your computer before you can continue with this step.

Download allows you to download either a Triscend configuration file, ***.cfg**, or an Intel MCS-86 Hex file, ***.hex**. During the Configuration step, we created a Triscend configuration file. Use the default settings and click **OK**.



Click **OK** to start downloading to the CSoC on the development board. When FastChip finishes downloading, you will see the 7-segment LED start counting up. The display should update about twice per second.

The resulting dialog box shows the progress of downloading the design. Click **OK** when downloading is finished.

In-System Debugging using the FastChip Device Link Utility

After you finish downloading the CSoC configuration data to the CSoC device, you should see the message "CSoC Running" in the status area in FastChip Device Link Utility. You can start performing in-system real time debugging using the FastChip Device Link Utility.

The A7 CSoC device—in conjunction with the FastChip Device Link utility, the visionPROBE software, and the visionCLICK II JTAG download/debug cable—provides nearly unparalleled debugging capabilities. Using the A7, you can debug your application …

- In real-time
- At full speed
- Using the same software that you plan to use in production
- Using the same hardware that you plan to use in production
- Without sacrificing resources inside the A7

The debugging capabilities of FDL supplement those of a good source-level debugger such as visionCLICK. FDL is not intended to perform as a source-level debugger but provides capabilities not offered by a source-level debugger, such as the ability to monitor flip-flop and logic output values buried deep within the CSL logic.

Debug Tools

The FDL toolbar offers five separate debugging functions.

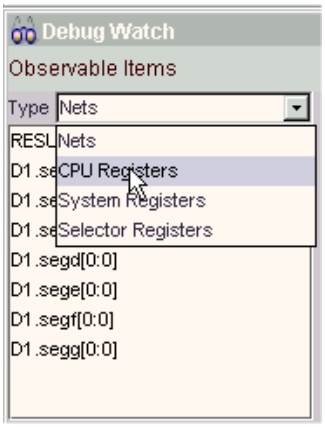
	The **Breakpoint** button lets you setup two hardware breakpoint units on the A7 CSoC. Clicking this icon displays the breakpoint dialog box. You will use breakpoint dialog box to setup a breakpoint later.
	The **Go** button restarts program execution whenever the CPU is in reset, halted, or has reached a breakpoint.
	The **Stop** button halts the CPU execution when it is running.
	The **Reset** button invokes a pull down menu offering the following options. • Reset the CPU only and halt it at the beginning of the application program • Reset the CPU only and start running the application program • Reset the entire CSoC device, and leave it halted. If you downloaded to internal SRAM and select this option, you must download your design again. If the configuration data was downloaded to external Flash, the CSoC device is staged to load itself from external Flash. • Reset the entire CSoC device and start the CSoC self-configuration process. Again, if you downloaded the design to internal SRAM and select this option, you must download your design again. If the configuration data was downloaded to external Flash, the CSoC device loads itself from external Flash and the CPU begins executing the application program.
	The **Single Step** button executes the next assembly instruction pointed to by the current program counter, when the CPU is halted or at a breakpoint. A single 'C'-code statement may compile into multiple assembly instructions. A source-level debugger, such as visionPROBE, allows you to single-step either assembly or 'C' instructions.

In addition to the buttons in the toolbar, there is also the **Debug Watch** window to display snapshot value of registers and nets within your FastChip project.

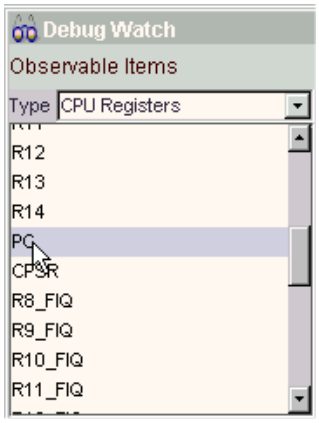
Observe and Control

Using FDL, you can monitor and control the A7 CSoC device from your computer via the JTAG visionPROBE II JTAG download/debug cable.

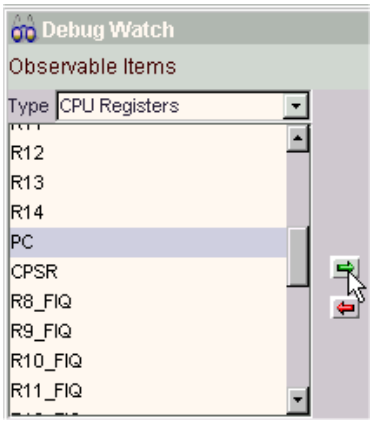
While not intended as a real-time debugger for the ARM7TDMI processor, FDL does provide some visibility into the ARM7's internal registers. In the **Debug Watch** window, select the **Type** drop list under **Observable Items** and select **CPU Registers**.



Scroll down through the list of available registers to locate the program counter (PC) and select it.



To place the program counter, PC, into the list of watched items, click the green arrow button while PC is highlighted.



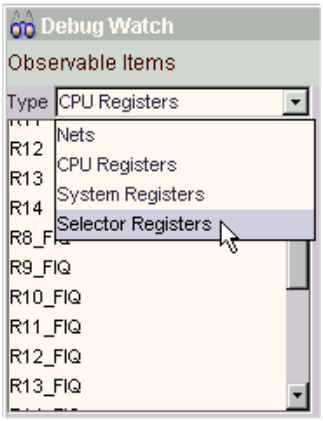
The PC register appears in the watched items display along with two associated text boxes. The **Observable Values** text box displays the value read back from the PC register. Use the **Set to Value** text box to modify the value of the associated item. Not all items, such as network connections, can be modified. Nets can only be monitored.

Also note the radix button to the right of each text box. Each time you click the button, you change the radix for the displayed value in the text box. The button cycles from **Hex** (hexadecimal), to **Dec** (decimal), to **Bin** (binary), and then back to **Hex**.

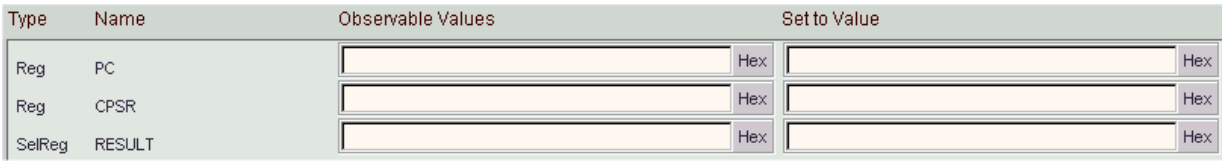
Now, try adding the current program status register (CPSR) to the watched items display.

You are not limited to just monitoring CPU registers. You can also watch any register that you added to the CSL matrix. For example, the RESULT register was a new register created earlier in the tutorial. FDL refers to these registers as Selector Registers because they use one of the CSoC's Selector functions to decode the register's address.

Again, in the **Debug Watch** window, click the **Type** list box under **Observable Items**. This time, however, choose **Selector Registers**.

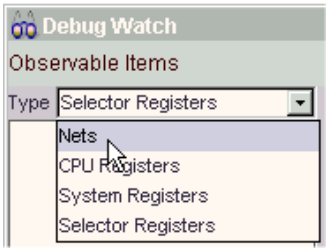


Only one register appears in this list, RESULT. Select the RESULT register and click on the green arrow, adding RESULT to the watched items display.



FDL now monitors the value stored in the RESULT register by reading RESULT at its assigned output. However, there are times when you may not want to read the actual register. For example, some peripherals have registers that are self-clearing when read. Monitoring such registers may cause strange consequences during debug.

Another safe way to monitor the contents of a register is to monitor the nets connected to the flip-flops that form the register. Monitor the output of the RESULT register by choosing **Nets** from the drop list.

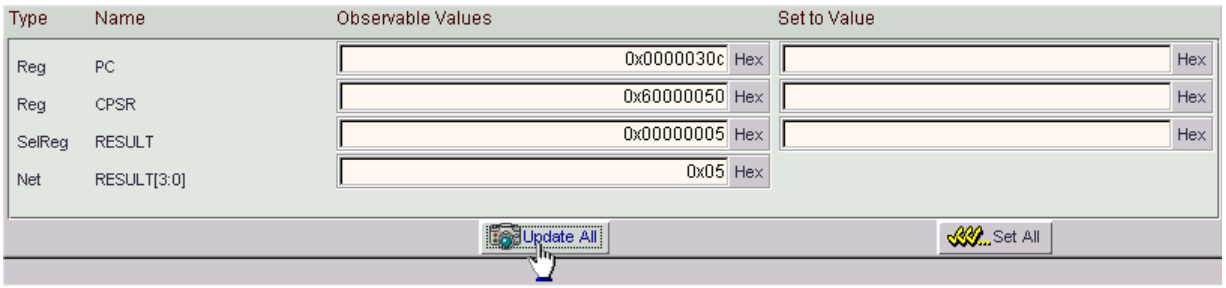


Select the nets connected to the RESULT register, which is a bundle of nets called RESULT[3:0]. Again, click the green arrow to add these nets to the watched items display. Note that nets only have

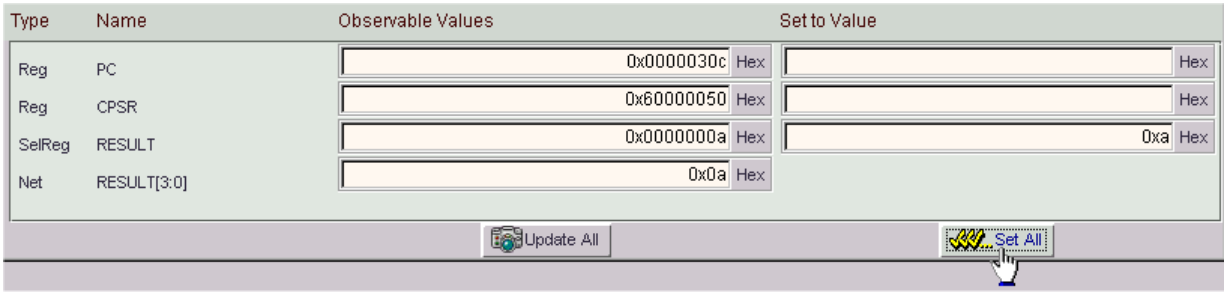
an associated Observable Values text box. There is no associated Set to Value text box because nets can only be monitored and not controlled.

At the bottom of the watched items display, there is a button labeled Update All. Every time you click the **Update All** button, the values in the **Observable Values** column are first read from the CSoC device via JTAG, uploaded to your computer, then displayed on your computer screen. The RESULT command register display should match the number displayed on the LED on the board at the moment.

Click the **Stop MCU** button, then click **Update All** to take a snapshot of the current device state.



To demonstrate how you can control logic buried deep inside the CSoC device over JTAG, type the value **"0xa**" in the **Set to Value** text box associated with the RESULT register. Then, click **Set All** to download this value to the RESULT register. You should see this new value reflected on the evaluation board's 7-segment LED display. If the Set to Value text box is left empty, then FDL does not download a new value to the associated item.

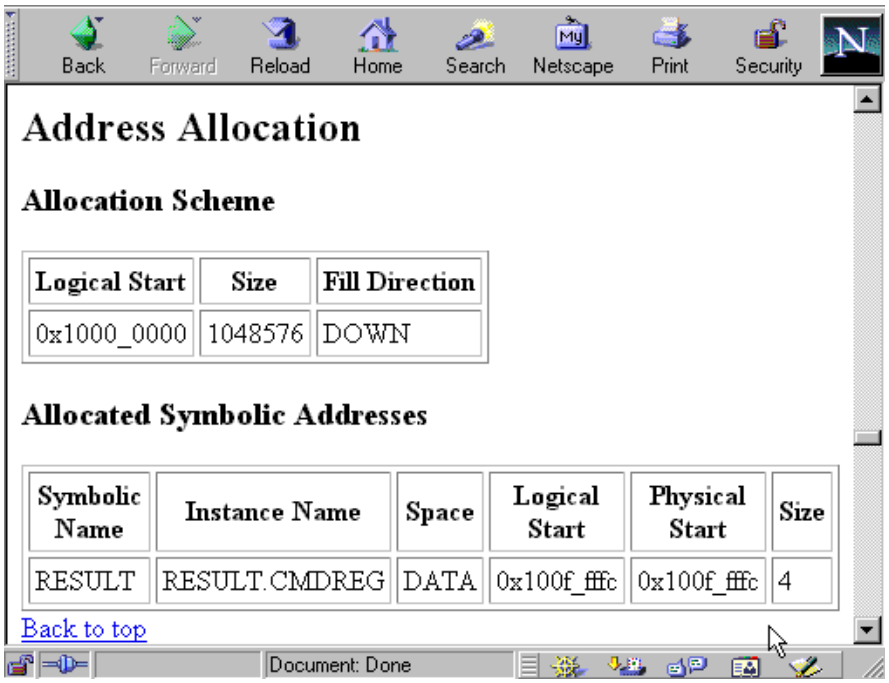


Click **Single Step** a few times. Observe the 7-segment LED display. Most likely, the display doesn't change. However, you may notice that the PC register in the watched items display changes. In this case, you are single-stepping the ARM7TDMI processor through a program loop. Using this approach, it may take a few hundred mouse clicks to advance the program to the point where the LED display is incremented.

A more productive approach is to set up a breakpoint.

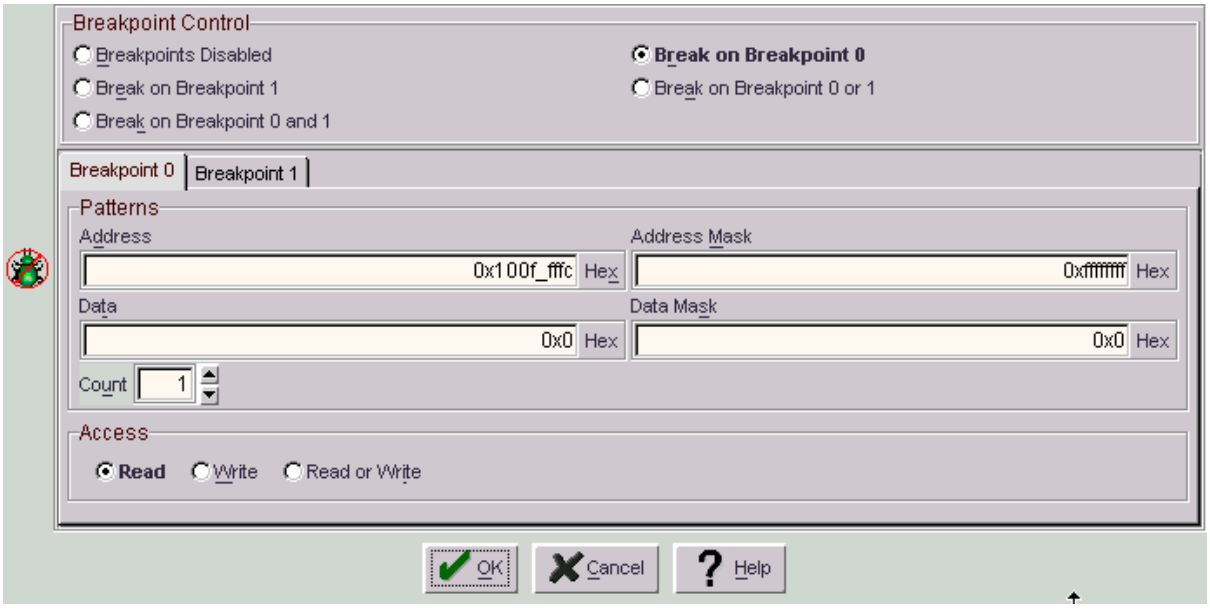
Setting Breakpoint

Next, you'll setup a breakpoint to stop the MCU whenever the 1st digit of the LED is displaying 2 or 5, using the two CSoC built-in hardware breakpoints. To do that, you will setup 2 breakpoints when address corresponding to the **value** command register is written into with the 2 or 5. To do this, you need to know the address the register on the CSoC's CSI bus.



You can find the address information in the project report you generated earlier. Bring up the HTML browser window from earlier. Select **Address Allocation** to display the CSL selector address mapping report.

FastChip allocated the RESULT register as a word-aligned register at address 0x100f_fffc. Now that you know **RESULT**'s assigned address, set up the hardware breakpoints. Click on the **Breakpoint** button to display the breakpoint dialog box.



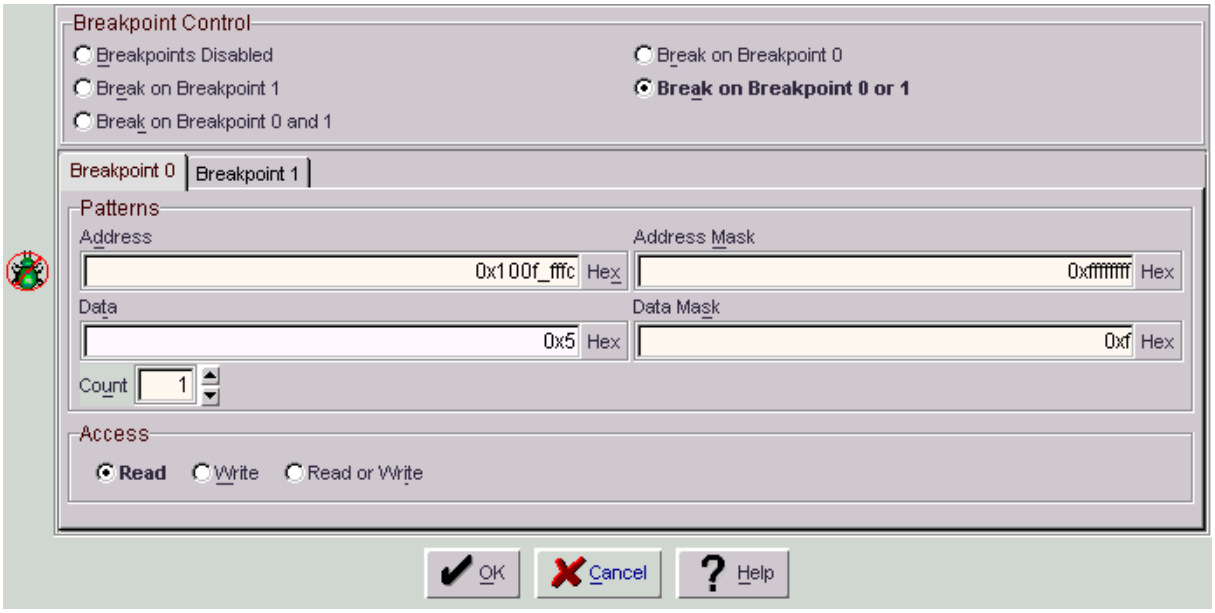
Enable the breakpoint by clicking **Break on Breakpoint 0**. In the **Breakpoint 0** tab panel, type in "**0x100f_fffc**" as the **Address** and "**0xffff_ffff**" as the **Address Mask**. The underscore character (_) is just used as formatting and can be eliminated. Setting the address mask to 0xffff_ffff indicates that we want to match all 32 address lines.

For this step, ignore the data value on the bus by typing "**0**" for both Data and Data Mask. Leave the **count** at 1; and set **Access** to **Read**. This sets up breakpoint 0 to match a CSI read to the RESULT register, regardless of the data value. Click OK to define the breakpoint value.

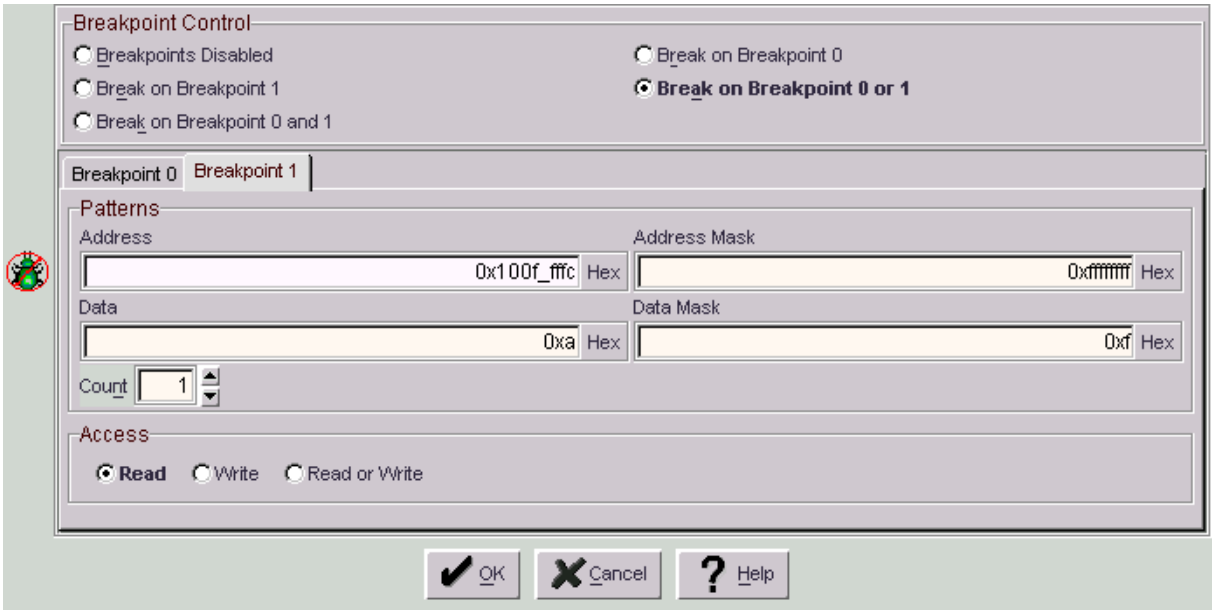
Next, click **Go** a few times. Note that the LED increments and stops. Breakpoint 0 detects a read to the indicated address and halts the processor. The status bar in the lower left-hand of the screen indicates "CSoC running. MCU halted." Clicking the **Go** button restarts the processor and the status bar indicates just "CSoC running."

