1 Help Contents

CoDeveloper™
from
Impulse Accelerated Technologies

Version 3.70

Please review the Before You Begin section for important information.

1.1 CoDeveloper Product Overview

Welcome to CoDeveloper™ Version 3.70

Welcome to CoDeveloper: advanced software tools enabling high-performance applications on FPGA-based programmable platforms. CoDeveloper allows you to:

- Generate optimized VHDL or Verilog from C.
- Partition your algorithms using a multiple process, parallel programming model.
- Iteratively optimize, pipeline and parallelize your C-language applications for high performance.
- Describe and generate I/O, in the form of streams, signals, memories and registers.
- Create HDL modules for use with other hardware components, or..
- Create complete hardware/software systems for high performance, FPGA-based computing.
- Target a wide variety of available FPGA platforms.

CoDeveloper is designed for use with leading tools for embedded software and programmable hardware development, including FPGA synthesis tools, desktop C/C++ development tools and embedded compiler and debugger tools.

The complete CoDeveloper environment consists of a set of libraries allowing Impulse C applications to be executed in a standard C/C++ desktop compiler, for simulation and debugging purposes, as well as cross-compiler and translation tools allowing Impulse C applications to be implemented on selected programmable hardware platforms. Additional tools for application profiling and co-simulation with other environments, including links to EDA tools for hardware simulation, are provided.

Introducing Impulse C

Impulse C is a library of functions and related datatypes that provide a programming environment, and a programming model, for highly parallel applications targeting FPGA-based platforms. Impulse C has been designed to simplify the expression, verification and compilation of complex applications consisting of multiple (potentially hundreds) of distinct parallel processes. Impulse C has been optimized for mixed software/hardware targets, with the goal of abstracting details of inter-process communication and allowing relatively platform-independent application design.

CoDeveloper includes the Impulse C libraries and associated software tools that help you use standard C language for the design of highly parallel applications targeting FPGAs. CoDeveloper includes an Application Manager, Application Monitor, compatibility with common development environments (including Microsoft Visual C++, Eclipse, and tools based on GCC), platform-specific FPGA compilers and other resources for Impulse C application developers.

For Module Generation and for Processor Acceleration

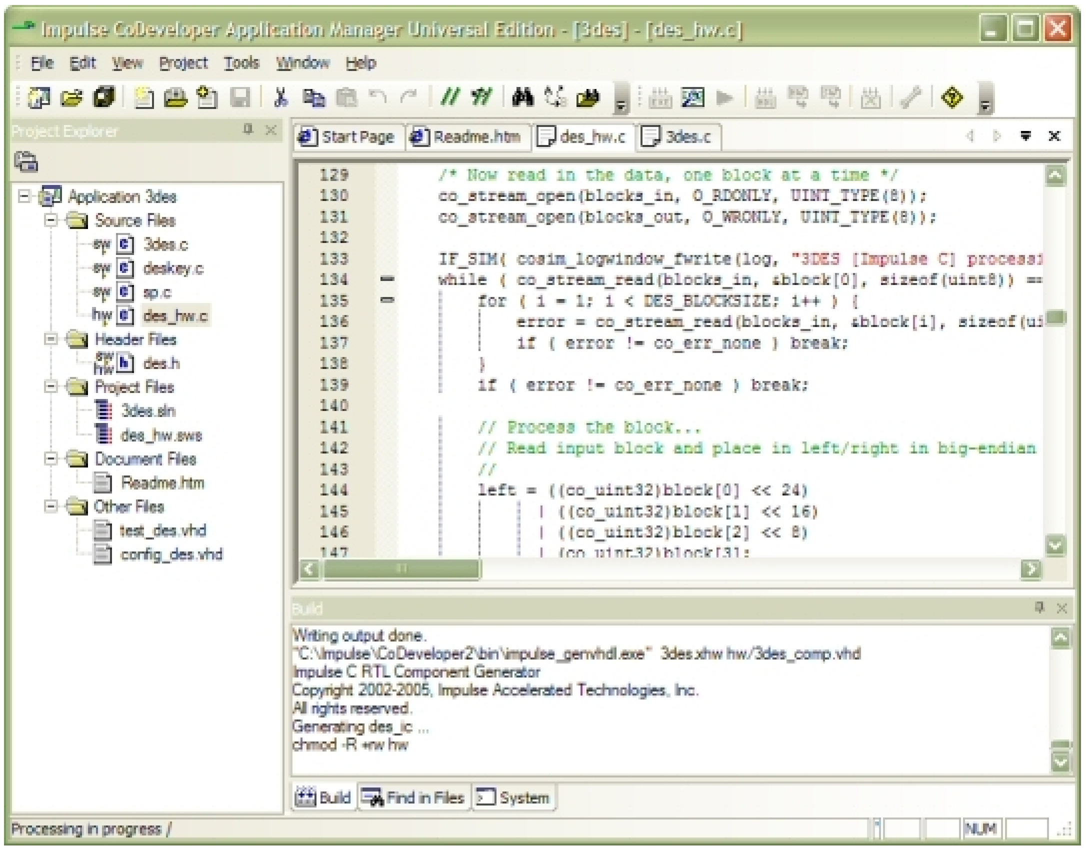
There are two primary uses for Impulse C, and which of these two uses best fits your particular needs will impact how you use the Impulse C in combination with other FPGA development tools.

Module generation refers to the use of Impulse C, along with other methods of design (including VHDL and/or Verilog, and possibly other design tools based on Simulink or other high-level entry methods) to create one or more modules, or blocks, of intellectual property (IP) that will be combined with other IP to create a complete system. In this mode of use, you will use Impulse C to describe one or more connected hardware processes, generate hardware for these processes, then connect them, using traditional hardware design methods and tools, to other components in your system. For this purpose, you will need some understanding of hardware design and hardware description languages (HDLs). With this knowledge, you will be able to use Impulse C as a front-end design tool to rapidly create synthesizable hardware components. To understand how to interface your Impulse C processes to other components, you will also need to have a good understanding of Impulse C stream, signal, memory and register interfaces and how they relate to the generated hardware interfaces.

Processor acceleration refers to the use of Impulse C, in combination with a Platform Support Package, to create a hardware-accelerated system. In this mode of use, your goal is to create one or more hardware accelerators that are directly connected, via predefined and automatically configured stream and memory communication channels, to an FPGA embedded processor, or to a host computer for server acceleration. To achieve this, you will not need to have deep knowledge of hardware design, and you do not need to be an experienced FPGA designer. You will, however, need to have a basic understanding of the platform tools provided to you by your FPGA vendor, or the supplier of your FPGA-based computing system, and you will need to study the platform-specific tutorials and examples provided with the CoDeveloper product. It is also helpful if you have performed at least one of the platform tutorials provided by your FPGA vendor (or FPGA system vendor) so you understand the basic steps for creating an FPGA-based computing system.

The CoDeveloper Application Manager

The CoDeveloper Application Manager can be used to quickly generate new Impulse C applications, as well as to manage existing applications and their constituent files. You can use the Impulse C Application Manager to create entirely new projects (perhaps using the Impulse C Design Assistant to help create new Impulse C files) or to help manage existing Impulse C projects.



```
    /* Now read in the data, one block at a time */
    co_stream_open(blocks_in, O_RDONLY, UINT_TYPE(8));
    co_stream_open(blocks_out, O_WRONLY, UINT_TYPE(8));

    IF_SIM( cosim_logwindow_fwrite(log, "3DES [Impulse C] processi
    while ( co_stream_read(blocks_in, &block[0], sizeof(uint8)) ==
        for ( i = 1; i < DES_BLOCKSIZE; i++ ) {
            error = co_stream_read(blocks_in, &block[i], sizeof(ui
            if ( error != co_err_none ) break;
        }
        if ( error != co_err_none ) break;

        // Process the block...
        // Read input block and place in left/right in big-endian
        //
        left = ((co_uint32)block[0] << 24)
                | ((co_uint32)block[1] << 16)
                | ((co_uint32)block[2] << 8)
                | (co_uint32)block[3];
```

```
Writing output done.
"C:\Impulse\CoDeveloper2\bin\impulse_genvhdl.exe"  3des.xhw hw/3des_comp.vhd
Impulse C RTL Component Generator
Copyright 2002-2005, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating des_ic ...
chmod -R +rw hw
```

Because Impulse C is compatible with standard third-party compilers, you can use CoDeveloper in combination with popular development and debugging tools to form a complete application development environment.

Getting Started

To get started using CoDeveloper and Impulse C, you should load one of the tutorial sample projects included in the Examples subdirectory of your Impulse C installation. The examples provided are intended to demonstrate a variety of useful Impulse C concepts, including various methods of specifying inter-process communication and the writing of Impulse C test routines (commonly called *test benches*). These examples will also help you to understand how to create and manage an Impulse C project.

To load a sample project, select the "Sample Projects" tab in the CoDeveloper Start Page. Select one of the sample projects.

When you have opened a sample project, you will see one or more source files listed in the Impulse C Projects view. You can double click on any file name listed to open a source file editing window.

Note: Additional examples and tutorials are included with each Platform Support Package that you have installed with your CoDeveloper software, and on the Impulse website.

See Also

About Impulse C
Application Monitor
Quick Start Tutorials

1.1.1 Before You Begin

Release Notes

This is release 3.70 of CoDeveloper, from Impulse Accelerated Technologies. For a list of changes from previous versions, please see Version History.

HOW TO GET STARTED

Tutorials

Before starting with CoDeveloper, we strongly suggest that you make use of the supplied tutorials, either by following the steps presented or by casual reading. Doing so will make your initial work with CoDeveloper and Impulse C much easier, and will provide you with useful insights into parallel programming and mixed hardware/software development methods.

Several Quick Start Tutorials are provided in this Help document.

The Impulse website hosts the most up-to-date version of the Tutorials and other documentation:

http://www.ImpulseC.com/Tutorials

Documentation

You can access this Help document, the *CoDeveloper User's Guide*, from the Help menu in the CoDeveloper Application Manager.

Documentation is provided with your CoDeveloper installation in the form of compiled HTML Help files (.chm). You can open any of these files, found in the "Help" and "Architectures" directories of your CoDeveloper installation, by double-clicking them.

Linux
The Linux edition of CoDeveloper does not include a graphical development environment. For documentation on developing Impulse C applications using command line tools, see the section Command Line Tools.

Platform-Specific Documentation

Tutorials and documentation for specific FPGA boards or platforms are included in the Platform Support Package (PSP) Help files installed with your software. Open the PSP Help files from the

CoDeveloper Application Manager's Help menu; select "Platform Support Package Help" and choose the .chm file for your platform from the dialog that appears.

Linux
Platform Support Package Help files are available in PDF format in the "Help" subdirectory of the CoDeveloper installation.

Sample Projects

A number of sample Impulse C projects are included in this installation. Once you have a sample project open, you can review the Impulse C source code, build and run a desktop simulation, or invoke the Impulse C hardware compiler to generate HDL and hardware/software interface files.

To work with these examples:

1. Start the CoDeveloper Application Manager by selecting Start > Impulse Accelerated Technologies > CoDeveloper 3.70 > CoDeveloper Application Manager.
2. The Start Page will appear; you can also open the Start Page by selecting the View > Start Page menu item.
3. Click the Sample Projects tab to display a page listing many Impulse C projects.
4. Click a project link to open the project.

Linux
Sample Impulse C projects are found in the $IMPULSEC_HOME/Examples directory. See the section Command Line Tools to learn how to build an Impulse C project in Linux.

VISUAL STUDIO USAGE NOTES

CoDeveloper includes an optional plugin for Microsoft Visual Studio. This plugin lets you simulate, debug, and generate HDL for an Impulse C project from within the Visual Studio environment. Documentation for the Visual Studio plugin can be found here:

http://impulsec.com/AppNotes/APP112_VISUAL_STUDIO_08/APP112_VISUAL_STUDIO_08.pdf

The Impulse C simulation libraries are compatible with and tested in the Visual Studio version 6, .NET (version 7), 2005 (version 8), and 2008 (version 9) environments. Sample projects for desktop simulation with Visual Studio may be provided with your installation. Please note that if you copy these sample projects out of the installation directory to your own working directory, you may need to modify the include and library file path settings within the Visual Studio project.

SYSTEM REQUIREMENTS

Windows

The following is required to make full use of CoDeveloper:

- CPU with 1GHz or better clock speed, 32- or 64-bit Intel/AMD architectures
- 1GB RAM
- Windows 2000, XP, or Vista

The following additional software is optional:

- Microsoft Visual Studio, *or*
- Any other Windows-compatible C/C++ development environment

Linux

- CPU with 1GHz or better clock speed, 32-bit Intel/AMD architectures (64-bit supported with a workaround, see below)
- 512MB RAM
- Red Hat Enterprise Linux (RHEL) 5.0 or later, or similar distribution
- gcc 3.4 or later
- GNU make 3.81 or later

INSTALLATION NOTES

The following comments apply to all releases:

- The CoDeveloper Version 3 software can be installed on the same computer as CoDeveloper Version 2, but you will need to change the environment variables IMPULSEC_HOME and IMPULSE_GCC_HOME to switch between versions. For example, if Version 3 is installed in "C:\Impulse\CoDeveloper3" and Version 2 is installed in "C:\Impulse\CoDeveloper2", then you will need to change IMPULSEC_HOME to reflect the version you are currently using.
- Due to limations in FPGA tools, it is not recommended that you install CoDeveloper into a directory that includes spaces in the directory path (such as "C:\Program Files") or create CoDeveloper projects in directories that include spaces (such as "My Documents"). Doing so may cause unexpected errors in third-party FPGA synthesis/place-and-route tools.

Linux

- After installation, the IMPULSEC_HOME, PATH, and LD_LIBRARY_PATH environment variables will be set automatically at login time by a shell script installed in /etc/profile.d. If these variables are not set upon login, they may be set manually by running "codeveloper-profile.sh" (found in the CoDeveloper installation directory).

Linux (64-bit)

- Install 32-bit gcc and libc libraries. These packages' names differ across distributions. For example, Kubuntu requires the lib32gccl and libc-dev-i386 packages.
- Edit $IMPULSEC_HOME/MakeRules/Makefile.rules, adding the "-m32" flag to both CFLAGS and LDFLAGS variables.
- Modify $LD_LIBRARY_PATH to include paths to 32-bit libraries *first*. For example: `export LD_LIBRARY_PATH=/usr/lib32:/lib32:$LD_LIBRARY_PATH`

IMPULSEC_HOME ENVIRONMENT VARIABLE

The CoDeveloper installation creates an environment variable, IMPULSEC_HOME, that must be set to the installation directory (typically "C:\Impulse\CoDeveloper3"). This environment variable is used to locate many resources required by the CoDeveloper environment.

For example, IMPULSEC_HOME must point to a directory containing a "Libraries" subdirectory, and the "Libraries" directory in turn must include the Impulse C desktop simulation libraries (impulsec.dll and impulsec.lib in Windows, or libImpulseC.so in Linux).

Windows
The "Libraries" subdirectory must be on the PATH so your compiled application can find impulsec.dll at run-time.

SYNPLICITY USAGE NOTES

To synthesize CoDeveloper outputs using Synplicity Synplify, you will need to create a Synplify project and import the Impulse component library files into that project along with the CoDeveloper-generated HDL files (which are normally written to the "Hardware\hw" subdirectory of your project). If you make use of the Export Generated Hardware feature of CoDeveloper, these library files will be copied to your specified Synplify project area automatically. The Impulse library files must be assigned to the library "impulse" in your Synplify project.

Note also that if you are synthesizing VHDL generated by CoDeveloper using the Synplify or Synplify Pro tools, you will need to modify the Impulse library file "impack.vhd" as follows (beginning at around line 153):

```
  -- NOTE: Synplicity Synplify does not support signatures. Use the commented
version
  -- of the following attribute statement if you encounter errors during analysis.
  --
  function sub(v1, v2 : std_ulogic_vector) return std_ulogic_vector;
  attribute ccs_op of sub [std_ulogic_vector, std_ulogic_vector return
      std_ulogic_vector] : function is  "-";
  -- attribute ccs_op of sub : function is  "-";  -- No signature
```

Change to:

```
  -- attribute ccs_op of sub [std_ulogic_vector, std_ulogic_vector return
      std_ulogic_vector] : function is  "-";
  attribute ccs_op of sub : function is  "-";  -- No signature
```

This change is required due to limitations in the Synplicity VHDL analyzer, which does not support the use of signatures. (Note that this change may result in warnings or other errors in certain VHDL simulators.)

