S5000 Development Tools

Quick Start Guide

Version 1.1

Confidential & Proprietary

Last modified: 03/23/2006

This preliminary publication is provided "AS IS." Stretch, Inc. (hereafter "Stretch") DOES NOT MAKE ANY WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF TITLE, NONINFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Information in this document is provided solely to enable system and software developers to use Stretch S5000 processors. Unless specifically set forth herein, there are no express or implied patent, copyright or any other intellectual property rights or licenses granted hereunder. Stretch does not warrant that the contents of this publication, whether individually or as one or more groups, meets your requirements or that the publication is error-free. This publication could include technical inaccuracies or typographical errors. Changes may be made to the information herein, and these changes may be incorporated in new editions of this publication.

Part #: RU-0104-0001-001

Contents

Chapter 1 Introduction

This document provides a step-by-step guide for using the Stretch Integrated Development Environment (IDE) to create, debug, profile, and run a Stretch application.

The first part of this document (Chapter 2, "Overview of the Stretch IDE" through Chapter 3, "Creating, Debugging, and Running an Application") introduces the Stretch IDE, and it teaches you how to install and launch the IDE. You use a simple application to walk through the steps of creating the project, and running and debugging the application.

The second part of this document (Chapter 4, "Stretch C Development Flow" through Chapter 6, "Stretch C Programming") describes the Stretch C programming flow. You create a Stretch C file to obtain the large performance gain in using Stretch C Extension Instructions. You also walk through the profile and optimization process using the IDE.

This document does not provide details regarding Stretch's S5 architecture, the Stretch C Compiler, simulator, and so on. Refer to the *SCP Architecture Reference*, the *Stretch C Compiler User's Guide*, the *Stretch Instruction Set Simulator User's Guide*, and the *Stretch Profiler User's Guide* for more technical details.

Chapter 2 Overview of the Stretch IDE

The Stretch Integrated Development Environment (IDE) provides users with a graphical user interface (GUI) to efficiently create applications targeted to run on Stretch processors. The major functions of the IDE include:

- Creating and managing projects
- Managing build targets
- Editing files
- Debugging with instruction set simulation
- Debugging on a target system
- Profiling code and analyzing the instruction pipeline

There are two versions of the Stretch IDE: a Windows® version supporting Windows XP [1], and a Linux version supporting Red Hat® Linux 7.3. The examples shown in this document are for the Windows version, but the two versions are similar enough that it is a simple matter to see how to perform the same tasks on the Linux version.

2.1 Installing and Launching the IDE for Windows XP

Refer to the Windows XP Stretch IDE installation guide for the installation and launch procedures.

This document occasionally refers to `$(STRETCHROOT)`, the directory where you install your Stretch tools, so you may want to make a note of this directory.

[1] Only Professional Edition has been tested.

2.2 Installing and Launching the IDE for Red Hat Linux 7.3

Refer to the Linux Stretch IDE Release Notes for the installation and launch procedures.

This document occasionally refers to `$(STRETCHROOT)`, the directory where you install your Stretch tools, so you may want to make a note of this directory.

Chapter 3 Creating, Debugging, and Running an Application

This chapter walks you through the steps for creating, debugging, and running an application with the Stretch IDE.

3.1 Project Options

Stretch IDE projects can be created with one of three options: Standard, External Makefile, and Executable.

The Standard option lets you use the IDE to build, debug, and profile applications from scratch. This option is recommended for most users. The project created in this quick start guide uses this option.

The External Makefile and Executable options are for users who prefer to create their own makefiles, but wish to take advantage of some of the IDE features. Appendix B, "Creating a Project with an External Makefile" gives a brief introduction to projects created using the External Makefile option. For information on how to use the external Executable option, refer to the online help in the Stretch IDE.

3.2 Target Options

Three types of targets are supported: Native, Simulator, and Remote. For the example in this guide, the Simulator option is used. It generates a binary file to be run on the Stretch Instruction Simulator (ISS).

The Remote option is used for debugging an application on a target board (for example, the development boards S56DB10 and S56DB20).

Using the Native option, the application is compiled for direct execution on the platform or O/S that is running the tools. Under this mode, even Stretch C code is executed natively, which lets you execute functions like `printf()` inside Stretch C functions.

Refer to Appendix A, "Target Types" for details about different target types.

3.3 Example Overview

This document uses a simple color conversion project as an example to illustrate the application development process using the Stretch IDE.

The application reads data from a static array of red, green, and blue pixel values (r, g, b), transforms them into luminance and blue and red chrominance values (y, cb, cr), and checks the results against a static array of expected outputs. Details on this transformation may be found in texts on image and video processing.

3.4 Creating a New Project

After launching the IDE (refer to Section 2.1 or Section 2.2), the first step is to create a new project.

To create a new project

1. Create an empty folder for the project.

   For this example, create a folder called `c:\work\rgb2ycc`.

2. Copy the following source files from `$(STRETCHROOT)/SCC/Examples/rgb2ycc`, where `$(STRETCHROOT)` is the root directory where you installed the Stretch tools, into the new project folder.

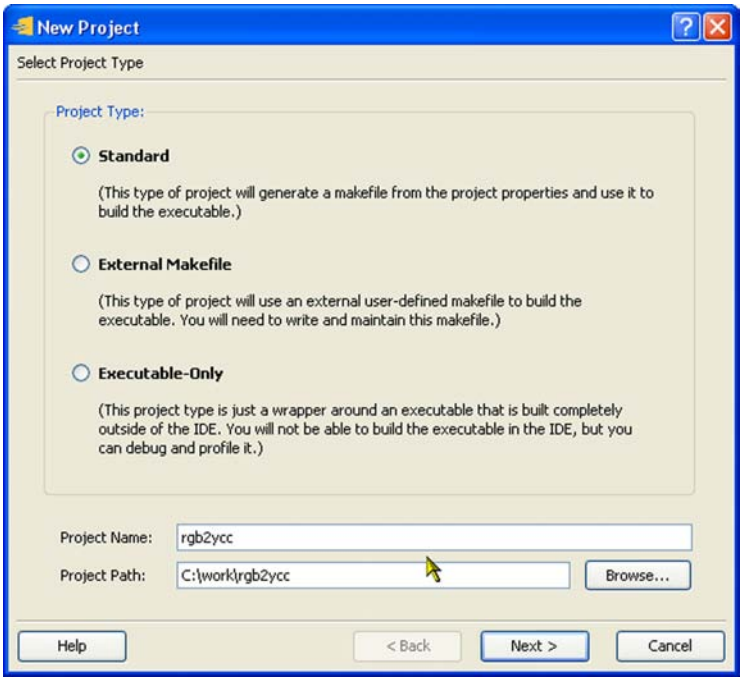
   ```
   data.h
   rgb2ycc_a.c
   rgb2ycc_b.c
   rgb2ycc.xc
   ```

   If these copies are read-only, you may want to make them writable.

3. From the menu bar, choose File →-> New Project to create a new project.

   The New Project dialog appears (see Figure 3-1).

Figure 3-1 New Project dialog



To select project options

1. From the Select Project Type dialog, click the Standard radio button.
2. In the Project Name field, type "rgb2ycc".
3. Click the Browse button to set the Project Path.

   For this project, simply navigate to the folder that contains the source code (that is, `c:\work\rgb2ycc`).
4. Click Next.

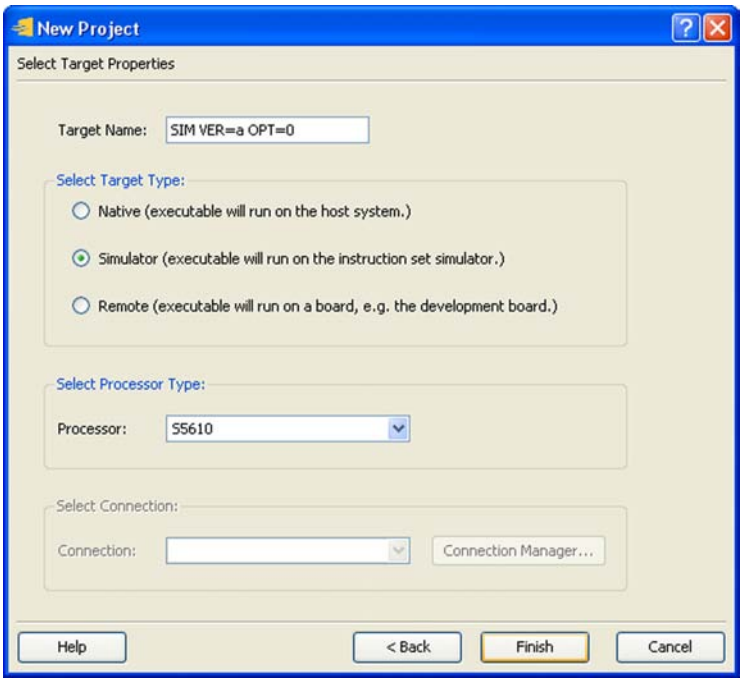
   The Select Target Properties dialog appears.

To select the target name and type

1. In the Target Name field, type "SIM VER=a OPT=0" (see Figure 3-2).
2. In the Select Target Type area, click the Simulator radio button.
3. Click Finish.