The two breakpoints units can be used independently or they can work together. In the last example, Breakpoint 0 halted any time the processor accessed the RESULT register. This time, halt the processor any time there is a read access to the RESULT register and the data value is either 0x5 or 0xa.

To accomplish this, set the breakpoint unit to **Break on Breakpoint 0 or 1**. Then, in the **Breakpoint 0** tab, type "**5**" for the Data match value and "**f**" for the Data Mask value.



Then, click the Breakpoint 1 tab. Fill in **"0x100f_fffc**" for the **Address** match value and "**0xffff_ffff**" for the **Address Mask**, or copy these values from the Breakpoint 0 tab. Set Data to "**a**" and Data Mask to "**f**". Click **OK** when finished.



Now, click **Go** from the toolbar. The LED should continue incrementing until it reaches either "5" or "A". Upon reaching either value, the system should halt.

Finally, click Breakpoint again to display the breakpoint dialog box. Change the **Breakpoint Control** to **Breakpoints Disabled**, and click **OK**. Click Go to restart the processor, if it's stopped. Leave the FastChip Device Link Utility running.

Importing CSL Logic Functions

So far, all of the CSL logic functions were created using soft modules from the FastChip library. While the library is extensive, it in no way covers all the potential functions that a designer may wish to implement. To address this, FastChip allows you to import functions created with third-party logic design packages, such as schematic capture and logic synthesis.

Supported Logic Design Packages

The logic design software packages that currently support Triscend include …

- Cadence/OrCAD Capture schematic capture, versions 7.2 and later
- ViewDraw schematic capture
- Synopsys FPGA Express logic synthesis, versions 3.4 and later
- Synplicity Synplify logic synthesis, versions 6.1.3 and later

You can find additional information on using third-party design tools in the following Acrobat file …

`Third_Party_Design_Methodology.pdf`

… stored in the following directory …

`<FastChip Installation Directory>\Docs\Design Methodology.`

Likewise, there is more detailed information on using these tools with FastChip in the various Acrobat files under the following directory …

<FastChip Installation Directory>\Docs\Third-party Flows and Tools

All these tools require a Triscend-specific design library for optimal results. For the FPGA Express and Synplify logic synthesis packages, the design library is provided by the synthesis vendor. For schematic capture, Triscend provides the design libraries.

The schematic capture libraries are found under

<FastChip Installation Directory>\Data\Libraries\Schematic

Documentation on the libraries is available under

<FastChip Installation Directory>\Docs\Triscend Tool Processes

… as the two following Acrobat files …

Triscend Primitives.pdf

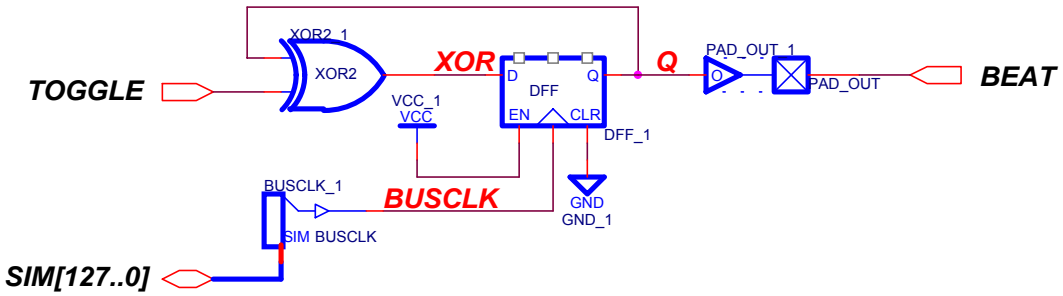
Triscend Schematic Macros.pdf

The various third-party logic design packages create an EDIF 2.0.0 netlist file, which FastChip can read and convert to a custom-created soft module. The next few steps describe the process to import an EDIF design.

Creating a Custom 'heartbeat' Module

In this example, assume that we want to create a custom logic function that toggles the decimal point on the LED display every time that the processor writes to the RESULT register. The logic design is purposely trivial, as shown in the OrCAD Capture schematic blow.

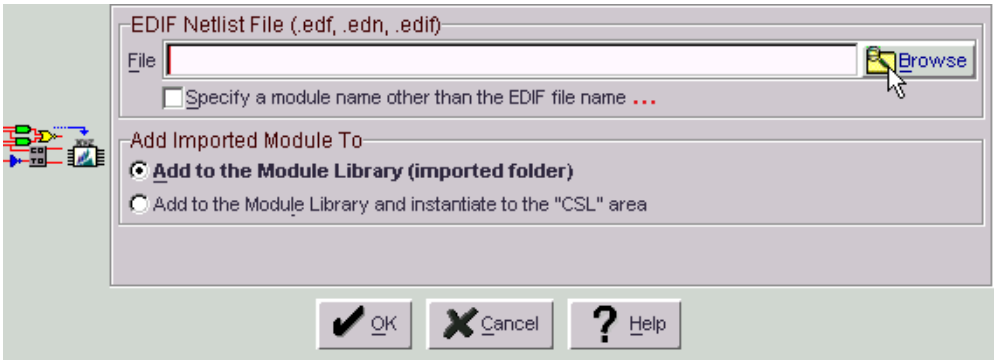
Essentially, when TOGGLE is High, the flip-flop Q toggles on the rising edge of BUSCLK. The flip-flop output, Q, connects to an output pad called BEAT. The source for BUSCLK is the CSI bus clock output, shown connecting to the CSI bus primitive, also called BUSCLK. All CSI bus primitives have a 128-bit bus port just for simulation purposes. The bus, SIM[127..0], does not actually appear in the graphical module interface created FastChip module. Neither does the port called BEAT because it is already connected to an output pad. The port called TOGGLE is an unconnected input port. This port will appear in the graphical module interface.



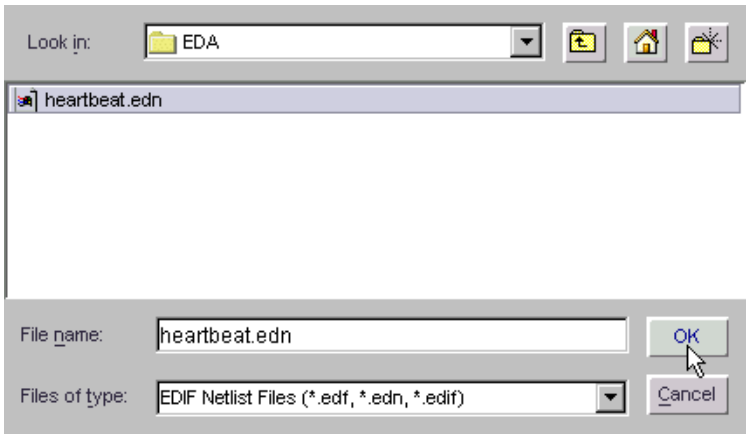
Importing the Design

Assume that this schematic has already been saved as an EDIF file. To bring this function into FastChip, click the **Import** button from the toolbar.

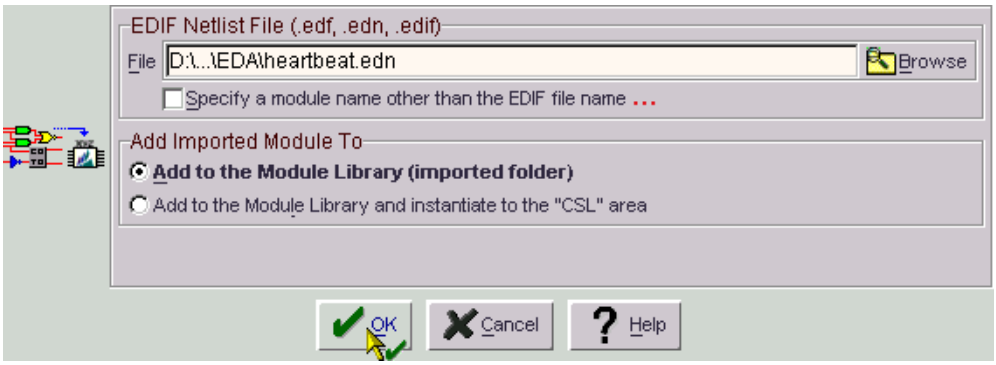
The resulting dialog asks you to specify the name of the EDIF netlist file. For this example, use the EDIF file saved as part of the MyDesignA7 project. Click **Browse**.



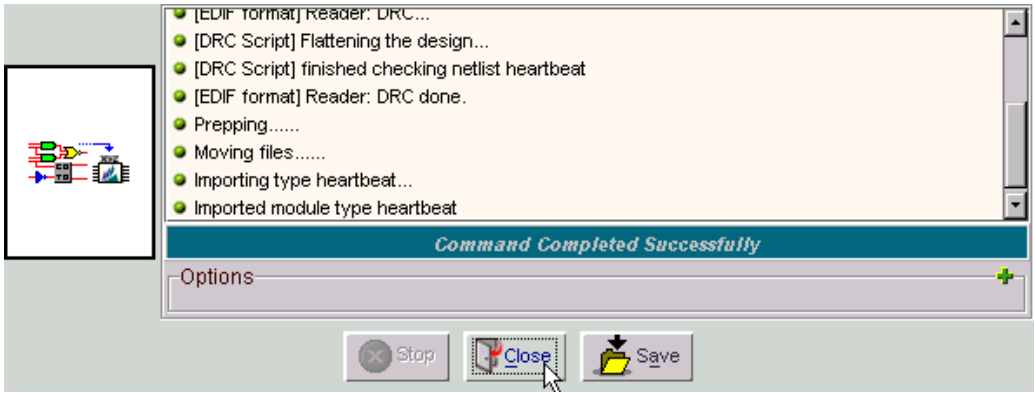
In the resulting file chooser dialog box, search for and select the file called **heartbeat.edn** under the directory **`<FastChip Installation Directory>\Projects\MyDesignA7\3rdParty\EDA`**. Click **OK** when finished.



The name of the EDIF netlist file appears in the Import dialog box. By default, FastChip uses the name of the imported file as the name of the newly-created module. The dialog allows you to specify another name and allows you to specify where the imported module will be placed. For this example, use the default settings and click **OK**.



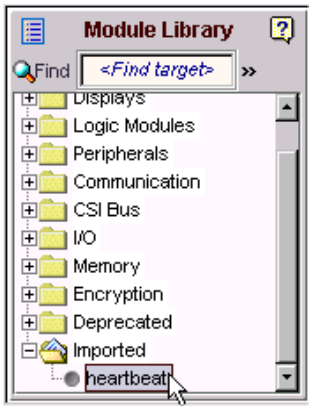
FastChip reads the specified EDIF netlist file, then flattens the logic, and then performs a design-rule check (DRC) for errors. Click **Close**.



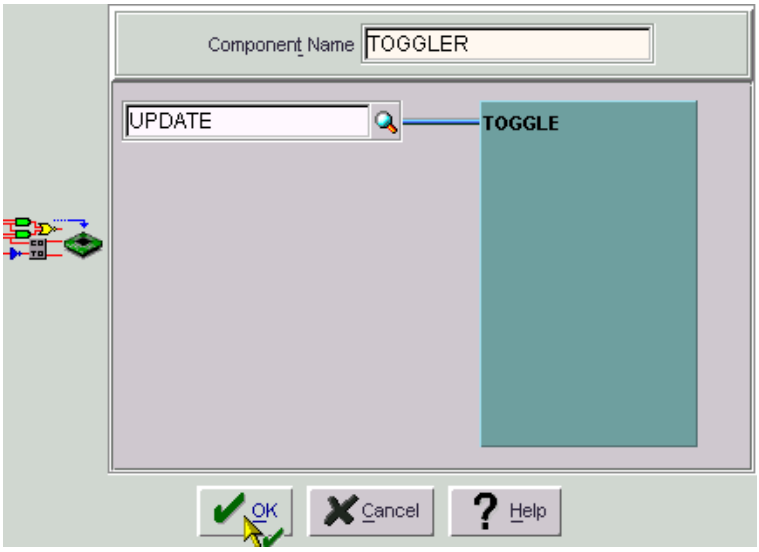
FastChip saves the imported function in the soft module library under Imported. Unlike the remainder of the library, the Imported library directory is unique to this design. It contains any and all imported functions specifically for this project. Because the design is imported as a module, you can instantiate the same module any number of times within a design.

Using the New Module

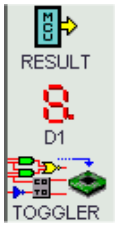
To use the newly imported module, double-click on **heartbeat** in the **Module Library** area.



Customize the heartbeat module by type "**TOGGLER**" as the **Component Name**. Connect the TOGGLE input to a new signal by typing "**UPDATE**" in the connection port text box. Click **OK** when finished.



Observe that the module was added to your CSL window and that the number of resources increased.

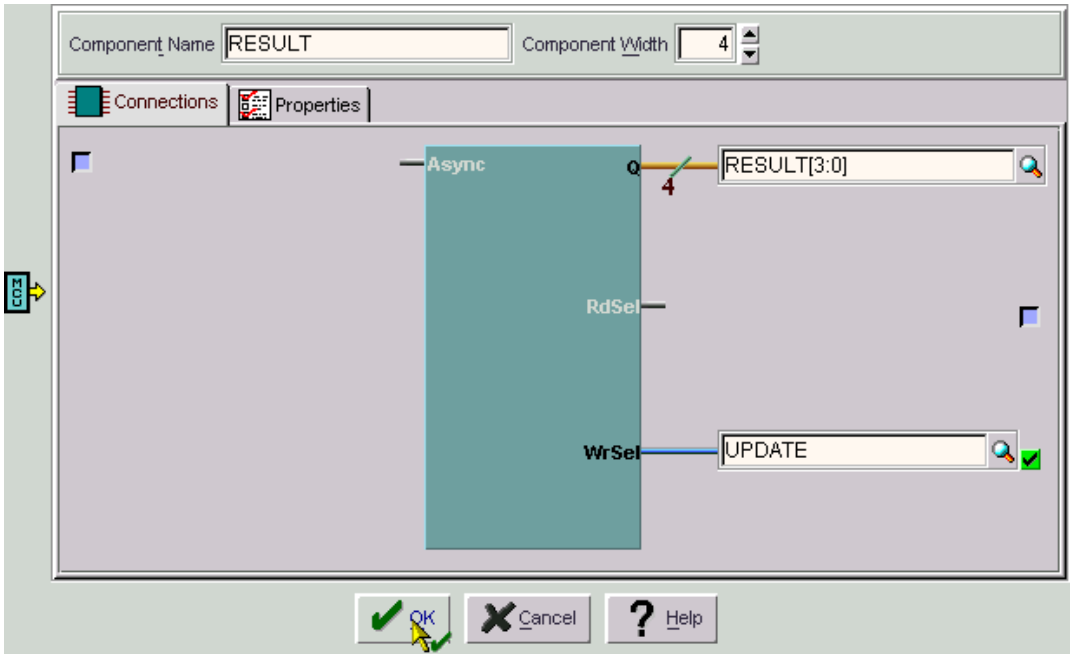


Adding this simple module changed the CSL logic. Observe the Bind button indicates that the binding is "(not current)". Don't Bind the design just yet.

Connect the new TOGGLER module to the RESULT register. Click the **RESULT** module to edit its settings.

The RdSel and WrSel outputs on the RESULT module are optional, and as such, there is an option box to the right of these outputs. Check the box next to **WrSel**, enabling the connection text box.

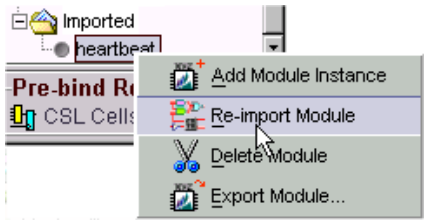
Type "**UPDATE**" in the text box to connect to the signal that drives the TOGGLER module. Click **OK**.



Other Imported Library Options

FastChip allows you to perform various functions on a library module. Imported modules support four functions while the standard FastChip library only supports adding and exporting a module.

To operate on a library module, click the right mouse button on the selected module. The resulting menu allows up to four different functions.



- **Add Module Instance** opens the module edit dialog and will add the module to your CSL logic when you click OK.
- **Re-import Module** is useful after modifying an imported module and created a new EDIF file. This option re-imports the new EDIF file based on the previous import settings.
- **Delete Module** removes an imported module from the library tree.
- **Export Module** allows you to create a Verilog or VHDL simulation model for the module, based on the module's parameter settings.

Finishing the Design

Complete the tutorial design by …

- Placing the output pad embedded within the heartbeat module on package pin 103.
- Clicking Bind to re-compile the CSL logic design.
- Creating a configuration file using FDL.

- Downloading the configuration file to the A7 Evaluation Board using FDL and the VisionPROBE cable.

After completing these steps, the design should behave much like before except that the decimal point on the right of the LED display should toggle every time the display updates. The decimal point should light up when the LED displays an even number.

Downloading to Flash

Previous revisions of the design were downloaded and executed from the A7's internal SRAM. In most designs, the application program is too big to fit into internal SRAM.

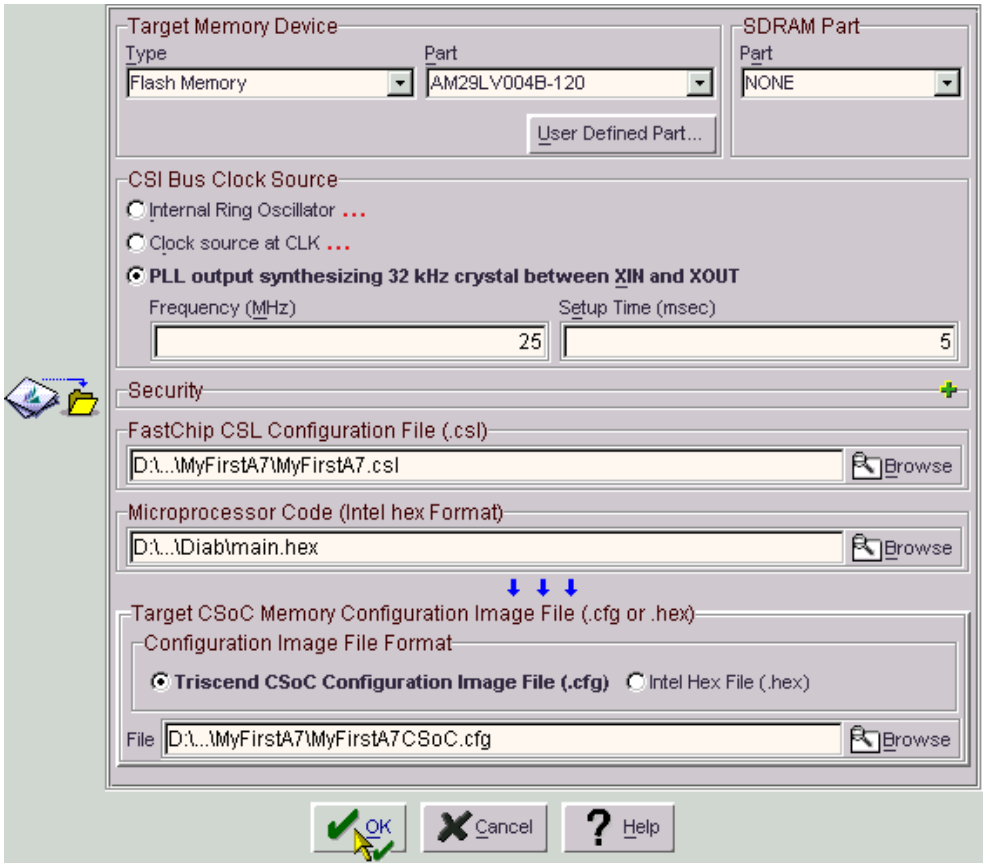
FastChip allows you to directly program a Flash device connected to the A7 device via the VisionPROBE cable. This is primarily useful during development and debugging. For high-volume production, you can also create an MCS-86 Hex file, which is compatible with most Flash and EPROM device programmers. That way, you can provide your manufacturing group with a Hex data file and they can program a large number of Flash devices on a device gang programmer.