See Also

Version History
License Agreement

1.1.2 Version History

Version 3.70.e

- Updated October 19, 2012
- Added plugin for Visual Studio C++ 2010 Express and Standard editions
- Added support for multi-bank memory in SM Debugger
- Added support for co_memoy_readblock/writeblock with multi-bank memory
- PSP:
  - Fix MicroBlaze DVI PSP HDL to compile correctly with ISE v13.4+
  - Added Altera Qsys PSP supporting Avalon-ST and NIOS II withn Qsys
  - Removed nonexistent Virtex-6 PLB and APU PSPs, please use relevant MicroBlaze PSPs instead
- Performance improvements for impulse_arch3 and impulse_genlib to speed up generating large (>100 processes) designs
- Fixed unintended latch generation in banked memory
- Fixes in primitive functions:
  - Fixed issue with return value timing
  - Fixed issue with multiple primitives executing in the same stage
  - Fixed array argument in primitive call under conditional control
  - Fixed address to array from output pointer
- Fixed extra stream read in pipeline when break appears before call to co_stream_read()
- Fixed crash due to unused global variable

- Fixed Verilog parameter generation in autoip
- Fixed generics mismatch error when running Altera PSPs with SM Debugger

Version 3.70.d

- Updated January 31, 2012
- CoValidator Testbench Generator: New support for Active-HDL
- New support for PSP options
- Bug fixes

Version 3.70.c

- Updated October 31, 2011
- Enabled shared memory configurations in multiple-PE designs
- Fixed a problem with Verilog generation related to 'x' assignments in final else clauses
- Modified the _top.v/.vhd output file to correctly include stream adapters

Version 3.70.b

- Updated July 28, 2011
- Improved optimization performance
- Fix issue with read-only global scalar variable initializations
- Update Xilinx floating point support to v4.0 of the Xilinx libraries
- Improved floating/integer conversion functions for Xilinx targets
- Bug fixes

Version 3.70.a

- Updated June 6, 2011
- Many new optimization features including floating point optimizations
- Improved Stage Master Explorer interface
- Support for register arrays
- Updated and improved PSPs
- Updated examples

Version 3.60.d

- Updated February 1, 2010
- New floating point optimization features
- Added support for math library
- Updated and improved PSPs including XD2000i and GiDEL
- Updated examples

Version 3.60.a

- Updated August 31, 2009
- New PSPs: Xilinx Virtex-6 APU and PLB v4.6
- Updated online tutorials for Xilinx ISE 11.2
- CoValidator: User-defined startup script for ModelSim (autorun.do)
- CoValidator: Refer to Altera libraries when using ModelSim Altera Edition
- CoValidator: Remove CoValidator stream traces on 'clean'
- CoValidator: Don't overwrite ModelSim batch files, to preserve user edits
- No limit to width of streams in desktop simulation
- Bug fix: Multidimensional array offsets calculated with missing bits
- Bug fix: Five bits used to compare 64-bit integers
- Bug fix: Maximum number of predicates exceeded
- Bug fix (XD1000 PSP): Generate makefile to include co_util.c when building library

- Pico Computing PSP for E-14 and later supports CoreGen

Version 3.50.a

- CoValidator Testbench Generator: Create ModelSim HDL testbenches
- New PSPs: Xilinx DVI Filter, for Video Starter Kit; additional PSPs add MicroBlaze FSL and PLB support
- New PSP: Virtex-5 PowerPC PLB v4.6; streams, signals, and registers only
- PSP- and board-specific tutorials updated and moved to web: http://www.ImpulseAccelerated.com/Tutorials/
- "Scalarize array variables" option works with all integer types
- Enforce support/maintenance expiration date in license
- Speed up license checkout, most noticeable in Linux release
- Corrections to Help sections on the CoDeveloper IDE
- Bug fix: Generate one co_port for a co_register with multiple connections to processes
- Bug fix: Visual Studio plugin uses wrong path when CoDeveloper installed in a non-default location
- Bug fix: Visual Studio plugin PSP listing was out of date
- Bug fix: Hardware processes writing streams flush data when closing
- Bug fix: Desktop simulation deadlock in co_stream_close after multiple reopen
- Bug fix: AppMonitor Help not found

Version 3.40.a

- Updated February 17, 2009
- SGI RC100 RASC 2.2 PSP: Includes shared memory (co_memory) support for onboard SRAMs
- New PSPs: Xilinx Open Source Linux Virtex-5 APU and Virtex-4 APU
- Linux installer: Environment variables set automatically on login, using /etc/profile.d
- Linux installer: Fix error w/partial context using chcon
- Linux installer: Non-root user install script, linked on website from install instructions
- Improved compiler reports on pipeline performance
- Updated documentation on pointer support
- Display support expiration date in CoDeveloper Application Manager
- Bug fix: impulse_sm crash when pipelining a primitive in limited cases
- Bug fixes related to pipelining primitives and pointer parameters to primitives

Version 3.30.a

- Updated December 15, 2008
- Improved pipeline scheduling, to allow rate=1 pipelines in some cases
- Microsoft Visual Studio 9 plugin for Impulse C projects
- New example: ComplexFIR for Xilinx MicroBlaze
- Updated Xilinx MicroBlaze tutorials for EDK 10.1i SP3
- Corrections to Impulse C Function Reference for co_register functions
- Clarified Help documentation on HDL testbenches using streams
- Corrections in PowerPC PSP documentation
- Add missing PDF Help documents to Linux release
- Bug fix: Add missing cosim_log.h to Xilinx MicroBlaze FSL PSP
- Bug fix: Buffer overflow in impulse_impc.exe
- Bug fix: Logical "not" precedence error in VHDL
- Bug fix: Stream operation return value comparison
- Bug fix: Out-of-order stream writes w/conditional
- Bug fix: Stage Master Explorer crashes when changing stageDelay several times
- Bug fix: Signed multiply generated mistakenly for int16*uint16

Version 3.20.b

- Updated September 3, 2008
- Stage Master Explorer for Linux
- Updated Xilinx PowerPC and MicroBlaze tutorials for EDK 10.1i
- Reduced logic usage for asymmetric multipliers
- Bug fix for byte-size DMA transfers with PLB v4.6
- Support hardware/software export in Linux for the XD1000 PSP
- Support multiple concurrent software processes communicating with hardware on the XD1000
- Updated XD1000 PSP to support stream base addresses in SOPC Builder >7.0
- Added DGEMM tutorial to XD1000 Help
- Updated Opal Kelly PSP documentation
- Added Opal Kelly Mandelbrot sample project
- Removed Cray XD1 PSP and examples
- Bug fix: Stage Master crashes under Windows Server 2003

Version 3.20.a

- Updated June 25, 2008
- Altera floating-point library for all Altera-based PSPs
- New PSP: Xilinx Virtex-5 APU (streams only)
- New PSP: Xilinx BlueCat Linux MicroBlaze FSL
- New PSP: Xilinx PowerPC PLB v4.6
- MicroBlaze v7 support
- XtremeData XD1000 PSP co_memory interface updated for better performance
- Microsoft Windows Vista support
- Microsoft Visual Studio 8 project generator (Beta)
- New example: DGEMM for XtremeData XD1000
- Updated ComplexFIR tutorial for Xilinx ISE 10.1i
- Updated the ImageFilter5X5 example for better pipelining
- Stage Master Explorer: Annotate HDL with original C source line numbers
- Fewer misleading messages from libCoreGenCommon.tcl
- Bug fix: Add missing mem_if.vhd to Opal Kelly PSP
- Bug fix: Bad names in co_stream_attach (Quartus 7.x)
- Bug fix: New File (Design Assistant) does not add file to project properly
- Bug fix: Help links broken in Eclipse IDE
- Bug fix: Correctly output >64-bit constants when generating HDL
- Bug fix: Latches were inferred for some registers in Verilog when using ISE
- Bug fix: Constant values not generated correctly in HDL under Linux
- Bug fix: Stage Master Debugger (ccycle.exe) not found in some installations
- Bug fix: co_signal outputs not captured correctly in VHDL
- Bug fix: Some co_memory logic was synthesized away in Verilog
- Bug fix: Variable latched on last stage of pipeline got incorrect data

Version 3.10.b

- Updated February 15, 2008
- **First release for Linux**
- Scheduler improvements and bug fixes
- Multiplier generation controlled with CO SET pragmas; support generation as pipelined or non-pipelined DSP blocks, or as LUTs, for Xilinx platforms
- Xilinx CoreGen library support, including pipelined divider and CORDIC functions
- Updated Xilinx PSP documentation on shared memory locations
- Remove Altera Nios Platform Support Package (Nios II still supported)

- Performance updates to some examples
- Fix minor issues in MicroBlaze ComplexFIR tutorial
- Updated SGI PSP docs on co_iterate_hardware and "done" flag, stream sizes; added section on troubleshooting
- SGI PSP automatically generates a bitfile using ISE upon hardware export
- Bug fix: Add missing signals to Verilog co_memory interface
- Bug fix: "Enable logic cannot be generated for net types" error w/Verilog input co_signals

Version 3.00.b

- Updated November 20, 2007
- **First public 3.0 release (Beta)**
- Updated documentation for Eclipse IDE
- New PSP: GiDEL PROCWizard
- New PSP: SGI RC100 (RASC)
- Replaced DRC RPU110-L200 PSP with updated VHDL version
- Updated Pico PSP for E-16; revised software driver for better performance, new bitfile build process
- Added ComplexFIR tutorial example for MicroBlaze and EDK 9.2i to replace 3DES
- Bug fix: Error finding environment variable with recent Altera tools releases
- Bug fix: Error generating memory interfaces for Verilog platforms

Version 3.00.a

- Updated October 13, 2007
- **First 3.0 release (Alpha)**
- Eclipse IDE for managing Impulse C projects, including built-in C debugger
- Support for array offsets and global arrays in co_memory block read/write operations
- Default locations for co_memory
- True dual-port memory for Altera FPGAs
- Xilinx PLB shared memory
- Shared memory I/O timing improvements
- Verilog floating-point support for Xilinx platforms, using XST
- Use divmod modules for modulus and division in Verilog
- New PSP: DRC RPU110-L200, v1.0
- PSP infrastructure: Pass streams' co_kind in connection list (for PSPs' GenerateBUS)
- Sync legacy IDE's State Page sample project list with V3 list
- Minor editing fixes to PSP Help files and sample projects
- Sample project: MicroBlaze MemoryIO example uses single-process producer/consumer
- Sample project: Fix 3DES example VHDL test bench stream timing errors
- Sample project: Fix compilation error in Mandelbrot sample for Virtex-4 APU under EDK 9.1i
- Bug fix: Block RAM generation problem for Verilog PSPs
- Bug fix: Close PLB hardware-to-software streams properly
- Bug fixes: FLATTEN pragma code generation
- Bug fix: Crash with pipelined primitive
- Bug fix: Pipeline with latency >64 does not start (Verilog)
- Bug fixes: Pipeline termination problems
- Bug fix: Floating-point operations inside conditionals

Version 2.20.h

- Updated May 8, 2007
- Updated XtremeData XD1000 Platform Support Package to version 1.1
- Bug fix: sema.vhd not exported for some platforms
- Bug fix: Semaphore counter off by one

Version 2.20.g

- Updated April 27, 2007
- Primitive functions can be pipelined
- Switch statements supported within if-statements
- Bug fix: Xilinx APU co_stream_close malfunctioning
- Bug fix: Xilinx APU HW_STREAM_READ_NB macro returns incorrect value on success
- Bug fix: Passing/storing library options from CoDeveloper options dialog
- Bug fix: Non-32-bit co_signals generated as co_ports have incorrect port sizes
- Bug fix: Various Verilog stream issues (e.g., data always zero)
- Bug fix: Global array logic synthesized away ("wr0" port tied to zero)
- Bug fix: Xilinx floating-point division scheduled incorrectly
- Updates to Sample Project and Help and Support listings on Start Page
- Minor updates to some example projects

Version 2.20.f

- Updated March 1, 2007
- Floating-point types are displayed in Application Monitor
- New project template: One-Process Testbench (Floating-Point)
- Updated Xilinx MicroBlaze tutorials for ML401 board and EDK 8.2
- Improved PLB wrapper timing
- FSL reset connection fix
- Active-low reset supported in FSL wrapper
- Improved support for APU instruction flushing
- Bug fix: APU conflict with FPU instructions
- Bug fix: "long long int" warning in impulse_prep
- Bug fix: Opal Kelly export failed to copy any files
- Bug fix: "driver" directory not exported for Xilinx platforms
- Bug fix: HDL generation problem with co_memory_ptr
- Bug fix: co_signal_wait not blocking
- Bug fix: Problem with multipliers in Altera Quartus
- "Allow double-precision types and operators" option always available

Version 2.20.e

- Updated January 10, 2007
- New optimization: Multiplier generation analyzes casts and constants to minimize size
- New optimization: Multiply/divide by two is replaced by bit shift operation
- New optimization: Modulo by power of two is replaced by logical AND operation
- co_register_read and co_signal_wait no longer require a cycle delay (Xilinx, Altera platforms)
- Non-constant operands supported on right-hand side of shift operations
- Initial support for custom I/O implementations
- Bug fix: Unsupported MPD file parameter causes warnings
- Bug fix: Verilog user IP not included in pcore
- New PointerSort example illustrating memory usage

Version 2.20.d

- Updated December 19, 2006
- New Mandelbrot tutorial for Opal Kelly
- New ComplexFIR example for Xilinx MicroBlaze FSL
- Improved hardware generation error messages
- Bug fix: Global variable access in multiples of same process gives bad HDL
- Bug fix: Export source dir is not taken from Generate Options

- Bug fix: Export Verilog files for Xilinx EDK
- Documentation update: stream burst operations

Version 2.20.c

- Updated December 4, 2006
- Added XtremeData Platform Support Package
- Corrected errors in project templates
- Added new single-process test bench template
- Updated Pro Tools installer
- Added 0-bit signals to Verilog output
- Updated Pico Computing E-14 platform support
- Updated Opal Kelly platform support for FrontPanel 3.0
- Added edge-detection example with 5X5 kernel
- Updated documentation
- Fixed various optimizer defects

Version 2.20.b

- Updated August 31, 2006
- Added support for array function parameters
- Added Lattice FPGA platform support files (Verilog output)
- Added National Instruments LabVIEW FPGA platform support files
- Added Pico Computing E-14 Cardbus platform files
- Fixed bug related to co_register types
- Improved Altera Quartus tutorial documents
- Added Verilog support for Altera Nios 2 platform
- Added quad-word read/write macros for Xilinx Virtex-4 FX platform
- Added library options field to Generate Options dialog
- Added co_stream_config function
- Various optimizer and hardware generation improvements

Version 2.10.g

- Updated June 13, 2006
- Added support for co_semaphore types and related functions
- Added support for n-bit signals via co_signal_create_ex function
- Added FFT32_Fixed and FFT32_Float examples
- Added support for external HDL functions via CO IMPLEMENTATION pragma
- Added support for dual port memories in Altera FPGAs
- Added ExtFunction, Cordic_Sine and CordicMath examples
- Improved support for co_port interfaces
- Fixed problem related to implicit latches in hardware primitives
- Fixed problem related to unary minus operations for float types

Version 2.10.e

- Updated March 28, 2006
- Added support for active-low reset
- Updated tutorial for Xilinx Virtex-4 APU (EDK 8.1)
- Added support for Nallatech DIMEtalk
- Added support for fast clock floating point cores (Xilinx)

- Documentation improvements

Version 2.10.d

- Updated February 28, 2006
- Added support for double-precision floating point (Xilinx targets)
- Resolved an optimization problem with complex control statements
- Fixed library incompatibility with EDK 7.1
- Fixed problem with shared OPB memories
- Added examples ImageFilterKernel and ImageFilterKernel5X5
- Fixed problem with certain mod operations

Version 2.10.c

- Updated February 3, 2006
- Updated examples mult32x32, mult18x18, DNA
- Fixed stream write blocking problem with OPB (Xilinx targets)
- Fixed problems with pointer arguments to hardware functions
- Fixed problem with CO FLATTEN pragma

Version 2.10.b

- Updated January 13, 2006
- Updated FFT example (Xilinx)
- New example: BubbleSort
- Updated Project Templates