   The new project is created. The project file `rgb2ycc.spf` is created under the project path you created in an earlier step.

Figure 3-2 Select Target Properties dialog



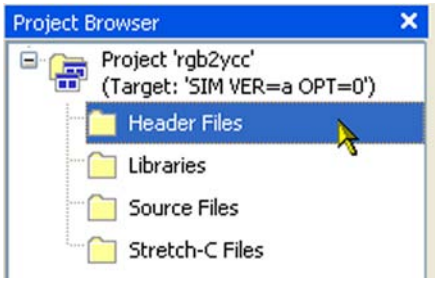
3.4.1 Adding Files to the Project

Files are divided into *Header Files*, *Libraries*, *Source Files* and *Stretch C Files*.

To add files to each category

1. In the Project Browser pane on the left-hand side of the IDE, click the Header Files file folder (see Figure 3-3). If the Project Browser window is not visible, from the Menu Bar choose View →-> Project Browser.

Figure 3-3 Selecting source file folder



2. From the menu bar, choose Project →->Add Existing Files.

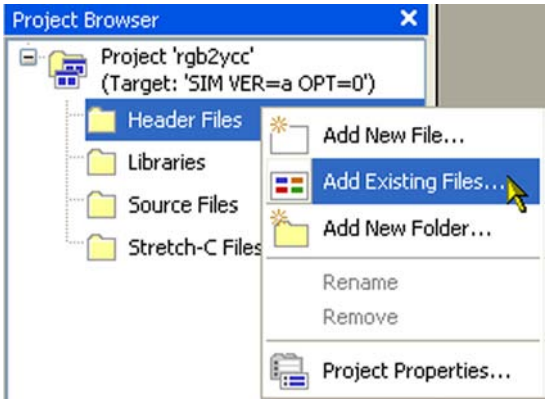
   A file browser appears.

3. In the file browser, click `data.h` to add it to the Header Files folder.

4. Browse to and select `C:\work\rgb2ycc\rgb2ycc_a.c` to add it to the Source Files folder.

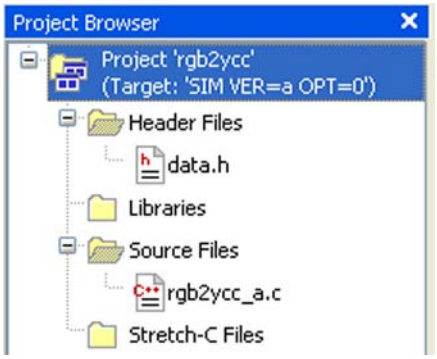
   TIP – You can also add files by right-clicking the File Category symbol and selecting Add Existing Files (see Figure 3-4).

Figure 3-4 Add existing files by right-clicking in Project Browser pane



After adding the files, your Project Browser pane should look similar to Figure 3-5.

Figure 3-5 Project Browser pane after adding source files



3.5 Building and Running the Application

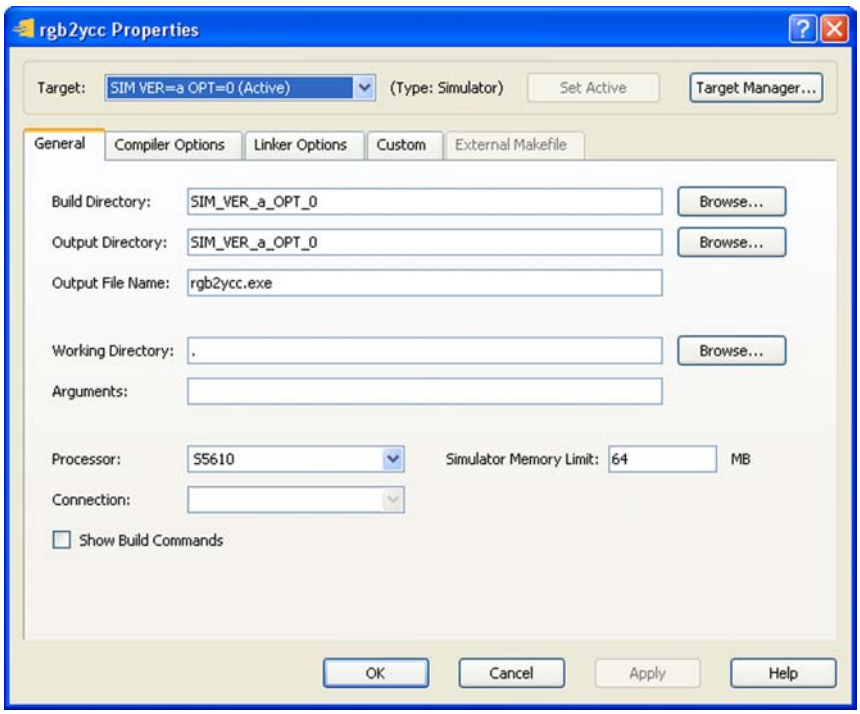
This section shows how to build and run the application.

To set up an application to be built and run

1. From the menu bar, choose Project →->Project Properties.

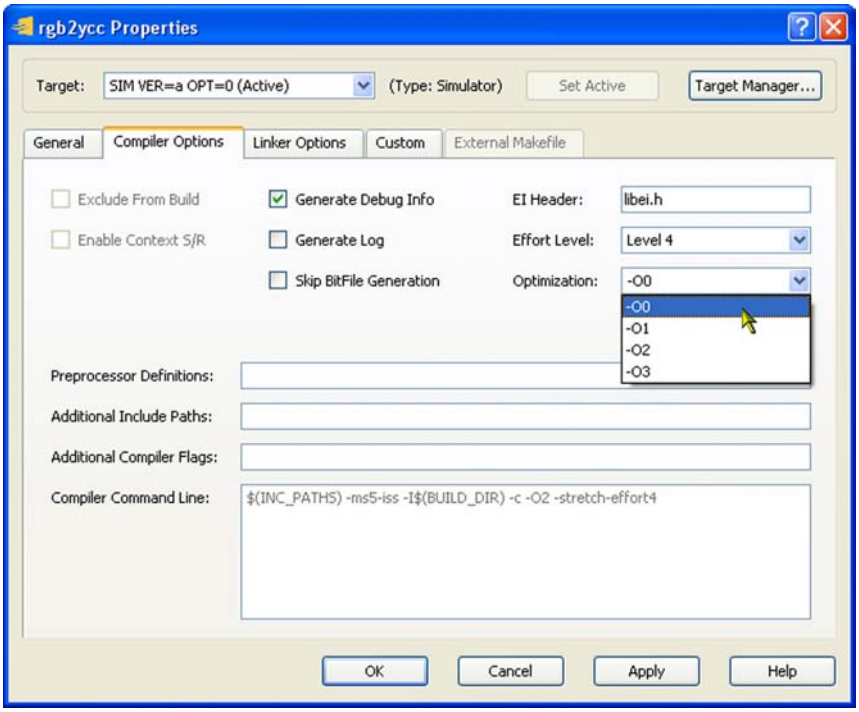
   This displays the Project Properties dialog (see Figure 3-6).

Figure 3-6 Project Properties dialog



2. Click the Compiler Options tab to display the compile options dialog (see Figure 3-7).

Figure 3-7 Compiler Options dialog



3. Click the Generate Debug Info box.

   This generates debug information for the debugger to use.