To create a configuration image for Flash, invoke FDL and click the **Configuration** icon from the toolbar.

Set **Type** to **Flash Memory** and select the Flash **Part** number. Specifying the entire Flash part number tells FastChip which Flash programming algorithm to use, the size of the Flash memory sectors, and the access time for the Flash device. Most A7 Evaluation Boards have an AM29LV004B-120 Flash device in socket U11. Double-check the part number for the Flash memory device installed on your A7 evaluation board and make the appropriate setting.

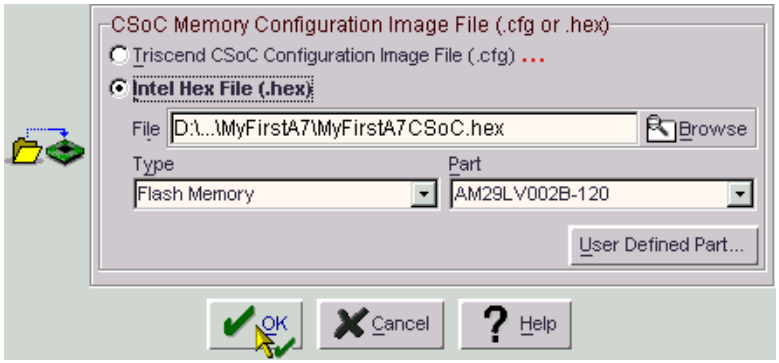
Again, use the **PLL output** and set the **Frequency** to **25** MHz and the crystal **Setup** time to **5** ms. The CSL configuration file and processor code settings remain the same. However, for the **Triscend CSoC Memory Configuration Image File**, choose to create an **Intel Hex File (.hex)**. This Hex file can be used by your manufacturing organization to program the specified Flash devices in volume.

Click **OK** when finished.

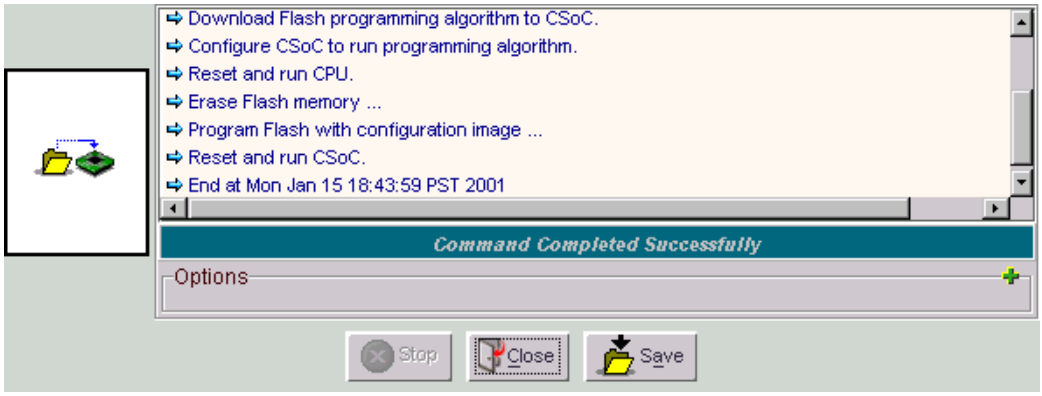


To download the design to the A7 Evaluation Board, click **Download** from the FDL toolbar.

Choose the **Intel Hex File** option, and then click **OK**.



FDL downloads the Flash programming algorithm into the A7 device, using the CSoC to provide the low-level Flash control. The A7 CSoC erases the Flash device, loads the configuration image, then resets itself to begin executing you application.



To prove that you are executing from Flash, turn off the A7 Evaluation Board and then re-apply power. The tutorial design should start executing. The LED should increment and the decimal point should alternately blink on and off every time the display updates.

Part 1 Complete!

Congratulations on creating your first A7 Configurable System-on-Chip (CSoC) design! So far, you assembled a small design using the FastChip library and downloaded it into working hardware. You viewed the internal registers and logic using the FastChip Device Link (FDL) utility. You added a custom logic function imported from schematic capture and downloaded the completed design to Flash on the Triscend A7 Development Board.

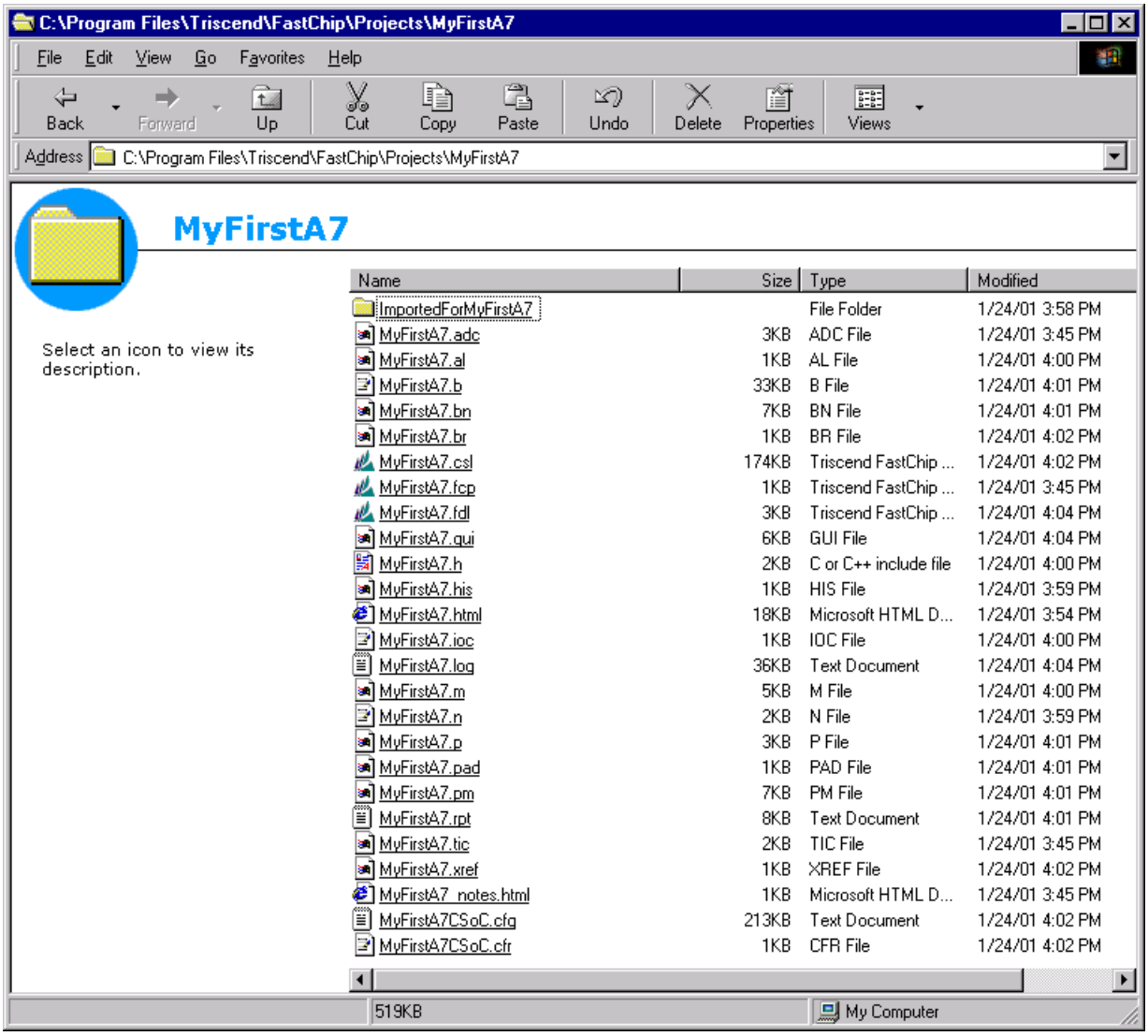
PART 2: Diab Compiler and visionCLICK

If you have not completed Part 1…

you can continue with Part 2, but only with the **MyDesignA7** project provided with FastChip. Whenever the tutorial refers to MyFirstA7, perform the same steps using the MyDesignA7 project.

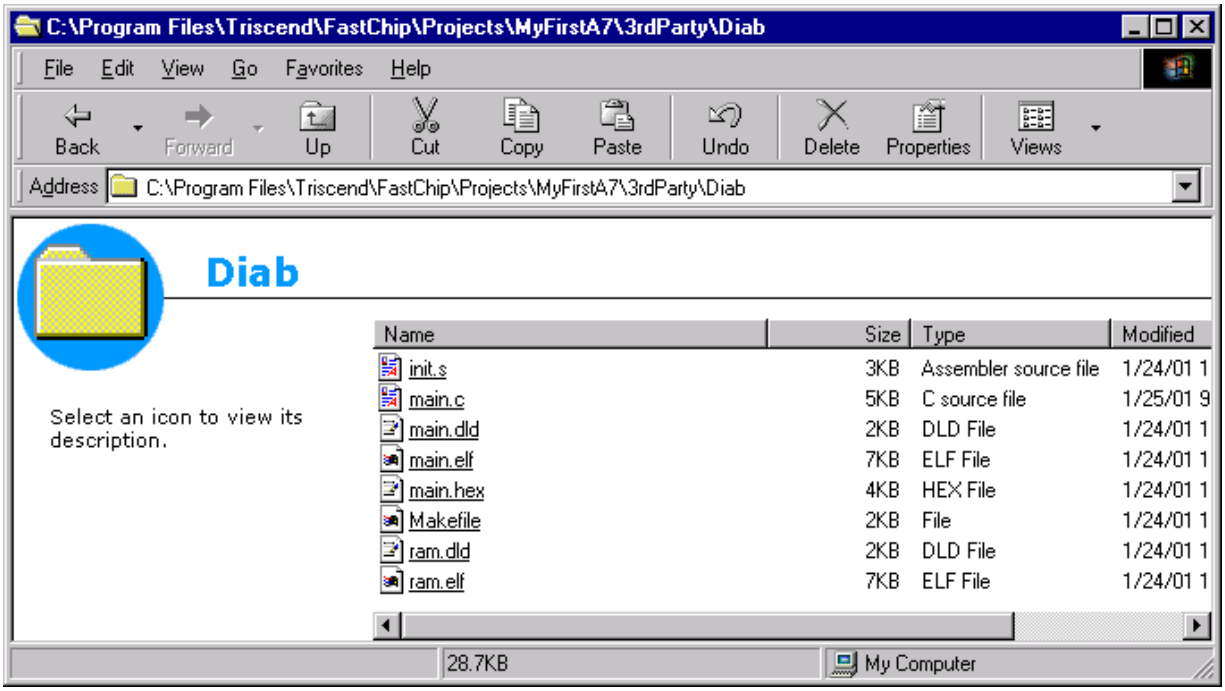
In Part 1, you completed a simple design using only FastChip and then downloaded and debugged the design using the FastChip Device Link (FDL) utility. In that exercise, you imported the application software, **`main.hex`**, from another project, MyDesignA7, and used the FDL Configuration function to combine the application code with the CSL design.

At this point in the tutorial, your MyFirstA7 project directory should look similar to the following graphic. FastChip generated all files in this directory—including the Folder ImportedForMyFirstA7. In Part 2 you will be working with the software application.



Part 2 uses the application example in the **\Diab** directory, which is designed to work with the Diab DCC compiler from WindRiver. Since A7 has a standard ARM7TDMI core, any standard compiler that supports ARM7TDMI can be used to compile application code for A7. This release of FastChip offers application examples with sample makefiles for both the ARM ADS compiler and the Diab DCC compiler from Wind River Systems. The tutorial exercises assume that you are using the Diab DCC Compiler.

In order to complete Part 2, you must have the Diab DCC Compiler installed on your host computer. Review the ***Getting Started with A7*** manual for installation instructions. In addition to the Diab compiler, you also need a text editor.



The **\Diab** directory contains the following files:

- **init.s:** This is a low-level initialization file that sets up a number of housekeeping items such as the initial vector table and the initial stack pointer.
- **main.c:** This is the main application source file. Later in the exercise we will make some modifications to this file to illustrate tools usage.
- **Makefile:** This sample makefile contains the rules for compiling and linking the application.
- ***.dld** files: These are Diab linker command files which instruct the linker to link your application to the A7's memory map. The **main.dld** file links the code for executing from external Flash memory; the **ram.dld** links the code for executing from the A7's internal SRAM.
- ***.elf** files: These are standard object files generated by the Diab compiler and linker. **Main.elf** was linked using the **main.dld** linker file and **ram.elf** was linked using information in the **ram.dld** file.
- **main.hex**: This file is the application code in Intel Hex format. This file is needed by FDL for combining with CSL configuration file to generate the final downlodable image. Since this release of FastChip uses only Intel Hex format files, **main.hex** was converted from **main.elf** during the build process.

Understanding the Application Code

Taking a closer look at the application code main.c, it contains the following routines:

The **`init()`** function initializes the dedicated resources on A7, such as the timers, the UARTs, etc. In this application, the only dedicated resource used is the Watchdog Timer. The **`init()`** function should be expanded if additional dedicated resources are used in the application.

The **`clockFreqHz()`** function returns the clock frequency in Hertz. In this application, the watchdog timer is programmed to generate an interrupt every 500 ms (half second). The watchdog driver code calls this function to calculate the appropriate watchdog timer time-out value.

The **`watchdogTimer()`** routine configures the watchdog timer to generate an interrupt once every 500 ms. Additionally, the Watchdog Control Register is configured to prevent the watchdog timer from generating a device reset in the event of a watchdog timeout.

The **`IRQ_Handler()`** function is the C-language part of the Interrupt Service Routine (ISR). This function is called from the **`IRQ_Wrapper()`** function inside **`init.s`**. Because this application only uses one interrupt—generated by the Watchdog Timer— the control logic is relatively simple. Simply clear a bit in the WATCHDOG_CLEAR_REG register, increment the RESULT register, and then return. There is no software prioritization of interrupts in this simple application.

The **`C_Enrty()`** function is the main application function. It initializes both hardware and software and then enters an infinite **`while`** loop. Most embedded applications never end.

The **`while`** loop runs until the watchdog interrupt is generated. The interrupt service routine increments the RESULT register, which is implemented in the A7's CSL logic. This, in turn, causes the 7-segment display value to change on the A7 evaluation board. Control is then returned to the **`while`** loop until the next interrupt occurs.

Editing main.c

In this section, we'll make a few changes to the source file, compile it and debug it with the visionCLICK debugger.

Edit the following locations inside the **`main.c`** module to do the following.

- At the top of the **`main.c`** module, there is a directive that includes a header file. Because this code was copied from another project, the referenced file is "MyDesignA7". Change this to "MyFirstA7", the name of your FastChip project.
- Add a global variable called **`see_result`**. The value of **`see_result`** is set equal to RESULT inside the while(1) loop. After an interrupt is serviced, RESULT will change. This change will be reflected in the **`see_result`** variable once control is returned to the main loop. You will use visionCLICK to monitor the value of this variable.
- In the original **`IRQ_Handler()`** function, the value of RESULT is incremented by 1. Change this so that RESULT increments by 2.

The required changes are shown in the following illustration in **bold** print. You need a text editor and for this example, Wordpad is sufficient. Start your favorite text editor and open the **`main.c`** file located in the **`3rdParty\Diab`** directory of your FastChip project directory. Make the necessary changes and save the file.

```
/* MyDesignA7.c */

/*
 ************************************************************
 *  MyDesignA7.c
 *  Copyright(C) 2000 Triscend Corporation
 ************************************************************
 */

#include "MyFirstA7.h"

.
.
.

/*
 ************************************************************
 *  GLOBAL VARIABLES
 ************************************************************
 */
unsigned int see_result;

.
.
.

void IRQ_Handler (void) {

    // if (IRQ is a Watchdog Interrupt) then
    //     Clear Watchdog Interrupt and indicate timeout.

    if (GET_BIT (INT_IRQ_STATUS_REG, IRQ_WATCHDOG_BIT)) {
        SET_BIT (WATCHDOG_CLEAR_REG, WD_INT_CLR_BIT);

   // If the watchdog timer times out, increment the
   // RESULT register
        TWORD(RESULT) += 2;
    }
}

/*
 ************************************************************
 *  MAIN FUNCTION
 ************************************************************
 */
void C_Entry () {

    // initialize the A7's dedicated resources used in this
    // design
    init ();

    // initialize variables
    TWORD(RESULT) = 0;

    // setup and start watchdog timer timer with interrupts
    watchdogTimer(TIMER_MS);

    // primary task waits until Watchdog Timer interrupt occurs
    while (1) {

        /*****************************************
         ***  AN EMBEDDED APPLICATION NEVER ENDS  ***
         ***           . . . . . .               ***
         ***    Add your application code here!   ***
         *****************************************/
    see_result = TWORD(RESULT);

    }
}
```

You now must recompile the application. To do so, perform the following steps.

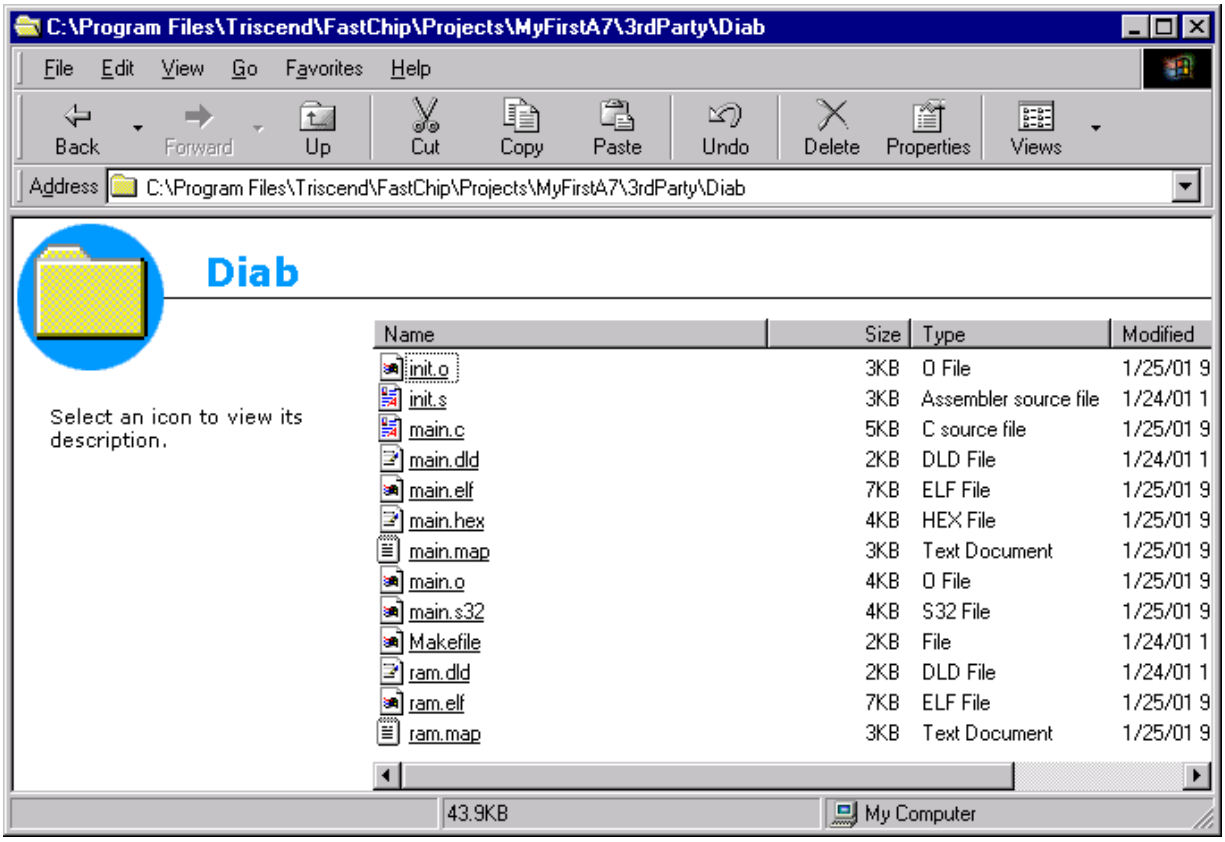
- Save the changes to the **main.c** file
- Open a DOS or command prompt window
- Change to the correct directory

  cd "<Installation Directory>\Projects\MyFirstA7\3rdParty\Diab"
- Type the command **make clean** to remove any unnecessary files
- Type the command **make**