Version 2.10.a

- Released December 28, 2005
- Support added for C structs (see restrictions)
- Support added for single precision floating point (Xilinx targets)
- Resolved error with multiple-cycle hardware primitive functions
- New example: EdgeDetect_struct
- Improvements made to Cray XD1 platform support package

Version 2.01.g

- Released November 4, 2005
- New Project Wizard templates added
- Support added for global static arrays added (Xilinx platforms)
- Problem with 64-bit assignments resolved
- Pipeline optimization issues related to type conversions resolved
- Sign-extension problem with division operations resolved
- Problem with FLATTEN pragma hardware generation resolved
- Improvements to Cray XD1 platform support (Beta release)
- Updated Xilinx MicroBlaze tutorials
- Pipeline graph format change

Version 2.01.f

- Updated October 15, 2005
- New Project Wizard added

- Problem with (0 downto 0) VHDL vector conversions fixed
- Problem with array size rounding fixed
- Added Synplicity project file generation (Tools menu)
- Verilog output improvements
- Various optimizer and code generator improvements
- New examples: SquareRoot, RadixSort and Mult32X32

Version 2.01.e

- Updated September 8, 2005
- Support for hardware functions and the PRIMITIVE pragma
- New generate option "Do not include co_ports in bus interfaces"
- New FLATTEN pragma for generating combinatorial logic
- New NONRECURSIVE pragma for forcing non-recursive pipeline optimizations
- Support for .mak files in the project explorer
- Support for abs() operation via macro
- Fixed error with array sizes (rounding up to nearest power of 2)
- Verilog output improvements
- New Prime example demonstrating use of a hardware function

Version 2.01.d

- Updated July 25, 2005
- Support for Flexlm license manager
- Support for std_logic (resolved) types in top-level interface
- New block diagram window in Application Monitor
- Support for external editors
- Optimizer and IDE bug fixes

Version 2.01.c

- Released June 30, 2005
- Support for Verilog output
- New co_port type for module generation
- New Application Monitor interface

Version 2.01.b (Beta)

- Released April 29, 2005
- New Application Manager user interface
- Redesigned Design Assistant
- Redesigned HDL generator
- Preliminary support for Verilog output
- Many optimizer and code generation improvements
- Improved error messages
- Reorganized examples directory

Version 1.22.d

- Released January 28, 2005
- Added co_par_break statement
- Resolved issue with multiple pipelines and redundant HDL declarations

- Resolved issues with "const" array declarations
- Updated Help file tutorials for updated Xilinx and Altera software
- Added log file output (CoDeveloper.log) for transcript messages
- Various optimizer and code generation improvements

Version 1.22.c

- Released November 30, 2004
- Added MicroBlaze FSL uClinux Platform support and related example
- Added Mandelbrot examples
- Resolved issues with co_bit_* functions
- Various optimizer and code generation improvements

Version 1.22.b

- Released September 26, 2004
- Added support for memory type "asyncrom" for Xilinx targets (co_array_config)
- Added FFT and DCT examples
- Added "fast" (two clock cycle rate) 3DES example for Xilinx targets

Version 1.22.a

- Released September 7, 2004
- Added support for shared OPB memory in Xilinx MicroBlaze FSL platform
- Various optimizer and code generation improvements

Version 1.21.e

- Released August 16, 2004
- Added support for Altera Nios II Platform Support Package
- Resolved issue with Makefile.rules format

Version 1.21.d

- Released July 30, 2004
- Resolves issues with Make and SHELL variable
- Improved transcript window. (Uncheck the "use external make window" option to use the internal transcript instead of using an external command window for Make processes.)
- Added support for Virtex II Pro (PowerPC) Platform Support Package
- Numerous RTL compiler and IDE bug fixes

Version 1.21.b

- Released July 2, 2004
- Improved support for shared memories
- New example: MemoryIO (demonstrates use of shared memories on Xilinx platforms)
- Automatic Makefile generation
- Impulse_unroll program renamed to impulse_prep
- Numerous bug fixes

Version 1.20.c

- Released April 22, 2004

- Support for integer division added for hardware processes
- Support for co_register and associated functions added
- Support for loop unrolling (UNROLL pragma) added
- Improved Support for shared memories
- New example: Fir51 (FIR filter with loop unrolling, demonstrates UNROLL pragma)
- New example: SimpleCounter (demonstrates use of streams and signals)
- New example: UpDownCounter (demonstrates use of co_register for hardware process)
- Numerous bug fixes

Version 1.10.j

- Released March 15, 2004
- Support for shared memories
- Improved documentation and examples for hardware interfaces (Universal Version)
- New hardware simulation tutorial (Universal Version)
- New examples (ImageFilter and ImageFilterDMA) for Altera
- Application Monitor peformance improvements
- Numerous bug fixes

Version 1.10.g

- Released February 20, 2004
- New text editing component
- New licensing system
- First commercial version

Version 1.10.c

- Released February January 5, 2004
- Release candidate 1 (final Beta)

1.1.3 License Agreement

Impulse CoDeveloper™ End User License Agreement

LICENSE. Impulse CoDeveloper is provided under license from Impulse Accelerated Technologies, Inc. Impulse Accelerated Technologies hereby grants you, as a Customer and Licensee, a non-exclusive license to use the accompanying software programs on a single CPU at any given point in time. The Licensee is bound by the terms of the license purchased and may use the software accordingly. A "Node-locked" license must be used on the computer for which the license was granted. A "Floating" license may be used on any compatible computer in a computer network in a local area network (LAN) configuration. For the purpose of this agreement a LAN is designated as a set of computers interconnected to each other wherein the computer serving as the license server is NOT physically separated by the client computer running the software by more than one kilometer.

Product(s) provided under this agreement are copyrighted and licensed (not sold). Impulse Accelerated Technologies and its Licensors retain title to the software and any patents, copyrights, trade secrets, and other intellectual property rights therein. Impulse Accelerated Technologies does not transfer title to the Products to Licensee.

TERM AND TERMINATION: This Agreement is effective from the date the software is received using any medium (downloaded from the internet/intranet, transferred via disks, etc.) until this Agreement is terminated. Any unauthorized reproduction or use of the Program and/or documentation will immediately terminate this Agreement without notice. Upon termination you are required to destroy/un-install the installed software and documentation.

COPYRIGHT AND PROPRIETARY RIGHTS: Both United States Copyright Laws and International Treaty provisions protect the Program and documentation. This means that you must treat the documentation and Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is No Possibility of its being used by two people at the same time.

RESTRICTIONS: The Software contains copyrighted material, trade secrets, and other proprietary information. You may not de-compile, reverse engineer, disassemble, or otherwise reduce the Software to a human-perceivable form. The Software may include software and/or hardware-based license management features. You may not reverse engineer the license management software. You may not modify the hardware or software configuration of your computer to create a false identity (host-id) for your computer. You may not modify or prepare derivative works of the Software in whole or in part.

DISCLAIMER AND LIMITATION OF LIABILITY: This software and documentation are licensed "AS-IS", without warranty as to performance. Impulse Accelerated Technologies EXPRESSLY disclaims ALL warranties, express or implied, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE. In no event shall Impulse Accelerated Technologies be held liable for any damages due to the use or inability to use of the software distrubution or parts thereof, even if Impulse Accelerated Technologies has been advised of the possibilities of such damages.

RESELLING: Only Impulse Accelerated Technologies authorized distributors may resell or distribute this product. Unauthorized resale of, or transfer of license to, the Software is specifically prohibited.

EXPORT RESTRICTION: You agree that you will not export or re-export the Software, reference images or accompanying documentation in any form without the appropriate government licenses.

© 2002 - 2015 Impulse Accelerated Technologies. All rights reserved.

1.1.4 About Impulse C

impulse accelerated technologies

About Impulse CoDeveloper

Impulse CeDeveloper™ allows you to compile C-language directly into optimized logic ready for use with popular FPGA devices. Use the Impulse tools to quickly prototype mixed software/hardware systems and perform design iterations in just minutes or hours, instead of days or weeks.

Impulse CoDeveloper includes the Impulse C software-to-hardware compiler, interactive parallel optimizer, and Platform Support Packages supporting a wide range of FPGA-based systems. Impulse tools are compatible with all popular FPGA platforms and tools.

Impulse CoDeveloper is for software application developers and FPGA designers seeking a faster path to FPGA hardware. CoDeveloper is ideally suited to image and video processing, digital signal processing, data compression/encryption, and hardware-accelerated computing. The Impulse C compiler is a high-level synthesis tool that is based on standard ANSI C. The Impulse tools enable highly iterative, software-oriented design methods and are compatible with a wide variety of FPGA-based platforms.

CoDeveloper allows you to use standard C tools and programming methods to accelerate your embedded, workstation and server applications.

- **Partition your code** between the processor and the FPGA accelerator.
- **Use multiple parallel processes** for increased performance.
- **Verify and debug** using familiar C-language development tools.
- **Compile and optimize** using interactive, graphical tools.
- **Parallelize and pipeline** your critical C code.
- **Generate FPGA hardware** ready for use with your selected FPGA-based platform.
- **Generate host-FPGA** interfaces for selected platforms.

Please note that Impulse C is not intended for directly converting large, monolithic C applications (typically consisting of many C subroutines called using a remote procedure call, or RPC, programming model) to hardware. The CoDeveloper tools include advanced C-to-hardware compilation capabilities, but these are intended to be used on a process-by-process basis. It is up to you, as the application developer, to make use of the CoDeveloper tools and Impulse C libraries to effectively develop applications appropriate for these new categories of programmable hardware platforms. In the tutorials included in this document (and in the Platform Support Package documents that are also included with CoDeveloper) we'll show you how.

See Also

Impulse C User Guide
The Programming Model
Product Overview
CoDeveloper Eclipse IDE
CoBuilder User Guide

1.1.5 Contents

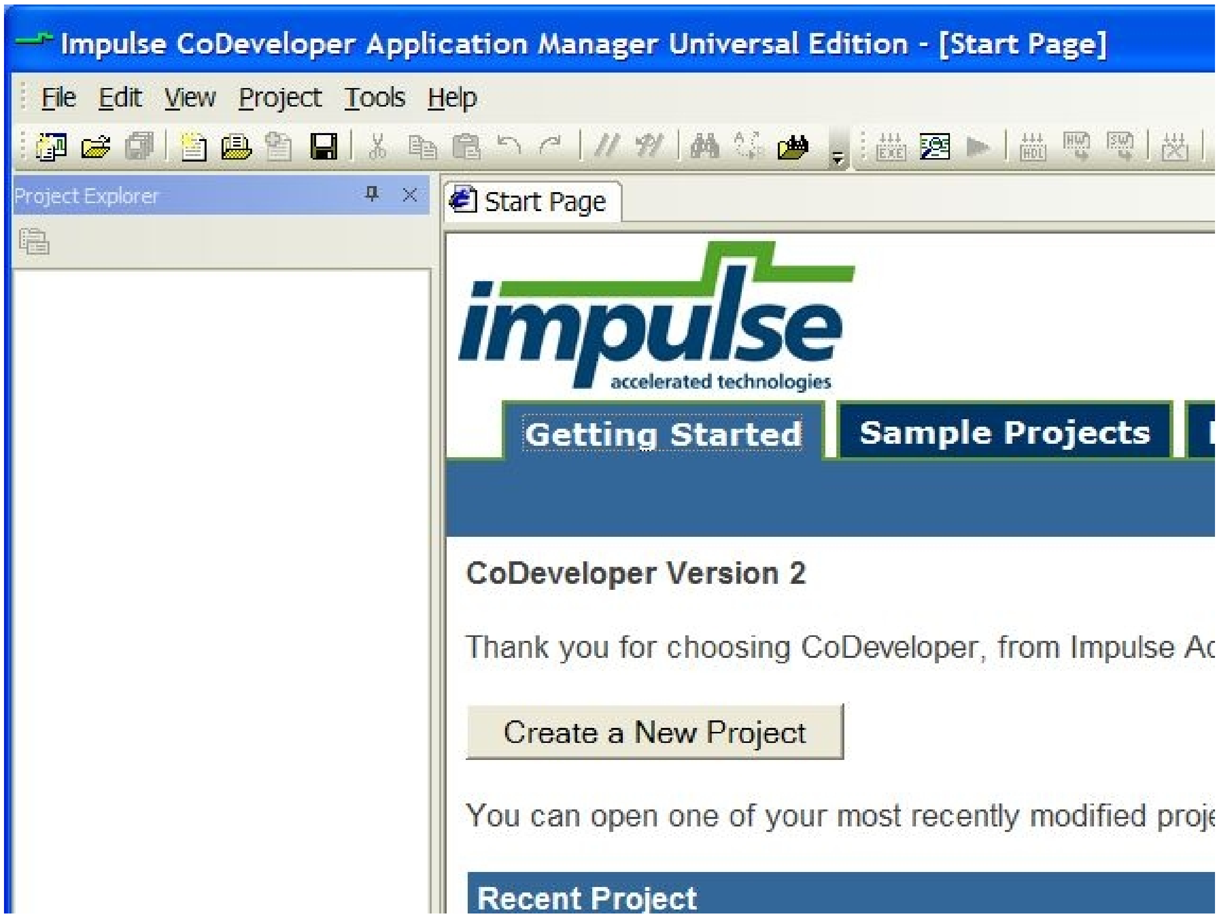
CoDeveloper™
from
Impulse Accelerated Technologies

Version 3.70

Please review the Before You Begin section for important information.

1.2 CoDeveloper Application Manager

CoDeveloper Application Manager IDE


Impulse CoDeveloper Application Manager Universal Edition - [Start Page]
File Edit View Project Tools Help
Project Explorer
Start Page
Getting Started
Sample Projects
Create a New Project
Recent Project

CoDeveloper Version 2

Thank you for choosing CoDeveloper, from Impulse Ac

You can open one of your most recently modified proje

Application Manager IDE Start Page

Simulation Features
Screen Displays
Design Management Features

1.2.1 Simulation Features

impulse accelerated technologies

Verify Designs with Three Levels of Simulation

The Impulse C development environment supports three levels of design simulation:

- Desktop software simulation
- Cycle-accurate hardware simulation
- HDL testbench generation

Each type of simulation can be run in a matter of seconds after making changes to your Impulse C code. These features give you the power to quickly track down defects, analyze performance, optimize parallel hardware, and verify the output of the Impulse C compiler.

Desktop Simulation Using Standard Programming Environments

Desktop, or "software", simulation involves running all the components of an Impulse C application on the desktop development PC. While an Impulse C application can include code whose final implementation is intended for both the FPGA fabric (in the form of hardware processes) and an embedded or host CPU (software processes), desktop simulation executes all of these components together as software running in the development PC's CPU. The advantages of software simulation are:

- **Rapid prototyping**: Change C code to change the design
- **Fast simulation times**: Run compiled C code natively on a desktop CPU
- **Iterative development**: Design changes are ready for testing at a software development pace--recompiling takes seconds

The CoDeveloper Application Manager and CoMonitor Application Monitor, when used in conjuction with a standard desktop compiler, form a complete system for the compilation, execution, and monitoring of Impulse C design descriptions. The system includes an Impulse C build facility (based on Make) and Impulse C instrumentation functions useful for simulating and debugging your Impulse C applications.

For desktop simulation, Impulse C applications are compiled into a native Windows executable that may be run directly or executed from within a debugger. While any C compiler and debugger can be used, CoDeveloper includes GNU gcc/gdb and also offers a Microsoft Visual Studio plugin. When launched, Impulse C executable files interact with the CoMonitor Application Monitor system to allow detailed investigation of your multi-process application. Impulse C instrumentation functions allow processes and streams to be monitored and log output to be written.

Cycle-Accurate Hardware Simulation

Stage Master Debugger simulates the Register Transfer Level (RTL) hardware generated by the Impulse C compiler alongside the software portions of the application. You can step through the hardware portions of your design clock cycle by clock cycle, as they would execute on the FPGA, using a simple graphical debugger. The original Impulse C source code is displayed, highlighting the statements that execute in parallel during simulation. Stage Master Debugger gives rapid feedback on how source code changes affect hardware performance, speeding the optimization process.

Generate HDL Testbenches and Close the Loop

The CoValidator Testbench Generator integrates into the entire Impulse C design flow to build confidence in the result of C-to-FPGA compilation. During desktop simulation, CoValidator traces events on the I/O channels of your design and creates test vectors. When generating hardware, CoValidator creates a custom HDL testbench for your application that reads the same test vectors used in desktop simulation. Finally, CoValidator generates scripts for multiple HDL simulators, so you can simulate your project with a single command. The result: a ready-to-run environment for verifying the RTL output of the Impulse C compiler. The outputs of HDL simulation are written to test vector files that can be compared directly to the software simulation's output, thus "closing the loop".