4. From the Optimization drop-down menu, choose the optimization level -O0.

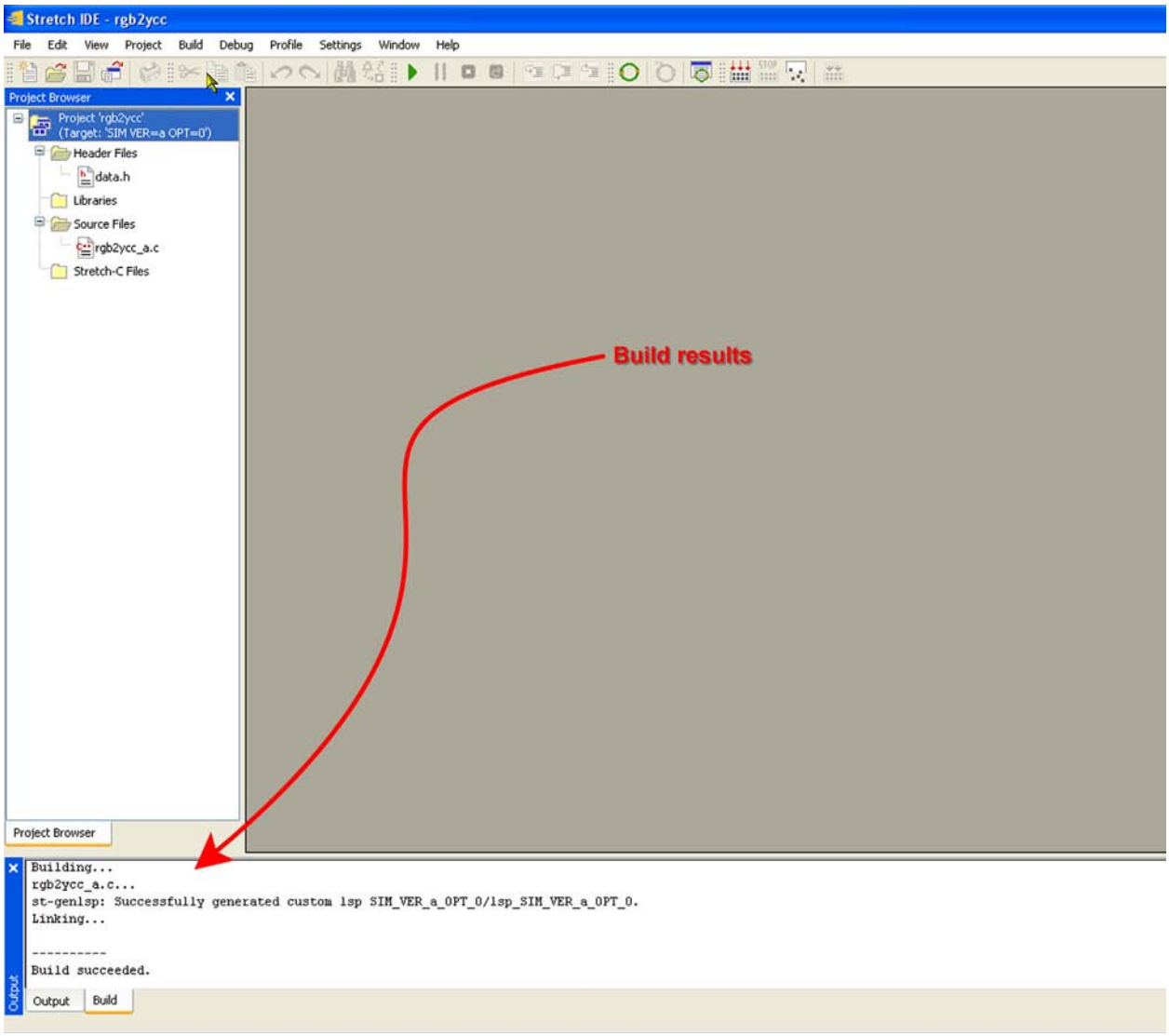
5. Click OK.

To build the application

1. From the menu bar, choose Build →->Build (or press F7).

   The build progress is displayed in the Build window in the bottom section of the IDE (see Figure 3-8).

Figure 3-8 Build progress window

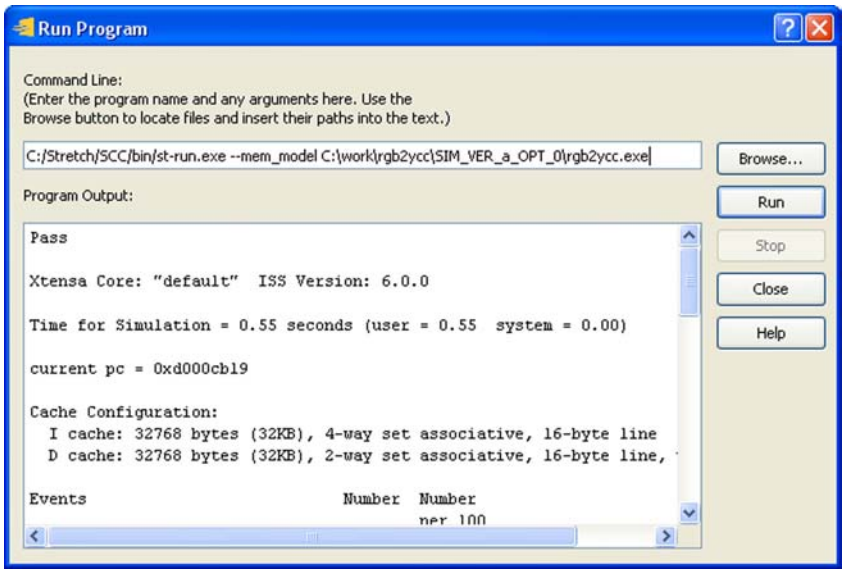


To run the program

1. From the menu bar, choose Debug →->Run Program.

   This displays the Run Program window with a scrolling text box that lists progress messages for the run (see Figure 3-9).

Figure 3-9 Run Program window



2. Click Run.

   You should see output similar to that in Figure 3-9.

3. Scroll to the top of the output and verify that "Pass" is printed, followed by some more information.

   Congratulations! You've created your first application!

4. Click Close.

3.6 Debugging the Application

The IDE provides useful debugging features such as breakpoints, variable watch, memory list, and so on. Some of these features are explored in this section.

To set a breakpoint

Set a breakpoint at the call to the function `rgb2ycc_wrapper()` as follows:

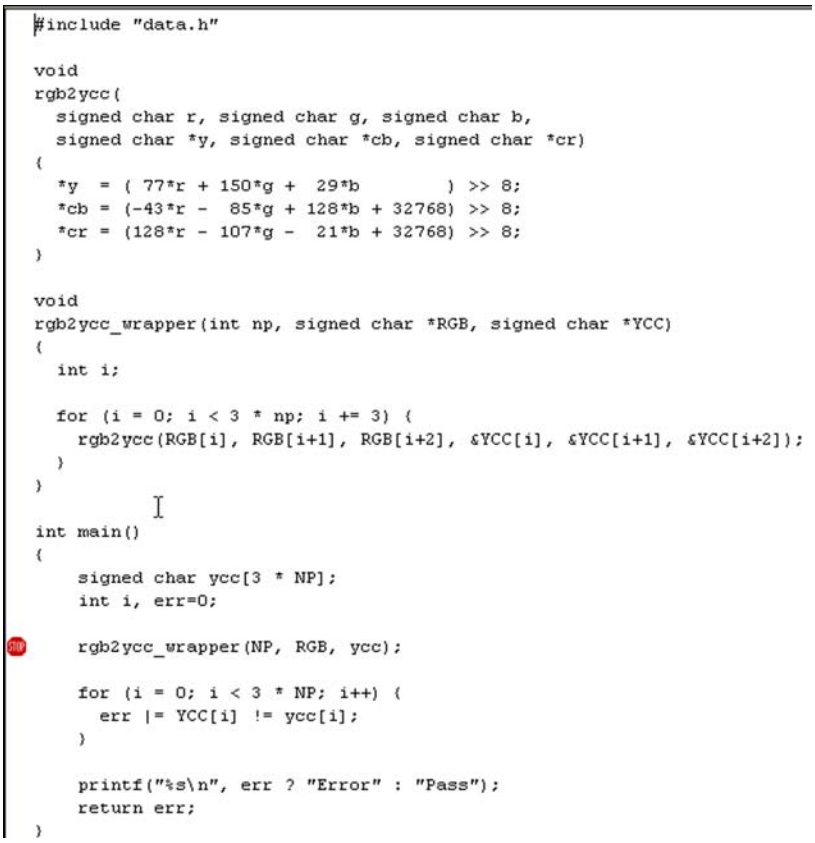
1. Open the file `rgb2ycc_a.c`.
2. Right-click the line "`rgb2ycc_wrapper(NP, RGB, ycc)`" in the `main` function.

   The Breakpoint pop-up menu appears.

3. From the pop-up menu, choose Insert Breakpoint.

A red stop sign appears to the left of the line of code (see Figure 3-10).

Figure 3-10 Setting a breakpoint

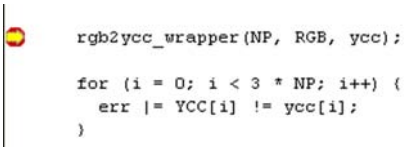


To start debugging

1. From the menu bar, choose Debug →->Start Debugging (or press F5).

   The program halts at the line calling the function `rgb2ycc_wrapper()` (see Figure 3-11).

Figure 3-11 Halted program



NOTE: If you get the message "Breakpoints cannot be set", review the Project Properties →->Compiler Options tab to make sure that the Generate Debug Info check box is checked, then rebuild the application.

To use the variable watch window

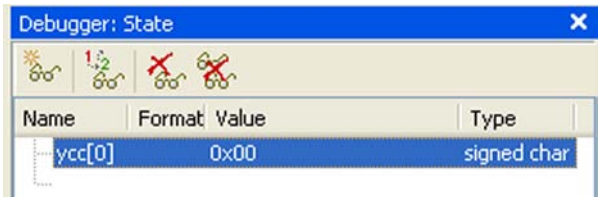
1. From the menu bar, choose, Debug →->Windows →->Watch →->Watch1.

This displays a watch window on the right-hand side of the IDE.

2. In the Name column, type the name of the variable to watch (next to the branching line).

   For example, we typed *ycc[0]* to watch the value of `ycc[0]` (see Figure 3-12).

Figure 3-12 The variable watch window



To step through the program

1. From the menu bar, choose Debug →-> Step Into.

   This steps into the `rgb2ycc_wrapper()` function.

2. From the menu bar, choose Debug →->Step Out.

   This steps out of the `rgb2ycc_wrapper()` function.