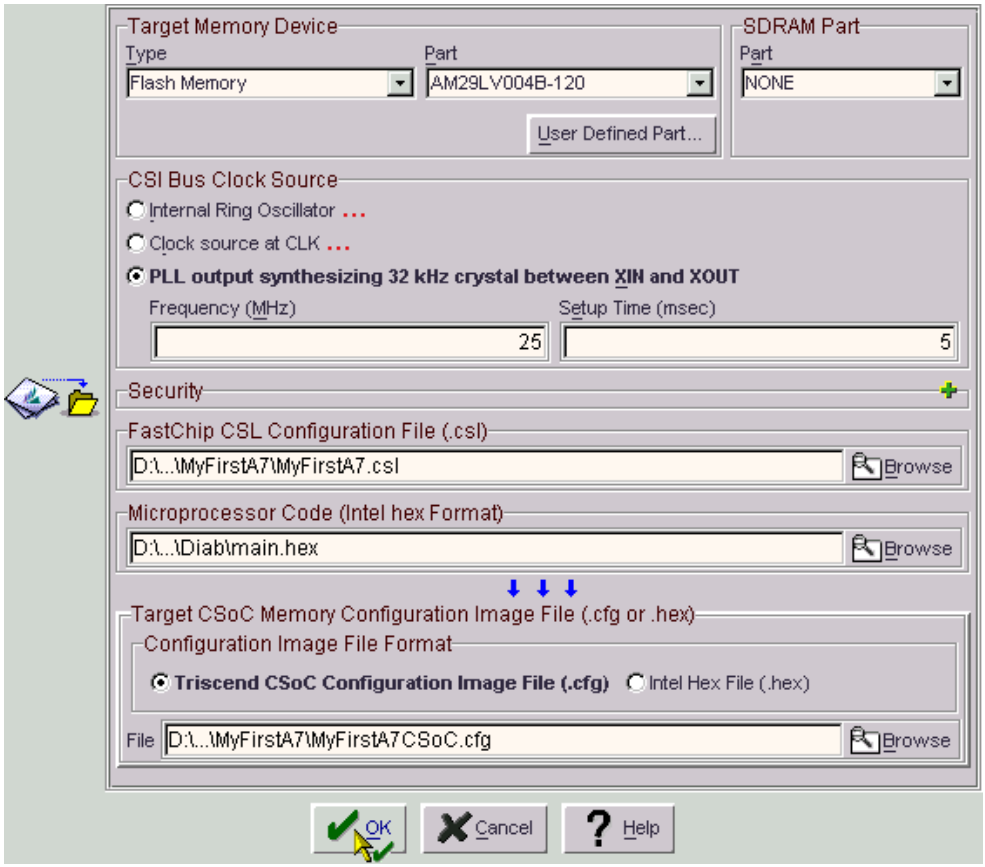
If you have correctly installed the Diab compiler, the application will be compiled and linked. If you experience error messages related to the Diab tools, make sure that you have set the path to include a pointer to the **\bin** directory under the Diab install directory. The Diab install program should have set the path. Also make sure that there is an environment variable called **LM_LICENSE_FILE** pointing to your **flexlm\license.dat** file. Finally make sure that the **license.dat** file contains a valid license for Diab. Refer to the **Getting Started A7 Manual** for Diab installation information.

```
C:\Program Files\Triscend\FASTCHIP\Projects\MyFirstA7\3rdParty\Diab>make clean
rm -f *.o *.elf *.map *.hex *.s32

C:\Program Files\Triscend\FASTCHIP\Projects\MyFirstA7\3rdParty\Diab>make
das -tARMLS:simple -g -o init.o init.s
dcc -tARMLS:simple -I../../../../include -I../../ -g -c -o main.o main.c
dld -tARMLS:simple -o main.elf init.o main.o main.dld -lc -m2 > main.map
ddump -Rv main.elf -o main.s32
mhc -h main.s32 main.hex
mhc Ver1.0.3 Motorola S data <-> Intel Hex data Converter.
(C)1997-1998 Cow Project Workshop.
Convert flag -h
Input file main.s32
Output file main.hex

Making Intel Hex file.
Input file size=3954
Output file size=0
Finished!
dld -tARMLS:simple -o ram.elf init.o main.o ram.dld -lc -m2 > ram.map
```

Once the application has been recompiled and re-linked, the contents of the directory should contain the following files. Check that the timestamp of the files are correct.



Repeat the FDL exercise from Part 1 of the Tutorial to reprogram the Flash with the latest reversion of the application. You do not need to rebind the FastChip project since you did not make any changes in the CSL portion of the project. You need only to change the FDL configuration.



In Part 1, the **Microprocessor Code (Intel Hex Format)** entry pointed to the **`main.hex`** file in the MyDesignA7 project. You need to change this entry to point to the **`main.hex`** that you just built. After you make this change, click **OK** and start the FDL download utility to download to the target device. Once download is finished, the 7-segment display should start incrementing by two instead of by one.

Debugging with visionCLICK

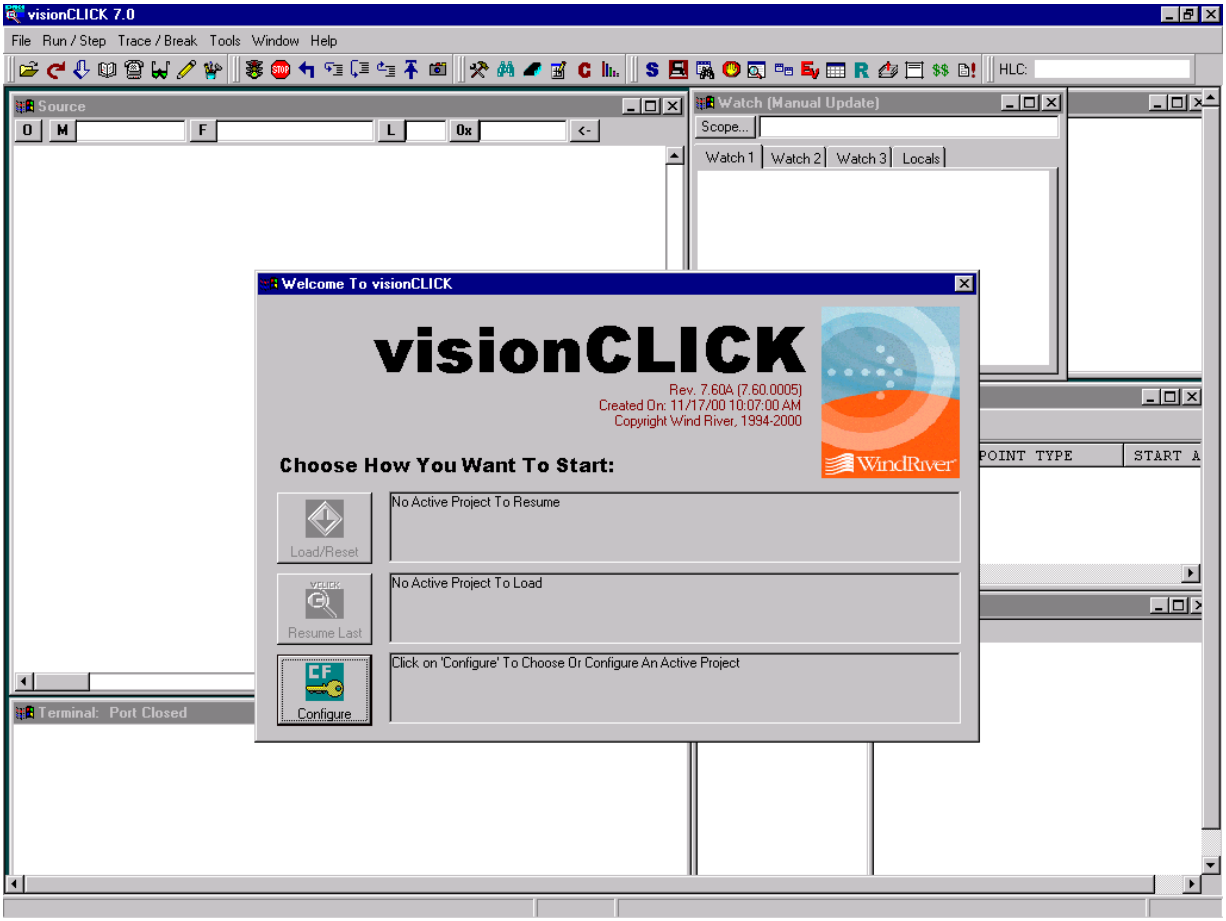
In the next part of the tutorial, you will debug the application using the visionCLICK source-level debugger. The purpose of this tutorial is to demonstrate some basic visionCLICK capabilities. For detailed information on how to use visionCLICK, please refer to the visionCLICK manuals included on the visionCLICK CD-ROM.

By now, you should have installed visionCLICK. The default installation directory for visionCLICK software is **c:\estii**. This part of the tutorial refers often to the **\estii** install directory, substitute it with the actual installation directory when needed.

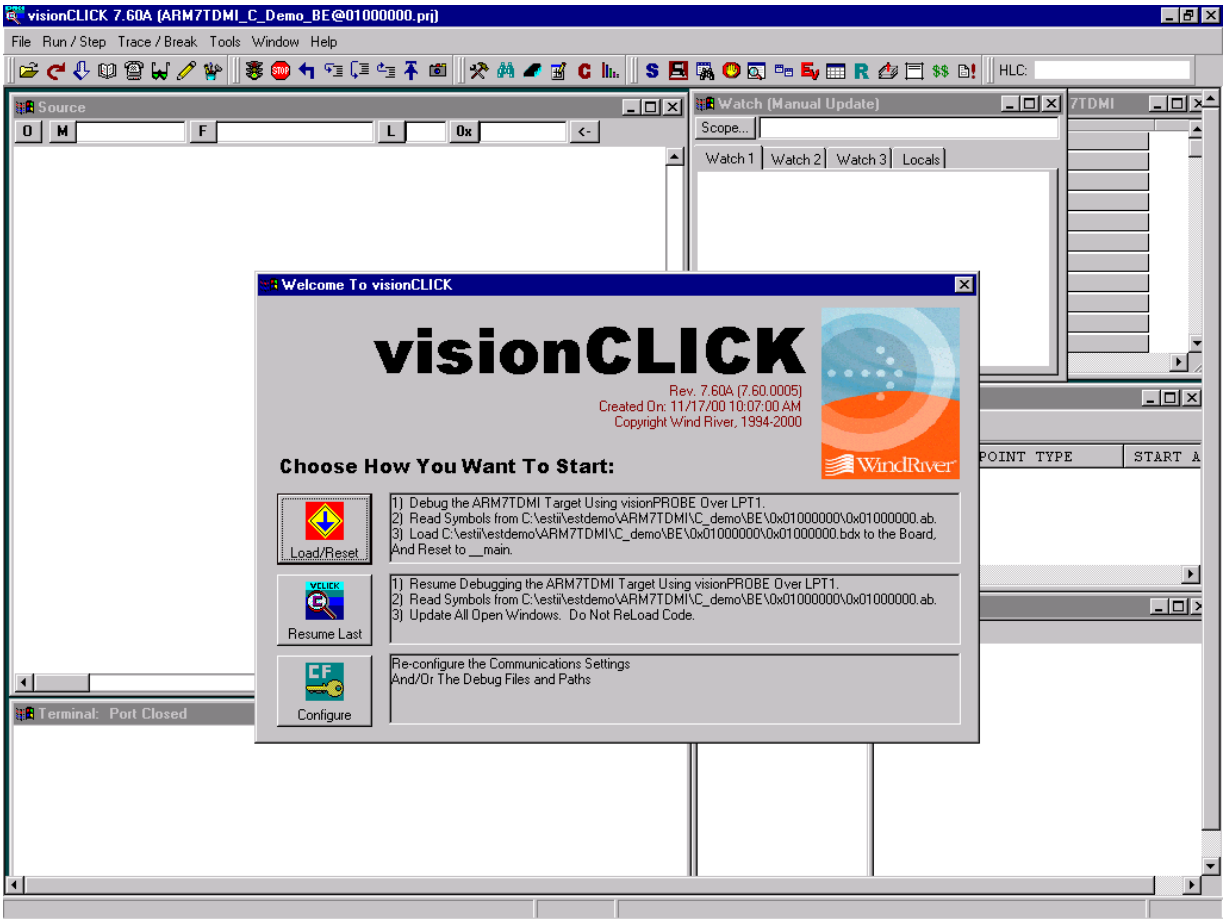
When you start visionCLICK for the first time, some general set up is required. This tutorial describes the required configuration steps. Then, you will download a version of the application you modified in the previous steps of this exercise.

Both visionCLICK and FDL communicate with the target board via visionPROBE and the two programs cannot share that link. Only one can be running at any given time. Make sure that the FDL utility is not running when you start visionCLICK.

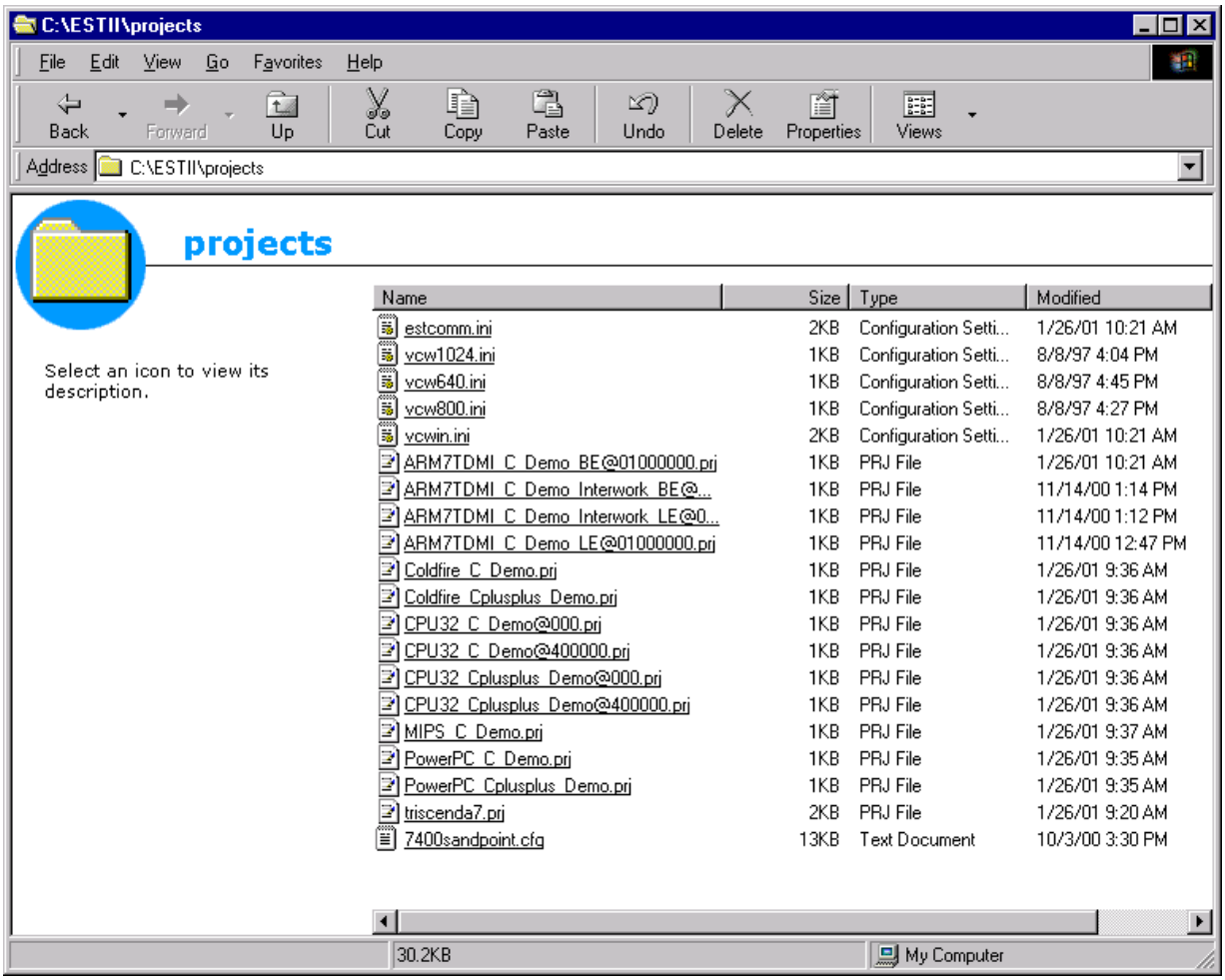
The first step is to start visionClick.



The first time that you start visionCLICK there will not be an active project. Once you complete the configuration, visionCLICK generates a file, **estii\projects\active.prj**. The first time you run visionCLICK, there is only one active button in the Start Window – Configure. Once an active project has been selected, all three buttons will be active when you start visionCLICK. If your display does not match the above graphic, it means that visionCLICK was previously started and a project was selected. In this case, the visionCLICK window may look like the following.

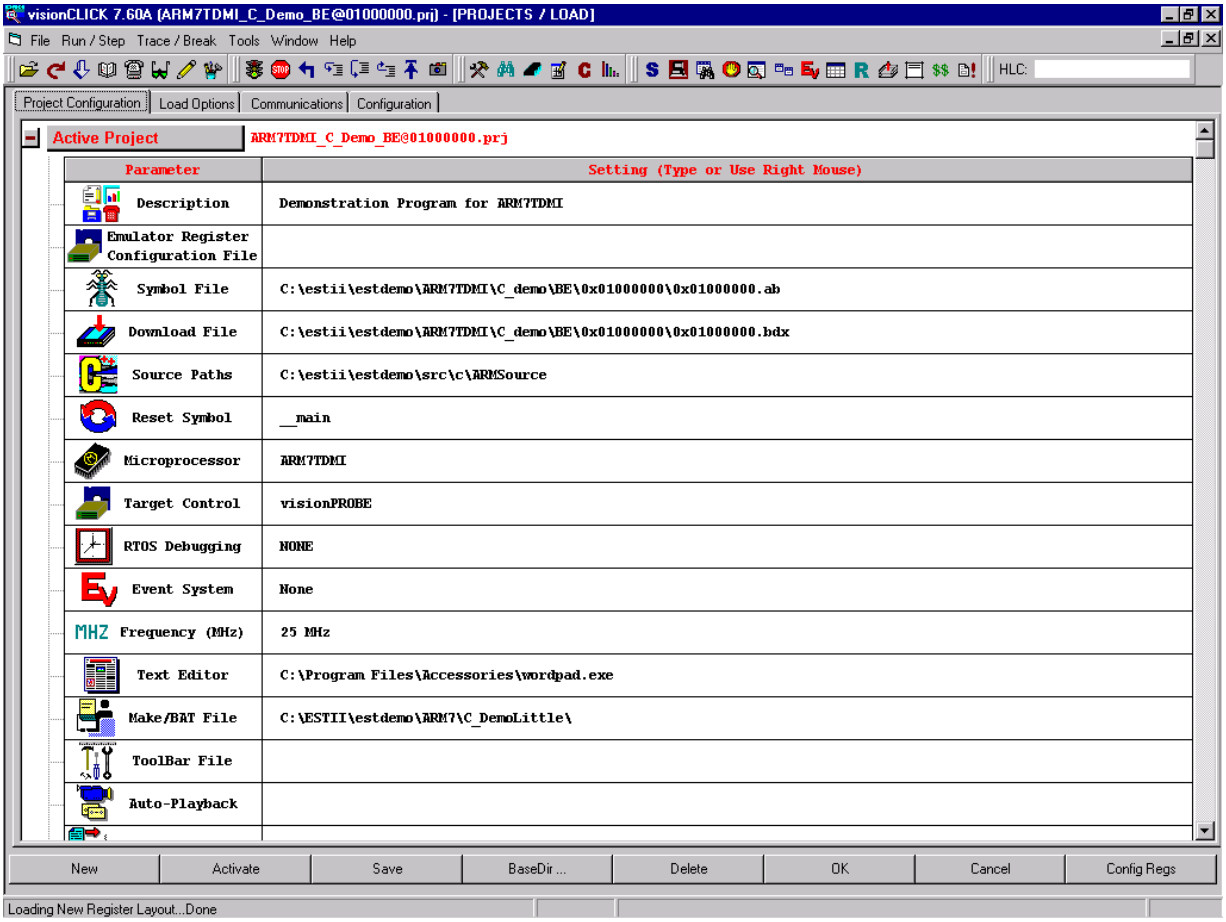


Your display will look different but the configuration steps will be the same. Look at the contents of the **`C:\estii\projects`** directory.