See Also

Simulating Impulse C Applications
Application Monitor
Build Software Simulation Executable
Launch Software Simulation Executable
Stage Master Debugger

CoValidator Testbench Generator
Simulation Options

1.2.1.1 Simulating Impulse C Applications

Simulation is an important element of the application development process. During simulation your goal will be to verify that your application and its component algorithms operate as expected, both in terms of algorithmic correctness and in terms of required performance.

Simulating for correct application behavior—behavioral simulation—is relatively easy once you have developed the necessary test stimulus and written a corresponding set of software tests for your application. Because Impulse C is compatible with standard C and C++ programming environments, you can use the full power of C programming to create complex, repeatable software test suites, and you can use standard C debugging tools to track down and fix errors in your application.

When simulating your application on your development machine ("desktop simulation", or "software simulation"), it is important to understand that the parallelism you express using Impulse C's stream, message, and process model are implemented by your desktop compiler using the threading model and associated libraries associated with your chosen environment. This means that the parallel behaviors you specify and the timing relationships between independently running processes may be very different from the behaviors and timing of your application when implemented on an actual programmable platform target.

Hardware simulation, using Stage Master Debugger, also executes on the desktop CPU. However, this time the hardware portions of your application run clock cycle by clock cycle as they would in the FPGA hardware. Software processes still execute as threads, at the whim of the operating system scheduler.

HDL simulation, where the hardware portions of your application are simulated as Register Transfer Level (RTL) logic in a third-party tool such as ModelSim, gives an even more accurate representation of how your Impulse C code will run on the FPGA. Performing HDL simulation requires a "testbench", an HDL program that stimulates the various hardware signals in your design to send and receive test data ("test vectors") between the simulator and your Impulse C hardware module. The CoValidator tool is capable of automatically generating testbenches for Impulse C applications, tracing the execution of desktop simulation to create test vectors used by the testbench.

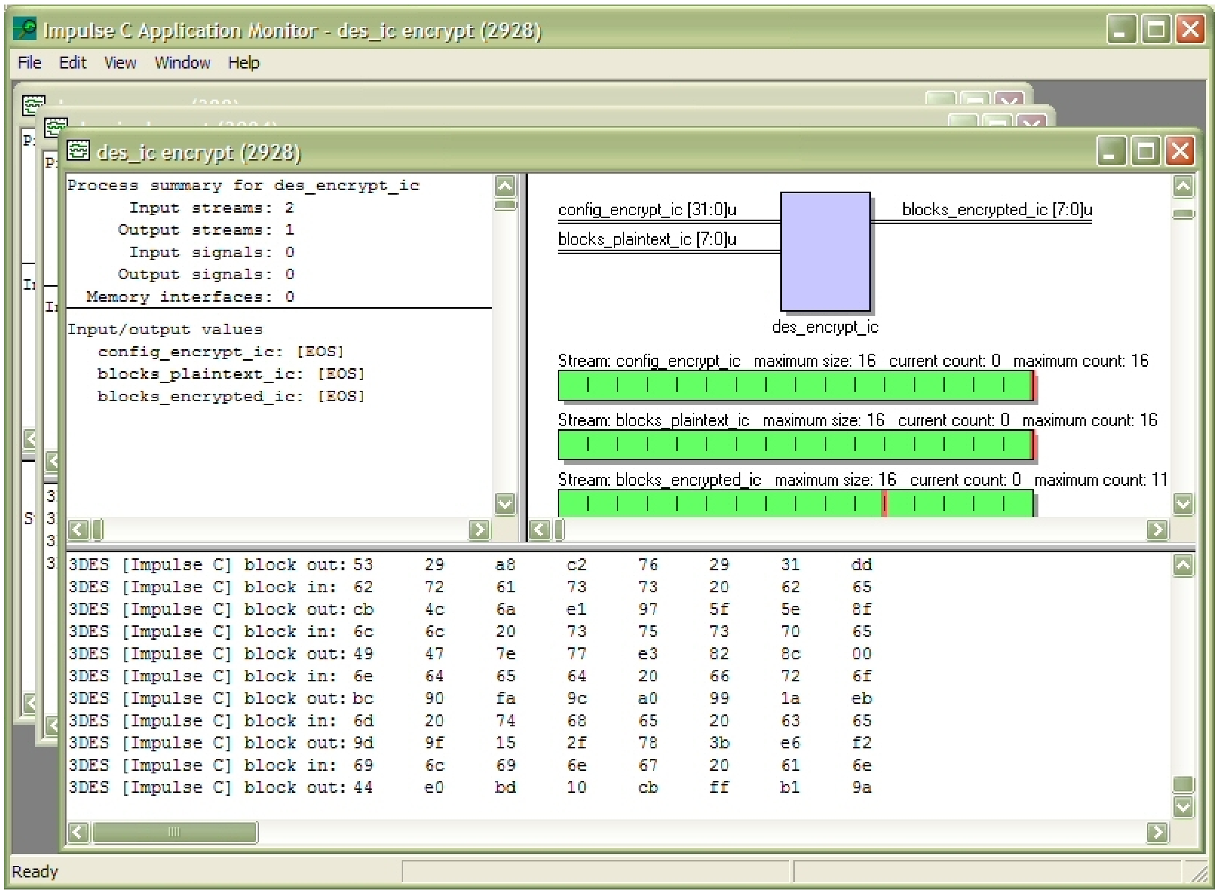
See Also

Simulation Features
Simulation Options
Software Simulation Executable

1.2.1.2 Application Monitor

Description

The CoMonitor Application Monitor allows you to observe your application as it executes, capturing messages, stream data values and other information generated as a result of instrumenting your application. One or more Application Monitor log windows may be created by your application, with each window corresponding to a separate Impulse C hardware or software process.


Impulse C Application Monitor - des_ic encrypt (2928)
File Edit View Window Help
des_ic encrypt (2928)
config_encrypt_ic [31:0]u
blocks_plaintext_ic [7:0]u
blocks_encrypted_ic [7:0]u
des_encrypt_ic
Stream: config_encrypt_ic maximum size: 16 current count: 0 maximum count: 16
Stream: blocks_plaintext_ic maximum size: 16 current count: 0 maximum count: 16
Stream: blocks_encrypted_ic maximum size: 16 current count: 0 maximum count: 11
Ready

```
Process summary for des_encrypt_ic
      Input streams: 2
     Output streams: 1
      Input signals: 0
     Output signals: 0
  Memory interfaces: 0
Input/output values
   config_encrypt_ic: [EOS]
   blocks_plaintext_ic: [EOS]
   blocks_encrypted_ic: [EOS]
```

```
3DES [Impulse C] block out: 53    29    a8    c2    76    29    31    dd
3DES [Impulse C] block in:  62    72    61    73    73    20    62    65
3DES [Impulse C] block out: cb    4c    6a    e1    97    5f    5e    8f
3DES [Impulse C] block in:  6c    6c    20    73    75    73    70    65
3DES [Impulse C] block out: 49    47    7e    77    e3    82    8c    00
3DES [Impulse C] block in:  6e    64    65    64    20    66    72    6f
3DES [Impulse C] block out: bc    90    fa    9c    a0    99    1a    eb
3DES [Impulse C] block in:  6d    20    74    68    65    20    63    65
3DES [Impulse C] block out: 9d    9f    15    2f    78    3b    e6    f2
3DES [Impulse C] block in:  69    6c    69    6e    67    20    61    6e
3DES [Impulse C] block out: 44    e0    bd    10    cb    ff    b1    9a
```

How You Use It

To make use of the Application Monitor, you must call the **cosim_logwindow_init** function in your Impulse C application's configuration function to set up the connection between your application and the Application Monitor.

To display an Application Monitor window for a process, call the **cosim_logwindow_create** function in that process.

The following functions may be used in your application to format and display text messages in a log window:

cosim_logwindow_write(logwindow, str) - This function writes an unformatted string value to the specified log window.

cosim_logwindow_fwrite(logwindow, formatstring, arg1, arg2...) - This function writes a formatted string value to the specified log window.

If your Impulse C application includes calls to these functions and you have invoked the Application Monitor prior to executing your simulation executable, then your simulation will communicate with the Application Monitor while it runs, displaying debugging messages and other information as appropriate.

Refer to the CoMonitor on-line help (Help > Help menu item) for additional details.

See Also

Build Software Simulation Executable

Launch Software Simulation Executable
Simulation Options
cosim_logwindow_init
cosim_logwindow_create
cosim_logwindow_fwrite

1.2.1.3 Build Software Simulation Executable

Description

This menu item builds a Software Simulation Executable. CoDeveloper runs the "make" program to invoke the C compiler and linker, generating a Windows-compatible executable.

What You Do

To generate a simulation executable, do the following:

1. (Optional) Select Project > Options from the menu bar to bring up the Build Options dialog. Alternatively, you can bring up the dialog by clicking on the Options toolbar button. Set "Desktop Simulation (.EXE) Build Options" as needed. Click on the Close button to close the dialog and save the options.
2. Select the Build Software Simulation Executable item from the Project menu, or click the Build Software Simulation Executable toolbar button.

See Also

Build Options
Software Simulation Executable

1.2.1.4 Launch Software Simulation Executable

Description

This option launches the project's software simulation.

Simulation options available in the Simulate tab of the Options dialog box allow you to:

1. Specify whether the simulation executable should always be updated (rebuilt) prior to execution, and
2. Specify whether the Application Monitor should be started prior to your simulation executable being launched. The Application Monitor allows you to observe the behavior of your application through the use of process-specific monitoring windows.

What You Do

To run the project's software simulation executable, do the following:

1. (Optional) Specify the name of your simulation executable, including path information as appropriate, in the Simulate Options dialog.
2. Select the Project > Lauch Software Simulation Executable item from the menu bar or click on the Launch Software Simulation Executable toolbar button. The selected simulation executable is executed in a separate Windows Command Prompt window.

See Also

Simulation Features
Simulation Options
Software Simulation Executable

1.2.1.5 Software Simulation Executable

Description

A **Software Simulation Executable** is a native Windows executable file created as a result of compiling and linking your Impulse C application. Simulation executable files are intended for desktop simulation of your application, and may be run under the control of a standard C debugger. The CoMonitor Application Monitor may also be used as a debugging aid if appropriate instrumentation functions have been included in the application.

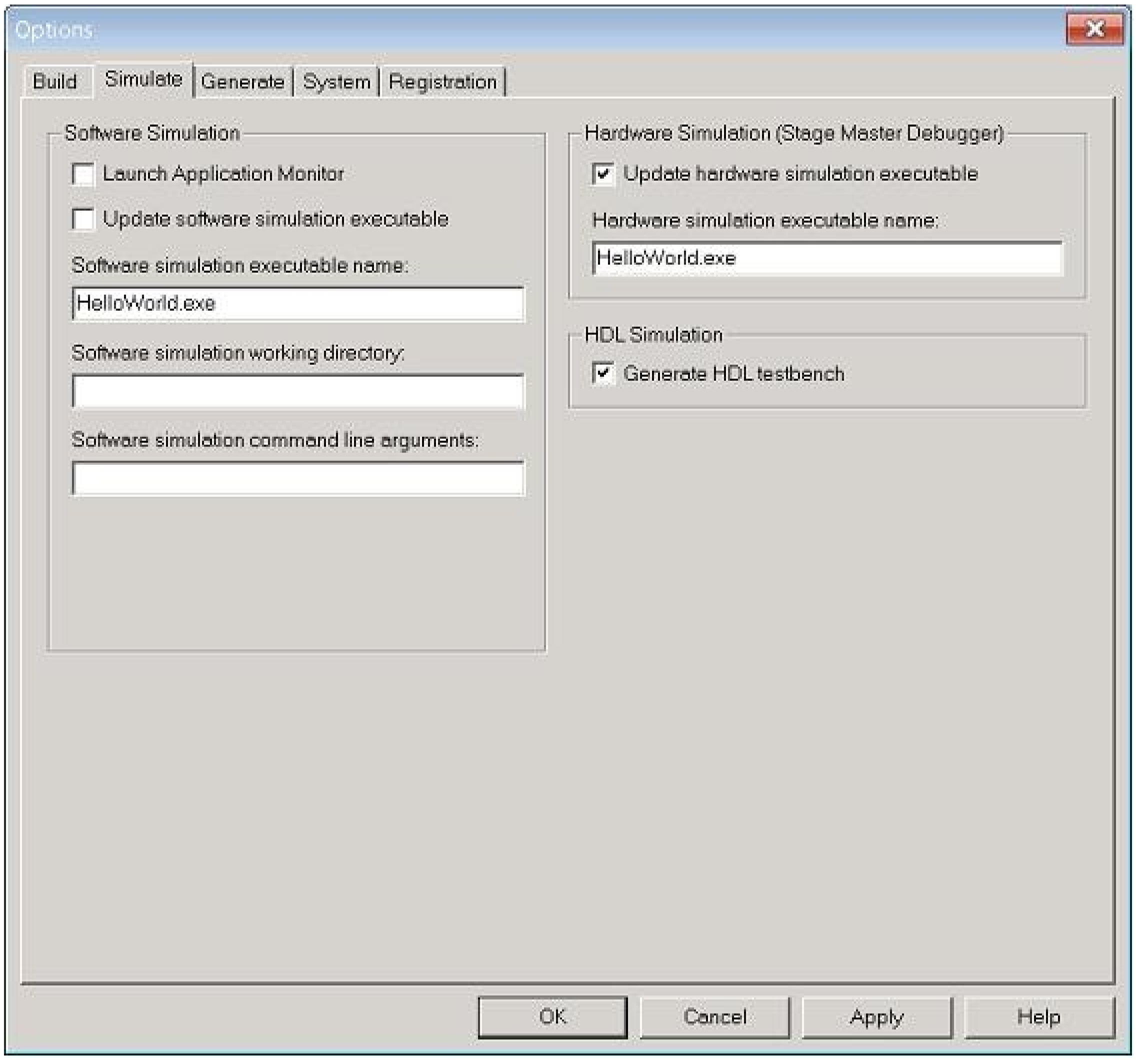
See Also

Simulation Features
Build Software Simulation Executable
Launch Software Simulation Executable
Simulation Options

1.2.1.6 Simulation Options

Description

The Options dialog contains a tab entitled "Simulation". The project options displayed here control desktop (software) simulation, hardware simulation using Stage Master Debugger, and HDL testbench generation using CoValidator.



Software Simulation
- [ ] Launch Application Monitor
- [ ] Update software simulation executable

Software simulation executable name: HelloWorld.exe

Software simulation working directory:

Software simulation command line arguments:

Hardware Simulation (Stage Master Debugger)
- [✓] Update hardware simulation executable

Hardware simulation executable name: HelloWorld.exe

HDL Simulation
- [✓] Generate HDL testbench

What You Do

To set Simulation options, select Project > Options from the menu bar to bring up the Options dialog. Alternatively, you can bring up the dialog by clicking on the Options toolbar button. Select the Simulation tab, then set options as needed. The various options are discussed below.

When you are finished setting options, click the OK button to close the dialog and update the current project options. Click the Apply button to update the options, but not close the dialog. Project settings are not saved to the project file (.icProj) until requested, for example using the File > Save Project menu item.

Software Simulation Options

Launch Application Monitor - Choose this option to cause the CoMonitor Application Monitor to be launched prior to running the simulation executable.

Update software simulation executable - Choose this option to cause the software simulation executable to be built prior to launching.

Software simulation executable name - Use this field to specify the filesystem path to the software simulation executable. You may enter either a path relative to the project's directory or an absolute path. When creating a new project, this field is automatically filled in with a name derived from the name of the project.

Software simulation working directory - Use this optional field to specify a working directory in which to launch a simulation executable. If blank, the current project directory is assumed to be the working directory.

Software simulation command line arguments - Use this optional field to specify any command line arguments to your simulation executable.

Hardware Simulation (Stage Master Debugger) Options

Update hardware simulation executable - Choose this option to cause the hardware simulation executable to be built prior to launching.

Hardware simulation executable name - Use this field to specify the filesystem path to the hardware simulation executable. The path entered here is interpreted as a path relative to the Stage Master Debugger build directory. When creating a new project, this field is automatically filled in with a name derived from the name of the project.