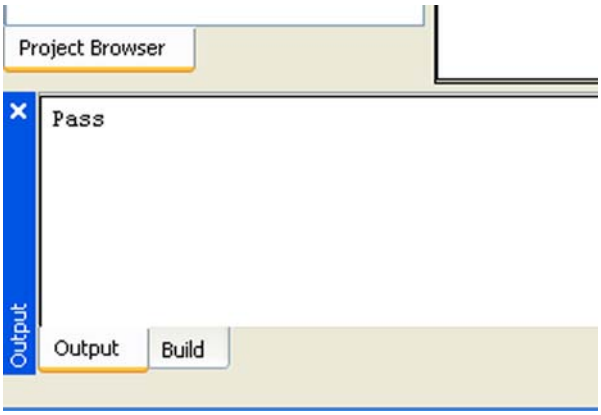
   Notice how the value of `ycc[0]` changes.

To continue through the end of the program

1. From the menu bar, choose Debug →-> Continue.

   This runs to the end of the program. Notice that "Pass" is printed in the output window at the bottom of the IDE window (see Figure 3-13).

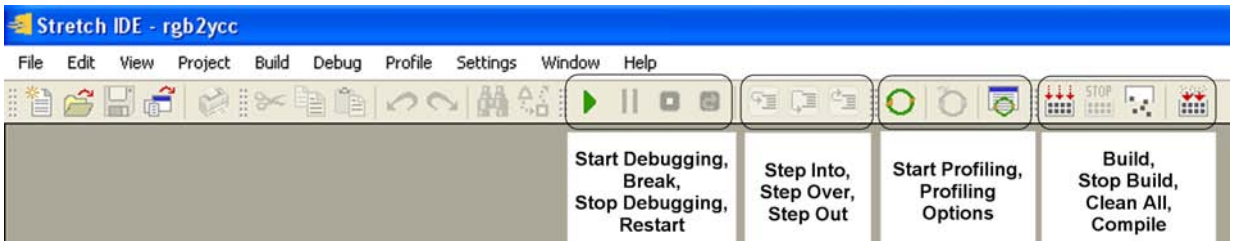
Figure 3-13 Result of complete run



3.7 Using the Toolbar and Hotkeys

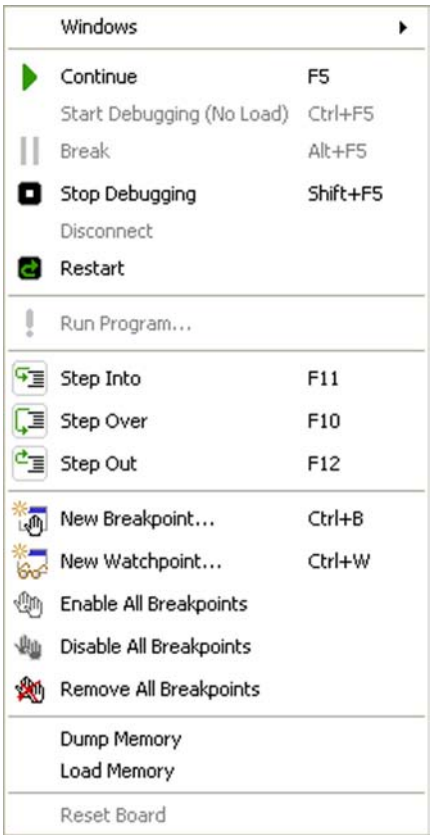
Many of the debugging features can be accessed using the tool bar buttons at the top of the IDE window. Figure 3-14 shows the locations of some of the commonly used functions.

Figure 3-14 Common tool bar options



Similarly, hotkeys can be used to access most of the features. Hotkeys are listed on the right-hand side of menu bar options (for example, Build is F7 and Step Into is F11).

Figure 3-15 Example of hotkeys



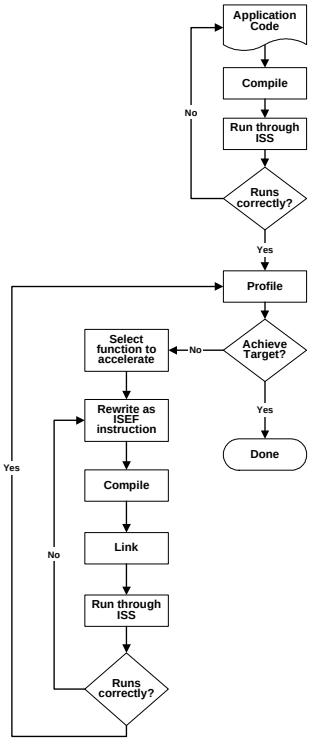
Chapter 4 Stretch C Development Flow

Programs frequently contain computational “hot spots”, or areas of code that consume most of the processing cycles by repetition of a small, but possibly complex computation.

A typical development flow (see Figure 4-1) is used to identify these hot spots by building the application using the Simulator option, just as we have done in the previous sections, then profiling the application using the IDE. Chapter 5, “Profiling the Application”, illustrates how to use the profiler to see the execution times of the various functions in the application. You then use the execution times to identify hot spots in your code. After you have identified the hot spots, you use Stretch C to write special instructions to accelerate their execution. Finally, you modify your application to invoke the special instructions.

When debugging your ISEF instructions, you may choose to first use Native mode to get them functionally correct, then switch to ISS or Remote mode to verify performance.

Figure 4-1 Typical development flow



Chapter 5 Profiling the Application

Profiling is important to identify the "hot spots" of an application. The Stretch IDE provides user-friendly profiling interfaces for analyzing important information like function cycle counts, instruction and data cache stalls and pipeline analysis.

In this section, basic features of the profiler are explored using the application created in Chapter 3, "Creating, Debugging, and Running an Application".

To select the target to be profiled

1. Open the `rgb2ycc` project.
2. From the menu bar, choose Project →–> Select Active Target.

   The Select Active Target dialog appears.
3. From the Select Active Target dialog, choose SIM VER=a OPT=0.
4. Click OK.

To start the profile

1. From the menu bar, choose Profile →–> Start Profiling (or press F3).

   The Profiler progress dialog appears (see Figure 5-1).

Figure 5-1 Profiler progress

To save the profile description

When the profiler is done, it asks if the profile should be saved as the default profile name. For purposes of this example, the default name is acceptable.

1. Click OK.

Figure 5-2 Save Profile As dialog

To view the profile summary

1. From the menu bar, choose Profile →–> Window →–> Profile Summary.

   Figure 5-3 shows the profile summary for all the functions.

Figure 5-3 Profile summary

Profiler: Profile

(Gprof Profile) ☐ Show All

%	Cumulative Cycles	Self Cycles	Calls	Self Cycles/call	Total Cycles/call	Name	Graph
49.71	63348.00	63348.00	??	??	??	ResetH	
24.50	94572.00	31224.00	160	195.15	195.15	rgb2ycc	
12.00	109858.00	15286.00	1	15286.00	62770.88	main	
9.74	122270.00	12412.00	1	12412.00	43636.00	rgb2ycc_wrapper	

In the profile summary, the function `ResetH()` is the reset handler function, which can be ignored. The function `rgb2ycc()`, which is called by the `rgb2ycc_wrapper()` function, consumes the most computation cycles. This is a good candidate for acceleration by a Stretch C Extension Instruction, which is described in the next chapter.

Chapter 6 Stretch C Programming

This chapter explains the basics of programming in Stretch C.

6.1 Example Review

The profile summary in Chapter 5 Chapter 5, "Profiling the Application" indicates that the `rgb2ycc()` function is a hot spot that can be accelerated. The function performs color conversion on a pixel from RGB format to YCbCr format. Following are the equations that apply to the conversion:

$$Y = 0.29900R + 0.58700G + 0.114B$$
$$Cb = -0.16874R - 0.33126G + 0.500B + 128$$
$$Cr = -0.50000R - 0.41869G - 0.08131B + 128$$

The preceding equations are transformed into fixed point equations as follows:

$$Y = (77R + 150G + 29B) \gg 8$$
$$Cb = (-43R - 85G + 128B + 32768) \gg 8$$
$$Cr = (128R - 107G - 21B + 32768) \gg 8$$

These operations are evaluated for every pixel in an image.

In the following steps, we are going to add two more files to the project: a Stretch C file (`rgb2ycc.xc`), which contains the Stretch C functions that implement Extension Instructions to perform the preceding operations; and a source file (`rgb2ycc_b.c`), which uses the Extension Instructions.

6.2 Building and Running a New Target

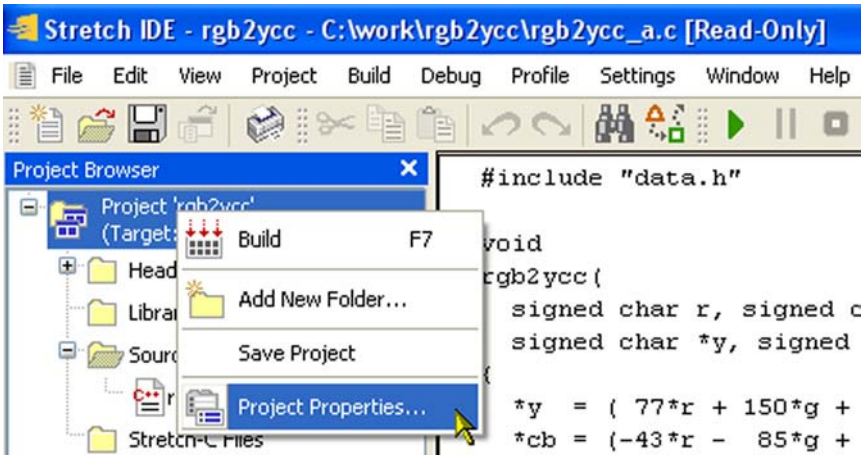
In this section, you learn how to build a target that uses Extension Instructions and to verify the results of the build.