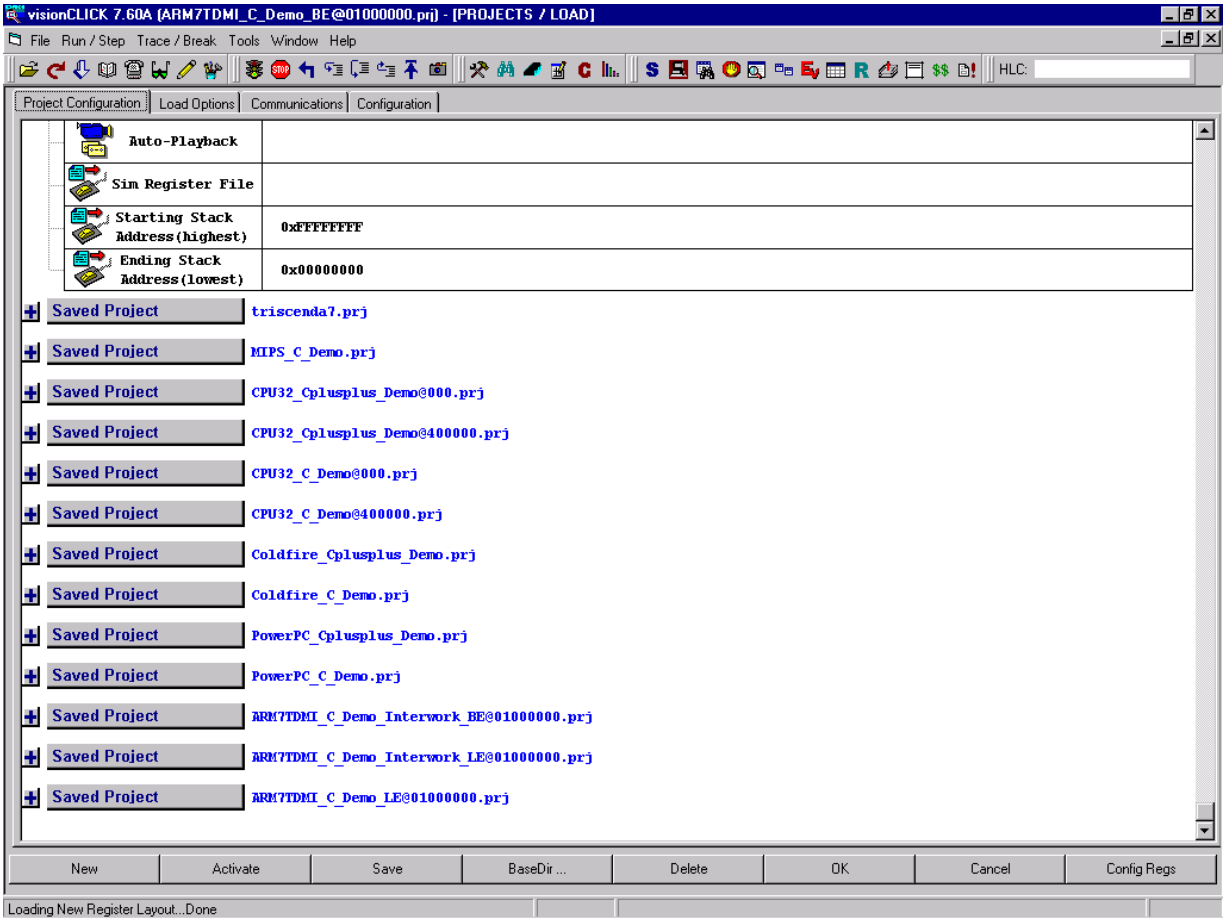


It contains a number of predefined projects, including **`triscenda7.prj`**. When you configure visionCLICK the first time, visionCLICK will select "`ARM&TDMI C Demo BE@01000000.prj`" as the default active project. It selects this project because it is first **`*.prj`** file in the list. Select the **`triscenda7.prj`** file.

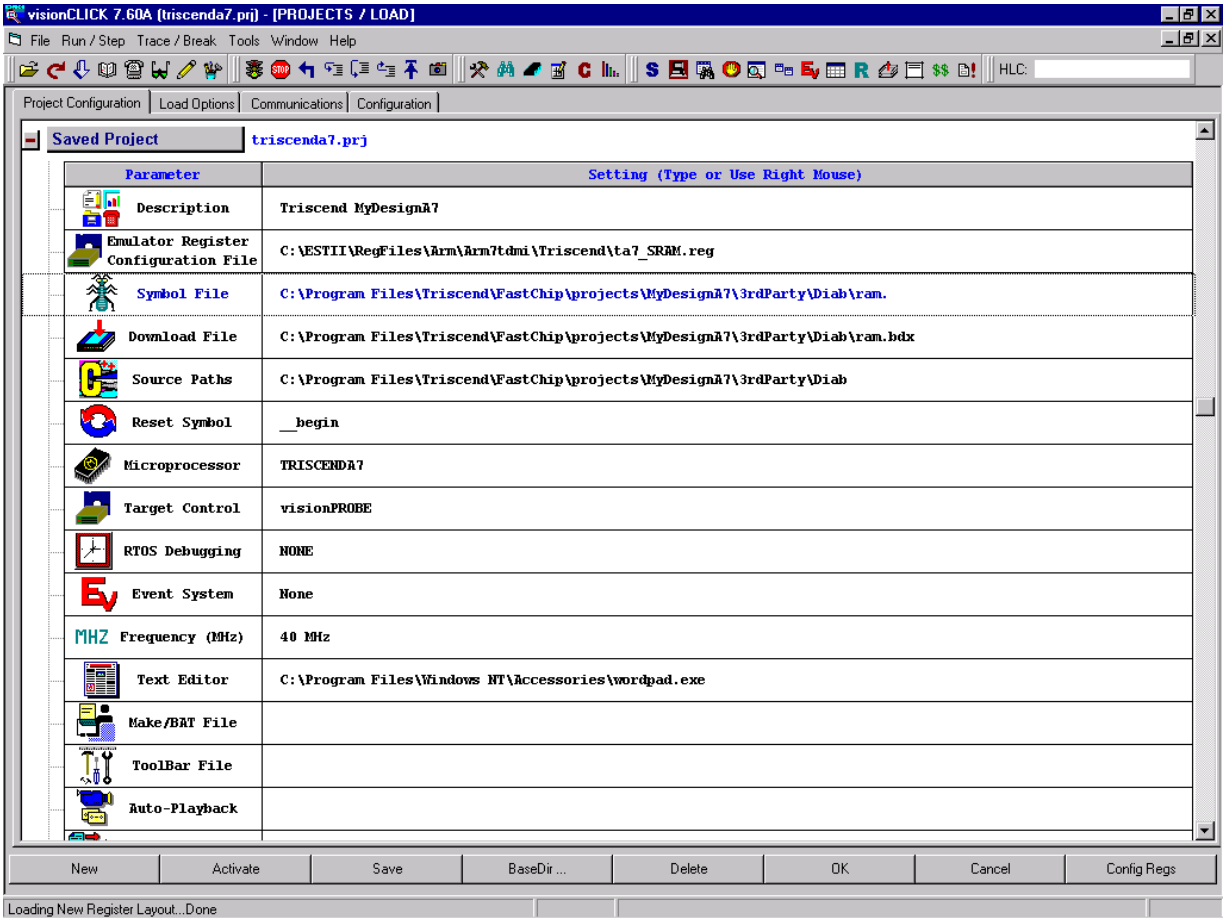
After starting visionCLICK, click the **Configure** button from the Welcome window to change the default configuration. The configuration window appears. Maximize this window so that you can see all the entries.



Scroll down to the bottom of this window until you locate the other projects.



Locate the triscenda7.prj file and double click on the file name. The opens the triscendA7 project as a "Saved Project". In this part of the exercise, you will modify some project settings and then make this your active project.



Entries in the project file are already set to use the MyDesignA7 FastChip project. You need to change several entries to point to the MyFirstA7 FastChip project you are currently working using. Previously when you built the application, Diab created a number of files in your project directory. One of the files created is named **`ram.elf`**. This file contains both the downloadable code and the application symbol table information. The visionCLICK program requires this information but uses proprietary file formats. The first two entries below relate to those files.

The following entries should be changed:

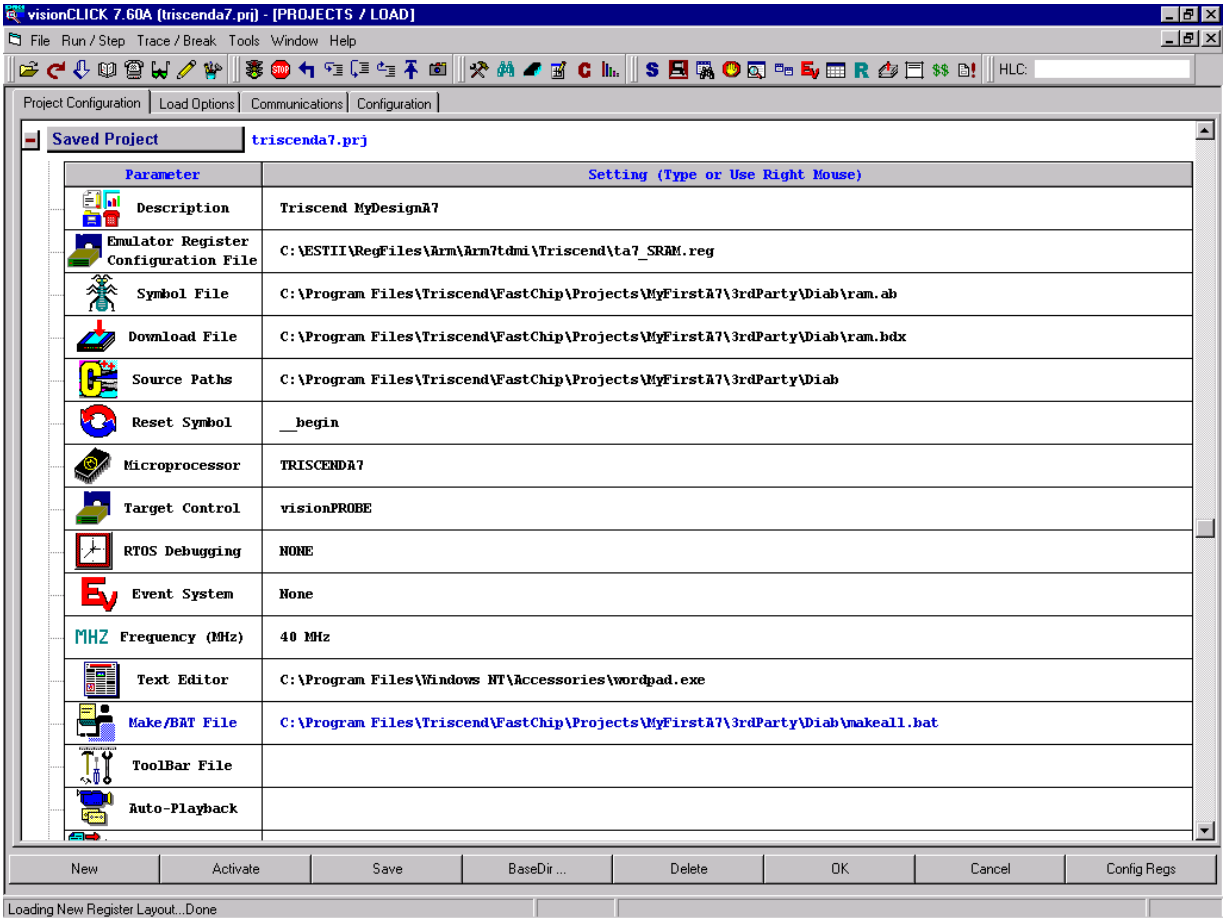
- **Symbol File** – edit the path to point to the **`ram.ab`** file. Place the cursor over this entry and click the right mouse button. Select **Edit this Setting** and change the directory to point to the **MyFirstA7** project instead of MyDesignA7. Be sure that the value points to the correct drive. The **`ram.ab`** file will contain all the application symbol information. It does not exist now. You will create it below.
- **Download File** - edit the path of the **`ram.bdx`** file. Again, place the cursor over this entry and click the right mouse button. Select **Edit this Setting** and change the directory to point to the MyFirstA7 project. The **`ram.bdx`** file will contain the downloadable file used by visionCLICK. It does not exist now. You will create it below.
- **Source Paths** - edit the path to point to the MyFirstA7 project. If you use full pathnames in the first two entries and if all the source code is located in a single directory—as it is in this exercise—this entry is not required. If source code is located is several different directories, each path should be listed. Pathnames should be separated by a semi-colon (;).

- **Make/BAT File –** This file does not exist yet. Later in the exercise you will create this file. By placing this batch file in your project directory you will be able to rebuild the application from within visionCLICK. For the time being, edit the path to:

  `<FastChip_install_directory>\projects\MyFirstA7\3rdParty\Diab\makeall.bat`

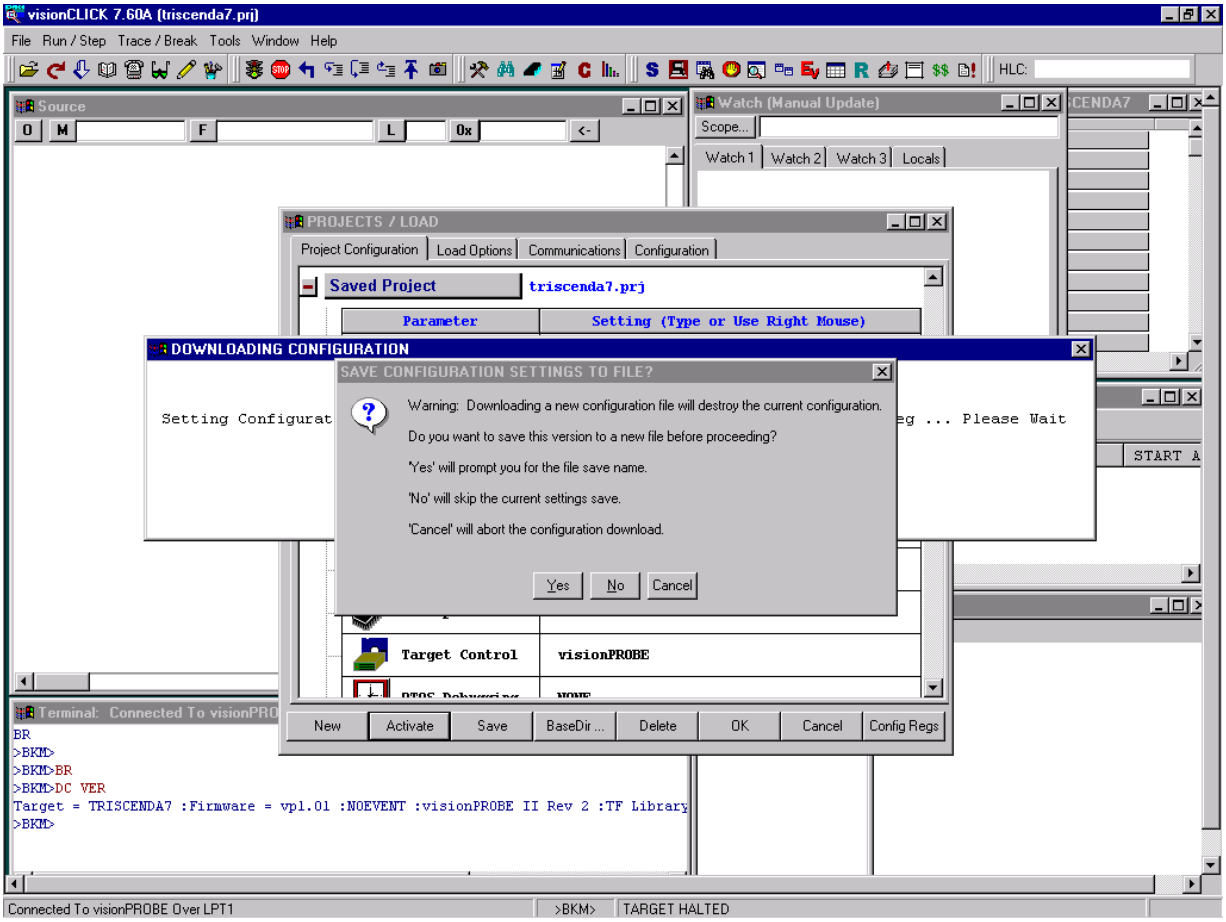
- **Text Editor** – edit the path to point to your favorite text editor. The current settings are for Wordpad installed on a Window NT machine using the default options.

This configuration step only needs to be done the first time you start visionCLICK for a new project. This information is saved in a file called **`triscenda7.prj`** in the **`estii\projects`** directory. Review the visionCLICK manuals for steps required for starting a new project. After you finish the configuration changes, the Project Configuration window should look like the following:

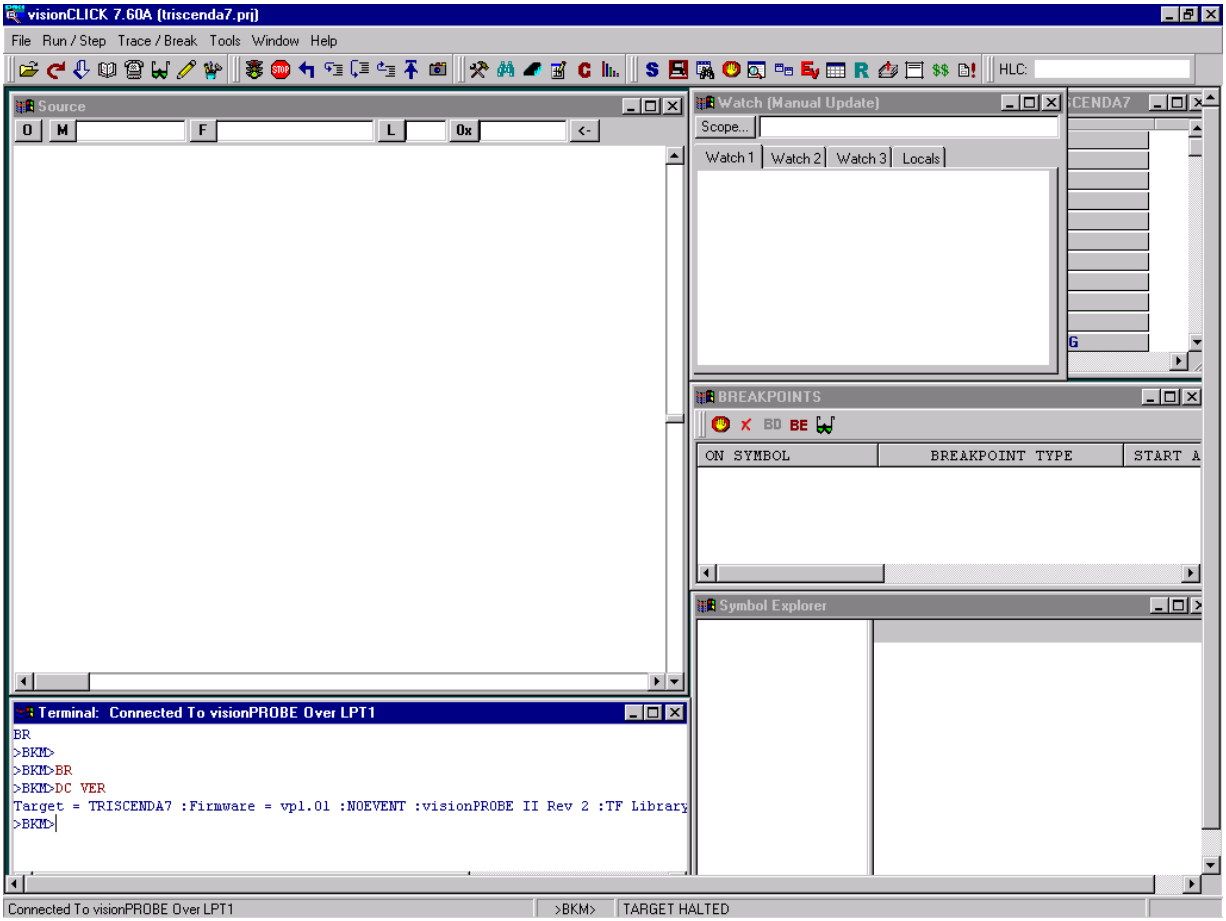


At this point, click **Activate** button.

The configuration window disappears and a temporary dialog box indicates that visionPROBE is being initialized. During this process, visionCLICK reads in a register file. The following dialog box appears.



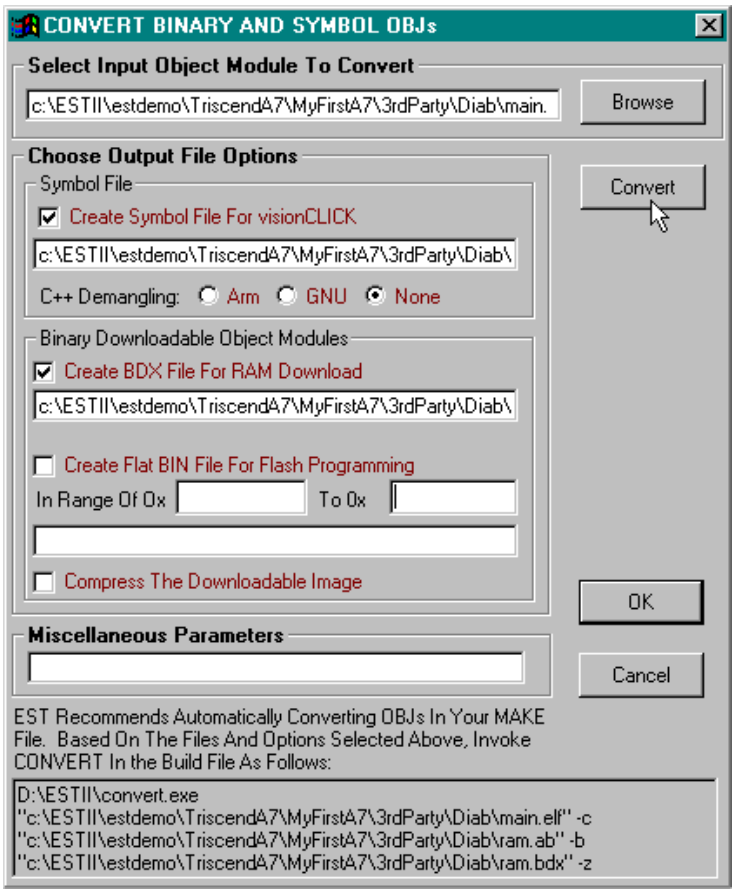
Once this has been completed, your display should look like the following.