HDL Simulation Options

Generate HDL testbench - Choose this option to generate test vectors during software simulation, and to generate a ModelSim HDL testbench during HDL generation. See the section of this document entitled "CoValidator Testbench Generator" for more details.

See Also

Simulation Features
Software Simulation Executable
Build Software Simulation Executable
Stage Master Debugger
CoValidator Testbench Generator

1.2.2 Screen Displays

impulse
accelerated technologies

The CoDeveloper Application Manager windows include:

Project Explorer Window
Transcript Window
Toolbar Buttons
Impulse C Design Assistant
Application Monitor

See Also

Design Management Features

Simulation Features
CoBuilder User Guide
Impulse C User Guide

1.2.2.1 Menu Bar

Description

The CoDeveloper main menu includes the following categories:

File
Edit
View
Project
Tools
Window
Help

See Also

Toolbar Buttons

1.2.2.1.1 File Menu

Description

The File menu options include:

New Project
Open Project...
Close Project
Save Project
Save Project As...
New File...
Open File...
Add File...
Close File
Save File
Save File As...
Exit

See Also

Edit Menu
View Menu
Window Menu
Design Management Features

1.2.2.1.1.1 New Project

Description

Creates a new blank or template project and closes the currently open project.

What You Do

To open a new project, select the File / New Project option from the menu bar or click on the New Project toolbar button.

If the current project has changed since it was last saved, a dialog box will appear asking if you want to save the current project. Click:

Yes to save the current project and start a new project.

No to start a new project without saving the current project.

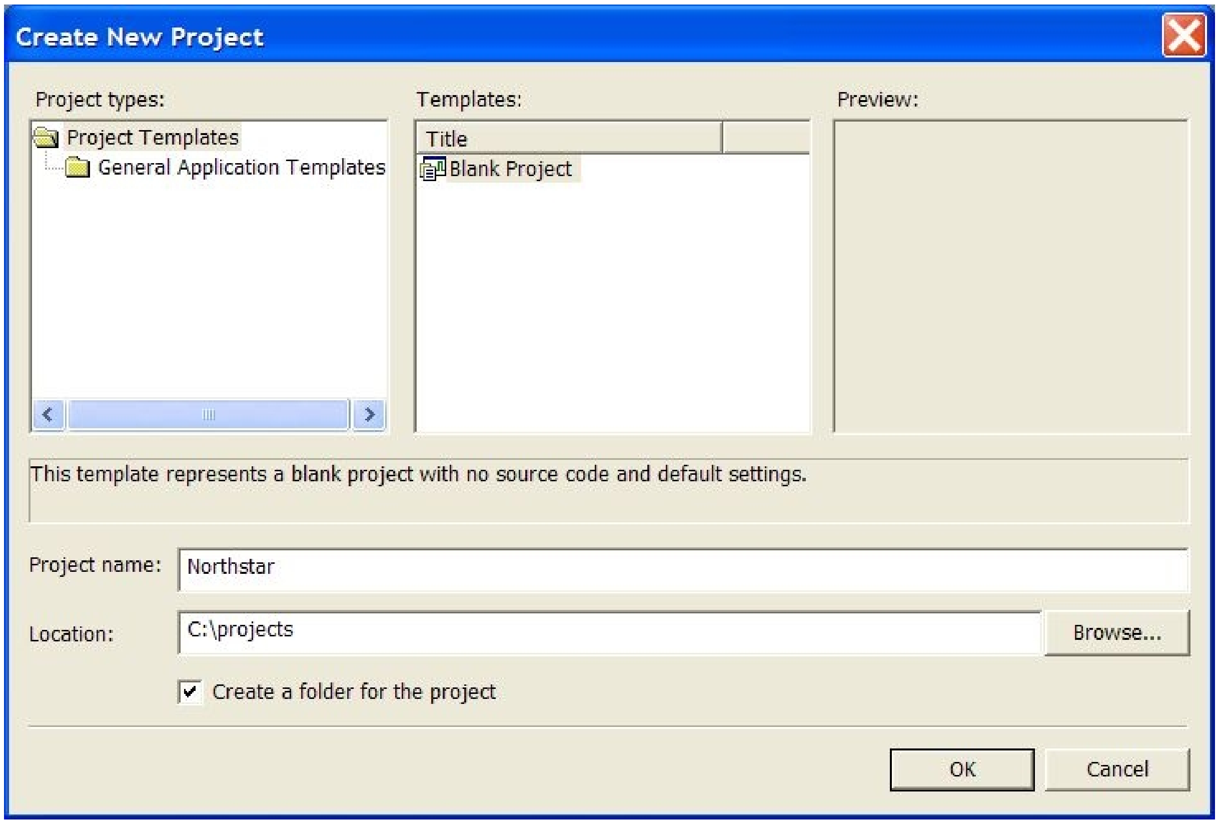
Cancel to cancel this command and return to the CoDeveloper.

The Create New Project dialog will appear. You can choose to create a blank project or one based on a template.

Creating a Blank Project

By default, the Create New Project dialog will select the "Blank Project" template when it appears.

1. Fill in the "Project name" field.
2. In the "Location" field, enter the absolute path (including drive letter) to the directory where the project will be stored. This directory must already exist.
3. (Optional) If you select the "Create a folder for this project" option, the project will be stored in a new directory with the same name as the "Project name", located under the "Location". For example, in the screenshot below, a new project named "Northstar" will be created in the directory "C:\projects\Northstar".
4. Click OK.
5. The Application Manager will now display your new blank project. You may now add new source files (File / New File) or existing source files (using the Project Explorer window or the Build Options dialog) to the project.



Project types: Project Templates, General Application Templates

Templates: Title — Blank Project

This template represents a blank project with no source code and default settings.

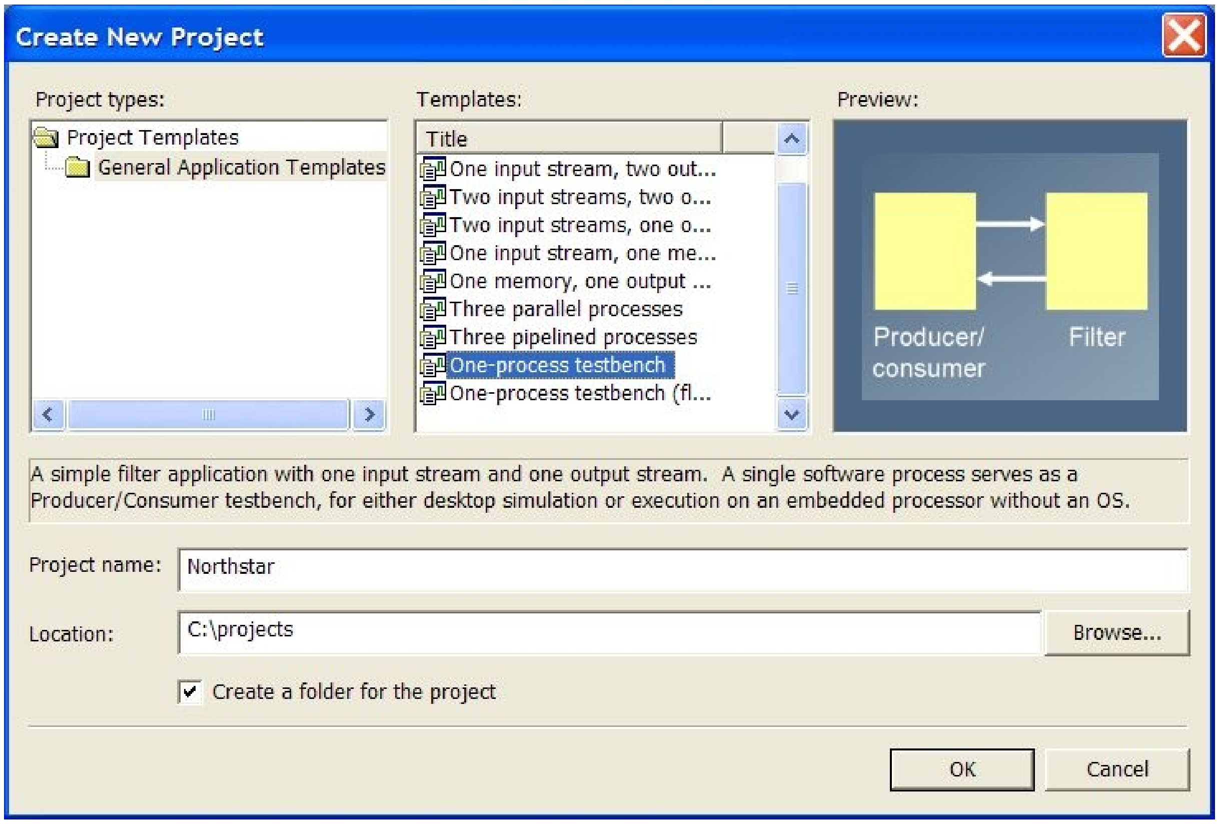
Project name: Northstar

Location: C:\projects

[✓] Create a folder for the project

Creating a Project from a Template

You can create a project that includes hardware and software source files implementing a complete Impulse C application by selecting one of the application templates in the Create New Project dialog. Such projects include all the necessary code for generating a simple FPGA design (processes, configuration function, main(), etc.). Several templates are available, representing a range of possible design choices.


Create New Project
Project types:
Project Templates
General Application Templates
Templates:
Title
One input stream, two out...
Two input streams, two o...
Two input streams, one o...
One input stream, one me...
One memory, one output ...
Three parallel processes
Three pipelined processes
One-process testbench
One-process testbench (fl...
Preview:
Producer/ consumer
Filter
Browse...
OK
Cancel

A simple filter application with one input stream and one output stream. A single software process serves as a Producer/Consumer testbench, for either desktop simulation or execution on an embedded processor without an OS.

Project name: Northstar

Location: C:\projects

[✓] Create a folder for the project

1. Select "General Application Templates" in the "Project types" window.
2. Select a template in the "Templates" window.
3. Fill in the "Project name" and "Location" fields as for a blank project, and optionally select the "Create a folder for the project" option.
4. Click OK.
5. The New Project Wizard will appear. Each template presents a different wizard, prompting you to fill in the template by providing names and other properties for processes and I/O channels (streams, signals, etc.). When you have made your selections, click Finish to close the wizard and generate the project.
6. The Application Manager will now display your new template project. You may now edit the source code, build and run desktop simulation, or generate hardware for your application.

See Also

New File
File / Add File...

1.2.2.1.1.2 Open Project...

Description

Open a CoDeveloper Impulse C project. Project files have a .icProj file extension.

What You Do

To open a project, select the File > Open Project... option from the menu bar or click on the Open Project toolbar button.

If a project is already open, a dialog box will then ask if you want to save the current project. Click on:

Yes to save the current project and open the project file dialog box. Then choose a project to open and click OK.

No to open the project file dialog box without saving the current project. Then, from the project file dialog box, choose a project to open and click OK.

Cancel to cancel this command and return to the current project.

See Also

File / Close Project
Project / Clean Project
Project Files Window
Open File...

1.2.2.1.1.3 Close Project

Description

Closes the currently open project.

What You Do

To close a project, select the File > Close option from the menu bar. The program will then ask if you want to save the current project.

Click on:

Yes to save the current project before closing it.

No to close the current project without saving it.

Cancel to cancel this command and return to the current project.

1.2.2.1.1.4 Save Project

Description

Saves the current project (and any associated open files). Note that the project is saved to a .icProj file.

What You Do

To save the current project, select the File -> Save Project option from the menu bar or click on the Save Project toolbar button. The program will then save the currently open project (and any associated open files) if the project is already named. If the project is not named, selecting this option brings up the Save Project As ... dialog box.

1.2.2.1.1.5 Save Project As...

Description

Saves the current project to a new file name. Note that the project is saved to a .icProj file.

What You Do

To save a project to a new or different file name, select the File > Save Project As... option from the menu bar. The program will then open a Save As ... dialog box. Enter a new name for the project and click OK.

1.2.2.1.1.6 New File...

Description

Creates a new Impulse C source file. When you select this option, the Impulse C Design Assistant will appear and allow you to create a new Impulse C application source file--either an blank file or one generated from a template.

1.2.2.1.1.7 Open File...

Description

Opens an existing file in a text editor window.

What You Do

To open a file in a text editor window, do one of the following:

Menu / Toolbar Button Method - Select the File > Open File... option in the menu bar or click on the Open File toolbar button. The Open dialog box will then appear. Select the file you wish to edit and click OK.

Double-click a Project File - Double-click on the filename in the Project Explorer window.

1.2.2.1.1.8 Add File...

Description

Adds a selected project-related file (.c. .h or other file) to the current project. Selected files are added to sections of your project (such as Source Files or Project Files) depending on the file name extension. Once added to your project, files may be moved in the project tree as desired.

What You Do

To add a module to your project, select the File -> Add File... option from the menu bar or click the Add File toolbar button. Select a file and click the Open button to exit the file selection dialog.

Note

You can rearrange the display order of files in the Project Explorer window, or move files to specific sections of the window, by right-clicking a file and using the Move Up or Move Down options in the context menu that appears.

Also note that files are, by default, added to your project with relative path information. The absolute path to the file, as understood by an open project, is displayed by right-clicking the file and selecting Properties from the context menu that appears.

See Also

New File (Design Assistant)

1.2.2.1.1.9 Close File

Description

Closes the currently active text editing window.

What You Do

To close the currently active text editing window, select the File > Close File option from the menu bar. If the contents of the file have changed, you will be prompted to save the file.

1.2.2.1.1.10 Save File

Description

Saves the contents of the currently active text editing window.

Tip

Saving a file will not automatically add the file to your project. To add a file to the currently open project, use the File / Add File option from the menu bar.

What You Do

To save the contents of the currently active text editing window, select the File > Save option from the menu bar.

1.2.2.1.1.11 Save File As...

Description

Saves the contents of the currently active text editing window to a new or different file name.

What You Do

To save a file to a new or different name, select the File > Save File As... option from the menu bar or use the shortcut keystroke. The program will then open a Save As ... dialog box. Enter a new name for the file and click OK.

1.2.2.1.1.12 Exit

Description

Closes the program.

What You Do

To exit the program, select the File > Exit option from the menu bar.

If the project has been modified, a dialog box will appear. Click on:

Yes save the current project and exit.

No to exit without saving.

Cancel to return to the program.

1.2.2.1.2 Edit Menu

Description

The Edit menu items perform standard Windows editing functions. A brief summary of each is given below.

Undo

This option undoes the most recent action in the text editing window. The shortcut key for this option is **Ctrl+Z** . This option can be used successively to undo up to 255 previous actions.

Redo

This option redoes the most recent text editing action that was undone using the **Undo** option above.

Cut

This option deletes a marked portion of text (block-marked using the mouse or the Shift and arrow keys) from a text editing window and copies it to the Clipboard. This action replaces the previous contents of the Clipboard. The shortcut key for this option is **Ctrl+X**.

Copy

This option copies a marked portion of text (block-marked using the mouse or the Shift and arrow keys) from a text editing window to the Clipboard. This action replaces the previous contents of the Clipboard. The shortcut key for this option is **Ctrl+C**.

Paste

This option copies the contents of the Clipboard into a text editing window at the location of the text cursor. The shortcut key for this option is **Ctrl+V**.

Clear

This option deletes a marked portion of text (block-marked using the mouse or the Shift and arrow keys) from a text editing window. The shortcut key for this option is the **Delete** key.

Select All

This option block-marks the entire contents of the currently active text editing window. The **Cut**, **Copy**, or **Clear** options can then be performed on the marked text. The shortcut key for this option is **Ctrl+A**.

Find

This option opens the **Search** dialog box which allows you to perform a text search in the currently selected text editing window. Enter the word or words you wish to search for in the **Search for** field, and then select from the following options:

Match case - finds only words that match your query exactly as to upper and lower case.

Match whole words only - ignores partial words and finds only whole words that match your query (e.g., a whole-words-only search for **init** would skip **initialize**).

Regular expressions - searches the file using regular expressions.

Once you have selected the desired options, click on the **Next Match** or **Previous Match** buttons to begin the search, forward or backward in the file, respectively, from the cursor's location. To repeat a Search, select the **FindNext Match** or **Previous Match** buttons. To cancel the search, click the **Close** button.

Replace

This option opens the **Search and replace** dialog box which allows you to perform a text search in the currently selected text editing window for a given text string and replace it with a second text string. To use the Replace feature, enter the word or words you wish to search for in the **Search for** field, enter the replacement word or words in the **Replace with** field, and then select from the following options:

Match case - finds only words that match your query exactly as to upper and lower case.