To add a new target to the project

1. From the menu bar, choose Project →->Project Properties.

This can also be done by right-clicking the Project icon in the Project Browser pane and choosing the Project Properties option (see Figure 6-1).

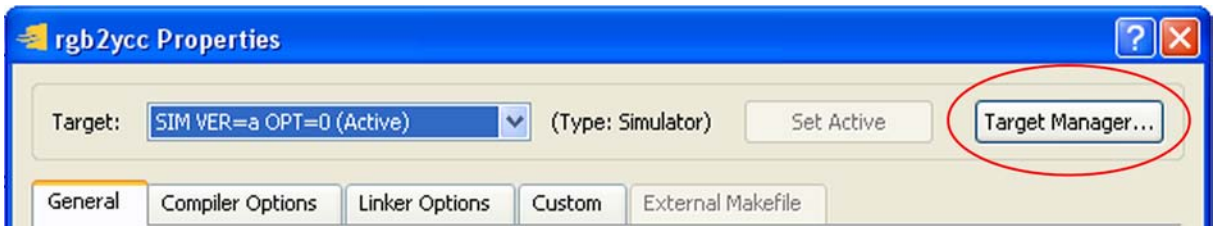
Figure 6-1 Adding a new target to the project



2. In the top right-hand corner of the Project Properties dialog, click the Target Manager button (see Figure 6-2).

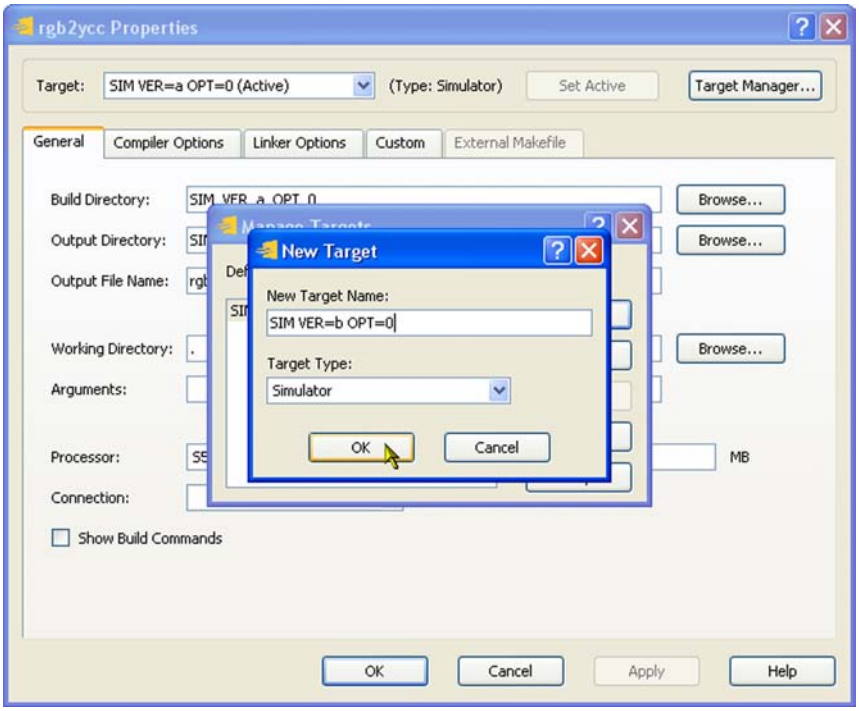
   The Manage Targets dialog appears (see Figure 6-3).

Figure 6-2 The Target Manager button



3. Click New.
4. In the New Target Name field, type "SIM VER=b OPT=0".
5. From the Target Type drop-down menu, choose Simulator.

Figure 6-3 Naming the new target

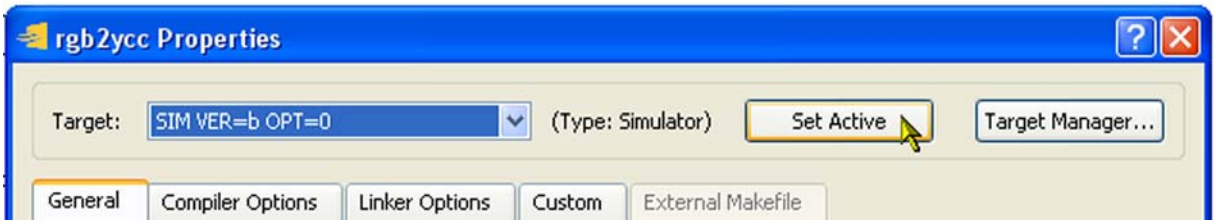


6. Click OK.
7. Click Close.

To set the new target as active

1. From the Target drop-down list on the top left-hand corner of the Project Properties dialog, choose SIM VER=b OPT=0.
2. Click the Set Active button to set the selected target as the active target (see Figure 6-4).

Figure 6-4 Setting new target active



To generate debug information

1. In the Project Properties dialog, click the Compiler Options tab.
2. Click the Generate Debug Info check box.

   This generates debug information for the debugger to use.

3. In the EI header field, change the value to `rgb2ycc.h` (instead of the default `libei.h`).

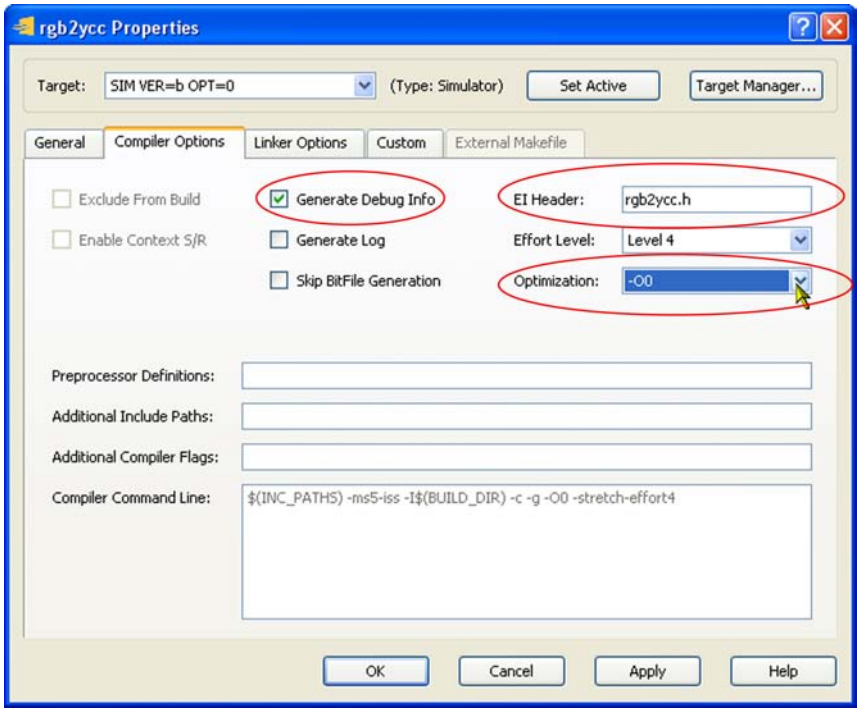
   `rgb2ycc.h` is generated from `rgb2ycc.xc`, and `rgb2ycc_b.c` does an `#include` of it.

4. From the Optimization Level drop-down menu, choose -O0 (see Figure 6-5).

   NOTE: Because this target uses the simulator, you can choose to check Skip Bitfile Generation. Doing so makes the compilation of the `.xc` file run faster, but does not verify that the Stretch C code will fit into the ISEF.

5. Click OK

Figure 6-5 Changing the optimization level



6. Follow the steps in Section 3.4.1, “Adding Files to the Project”, on page 3-4 to add `rgb2ycc_b.c` to the Source File category, and `rgb2ycc.xc` to the Stretch C File category.

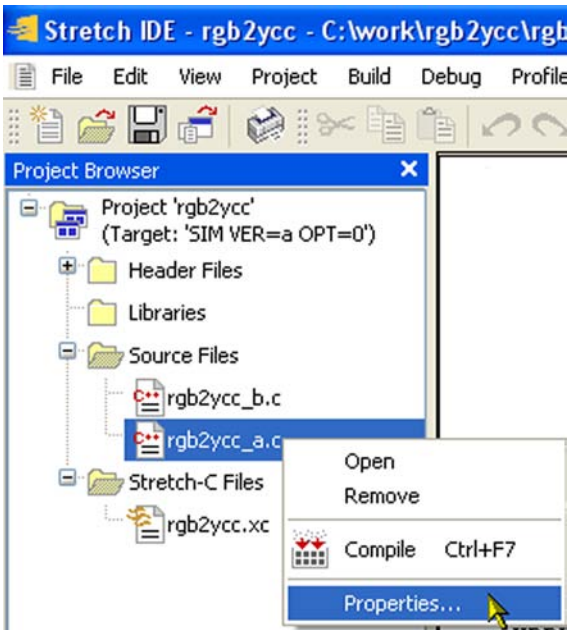
To exclude unneeded files from the build

Exclude `rgb2ycc_a.c` from the build, because only `rgb2ycc_b.c` and `rgb2ycc.xc` are needed in the build, and leaving it in will result in two `main()` functions.