One more configuration step is required at this point and need only be performed the first time that you start visionCLICK after changing the project configuration information.

Convert Object Modules

Before you can download the application using visionCLICK you need to convert the **`ram.elf`** file into the proprietary **`ram.ab`** and **`ram.bdx`** files required by visionCLICK. A utility program included with visionCLICK performs this step. From the upper toolbar, select **Tools → Convert Object Modules.** The following configuration window appears.



Edit the three pathnames to change them from MyDesignA7 to MyFirstA7. After you complete this edit, the window should appear as shown.

Click **Convert** to create the symbol and downloadable files. The following DOS box window appears.

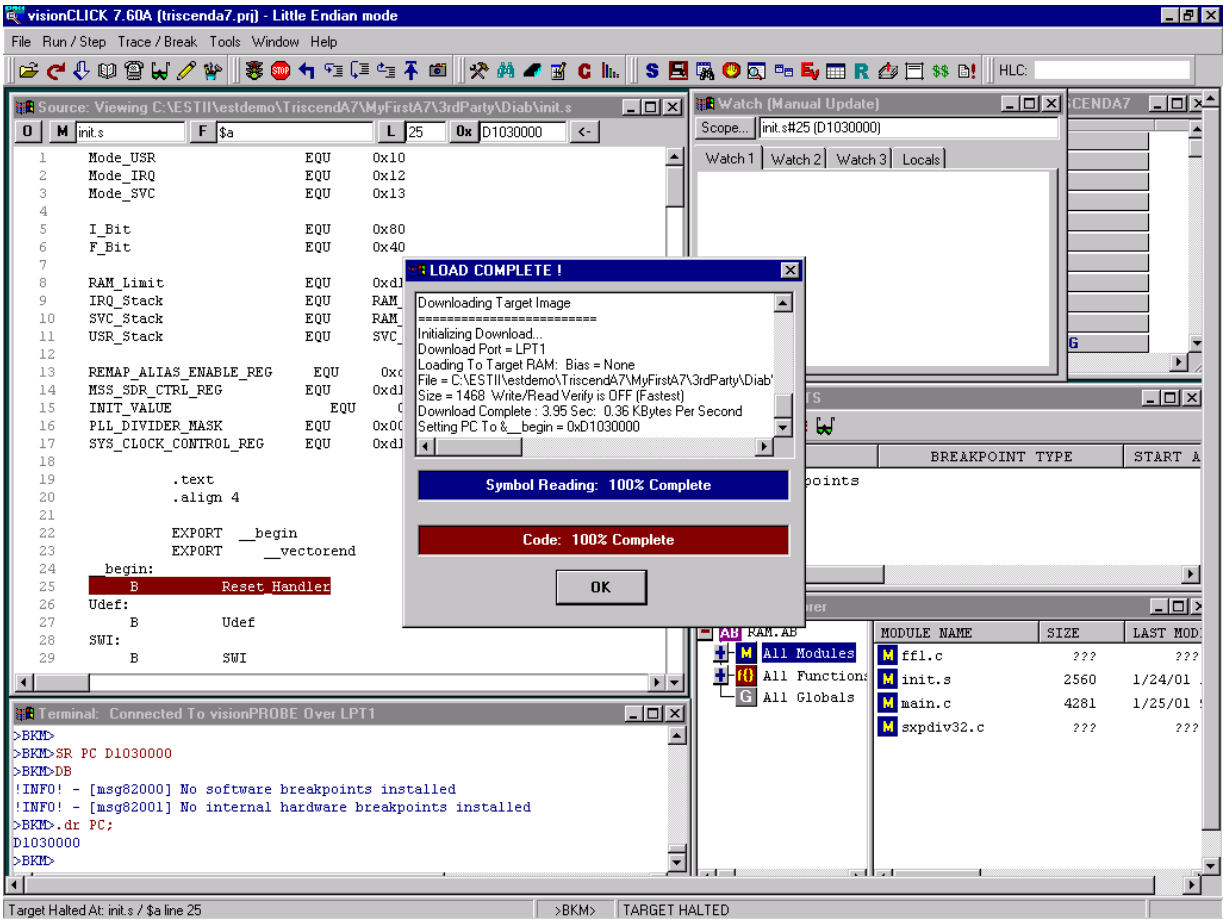
```
CONVERT.EXE v7.6A Copyright (c) 1996-2000 Wind River HSI
convert ELF file ram.elf to CLICK file ram.ab
Extracting symbols from 'ram.elf'
Symbols written
Processing time: 0.160 seconds
convert ELF file ram.elf to BDX file ram.bdx
Extracting image from 'ram.elf'
Writing binary download image to 'ram.bdx'
Maximum packet size: 0x100
Lower address: 0x0
Upper address: 0xffffffff
Execution address: 0xd1030000
Image written
Processing time: 0.060 seconds
press a key please
```

Press any key to close the DOS box window. To complete the convert process, click **OK**. If you check the contents of your MyFirstA7 directory you should note that a number of files have been created.

Download with visionCLICK

Being able to set software breakpoint is almost synonymous with source-level debugging. Setting software breakpoint with a source level debugger is possible only when the software is executing out of RAM. Therefore, we'll switch to work the **`ram.elf`** file, which has been built to run out of A7's internal SRAM.

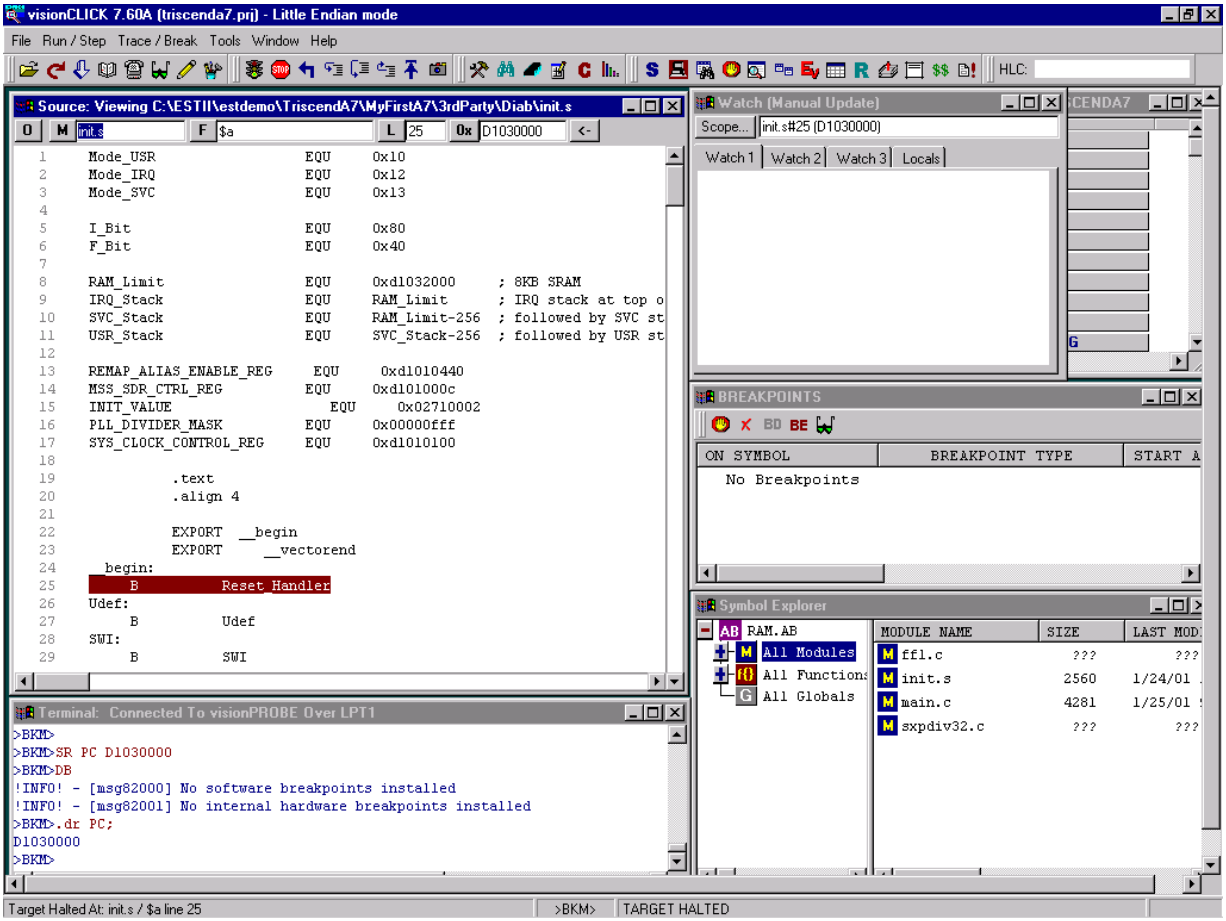
The next part of the exercise is to download the RAM-based application to the Target board. Select **File → Reset, Load Target and Symbols, Set PC** from the upper visionCLICK toolbar. VisionCLICK reads the **`ram.ab`** file containing the symbol information and downloads the **`ram.bdx`** file to the target board. Your visionCLICK screen will look like the following.



Click the **OK** button to close the "Load" window. The RAM-based application has been loaded and is ready to run. From this point, you can debug your application. In the following sections, you will perform some basic visionCLICK debugging.

visionCLICK debugging – Basics

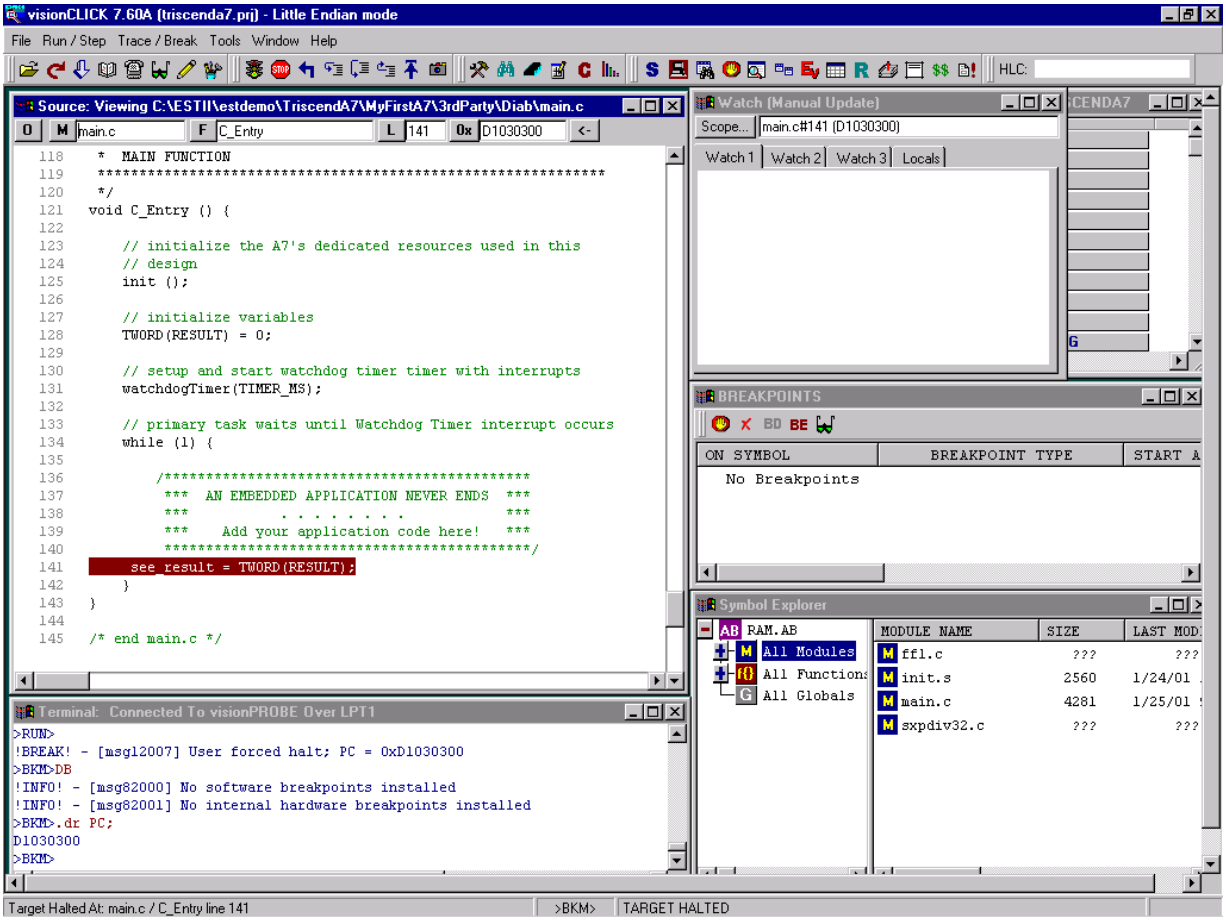
Your display should now look like:



The program counter (PC) is pointing to the first instruction in the **`init.s`** module. In the visionCLICK project configuration window, the RESET Symbol was set to the address of **`__begin`**.

In the visionCLICK toolbar, click the icon that looks like a “Traffic Light” (). This starts your application running. The 7-segment display should begin counting by two again.

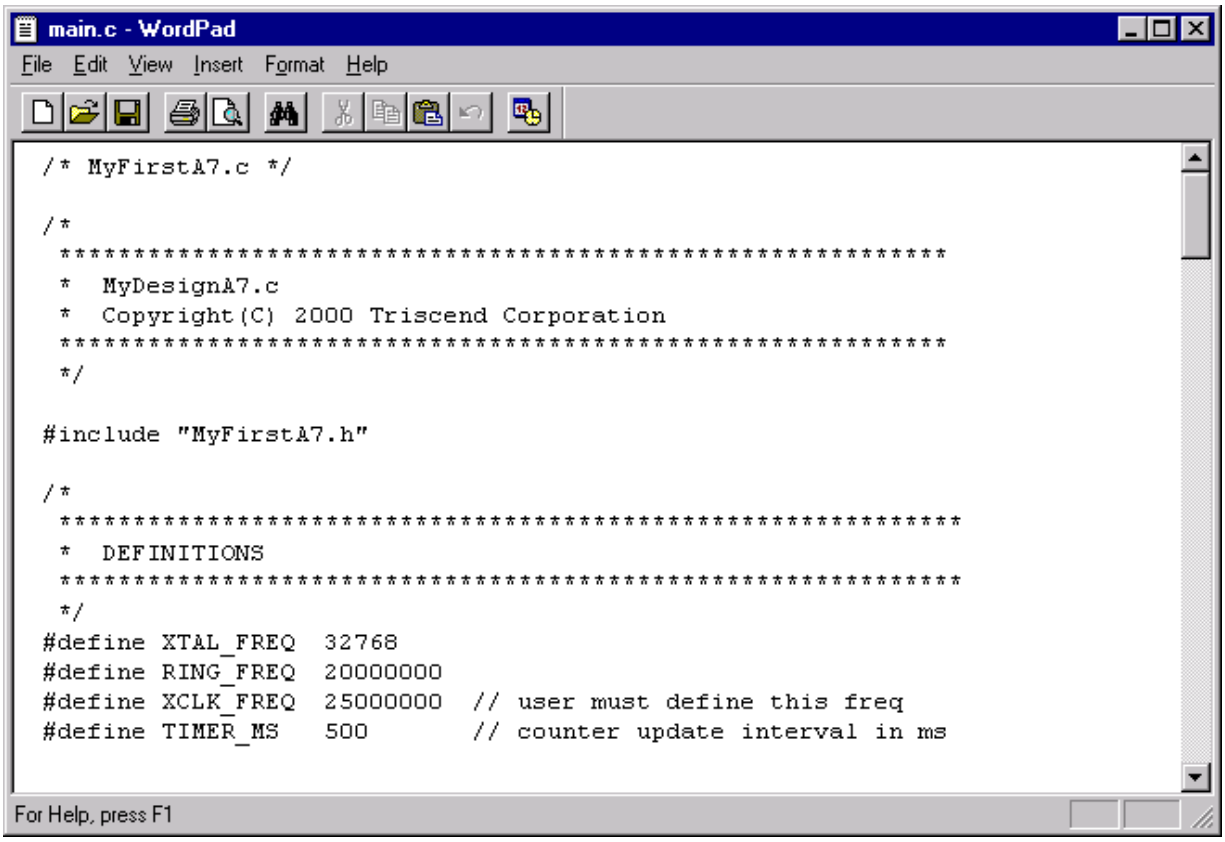
If you hit the icon the looks like a STOP Sign (), the application stops. The source window indicates which instruction will execute next when you resume the application. Most likely, the program will stop inside the **`while`** loop on the **`C_Entry`** function.



If you do not see C source code, you probably did not properly configure visionCLICK.

Editing from within visionCLICK

The next step is to modify the application. In the current version, the 7-segment display is incremented by two each time that the watchdog timer interrupt is serviced. Edit the **`main.c`** module to restore the original application. With the application halted in the C module, right click inside the Source Window and select the **Edit the Current Module** option from the pop up window. The text editor you specified in the Project Configuration window opens **`main.c`** for editing.



The examples in this tutorial assumes that you are using WordPad, however, any text editor will work equally well. Scroll down until you locate the following line of code:

```
TWORD(RESULT) += 2;
```

Modify the increment value to 1.

```
TWORD(RESULT) += 1;
```

Save the file but do not exit from the editor.

Recall that in the visionCLICK project configuration window there was an entry for Make/BAT file. Create the following text file as shown and save it as **`Makeall.bat`**. This batch file enables you to rebuild the application from within visionCLICK.

```
make clean
make
d:\estii\convert –w “ram.elf” –c “ram.ab” –b “ram.bdx” -z
```

The file should include the statements shown in the above example. After entering these three lines, save the file as **`Makeall.bat`**. Make sure that you save it in the correct directory.

`c:\estii\estdemo\TriscendA7\MyFirstA7\3rdParty\Diab`

The example assumes that you installed visionCLICK in the default directory. If you installed visionCLICK in another location, use the necessary pathname.

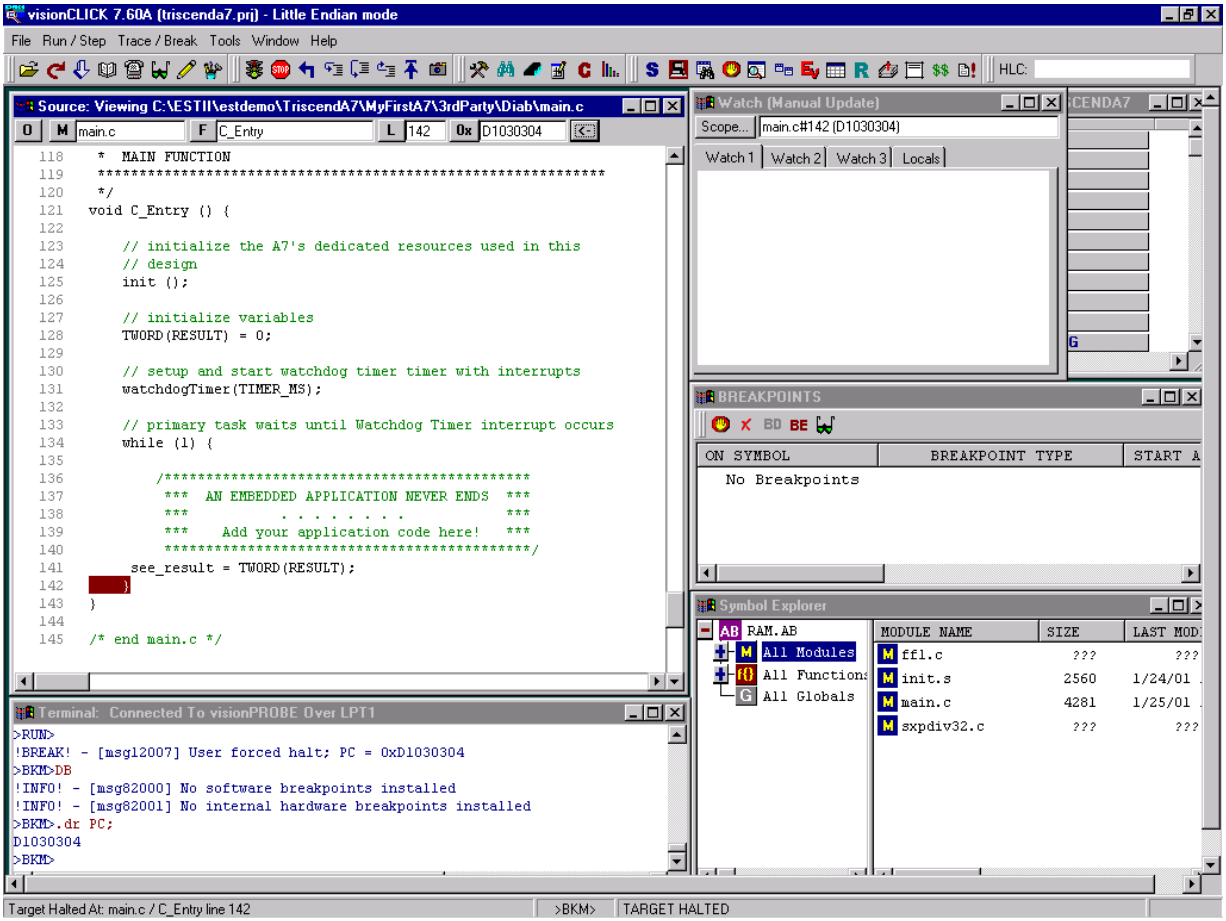
A full pathname to the **`convert.exe`** utility is highly recommended as there may be other **`convert.exe`** programs installed on your computer. Running the wrong **`convert.exe`** could cause your system to crash. The FastChip **`convert.exe`** utility converts the **`ram.elf`** file into visionCLICK symbol and downloadable files.