Match whole words only - ignores partial words and finds only whole words that match your query (e.g., a whole-words-only search for **init** would skip **initialize**).

Regular expressions - searches the file using regular expressions.

Replace in selection only - finds only strings within the selected text.

Once you have selected the desired options, click the **Replace** button to replace the first string found. Control the rest of the search and replace operation with the following buttons:

Replace - Performs the replacement on the highlighted match and moves to the next match.
Replace All - Perform the replacement on all of the remaining matches.
Next match - Find the next match and highlight it, but do not replace it. Pressing **Replace** will now continue the replace operation in the forward direction.
Previous match - Find the previous match (searching toward the beginning of the file) and highlight it, but do not replace it. Pressing **Replace** will now continue the replace operation in the backward direction.
Close - Ends the replacement operation.

Find Next

This option repeats the previous **Find** command. The shortcut key for this option is **F3** .

Find in Files

The option searches for a string in the current project or in any directory in the filesystem. See the topic Find in Files for details.

See Also

File Menu
View Menu
Project Menu
Window Menu
Design Management Features

1.2.2.1.2.1 Find in Files

Description

This option opens the Find in Files dialog box which allows you to perform a multi-file text search in the current project, or elsewhere in the filesystem.

Enter the text you wish to search for in the **Find what** field, then select from the following options:

Look in - Select the scope of the search. You can search in the "Current Project" (default) or click the "..." button to open a dialog box where you can choose any directory in the filesystem.

File types - Enter comma-separated file globs in this field to specify which types of file to search in. Filtering is often done by filename extension (e.g., "*.c,*.h", the default).

Match case - Finds only words that match your query exactly as to upper and lower case.

Whole words only - Ignore partial words and find only whole words that match your query (e.g., a "Whole words only" search for **in** would skip **inout**).

Look in subfolders - Recursively search subfolders in the "Look in" location.

Examining Results

Once you have selected the desired options, click on the **Search** button to begin the search. Any occurrences of the search string found in the selectedfiles will be displayed in the Find in Files transcript window. You may double-click any search result to open the relevant file and highlight the matched text.

See Also

Edit Menu
Design Management Features

1.2.2.1.3 View Menu

Description

The View menu normally includes the following items:

Start Page
Toolbars
Docking Windows
Status Bar
UI Theme
MDI Tabs
Line Numbers
Selection/Bookmark Numbers
Folding Margin
Add Bookmark
Delete Bookmark
Find Next Bookmark
Find Previous Bookmark
Comment
Uncomment

See Also

File Menu
Edit Menu
Project Menu
Window Menu
Design Management Features

1.2.2.1.3.1 Toolbars

Description

This option allows you to toggle display of the toolbars.

What You Do

The View > Toolbars option on the menu bar allows you to specify the toolbars (whether or not to display them) for standard file operations (saving files, etc.) and for Impulse functions such as building hardware and invoking the Application Monitor.

If toolbars are displayed and you would like to hide the toolbars to get more screen space, navigate to the View > Toolbars menu and select one of the toolbar names. The selected toolbar will disappear from your screen. Similarly, hidden toolbars can be displayed by selecting them again in the View > Toolbars menu.

Note that toolbars are dockable, so you can move them to alternate locations by dragging with the mouse.

See Also

Toolbar Buttons
Menu Bar Options

1.2.2.1.3.2 Docking Windows

Description

This option allows you to toggle display of the docking windows.

What You Do

The View > Docking Windows option on the menu bar allows you to specify whether or not to display various windows in the application.

Note that the Project Explorer and Transcript windows are dockable, so you can move them to alternate locations by dragging with the mouse. These windows also have auto-hide features that may be toggled by clicking on the thumbtack icon at the upper-right corner of the window.

See Also

Toolbar Buttons
Menu Bar Options

1.2.2.1.4 Project Menu

Description

The Project menu options include:

Clean Project
Build Software Simulation Executable
Launch Software Simulation Executable
Generate HDL
Build/Launch Hardware Simulation Executable
Export Generated Hardware (HDL)
Export Generated Software
Options

See Also

File Menu
Edit Menu
Window Menu
Design Management Features

1.2.2.1.4.1 Clean Project

Description

This option removes compiler-generated files from your project's working directory, including:

- All *.obj and other intermediate files from building desktop simulation
- All *.xic and other intermediate files from generating HDL
- All *.exe, *.vhd and other output files in the "Hardware build directory" and the "Software build directory"

What You Do

To remove all of the compiler-generated intermediate and output files from your project working directory, select the Project > Clean Project option from the menu bar.

Custom Makefile users can change the types and number of files deleted by a "clean" operation by editing their project makefiles accordingly. The name of the project makefile is specified in the Build Options dialog.

See Also

Build Options

1.2.2.1.5 Window Menu

Description

The Window menu items perform standard Windows management functions on the text editor windows. A brief summary of each menu option is given below.

After the list of menu options, each open file will be listed by name. Selecting a file name will switch the editor view to that file.

Cascade
This option cascades all open windows in the main application window.

Tile
This option tiles all open windows in the main application window.

Arrange Icons
This option is unused.

Close All Documents
Close all open documents. If any documents have been changed, a dialog box will prompt you to save each one.

Windows...
Displays a dialog box that will allow you to switch to ("Activate") or close any of the open editor windows.

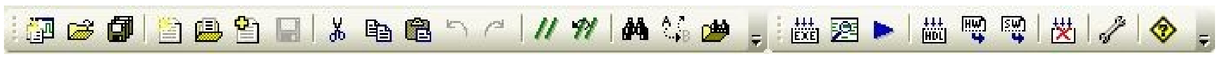
See Also

File Menu
Edit Menu
View Menu
Project Menu
Design Management Features

1.2.2.1.6 Toolbar Buttons

Description

The main application window includes a toolbar providing access to commonly-used functions. These buttons, which can be toggled on or off (see View / Toolbars) are summarized below. Note that as you move your cursor over a toolbar button, a tip appears that explains the function of that button.



From left to right, the toolbar buttons are:

New Project - same as File / New Project. The currently open project is closed before another project is created.

Open Existing Project - same as File / Open Project. The currently open project is closed before another project is opened.

Save Project - same as File / Save Project. All files in the project, including currently open source files, are saved.

Design Assistant - same as File / New File. Create a new source file (empty or from a template) and add it to the project.

Open - same as File / Open File. This button is useful for examining files not normally included in your project, such as generated HDL files.

Add File - same as File / Add File. This button allows you to add an existing file to your project.

Save - same as File / Save File. Save the file currently displayed in the editor window. This button is active when the contents of the current file have changed.

Cut - same as Edit / Cut. This button is active when a source file is being edited.

Copy - same as Edit / Copy. This button is active when a source file is being edited.

Paste - same as Edit / Paste. This button is active when a source file is being edited.

Undo - same as Edit / Undo. This button is active when a source file is being edited.

Redo - same as Edit / Redo. This button is active when a source file is being edited.

Comment - same as View / Comment. Comment selected C source code. This button is active when a source file is being edited.

Uncomment - same as View / Uncomment. Uncomment selected C source code. This button is active when a source file is being edited.

Find - same as Edit / Find. Find a string in the file displayed in the editor window. This button is active when a source file is being edited.

Replace - same as Edit / Replace. Replace a string in the file displayed in the editor window. This button is active when a source file is being edited.

Find in Files - same as Edit / Find in Files. Find a string in files in the current project or in a selected directory.

Build Software Simulation Executable - same as Project / Build Software Simulation Executable.

Application Monitor - same as Tools / Application Monitor. This button invokes the CoMonitor application.

Launch Simulation Executable - same as Project / Launch Software Simulation Executable. This button starts the software simulation executable specified in the Simulation Options dialog.

Generate HDL - same as Project / Generate HDL.

Export Generated Hardware (HDL) - same as Project / Export Generated Hardware (HDL).

Export Generated Software - same as Project / Export Generated Software.

Clean Project - same as Project / Clean Project.

Options - same as Project / Options. Opens the Options dialog.

Help - same as Help / CoDeveloper User's Guide. Opens this document.

See Also

Menu Bar Options

1.2.2.2 Transcript Window

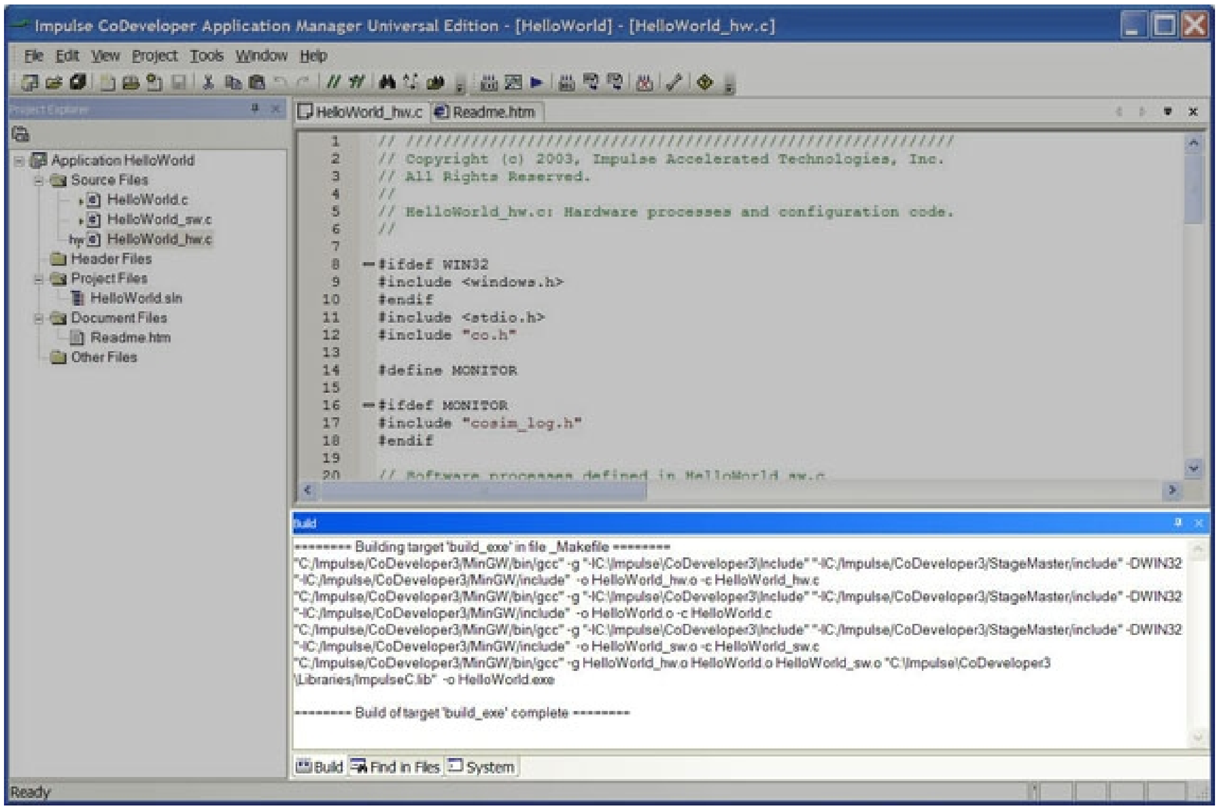
Description

The transcript window collects and displays messages generated by the build process (in the Build tab), search results (Find in Files tab), and system messages (System tab).

If you wish to save or print the text of a transcript, select the trancript text, then press Ctrl-C to copy the transcript text to the clipboard. To print the transcript text, you can use the clipboard to copy and paste the text to Windows Notepad or a similar editing application, then print the resulting document.

Right-click in the transcript window and a menu will appear, with options to cut/copy/paste, clear, and select all text in the window.

The contents of the transcript window are cleared if the transcript window is closed.



```
// ////////////////////////////////////////////////////////////////
// Copyright (c) 2003, Impulse Accelerated Technologies, Inc.
// All Rights Reserved.
//
// HelloWorld_hw.c: Hardware processes and configuration code.
//

#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include "co.h"

#define MONITOR

#ifdef MONITOR
#include "cosim_log.h"
#endif

// Software processes defined in HelloWorld_sw.c
```

```
======== Building target 'build_exe' in file _Makefile ========
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include" -DWIN32
"-IC:/Impulse/CoDeveloper3/MinGW/include"  -o HelloWorld_hw.o -c HelloWorld_hw.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include" -DWIN32
"-IC:/Impulse/CoDeveloper3/MinGW/include"  -o HelloWorld.o -c HelloWorld.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include" -DWIN32
"-IC:/Impulse/CoDeveloper3/MinGW/include"  -o HelloWorld_sw.o -c HelloWorld_sw.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g HelloWorld_hw.o HelloWorld.o HelloWorld_sw.o "C:\Impulse\CoDeveloper3
\Libraries/ImpulseC.lib"  -o HelloWorld.exe

======== Build of target 'build_exe' complete ========
```

1.2.2.3 System Options

Description

The System tab of the Options dialog allows you to view and edit several system-wide options used by Impulse C projects.

Tip

Options set in the System Options dialog are not saved with your project. To save the current project's options as system-wide defaults after exiting, check the **Save these options as defaults** box. These options will then be selected in any new project you create.

What You Do

To set System options, select Project / Options from the menu bar and select the System tab in the dialog that appears. Alternatively, you can bring up the dialog by clicking on the Options toolbar button and then clicking on the System tab. The various system options are discussed below.

When you are finished setting options, click on the Close button to close the dialog and update the options, or click the Apply button to update the options and leave the dialog open.

General Options

Save these options as defaults - If this box is checked, the current project's options will be saved to the Windows Registry when you exit the program. The saved options will then be selected in any new project you create.

Save all files before processing - If this box is checked, all open text files will be saved before any build operation is invoked.

Clear transcript before processing - If this box is checked, the Build Transcript Window will be cleared before any build operation is invoked.

Editing Options

Color syntax - If this box is checked, the text editor will color code source files being edited. Coloring rules are determined by the file extension (e.g., ".c" for C language files).

Convert tabs to spaces - If this box is checked, the text editor will convert uses of the Tab key to the specified number of spaces (**Tab size**) to advance to the next tab position. *Note: Do not select this option when editing Makefiles.*

Show indent guides - If this box is checked, the text editor will display vertical guidelines at each calculated tab position. These guidelines can be useful for source code formatting.

Font size - Use this option to specify the size of the font used in the text editing window.

Tab size - Use this option to specify the character width of tabs in the text editing window.

Note: The above Editing Options are not immediately effective for currently-opened editing windows. You must close, then re-open the editing windows to apply the new settings.

Make Program Options

Use external Make window - When this option is selected, each Make process will open in a separate console window, rather than display its output in the Transcript.

Use Impulse Make (GNU Make) - When this option is selected, build operations will use the Impulse Make program, based on GNU make.

Use external Make: - When this option is selected, a text field will appear. The Make executable whose name is entered in this field will be invoked for each build operation.

Other System Options

Impulse C Library Directory - This field specifies the path the program uses when searching for the Impulse C libraries (or other libraries) during build operations.

Impulse C Include Directory - This field specifies the directory the program uses when searching for the Impulse C header (include) files during build operations.