1. In the Project Browser pane, right-click `rgb2ycc_a.c`.

   This displays a pop-up menu (see Figure 6-6).

Figure 6-6 Displaying right-click pop-up menu



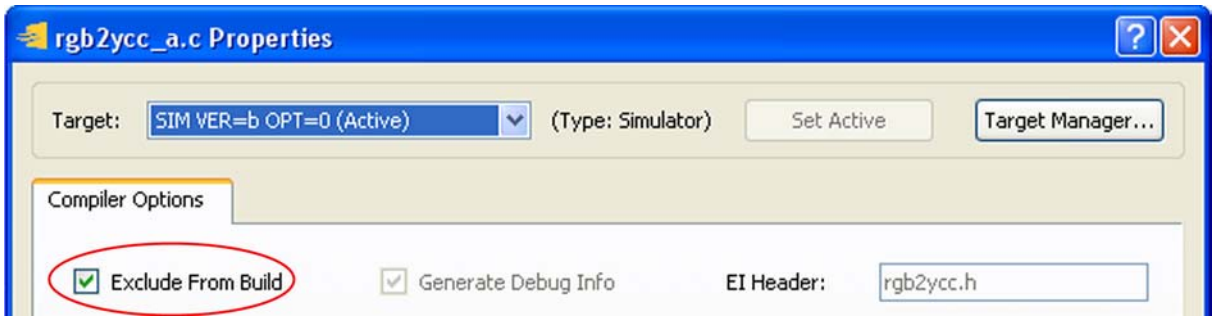
2. From the pop-up menu, choose Properties.

   This displays the Properties dialog for `rgb2ycc_a.c`.

3. From the Properties dialog, click the Exclude From Build check box (see Figure 6-7).

4. Click OK.

Figure 6-7 Exclude file from build



NOTE: If you want to compile the SIM VER=a OPT=0 target again, use the same procedure to exclude the two new files from that target: Choose the SIM VER=a OPT=0 target; choose `rgb2ycc_b.c` and exclude it from the build, then exclude `rgb2ycc.xc`.

To build the new target

1. From the menu bar, choose Build →-> Build (or press F7).

To run the program

1. From the menu bar, choose Debug →->Run Program.

   The Run Program dialog appears.

2. Click Run.

   The program `rgb2ycc_b.c` uses the `rgb2ycc.xc` ISEF Extension Instruction to accelerate the color conversion operation. It prints "Pass" to indicate that the result is correct.

3. Click Close.

6.3 Understanding the Stretch C Accelerated Application

This section explains in some detail what is happening in the Stretch C file and the Stretch C application. Refer to the *SCP Architecture Reference Part I*, Chapter 2, for a complete description of Stretch C.

6.3.1 The Stretch C File

Figure 6-8 shows the ISEF function, `rgb2ycc()`, that defines the Extension Instruction for the inner loop RGB to YCC conversion computations. `rgb2ycc()` is defined in the Stretch C file `rgb2ycc.xc`. Table 6-1 explains what each part of the code does.

Figure 6-8 ISEF function rgb2ycc()

```
A   #include <stretch.h>

    /* Extension instruction converting 5 pixels */
B   SE_FUNC void rgb2ycc(WR A, WR *B)
    {                                              D
C     se_sint<8> r[5], g[5], b[5];
      se_sint<8> y[5], cb[5], cr[5];
      int i, j;

      /* unpack A to RGB data, does not use any ISEF logic */
      for (i = 0; i < 5; i++) {
E       j = i * 3 * 8;
        r[i] = A(j+7, j);
        g[i] = A(j+15, j+8);
        b[i] = A(j+23, j+16);
      }

      /* converting 5 pixels */
      for (i = 0; i < 5; i++) {
F       y[i]  = ( 77*r[i] + 150*g[i] +  29*b[i]        ) >> 8;
        cb[i] = (-43*r[i] -  85*g[i] + 128*b[i] + 32768) >> 8;
        cr[i] = (128*r[i] - 107*g[i] -  21*b[i] + 32768) >> 8;
      }

      /* pack YCbCr to B */
      *B = (cr[4],cb[4],y[4],
G           cr[3],cb[3],y[3],
            cr[2],cb[2],y[2],
            cr[1],cb[1],y[1],
            cr[0],cb[0],y[0]);
    }
```

Table 6-1 Components of the rgb2ycc() function

A This header file defines arbitrary-sized integer types and the usual arithmetic operations on them. It must be included in all source files that use Stretch-defined declarations and data types.

B The ISEF function is declared as type `SE_FUNC void` and named `rgb2ycc`. All ISEF functions must be declared as type `SE_FUNC void`.

C Variables of type `se_sint<n>` are arbitrary-width data types used to define variables `<n>` bits wide. The term `se_sint<>` is used for signed quantities. Because the width of these data types is arbitrary, `<n>` can be any value required for your application, within the physical limitations of the ISEF.

D The Extension Instruction `rgb2ycc` is defined to have two arguments: A and B. They are declared as type `WR`, which means that they are wide registers. In this example, A is input and B is an output. The asterisk(*) before B means that B is an output. In some cases, we can also use B as an input (that is, in–out). In such cases, we would still declare B in this fashion if the result were to be written out using B.

E Fifteen bytes of the input A are unpacked using the Stretch C extraction operator to create the r, g, and b arrays.

Table 6-1 Components of the rgb2ycc() function

F Computation loop. Because the inputs to the conversion of the five pixels are independent, the Stretch C compiler (scc) recognizes this and executes the five iterations in parallel.

G The values are packed together using the Stretch C concatenation operator and written to the output wide register.

6.3.2 The Accelerated Application

Section 6.3.1 explained each of the parts that make up a Stretch Extension Instruction; this section shows how `rgb2ycc_b.c` uses the Extension Instruction (see Figure 6-9). Table 6-2 describes what each of the components of the code does.

Figure 6-9 Using the Extension Instruction

```
          #include "data.h"
A ————    #include "rgb2ycc.h"

          #if (!defined(__STRETCH_ISS__)&&!defined(__STRETCH_NATIVE__))
          #include <s5000\sx-isef.h>
          #endif

          void
          rgb2ycc_wrapper(signed char *RGB, signed char *ycc)
          {
B ————      WR A, B;
            int i;

C ————      WRGET0INIT(0, RGB);     /* initialize input stream from RGB */
            WRPUTINIT(0, ycc);      /* initialize output stream to ycc */

            /* loop over RGB data, converting 5 pixels at a time */
            for (i = 0; i < NP/5; i++) {

D ————————  WRGET0I(&A, 15);      /* load 5 RGB pixels to A */

E ————————  rgb2ycc(A, &B);        /* convert 5 pixels */

F ————————  WRPUTI(B, 15);         /* store 5 YCbCr pixels from B */
            }

G ————      WRPUTFLUSH();          /* flush output stream */
          }

          int main()
          {
            signed char ycc[3 * NP];
            int i, err=0;
              ..
              ..
          }
```

Table 6-2 Code components

A This is the generated header file from rgb2ycc.xc

B Wide Register definitions for the Input and Output samples.

Table 6-2 Code components

C The `WRGET0INIT()` intrinsic initializes the input byte stream mechanism with the memory address `RGB`.
The `WRPUTINIT()` intrinsic initializes the output byte stream mechanism with the memory address `ycc` where the data is to be written, and the manner in which that address is to be updated for subsequent `PUT` instruction.

D The `WRAGET0I` instruction loads 1–15 bytes from an unaligned memory location into a wide register.

E The `rgb2ycc` invokes the Extension Instruction using function call notation. The calling module supplies all the required I/O arguments when invoking the instruction. This instruction performs five pixels of RGB to YCbCr conversion for every call.

F The `WRPUTI` instruction stores 1–15 bytes to an unaligned memory location from a wide register.

G The `WRPUTFLUSH` intrinsic flushes the output byte stream. It is required when using `WRPUT` instructions to write data to memory.

6.4 Profiling the Stretch C Application

In this section, we show the results of profiling the SIM VER=b OPT=`0` target (the steps are not listed as they are the same as those in Chapter 5, “Profiling the Application”).

Figure 6-10 shows the results of profiling the Stretch C application.