Now rebuild the application. From the toolbar, select **Tools → Compile/Make Project**. A DOS box command window appears allowing you to monitor the progress of the build. Press any key to terminate the build process.

The next step is to download the changed application. From the toolbar, select **File → Reset, Load Target and Symbols, Set PC**. When the download is completed, click **OK** in the download dialog box. Start the application running by clicking the **Go** icon (the traffic light). If you were successful, the 7-segment display should start incrementing by one.

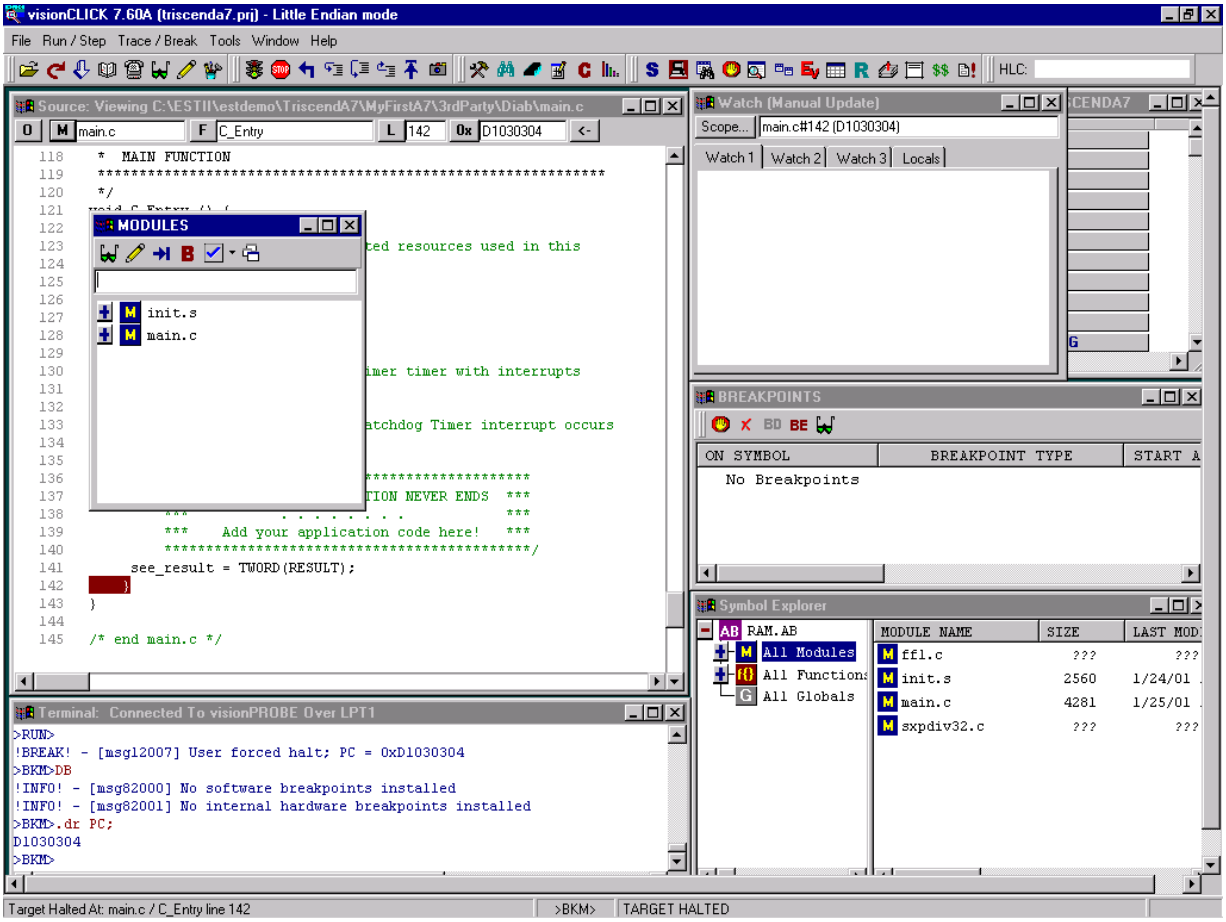
Breakpoints in visionCLICK

You built the application to execute code from internal SRAM so that you could set software breakpoints. The current application example is very simple. By alternately clicking **Stop** and **Go**, you will almost always stop on the same instruction in the **`while(1)`** loop.



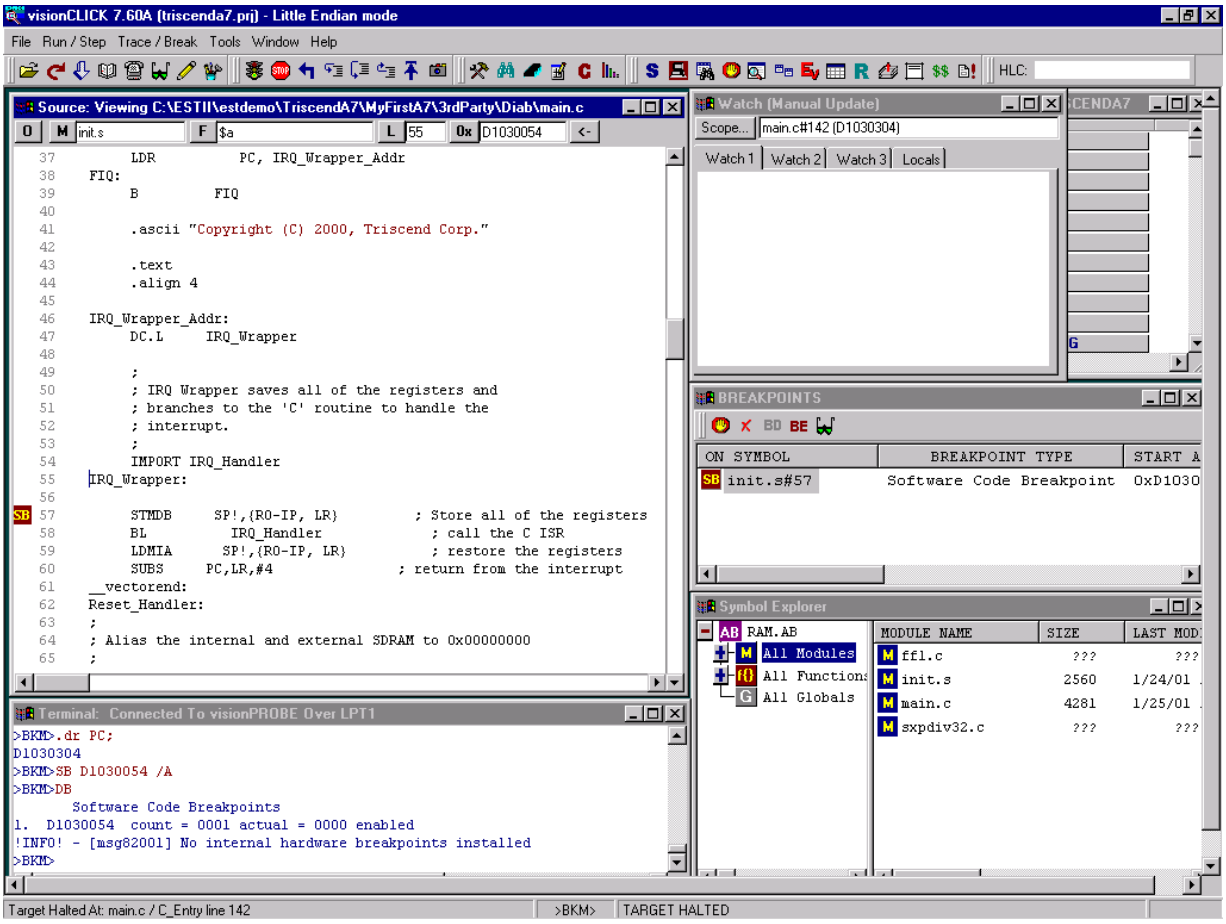
The only indication that the application is executing is that the 7-segment display is changing. Beyond the simple **`while`** loop, there is another significant part of the application. The Watchdog Timer generates a periodic interrupt. Each time that an interrupt occurs, program control passes from the **`while`** loop to the Interrupt Service Routine (ISR). Recall that there are two parts to the ISR, the **`IRQ_Wrapper`**—which is written in assembler and contained in the **`init.s`** module—and the **`IRQ_Handler`**, which is written in C and located in **`main.c`**. Set a breakpoint to observe that the interrupt is actually working. For this exercise, set the breakpoint on the first instruction of the **`IRQ_Wrapper`** code.

First stop the application by click the **Stop** icon. Next, locate the "**M**" button at the top of the visionCLICK source window. In this case, "**M**" stand for module. Click this button to open a module window.

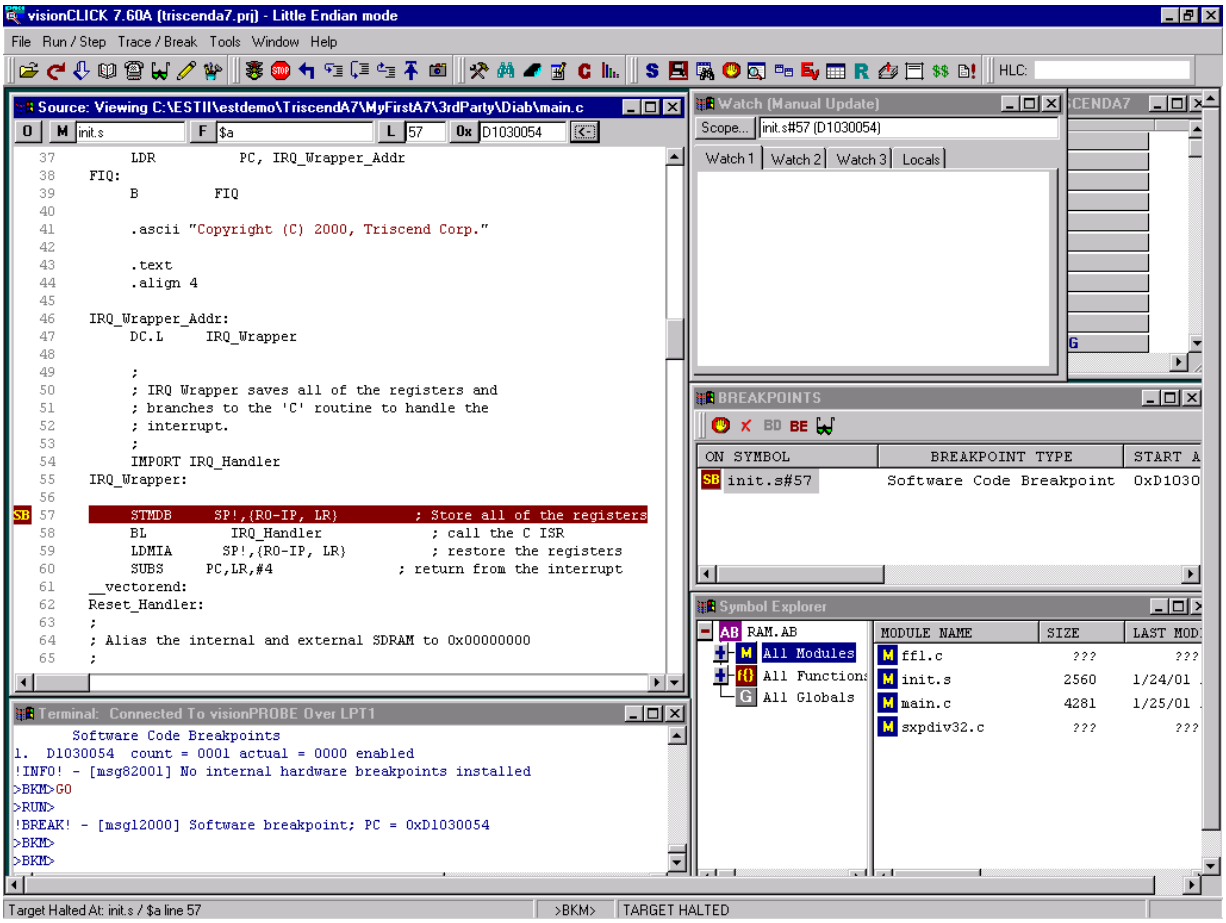


There are two modules in this application – **`init.s`** and **`main.c`**. Double click on the file named **`init.s`**. The source code for the **`init.s`** module appears in the source window. Close the Modules Window.

Scroll down until you find the start of the **`IRQ_Wrapper`** code (line 55). Double click on line 55. An "**SB**" (Software Breakpoint) icon appears immediately to the left of line 57. VisionCLICK sets the breakpoint at the first executable code address.



Click **Go** to start the application running. Execution stops when the interrupt occurs and begins the service routine.



The highlighted line indicates which line of code will execute when the application resumes. Locate the step instruction icon ( ) and click it to step through the code. On the second step, your program branches to the **`IRQ_Handler`** in **`main.c`**.

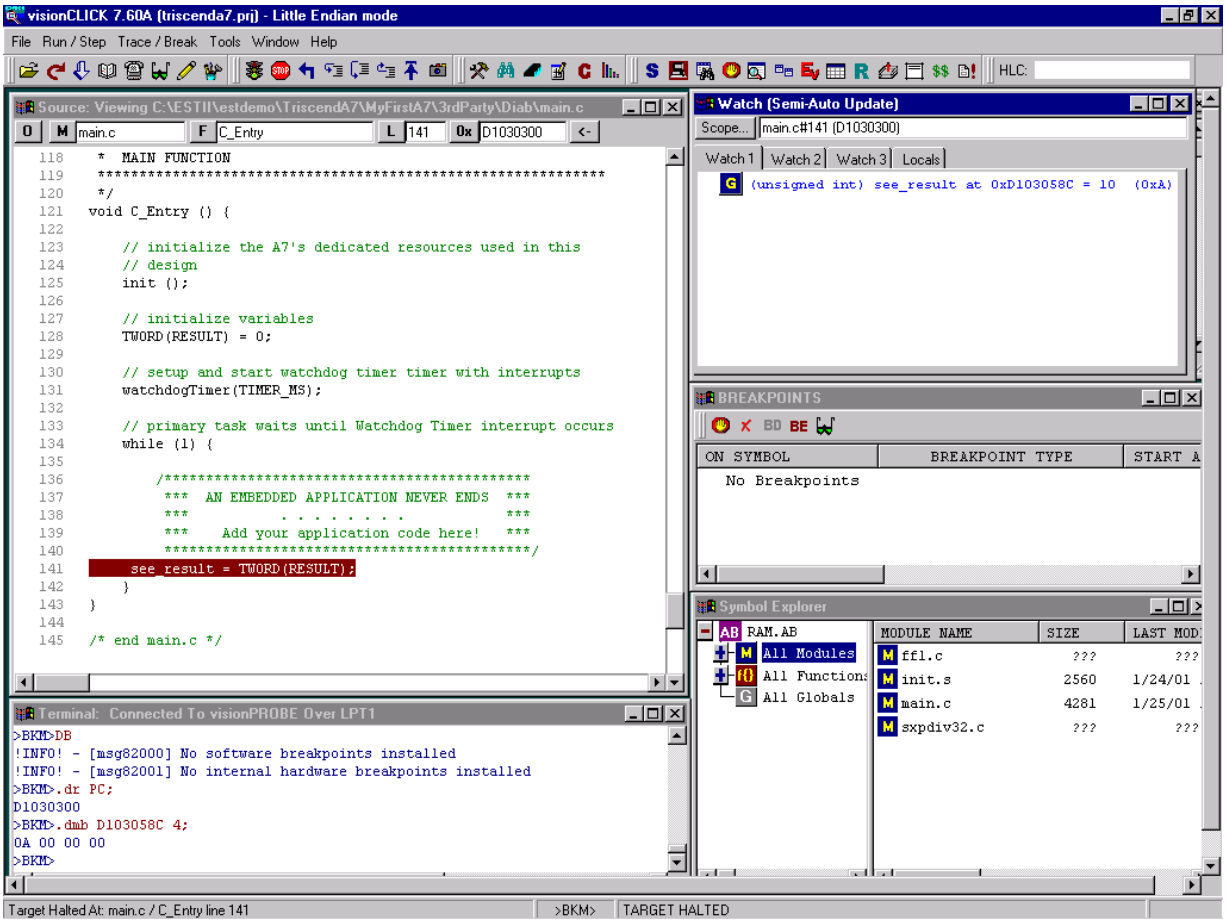
Locate the visionCLICK Breakpoint window and see that there is now one entry in the table.

There are a number of ways to clear the interrupt. Run the application until you once again stop at the breakpoint. Double click the SB icon. Note also that the entry in the Breakpoint Window disappears as well as the SB icon. You can also clear the breakpoint from the Breakpoint Window. First Reset the Breakpoint. Right click on the entry in the Breakpoint window. The first option in the pop-up window provides another way to remove the breakpoint. Note that the SB icon is cleared as well.

Watching Variables

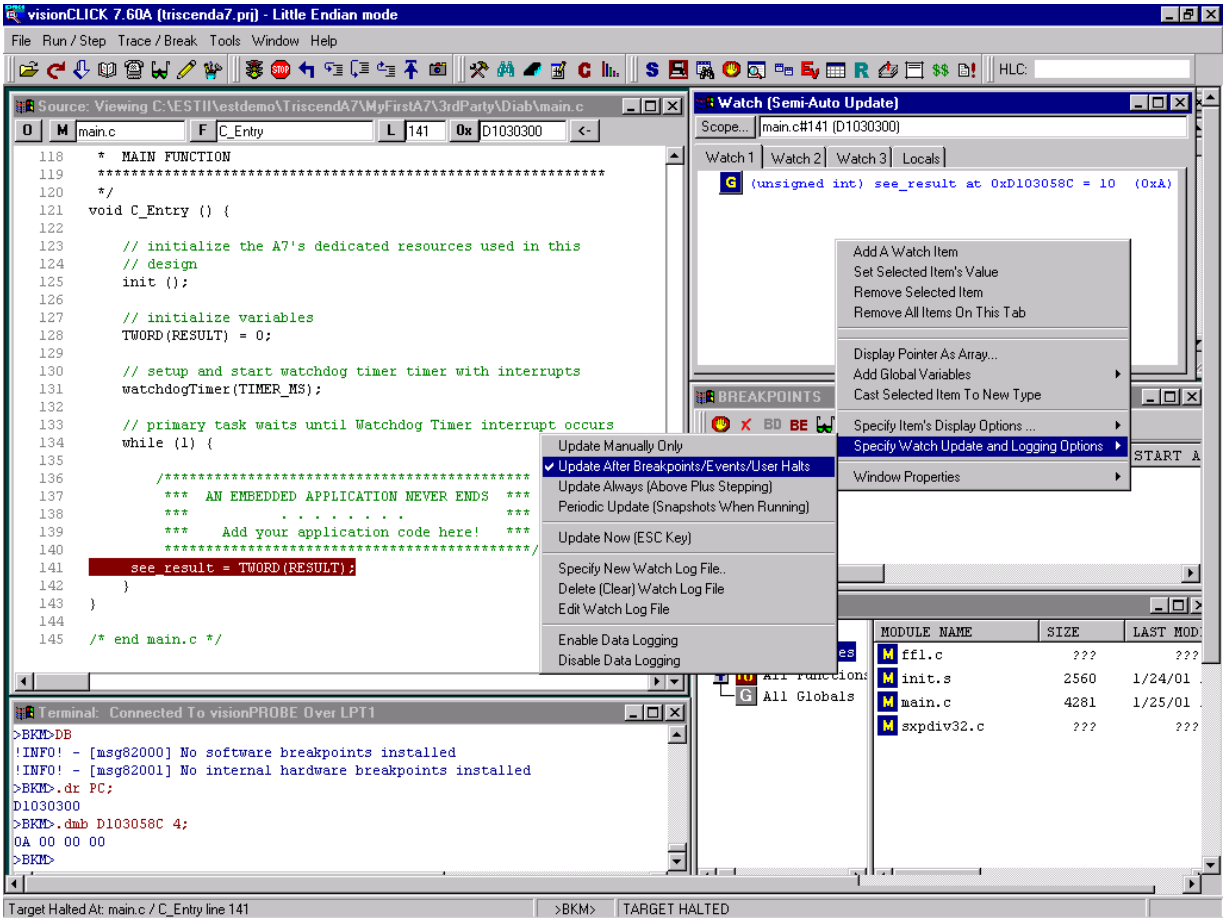
Earlier in this exercise, you added the **see_result** variable to the C_Entry function. You can monitor or watch the value of **see_result** change with visionCLICK. Whenever RESULT increments, the value of **see_result** also changes.