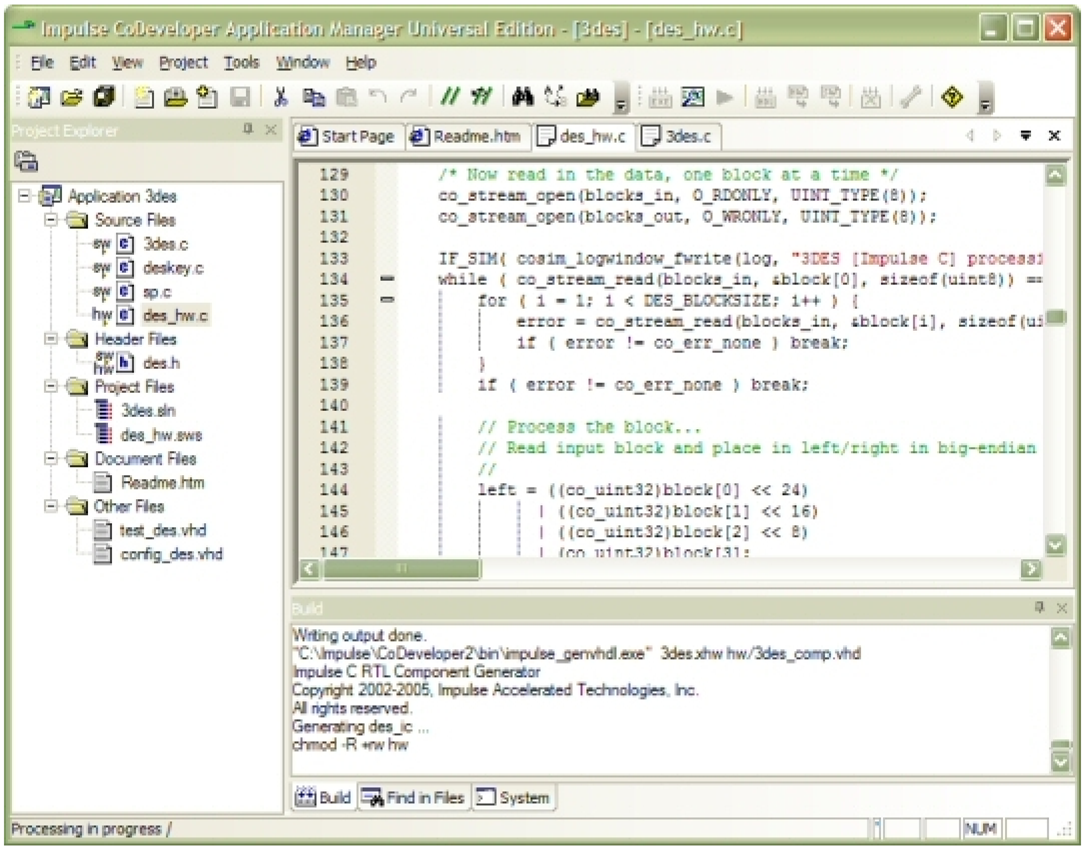
See Also

Build Options
Simulation Options
Generate Options

1.2.3 Design Management Features

Application Manager

The Impulse C Application Manager includes many useful features to help you create, modify and process your Impulse C projects.



```
    /* Now read in the data, one block at a time */
    co_stream_open(blocks_in, O_RDONLY, UINT_TYPE(8));
    co_stream_open(blocks_out, O_WRONLY, UINT_TYPE(8));

    IF_SIM( cosim_logwindow_fwrite(log, "3DES [Impulse C] processi
    while ( co_stream_read(blocks_in, &block[0], sizeof(uint8)) ==
        for ( i = 1; i < DES_BLOCKSIZE; i++ ) {
            error = co_stream_read(blocks_in, &block[i], sizeof(ui
            if ( error != co_err_none ) break;
        }
        if ( error != co_err_none ) break;

        // Process the block...
        // Read input block and place in left/right in big-endian
        //
        left = ((co_uint32)block[0] << 24)
                | ((co_uint32)block[1] << 16)
                | ((co_uint32)block[2] << 8)
                | (co_uint32)block[3];
```

```
Writing output done.
"C:\Impulse\CoDeveloper2\bin\impulse_genvhdl.exe"  3des.xhw hw/3des_comp.vhd
Impulse C RTL Component Generator
Copyright 2002-2005, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating des_ic ...
chmod -R +rw hw
```

Project Explorer Window

The Project Explorer Window helps you track and manage the source files, header files, external project files, build files and document files that make up your complete application. This is particularly useful for projects involving multiple Impulse C source files and/or multiple build targets. This window also allows you to quickly identify which source files represent hardware (for HDL generation) and which represent software (for an embedded microprocessor or for desktop simulation).

New Project Wizard

The New Project Wizard helps you get started quickly with a new Impulse C project by generating a complete set of project files and source code from one of several customizable design templates.

Design Assistant

The Impulse C Design Assistant helps you create new Impulse C applications by first asking you a series of questions about your application requirements. The Design Assistant then generates template Impulse C source files based on those requirements and embeds useful comments helping you complete and verify your application.

Make integration

Built-in Make features help you streamline the processing of your design for simulation and for programmable hardware compilation. For example, when you are ready to simulate your design, simply click a single button and Makefiles run behind the scenes to produce an simulation executable. There is no need to compile each source file in the design separately or keep track of which source files have been updated since they were last compiled. Advanced users can even customize the Make process for their own needs.

External projects

Keep track of external project files in one place. The Application Manager allows you to include references to external project files (such as Visual Studio Solution files, HDL simulation projects or FPGA synthesis projects) right in the Application Manager source file tree. By double-clicking on a project file you can invoke the needed external applications quickly and easily. Document files (such as Microsoft Word files or Powerpoint files) may also be included in this way.

See Also

Build Features
Simulation Features
Screen Displays
Make Integration

1.2.3.1 Project Explorer Window

Description

The Project Explorer window (normally appearing in the left part of the Application Manager screen) allows you to:

1. Organize the files that make up your project.
2. Specify hardware, software and simulation source files.
3. Control the processing of your project.
4. Invoke a source file editor to modify the elements of your design.

Context menus allow you to operate on the elements of the project show in the Project Explorer window. Right-click any element (file, category, application) to show the context menu for that element.

Organizing Project Files

The Project Explorer window consists of:

- File categories
- File entries

To add a source file to your project, right-click a category or the application (the root of the tree). Select the "Add File..." menu item and select the file in the dialog that appears. Depending on the extension of file added, the file may be moved to a specific category. For example, .c files will be added to the Source Files category by default.

Project files are grouped in the following categories:

Source Files – This category generally includes the C files associated with your application. C files may represent one or more Impulse C processes that will be compiled to hardware or software, as well as test-related files needed for running simulations. Double-clicking on any source file will open it in a source code editor window.

Header Files – This category includes .H (header) files needed when your application is compiled. Typically you will include only those header files that are specific to the project, and will not include header files that are global or system-wide (such as the Impulse C "co.h" header file, which is located in the CoDeveloper installation tree and located through the use of compiler-specific include paths). Double-clicking on any header file will open it in a source code editor window.

Project Files – This category includes external project files that may be associated with your Impulse C project. Types of project files may include:

- Visual Studio solution/project files
- CodeWarrior project files
- Dev-C++ project files
- FPGA vendor tool project files
- Other project files needed to process or document your complete application

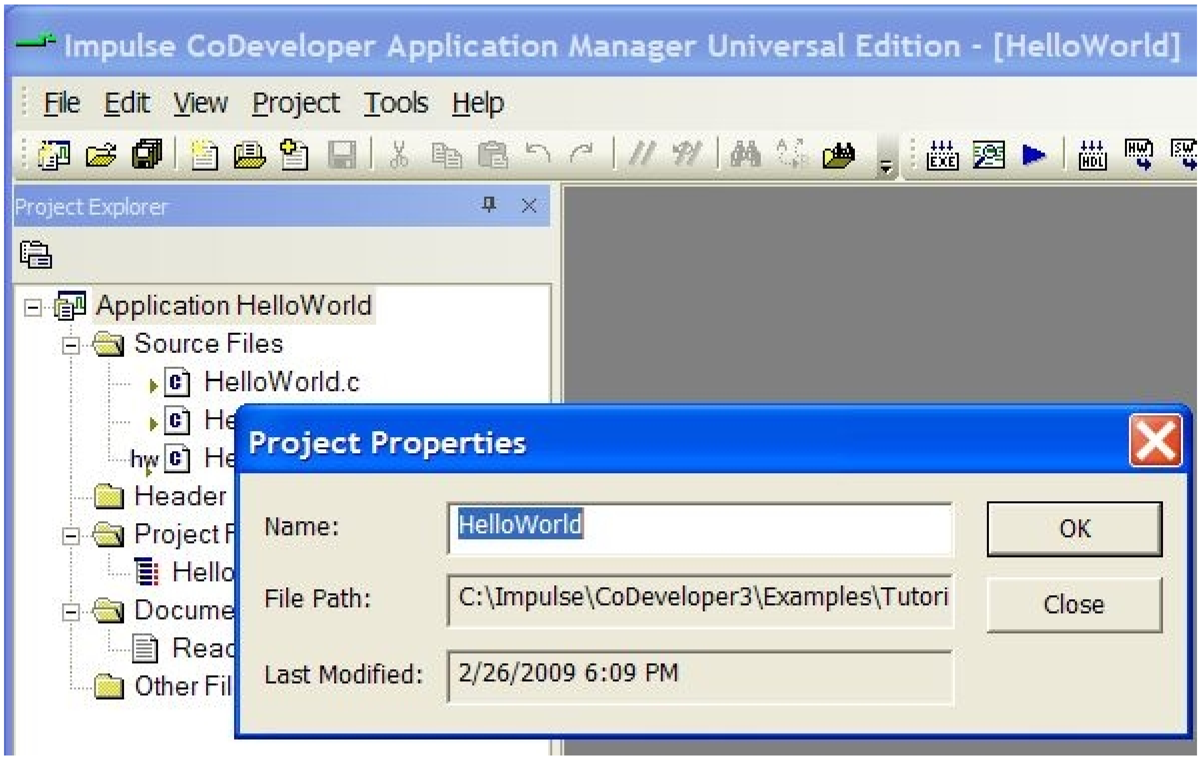
Double clicking on any project file will launch the application that is currently associated with the type of project file you have selected, as indicated by its filename extension.

Document Files – This category includes documents related to your project. Documents might include .DOC, .TXT, .PDF or other files of other formats. Double-clicking on any document file will launch the application that is currently associated with the type of document file you have selected, as indicated by its filename extension.

Other Files – The category may be used to collect and manage files that do not fit into one of the above categories. You may wish to include references to generated output files here, or include data files related to application testing. Depending on the type of file referenced in this section, double clicking may result in the file being opened for viewing and editing.

Changing the Application Name

To change the application/project name, right-click the Application entry to invoke the context menu. Select the Properties option and change the Name field, then click OK:



Project Properties

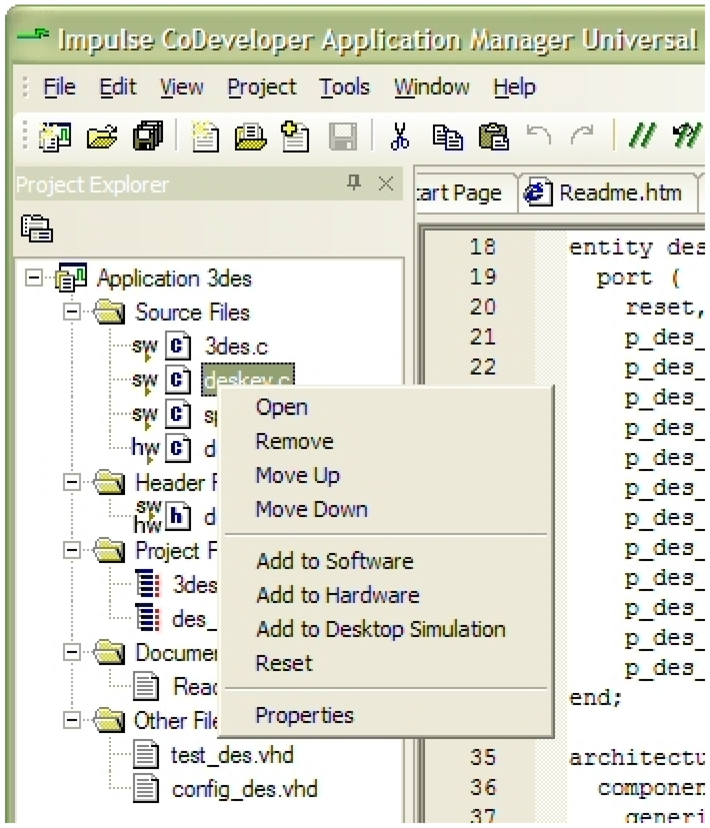
Name: HelloWorld

File Path: C:\Impulse\CoDeveloper3\Examples\Tutori

Last Modified: 2/26/2009 6:09 PM

Changing File Properties

Right-click a file name in the Project Explorer window to display the context menu:



This menu allows you to open a file, remove it from the project, or move it up or down the list. You can also assign source files to one (or more) of three groups including Software, Hardware, and Desktop Simulation, or reset the file's assignments. The Properties item allows you to view file-specific properties including the filesystem path to the file.

Understanding Software and Hardware Files

The Project Explorer window allows you to assign each of your project source files (typically .C and .H files) into one or more groups, depending on the desired compilation target. There are three groups, as described below:

Hardware Source Files - Files identified as belonging to the "hw" group (as indicated by the small icon to the left of the file name) are to be processed by the CoBuilder compiler to generate hardware, in the form of VHDL or Verilog output files. These files must include (at a minimum) your application's configuration subroutine, a co_initialize function, and one or more C subroutines representing hardware processes.

Software Source Files - Files identified as belonging to the "sw" group are to be included for exporting to the target cross-compiler environment for compilation on an embedded or host processor. These files may be uniquely written for the cross-compiler environment (for example, making use of processor-specific library calls) or may be written in such a way that they can be compiled for desktop simulation as well as cross-compilation (perhaps using #ifdef).

Desktop Simulation Source Files - Files identified as belonging to the desktop simulation group are files that are to included when compiling the application for simulation using a standard C compiler (gcc). These files are identified by the small, triangular "play" icon.

See Also

New Project
File / Open Project...
File / Add File...
Project / Clean Project
File / Close Project
Design Assistant

1.2.3.2 File / New Project

Description

Creates a new blank or template project and closes the currently open project.

What You Do

To open a new project, select the File / New Project option from the menu bar or click on the New Project toolbar button.

If the current project has changed since it was last saved, a dialog box will appear asking if you want to save the current project. Click:

Yes to save the current project and start a new project.

No to start a new project without saving the current project.

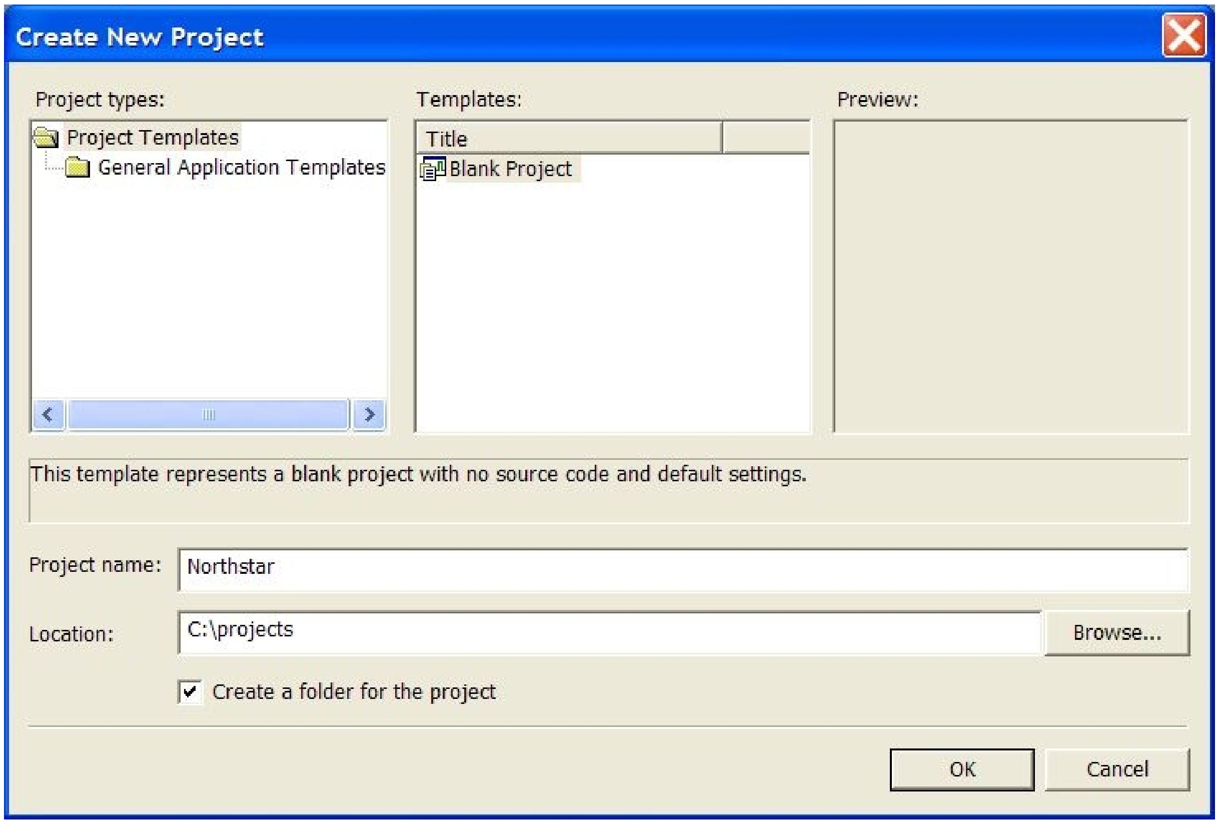
Cancel to cancel this command and return to the CoDeveloper.