Figure 6-10 Results of profiling the Stretch C application

Profiler: Profile

(Gprof Profile) ☐ Show All

%	Cumulative Cycles	Self Cycles	Calls	Self Cycles/call	Total Cycles/call	Name	Graph
74.61	63348.00	63348.00	??	??	??	ResetH	
17.41	78130.00	14782.00	1	14782.00	20396.25	main	
2.06	79878.00	1748.00	1	1748.00	1748.00	rgb2ycc_wrapper	
1.33	81008.00	1130.00	1	1130.00	3315.25	_vfprintf_r	

You can see that the ISEF-accelerated `rgb2ycc_wrapper()` function, which replaced the two functions `rgb2ycc_wrapper()` and `rgb2ycc()` in the original code, produces a significant performance gain.

To further optimize the performance, you can choose different optimization levels, which instruct the compiler to perform other optimizations like loop unrolling. Refer to the *SCP Architectural Reference Part 1* and *Color Conversion Application Note* for details regarding optimization of Stretch C functions.

As an exercise, follow the steps in this chapter to create, build, and profile a new target named SIM VER=b OPT=3, but set the optimization level to -O3. You will see further cycle reductions for `rgb2ycc_wrapper ()` function.

Appendix A Target Types

The Stretch IDE can create three types of targets: Native, Simulation, and Remote. This appendix explains how to use each of them.

A.1 Native Target Type

Under Native mode, the compiler builds the application for running on an x86 platform (Windows or Linux). Typically, you use this mode to verify the correctness of your application and to debug Extension Instructions.

There are several characteristics of Native mode:

- Compilation and run time are considerably faster, but this mode is not cycle-accurate.
- All debug features are supported by the IDE under this mode.
- It is easier to debug under Native mode. `printf()` as well as other `stdio` functions are supported within Stretch C files for Native mode. In addition, breakpoint and step through are supported.
- Native mode is fully compatible with the cross-compiler for chip or simulator.
- The processor is simulated, but peripherals are not. Extension Instructions behave like function calls.
- It can be used with standard GNU tools.
- The IDE profiling feature is disabled for Native targets.

A.2 Simulator Target Type

Under Simulator mode, the compiler builds the application for running on the Stretch Instruction Set Simulator (ISS). This mode is cycle accurate, which gives you correct performance evaluation. The ISS, however, simulates only the chip's processor, not its peripherals or DMA engine. Extension Instructions are simulated, but ISEF loading is not—the ISS behaves as if all ISEF

configurations were loaded and are available at all times. Furthermore, memory access times are cycle accurate only for data in the cache and on-chip data RAM.

You can elect to generate the ISEF configuration bitstream by clearing the Skip Bitfile Generation check box in the Project Properties dialog's Compiler Options tab. This can verify that the compiled application can be implemented in hardware. Checking Skip Bitfile Generation makes `.xc` files compile faster.

There are several characteristics of Simulator mode:

- This mode is cycle accurate for the S5 processor, caches, and data RAM, but not for the SRAM or DDR memories.
- The processor caches and data RAM are simulated, but the chip's peripherals are not.
- `printf()` as well as other `stdio` functions are *not* supported within Stretch C files.
- Profiling is supported under this mode.
- You may skip bitfile generation to save compilation time used for placing and routing ISEF resources.

A.3 Remote Target Type

Under Remote Target mode, the compiler builds the application for running on development boards. ISEF configuration bitfiles *must* be generated under this mode.

There are several characteristics of Remote mode:

- `printf()` and other `stdio` functions are directed to the UART in OCD mode, and directed to the IDE's output window in Redboot mode ; they are *not* supported in Stretch C (that is, from within Extension Instructions).
- Statistical profiling is supported, but ISS-style profiling is not.

Appendix B Creating a Project with an External Makefile

You can use a preexisting makefile to build target applications. Do this by selecting External Makefile when creating a new project. Debug features are supported under this mode. You are responsible, however, for setting the correct build path, output executable file path, file name, and so on to match the settings in the makefile.

To use an external makefile

1. Create an empty project folder `C:\work\rgb2ycc_ext`.
2. Copy all files from `$(STRETCHROOT)\SCC\examples\rgb2ycc`, where `$(STRETCHROOT)` is the root directory where you installed the Stretch tools, into the newly created choose folder.
3. Start the IDE.

To create a new project

1. From the menu bar, choose File → New Project.

   The New Project dialog appears.

2. In the Project Type area, click the External Makefile radio button (see Figure B-1 on page B-2).
3. In the Project Name field, type a name for the project.
4. Click the Browse button to set the project path to the folder you have created.
5. Click Next.

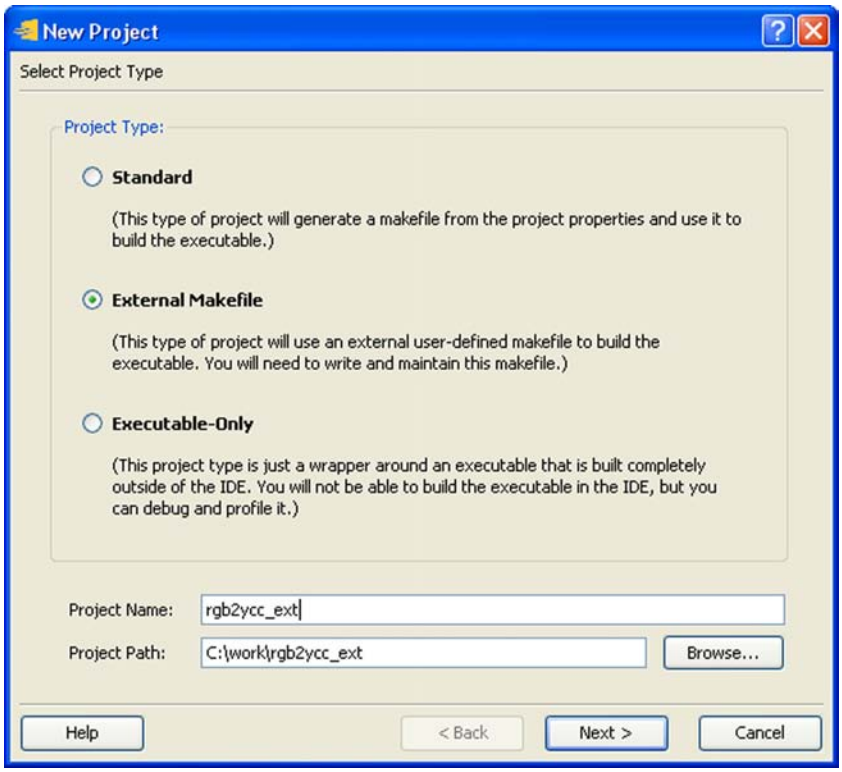
   The Select Target Properties dialog appears (see Figure B-2 on page B-3).

6. In the Target Name field, type a name for the target.
7. In the Select Target Type area, click the Simulator radio button.
8. Click Finish.

To set project properties for a build

1. From the menu bar, choose Project → Project Properties.
2. In the Project Properties dialog, click the External Makefile tab.
3. In the Command line to build target field, type "st-make VER=a TARGET=ISS build" (see Figure B-3 on page B-3).

Figure B-1 Start a new project



4. In the Command line to clean target field, type "st-make clean".
5. Click OK.

To start the build

1. From the menu bar, choose Build → Build

   This makefile generates an executable called `rgb2ycc_a-O0-S5ISS.exe`, which is located in the project directory.

   For the IDE to correctly execute the program, you must set the output directory path to the project path, and you must rename the executable (see Figure B-4 on page B-4).

2. From the menu bar, choose Project → Properties.
3. In the Project Properties dialog, click the General tab.
4. In the Output Directory field, type "." (a single period).
5. In the Output File Name field, type "`rgb2ycc_a-O0-S5ISS.exe`".

   NOTE: In the preceding output file name, "`O0`" is the letter "O" followed by the number zero.