To watch this variable, do the following. First, stop the application. Double click on the name **see_result** in the source code and while holding down the left mouse button, drag it to the Watch window.



VisionCLICK displays the current value of the variable **see_result**, which should also be the value shown on the 7-segment display. In the example shown above, the current value is 10 or 0xA.

Start the application running. Note that the value in the Watch window does not change. A watched value is only updated when you stop the application. Right click in the Watch window. There are options that allow you to control visionCLICK's update behavior. The default setting is Update Manually Only. When you start and stop the application, the value displayed in the Watch Window will not change. Change the setting to **Update after Breakpoints/Events/User Halts**.



Now run and stop the application. You should note that each time the application halts that the value associated with the **see_result** is updated.

Part 2 Complete!

Congratulations on completing Part 2 of the tutorial!

At this point, you created and compiled an application program for the A7's embedded ARM7TDMI RISC processor. Then, you used the visionPROBE JTAG download/debug cable with the visionCLICK source-level debugger to control and monitor your application program operating in real-time on the A7 Evaluation Board.

PART 3: The FastChip Command-Line Interface

All functions in the FastChip's Graphical User Interface (GUI) are also available through the FastChip command line interface. All FastChip commands are accessible through **`csoc.exe`** found in the **`<FastChip Installation Directory>\bin`** directory. Most of the Triscend supported third-party design tools also include command-line or batch-file interfaces.

Graphical and command line FastChip interfaces are useful for different kinds of design activities. The graphical interface is useful for tasks performed rarely, such as entering a design, instantiating and parameterizing a soft module library, connecting modules, etc. The command interface, by contrast, is useful for RTL design flow, and automating repetitive tasks.

The power of the command line is truly unleashed when you chain multiple commands together in a script file. With scripts—such as DOS batch script, shell script, perl script or makefile—repetitive tasks become manageable, if not automatic.

A script can generate headers, compile C code, and link the microprocessor code. A script can also run logic synthesis tools on an RTL design, import the result into FastChip, bind, export and verify the resulting design against a simulation test bench. Likewise, a script combine the microprocessor code with the CSL configuration file and download the result to the CSoC under test. Finally, a script can perform in-circuit debugging of your hardware and software design as part of your regression suite to ensure your application still functions correctly after a design iteration.

Unlike other scripting solutions, make files provide the capability to specify dependency between different files. It is very convenient to rerun only the changed portion of your design once the dependency is setup correctly.

Triscend FastChip ships with a number of demonstration projects with makefiles included. Makefiles are commonly used during software development, particularly by command-line tool users. However, there are many similar but not mutually compatible makefile tools available. Triscend chooses GNU make as the reference standard for the project makefiles.

GNU make is part of the popular GNU free software available on many operating system platforms, including many variants of UNIX and Microsoft Windows. The GNU tools are available from GNU's web site at http://www.gnu.org. The Cygwin tools are ports of the popular GNU development tools and utilities for Windows 95, 98, 2000 and NT (http://sources.redhat.com/cygwin).

You only need to install a very small subset of the cygwin tools in order to use the bundled makefiles. For your convenience, we have selected that subset and put it in the directory **`ThirdParty\Redhat\cygwin\bin`**. We recommend you copy the entire **`\cygwin`** directory and its **`\bin`** sub-directory to **`C:\ on`** your hard disk. Also, add "C:\cygwin\bin" to your Windows PATH environment variable.

To comply with the GPL license, Triscend re-distributes the unmodified binary and source files on the FastChip CD-ROM under the **`Thirdparty\Redhat\cygwin_install`** directory. The complete Cygwin package can be installed by running "setup.exe" directly from that directory on the CD-ROM. If you have previously installed "cygwin\bin", you do NOT need to run "setup.exe".

Windows PCs running the McAfee anti-virus software may generate warning or error messages on some of the .tar.gz files if you attempt to copy the cygwin_install directory from the CD-ROM to your hard disk. Apparently, the anti-virus software is cautious about a non-Windows-native compression format it doesn't quite understand. Turn off checking on compressed files to proceed with copying if desired.

The third portion of the tutorial assumes that you are a fairly advanced PC user. You need to be comfortable using the MS-DOS command shell and DOS batch files.

In this exercise, you will add some functionality to the source code and change the design. You will use the command line interface to compile, link, bind and download the code into the TA7S20. Portions of the project are performed using the GUI for convenience.

DOS Environment Variable

To use the FastChip command line, the **FastChip\bin** directory must be in your executable search path. FastChip automatically sets this environment variable for you during installation. In order to compile and link application, the **Diab\4.4a\Win32\bin** directory must also be in your search path. Again, Diab should have set this path variable during the installation. You can double check this variable by typing …

```
set path
```

… in DOS command window. Make sure that the **FastChip\bin** directory appears somewhere in the search path.

You can also type …

```
csoc
```

… and see if the **csoc** general help message is displayed:

```
C:\>CSOC

    csoc -- Triscend FastChip command line program

    csoc [options] commandName [commandArguments]

        Run the given csoc command.  Recognized commands are listed below.

        Use 'csoc help <commandName>' or 'csoc man <commandName>' for
        information on each command.

        Use 'csoc man csoc' for details.

        help        - Show brief help for a command
        man         - Show the manual page for a command
        fcid        - Show FastChip ID on local machine.
        migrate99   - Convert a FastChip 1999 project
        create      - Create a project
        import      - Import a module type into a project
        export      - Export a simulation model
        bind        - Bind logic elements to CSL resources of CSoC
        generate    - Generate source code
        listdev     - List Triscend CSoC parts supported
        listmem     - List memory parts supported
        config      - Configure an image file for download
        download    - Download to CSoC and memory devices
        debug       - Debug CSoC
        report      - Generate a project report
        listmod     - List module library information
        tjr         - Show Tjr file information
```

To verify that the proper version of the Diab tools are installed correctly and the search path is correct you can type …

```
dcc -V
```

… in the command window.

The following information should be displayed.

```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1999.

C:\WINDOWS>dcc -V
C:\DIAB\4.4A\WIN32\BIN\DCC.EXE Rel 4.4 Rev a
Copyright 1986-2000 Wind River Systems, Inc.
build date and time: Oct 26 2000 14:09:34
Table: ARM  Rel. 4.4a
```

If the search path is not setup properly, you can set it yourself by typing:

```
set PATH=<FastChip install dir>\bin;%PATH%
```

FastChip Makefile/Batch File

A Makefile is provided in the **FastChip\projects\MyDesignA7** directory. You can modify this file to use any of the various **csoc** commands available from the command line interface. In fact, you must make some minor edits to use this file with other projects.

This section of the Tutorial examines the Makefile organization and describes the necessary changes to use it with the MyFirstA7 project.

First, copy the **FastChip\projects\MyDesignA7\makefile** into your MyFristA7 project directory.

Makefile Header

In the header section of the Makefile, the project name should be changed to MyFirstA7. This is optional change. The reference to a project could just as easily be deleted.

```
#-------------------------------------------------------------------------------
# Triscend sample makefile for demo design: MyFirstA7
#
# Copyright (c) 2000 Triscend Corporation.  All right reserved.
#
#-------------------------------------------------------------------------------
#
#  This file has been tested with GNU make
#
#-------------------------------------------------------------------------------
```

Makefile Variables

The variables section of the makefile descibes the hardware configuration of your target hardware and specifies the included application components. The hardware configuration information includes such items as the CSoC Flash, SRAM, and SDRAM device part names. These values are correct for the Triscend A7 Evaluation Board. If you use the default hardware configuration, you do not need to make any changes to these entries.

The DESIGN_NAME entry needs to be modified to reflect the correct project name. For this tutorial change MyDesignA7 to MyFirstA7. This setting is used in the Makefile rules to specify a pathname to the application source directory and is therefore a required change.

For the purpose of this Tutorial, no other software reference changes will be required.

```
#===============================================================================
# Variables
#===============================================================================
# You should customize the following variables to your environment and project
DESIGN_NAME=MyFirstA7
USER_CODE=main
CSOC_NAME=TA7S20-ESQ
FLASH_PART_NAME=AM29LV004B-120
SRAM=CYM1851V33-20
SDRAM=DIMM100-08M32-662  # == MT8LSDT864HG-662
IMPORTED_MODULE=heartbeat
EDIF_DIR=3rdParty/EDA
SW_DIR=3rdParty/Diab
RM=rm -f

DESIGN_FILES=$(DESIGN_NAME).fcp $(DESIGN_NAME).n ImportedFor$(DESIGN_NAME)/*.n
CONSTRAINTS=$(DESIGN_NAME).ioc $(DESIGN_NAME).adc $(DESIGN_NAME).tic
```

Makefile Rules

Target

```
#=============================================================================
# Targets
#=============================================================================

# build is the default target if you just type make
build:  import generate code csl

# Call the EDA makefile to see if it needs to be recompiled
import:

# If the dependency check fails, generates the project header and source file
# for the Keil project
generate: $(SW_DIR)/$(DESIGN_NAME).h

# Call the ARM/Diab Project makefile to see if it needs to be recompiled
code:
   @$(MAKE) -C $(SW_DIR)

# If the dependency check fails, bind the CSL design and produce a project
# report
csl:   $(DESIGN_NAME).csl
```

Make and Download Options

There are three download options available in the default Makefile: **`Direct`**, **`Flash`**, and **`SRAM`**. In all three cases, two separate **`csoc`** commands are used: **`config`** and **`download`**. The **make direct** option uses the current binder file and downloads directly to the CSL configuration memory. The application code is download to the internal SRAM.

The **make flash** option builds the application and downloads both the CSL and application code to Flash Memory. The **make SRAM** option is similar to **make flash** except that it downloads CSL and application code to an external SRAM added to your board. By default, the A7 Evaluation Board is shipped from Triscend without any external SRAM. To use the **make SRAM** option, you must add an SRAM module and modify several of the default switch settings, described in the ***Triscend A7 Evaluation Board User Manual***, under **`FastChip\Docs`**.

```
#============================================================================
# Short Hand Targets
#============================================================================
#  These targets execute unconditionally, regardless of whether the project
#  files are up-to-date.

#  Take the current binder output and directly program the CSoC
direct:DIRECT.cfg directp

#DIRECT.cfg: $(DESIGN_NAME).csl $(SW_DIR)/$(USER_CODE).hex
DIRECT.cfg:
    csoc config -code $(SW_DIR)/$(USER_CODE).hex \
             -csl $(DESIGN_NAME).csl -dev $(CSOC_NAME) -mem NONE \
             -clk xtal32K:20M -swt 5 \
             -out DIRECT.cfg

#  Reprogram using the current image file DIRECT.cfg
directp::
    csoc download DIRECT.cfg -pgm

#  Take the current binder output and program the flash memory
flash: FLASH.cfg flashp

#FLASH.cfg: $(DESIGN_NAME).csl $(SW_DIR)/$(USER_CODE).hex
FLASH.cfg:
    csoc config -clk ring -code $(SW_DIR)/$(USER_CODE).hex \
             -csl $(DESIGN_NAME).csl -dev $(CSOC_NAME) -mem $(FLASH_PART_NAME)
\
           -miu 512K:8 -sdmiu 32M:32:1 -sdram $(SDRAM) \
             -out FLASH.cfg

#  Reprogram using the current image file FLASH.cfg
flashp::
    csoc download FLASH.cfg -pgm

#  Take the current binder output and program the external SRAM
sram: SRAM.cfg sramp

#SRAM.cfg: $(DESIGN_NAME).csl $(SW_DIR)/$(USER_CODE).hex
SRAM.cfg:
    csoc config -clk ring -code $(SW_DIR)/$(USER_CODE).hex \
             -csl $(DESIGN_NAME).csl -dev $(CSOC_NAME) -mem $(SRAM) \
           -miu 4M:32 -sdmiu 32M:32:1 -sdram $(SDRAM) \
             -out SRAM.cfg

#  Reprogram using the current image file SRAM.cfg
sramp::
    csoc download SRAM.cfg -pgm

#  Perform in-system-debugging by reading back the Result Register in CSL and
#  the Watchdog timer control SFR register
observer::
    csoc debug getreg Result -xref $(DESIGN_NAME).xref

#  Generate a project report file
report::
    csoc report $(DESIGN_NAME).fcp
```

Make Clean

This option scours the project directory by removing everything except for the source files. See the ***Getting Started Manual*** for details on how to install the necessary GNU commands such as **`rm`**.

```
#  Remove all except the source files
#  Since the DOS del command errors out if file does not work
#  It is recommended that your call the clean target using
#  make -i to ignore the file missing error
clean::
    $(RM) $(DESIGN_NAME).m
    $(RM) $(DESIGN_NAME).al
    $(RM) $(DESIGN_NAME).p
    $(RM) $(DESIGN_NAME).pm
    $(RM) $(DESIGN_NAME).pad
    $(RM) $(DESIGN_NAME).rpt
    $(RM) $(DESIGN_NAME).bn
    $(RM) $(DESIGN_NAME).b
    $(RM) $(DESIGN_NAME).br
    $(RM) $(DESIGN_NAME).htm
    $(RM) DIRECT.cfg
    $(RM) FLASH.cfg
    $(RM) SRAM.cfg
```

Exercises

In the next part of the tutorial, you will use several command line rules. First open a DOS box command window and change directory to point to your **`FastChip\projects\MyFirstA7`** directory. If you have not already done so, copy the **`makefile`** from the MyDesignA7 project. Use a text editor to change all references in that file from MyDesignA7 to MyFirstA7.

You should also be connected to the A7 Evaluation Board using the visionPROBE cable. Make sure that the visionCLICK software is not currently running. Both the board and the visionPROBE should be powered.

In this exercise we are going to use two of the three short-hand target rules: Direct and Flash. From the command line, enter the command …

```
make flash
```

```
C:\Program Files\Triscend\FASTCHIP\Projects\MyFirstA7>make flash
csoc config -clk ring -code 3rdParty/Diab/main.hex \
            -csl MyFirstA7.csl -dev TA7S20-ESQ -mem AM29LV004B-120 \
                -miu 512K:8 -sdmiu 32M:32:1 -sdram DIMM100-08M32-662   \
            -out FLASH.cfg
Read CSL file.
Read code file(s).
Write configuration image file 'FLASH.cfg'
Write configuration report file
csoc download FLASH.cfg -pgm
Begin at Tue Feb 06 14:18:17 PST 2001
Download to 'AM29LV004B-120' via 'TA7S20-ESQ' ...
Read configuration image file 'FLASH.cfg'
skip check id for TA7 for now
Reset and halt CSoC.
MCU halted. MCU in reset.
Download Flash programming algorithm to CSoC.
Configure CSoC to run programming algorithm.
Reset and run CPU.
CSoC running.
    total progress = 1.0%
Erase Flash memory ...
    total progress = 15.0%
Program Flash with configuration image ...
    total progress = 16.3%
    total progress = 18.2%
    total progress = 20.2%
     •
     •
     •
    total progress = 97.3%
    total progress = 99.2%
    total progress = 100.0%
Reset and run CSoC.
MCU halted. MCU in reset.
CSoC running.
CSoC running.
    total progress = 100.0%
End at Tue Feb 06 14:24:04 PST 2001
```

You should see various commands execute from the script.

The first command executed is **`csoc config`**. This command uses information in the variables section of the makefile to generate a **`flash.cfg`** file. You will recall that a .cfg file is an Intel Hex file with a short header prepended to the top of the file. Once the .cfg file is created, the MCU is halted a Flash programming algorithm is downloaded to CSoC device. Then the Flash memory device is programmed from your computer over the JTAG cable.

The Flash programming algorithm used is based on the FLASH_PART_NAME specified in the Makefile. Once the programming is complete, the CSoC device is reset and the application begins executing from Flash and the 7-Segment display should increment.

In a previous section of the Tutorial, you modified the **`main.c`** module so that the LED display would increment by two. Now change the application so that the LED increments by three. Using a text editor, open the **`main.c`** file for editing. Locate and modify the following line of C code.

```
TWORD(RESULT) += 3;
```

Save the file and exit the editor. From the DOS window, enter the following command.

```
make code
```

```
C:\Program Files\Triscend\FASTCHIP\Projects\MyFirstA7>make code
MAKE.EXE[1]: Entering directory `C:/Program Files/Triscend/FASTCHIP/Projects/MyF
irstA7/3rdParty/Diab'
dld -tARMLS:simple -o main.elf init.o main.o main.dld -lc -m2 > main.map
ddump -Rv main.elf -o main.s32
mhc -h main.s32 main.hex
mhc Ver1.0.3 Motorola S data <-> Intel Hex data Converter.
(C)1997-1998 Cow Project Workshop.
Convert flag -h
Input file main.s32
Output file main.hex

Making Intel Hex file.
Input file size=3954
Output file size=0
Finished!
MAKE.EXE[1]: Leaving directory `C:/Program Files/Triscend/FASTCHIP/Projects/MyFi
rstA7/3rdParty/Diab'
```

You are using another option in the Makefile. Now, rebuild the entire application by entering the following command.

```
make direct
```

You will see the following information in your command window.

```
C:\Program Files\Triscend\FASTCHIP\Projects\MyFirstA7>make direct
csoc config -code 3rdParty/Diab/main.hex \
            -csl MyFirstA7.csl -dev TA7S20-ESQ -mem NONE \
            -clk xtal32K:20M -swt 5 \
            -out DIRECT.cfg
Read CSL file.
Read code file(s).
Write configuration image file 'DIRECT.cfg'
Write configuration report file
csoc download DIRECT.cfg -pgm
Begin at Tue Feb 06 14:26:50 PST 2001
Download to internal RAM of 'TA7S20-ESQ' ...
Read configuration image file 'DIRECT.cfg'
skip check id for TA7 for now
Reset and halt CSoC
MCU halted. MCU in reset.
Use internal ring oscillator
Enable CSL programming
Clear CSL
    total progress = 10.0%
Download bytes to CSoC
    total progress = 11.4%
    total progress = 11.5%
     •
     •
     •
```

In this case, the makefile command builds a file called **`direct.cfg`** and downloads it directly to the CSoC's internal configuration memory and executes code from the internal SRAM. After the download is complete, the 7-segment display increments by three.

If you hit the reset button on the A7 Evaluation Board, the program previously programmed into Flash memory will start and the LED again increments by two.

Conclusion

In just a few hours, you developed, downloaded, and debugged your own custom system-on-a-chip design operating in working silicon—assuming that you completed the entire tutorial.

FastChip provides an easy-to-use, graphical environment for creating your customized system-on-a-chip design. The FastChip Device Link (FDL) utility combines your custom logic design with your application code. Using FDL, you can download your design onto your target board and debug internal logic using the visionPROBE II JTAG-based download and debug cable.

The WindRiver Diab compiler provides optimized program code for the ARM7TDMI processor embedded within the Triscend A7 Configurable System-on-Chip (CSoC) device. The Diab compiler is just one of many that supports the popular ARM7 RISC processor architecture.

The WindRiver visionCLICK source-level debugger provides additional visibility into your application program as it operates, in real-time, on your target hardware. With visionCLICK, you can set breakpoints, single-step your code, and monitor or preload registers—all while viewing your C or C++ source code.

The Triscend A7 offers you unparalleled time-to-market advantages in embedded applications. With a minimum investment in development tools, you can employ the cost-effective A7 (CSoC) device to create custom-tailored designs embedding the powerful ARM7TDMI 32-bit RISC processor.

Triscend Corporation
301 N. Whisman Rd.
Mountain View, CA 94043-3969
U.S.A.

Tel: 1-650-968-8668
Fax: 1-650-934-9393

E-mail: contact_us@triscend.com
Web: www.triscend.com

SKK/JB 2001