6. Click OK.

Figure B-2 Specify target name

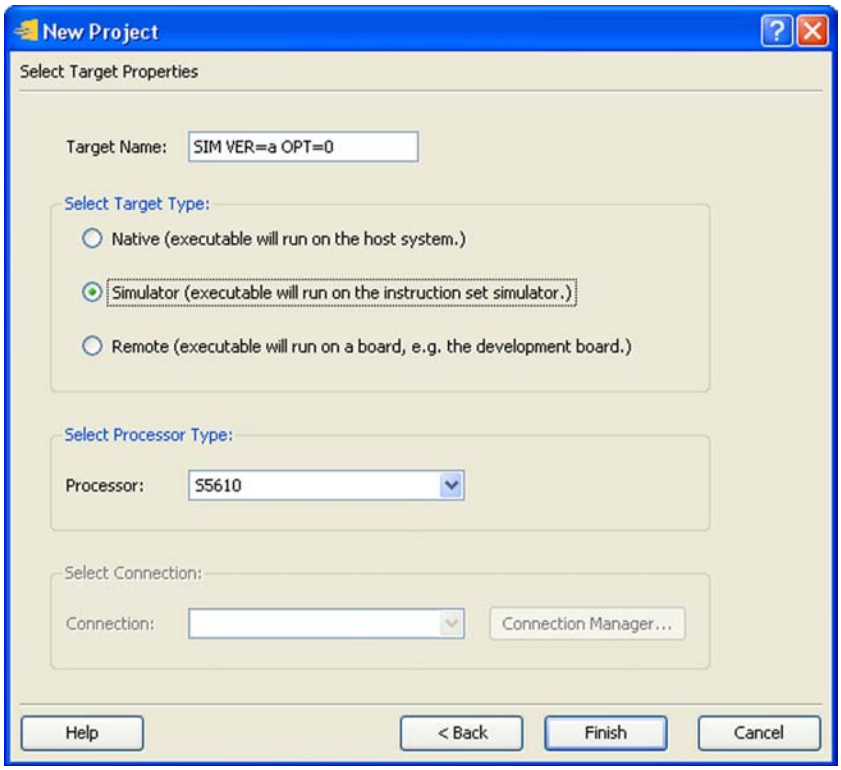


Figure B-3 Supply build and clean information

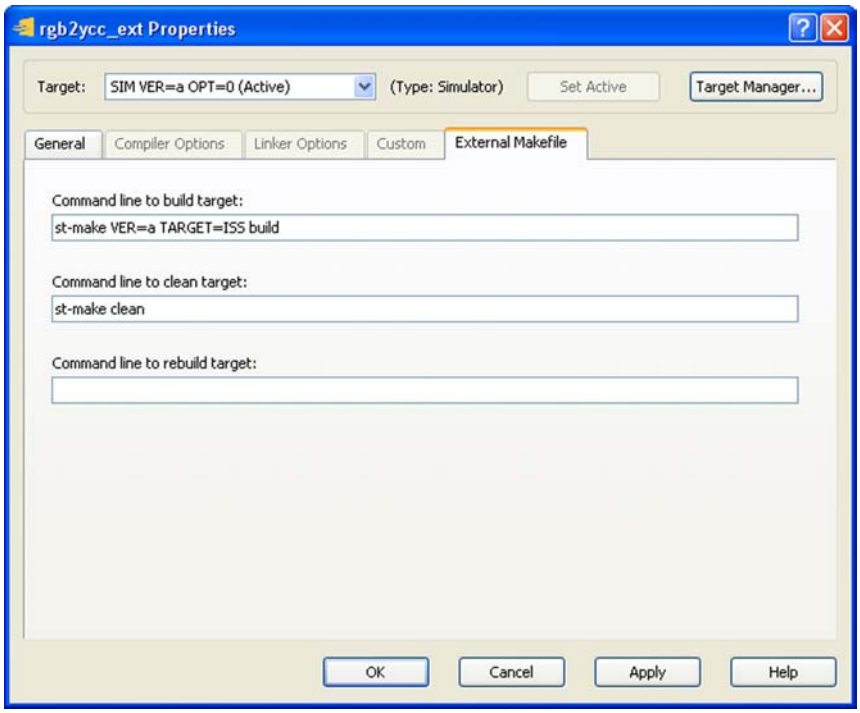
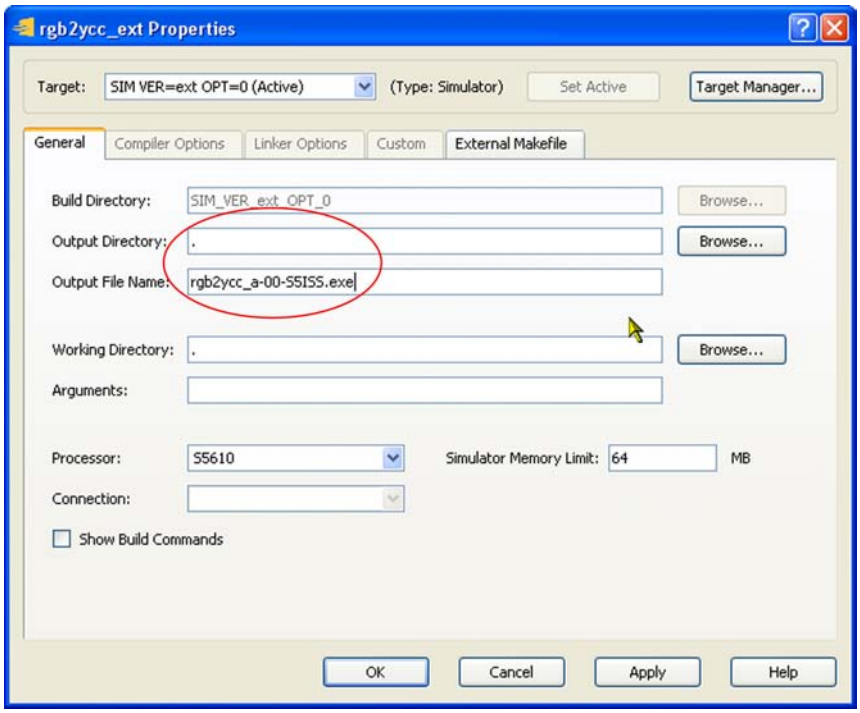


Figure B-4 Set build directory and rename output file



To run the executable

1. From the menu bar, choose Debug → Run Program.

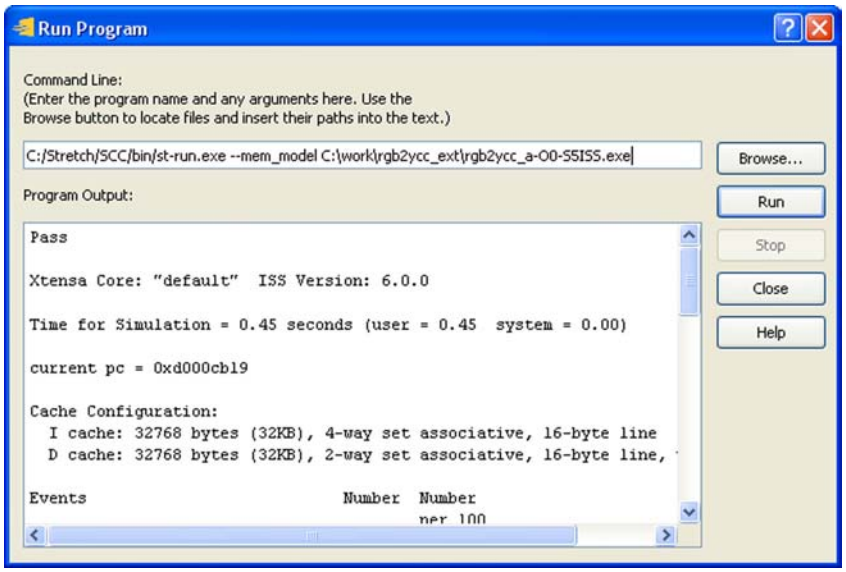
   The Run Program dialog appears.

2. Click Run.

   You should see "Pass" printed at the top of the output window (see Figure B-5 on page B-5).

3. Click Close.

Figure B-5 Run results



Appendix C Compiling and Running Using the Command Line

In addition to using the IDE, you can compile and run the applications using the command line.

C.1 Basic Building and Running

To compile the Stretch-C file

```
> scc -g -O0 -stretch-h rgb2ycc.h -o rgb2ycc.a rgb2ycc.xc
```

To compile the application C file and link with the Stretch C file

```
> scc -g -O0 -o rgb2ycc_b.exe rgb2ycc_b.c rgb2ycc.a
```

To run using the simulator

```
> st-run rgb2ycc_b.exe
```

To profile using the profiler

```
> st-run --mem_model --profile=gmon.out rgb2ycc_b.exe
> st-gprof rgb2ycc_b.exe > rgb2ycc_b.exe.prof
```

This generates a profiler output file called `rgb2ycc_b.exe.prof`

C.2 Debug Using st-debug

`st-debug` is a command line debugger that lets you perform debugging operations like loading, setting breakpoints, single-stepping, and so on. You can use `st-debug` for debugging in Simulator, Native, and Remote modes. For detailed information about `st-debug`, refer to the *Stretch Command Line Debugger User's Guide*.

C.2.1 Native Mode

To build the application, type

```
> scc -g -O0 -ms5-native -stretch-h rgb2ycc.h -o rgb2ycc.a
      rgb2ycc.xc
> scc -g -O0 -ms5-native -o rgb2ycc_b.exe rgb2ycc_b.c
      rgb2ycc.a
```

To run the application using st-debug

1. Type the following at the prompt:

```
> st-debug rgb2ycc_b.exe
```

This starts the debugger mode and displays the following:

```
Stretch Command-line Debugger
(st-debug)
```

2. Type the following at the st-debug prompt:

```
run
```

This displays the following:

```
Starting program: rgb2ycc.exe
Pass
Program exited normally.
(st-debug)
```

C.2.2 Simulator Mode

To build the application

1. Type the following at the prompt:

```
> scc -g -O0 -ms5-iss -stretch-h rgb2ycc.h -o rgb2ycc.a
      rgb2ycc.xc
> scc -g -O0 -ms5-iss -o rgb2ycc_b.exe rgb2ycc_b.c
      rgb2ycc.a
```

To run the application using st-debug

1. Type the following at the prompt:

```
> st-debug rgb2ycc_b.exe
```

This starts debugger mode:

```
Stretch Command-line Debugger
(st-debug)
```

2. Type the following at the st-debug prompt:

```
run
```

This displays the following:

```
Starting program: rgb2ycc.exe
Pass
Program exited normally.
(st-debug)
```