The Create New Project dialog will appear. You can choose to create a blank project or one based on a template.

Creating a Blank Project

By default, the Create New Project dialog will select the "Blank Project" template when it appears.

1. Fill in the "Project name" field.
2. In the "Location" field, enter the absolute path (including drive letter) to the directory where the project will be stored. This directory must already exist.
3. (Optional) If you select the "Create a folder for this project" option, the project will be stored in a new directory with the same name as the "Project name", located under the "Location". For example, in the screenshot below, a new project named "Northstar" will be created in the directory "C:\projects\Northstar".
4. Click OK.
5. The Application Manager will now display your new blank project. You may now add new source files (File / New File) or existing source files (using the Project Explorer window or the Build Options dialog) to the project.



This template represents a blank project with no source code and default settings.

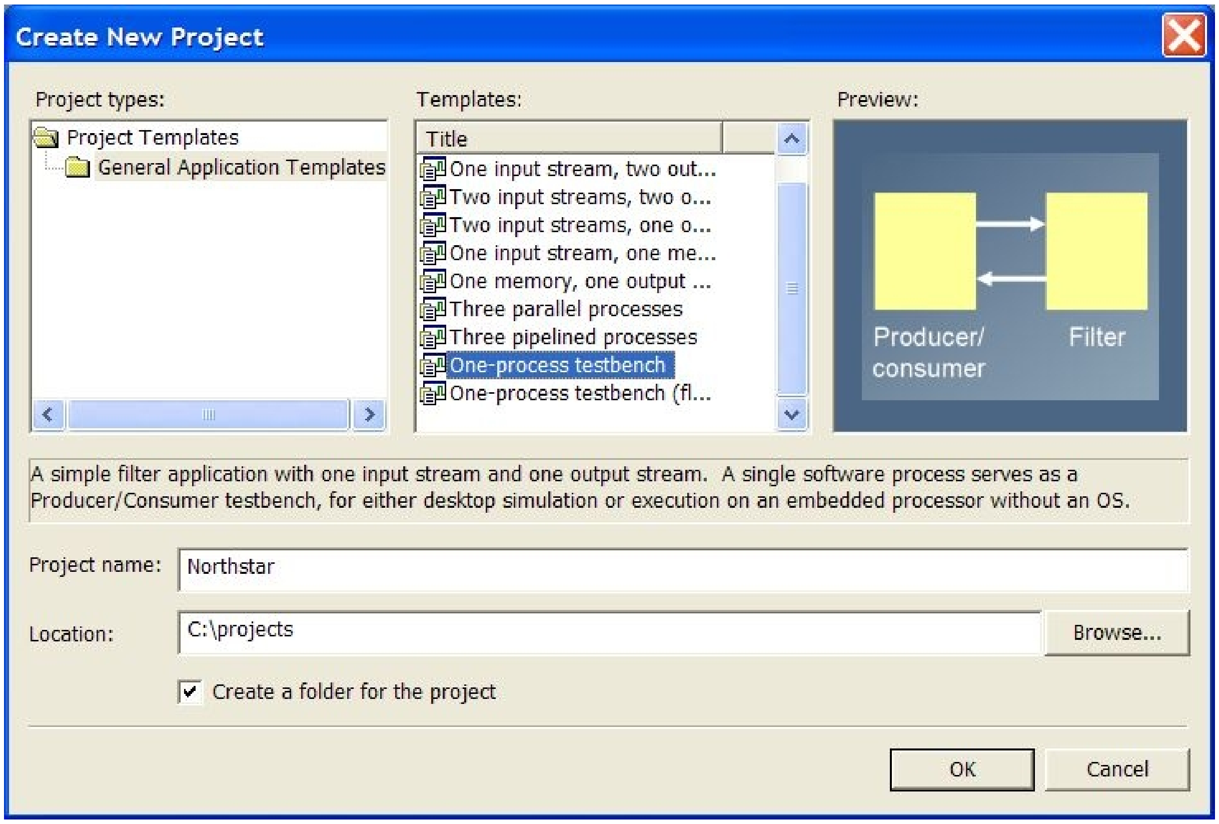
Project name: Northstar

Location: C:\projects

[✓] Create a folder for the project

Creating a Project from a Template

You can create a project that includes hardware and software source files implementing a complete Impulse C application by selecting one of the application templates in the Create New Project dialog. Such projects include all the necessary code for generating a simple FPGA design (processes, configuration function, main(), etc.). Several templates are available, representing a range of possible design choices.


Create New Project
Project types:
Project Templates
General Application Templates
Templates:
Title
One input stream, two out...
Two input streams, two o...
Two input streams, one o...
One input stream, one me...
One memory, one output ...
Three parallel processes
Three pipelined processes
One-process testbench
One-process testbench (fl...
Preview:
Producer/ consumer
Filter
OK
Cancel
Browse...

A simple filter application with one input stream and one output stream. A single software process serves as a Producer/Consumer testbench, for either desktop simulation or execution on an embedded processor without an OS.

Project name: Northstar

Location: C:\projects

[✓] Create a folder for the project

1. Select "General Application Templates" in the "Project types" window.
2. Select a template in the "Templates" window.
3. Fill in the "Project name" and "Location" fields as for a blank project, and optionally select the "Create a folder for the project" option.
4. Click OK.
5. The New Project Wizard will appear. Each template presents a different wizard, prompting you to fill in the template by providing names and other properties for processes and I/O channels (streams, signals, etc.). When you have made your selections, click Finish to close the wizard and generate the project.
6. The Application Manager will now display your new template project. You may now edit the source code, build and run desktop simulation, or generate hardware for your application.

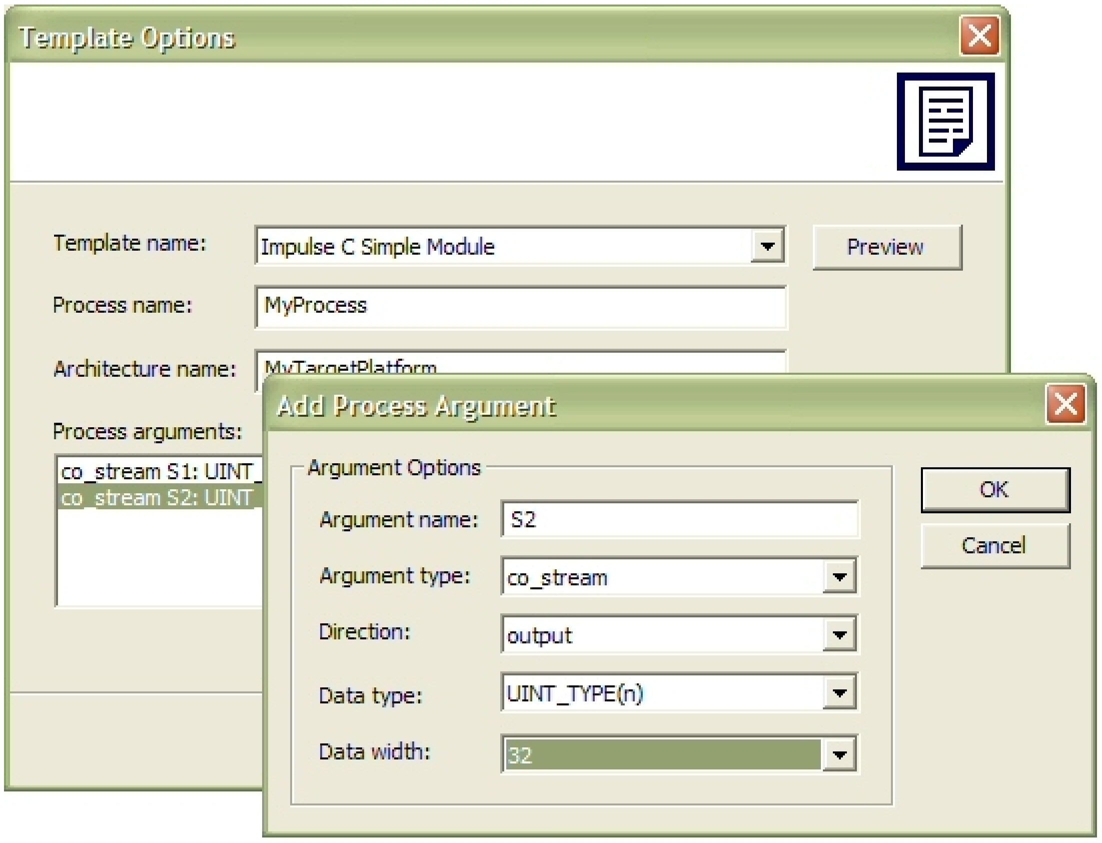
See Also

New File
File / Add File...

1.2.3.3 Design Assistant

Description

The Impulse C Design Assistant helps you create new Impulse C application source files by first asking you to describe your application's general structure and the required interfaces (inputs and outputs) to its component hardware and software processes. The Design Assistant then generates prototype source files based on those requirements, using template files that are provided as part of your Impulse C installation.



Template name: Impulse C Simple Module
Process name: MyProcess
Architecture name: MyTargetPlatform
Process arguments:
co_stream S1: UINT_
co_stream S2: UINT

Argument name: S2
Argument type: co_stream
Direction: output
Data type: UINT_TYPE(n)
Data width: 32

What You Do

To access the Design Assistant, first select the File / New File option from the menu bar or click on the Design Assistant toolbar button. Then choose the type of appliction source file (or files) you wish to generate and click on the Next button. A Design Assistant window will appear. Once the window is displayed, set options as needed to define the interface to your design. The various sections of the Design Assistant window are discussed below.

Template Name - Select the name of the template you wish to use for generating your application source file. (Template source files are stored in the Templates subdirectory of your Impulse C installation, and may be modified or extended as needed.) If you wish to preview a selected template prior to using it, click the Preview button that appears next to the Template Name selection box. Note that the type of template you select will depend on the nature of your application or algorithm requirements. The template names are somewhat descriptive, but you may wish to use the Details button to view more detailed comments about each template's specific features, and to view a sample file output.

Process Name - Enter the name to be used for the generated run process (the run process is the process that will include the actual C code for the algorithm you are developing). For example, if you enter the entity name "my_process" in the Process Name field of the Design Assistant, then "my_process_run" will be used as the run process name of the generated module. "My_process" will also be used as the basis for sample producer and consumer test processes generated by the Assistant.

Architecture Name - When using the Design Assistant, the application architecture generated will have a name that is based on the name you specify here. The name you select should refer in some way to the type of platform you are targeting with your appliction. For example, the name "my_architecture_virtexFPGA" might be appropriate for naming an FPGA-based platform. If this field is left blank, the architecture name "my_architecture" will be used.

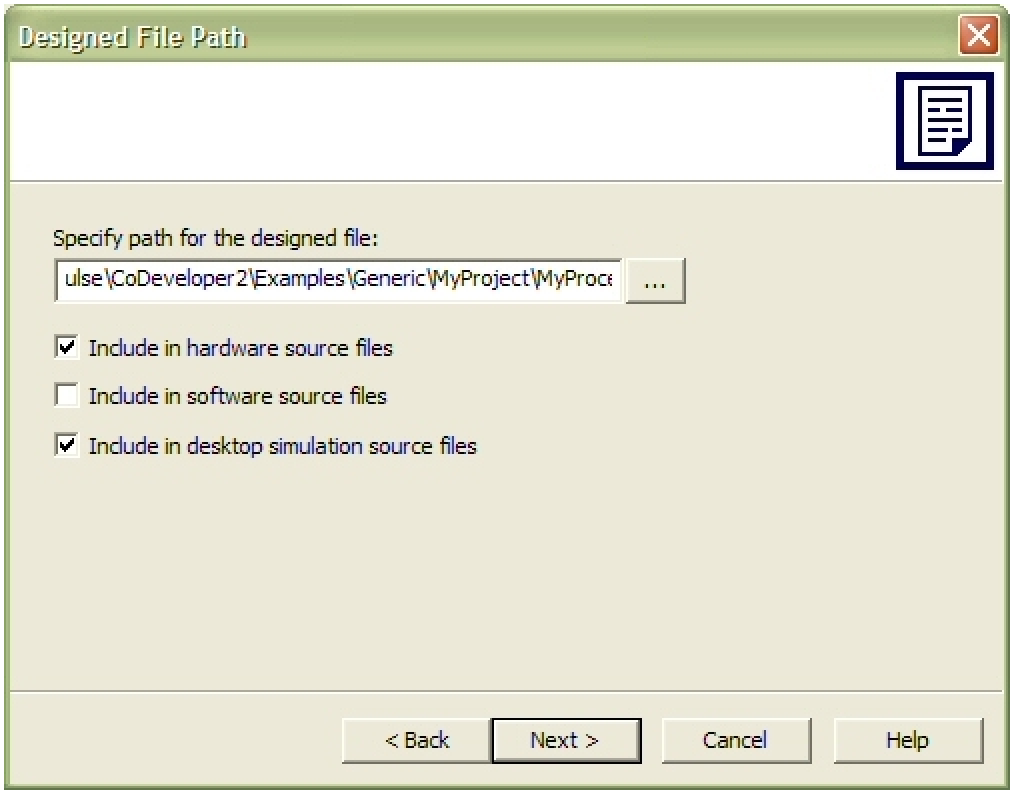
Port Name, Direction, Type and Width - Use these fields to enter the names, types and directions of input and output ports (streams, signals, memories or parameters) of your new process. You must enter ports one at a time and use the **Add** button to add each port to the port declarations. After adding a port declaration you may (if needed) edit the resulting declaration to correct any entry errors.

Add - Click this button to add the specified port to the Port Declarations field.

Edit - Click this button to edit a selected port.

Process Arguments - This field is constructed automatically when you click the **Add** button for each port of your entity.

Next - When you are satisfied with your input and output port declarations, click this button to generate the new application source file. The Design Assistant will prompt you for the file name of your new file, and prompt you to specify which of the three groups (Hardware, Software and/or Desktop Simulation) the new file belongs:


Designed File Path

Specify path for the designed file:
ulse\CoDeveloper2\Examples\Generic\MyProject\MyProce

- [x] Include in hardware source files
- [ ] Include in software source files
- [x] Include in desktop simulation source files

Cancel - Click this button if you do not want to generate the new source files.

After the template process run function and other files have been been generated, you can modify them as needed to add functionality, change source code formatting as desired and prepare for processing.

See Also

File / Add File...

1.2.3.4 Make Integration

Overview

The CoDeveloper Application Manager makes use of the well-known Make program to control the processing of your application files for the purpose of compilation, desktop simulation, and generation of target files such as HDL files for FPGA synthesis.

By default, CoDeveloper will generate the Makefile (named "_Makefile") that defines its various build processes, but you also have the option of writing your own Makefile to customize the CoDeveloper build processes. See Build Options to learn how to configure a project to use a custom Makefile.

Make Targets

CoDeveloper Application Manager invokes Make with a different "target" for each different compilation command in the Project menu:

Make Target	CoDeveloper Project Menu Command
clean	Clean Project
build_exe	Build Software Simulation Executable
build	Generate HDL
build_testbench	Generate HDL (when "Generate HDL testbench" option is selected)
build_smd	Build/Launch Hardware Simulation Executable
export_hardware	Export Generated Hardware (HDL)
export_software	Export Generated Software

The Make target being built by CoDeveloper is indicated in the Build Transcript Window when the compilation command runs.

Make Macros

When processing is invoked, CoDeveloper passes certain variables (or "macros" in Make-speak) to the Make process via an automatically-generated _Makefile.defs file. The values of these variables are derived from the Project Options and include the following:

Variable Name	Description	Source
ARCH_ID	Name of the architecture file (XML format) that defines the selected Platform Support Package (PSP).	Generate Options
CO_INC	The Impulse C include file directory (typically C:\Impulse\CoDeveloper3\Include\).	System Options
LIB_DIR	The Impulse C library file directory (typically C:\Impulse\CoDeveloper3\Libraries\).	System Options
DEBUG	This flag will be set if the desktop simulation executable is to be built with debugging symbols.	Build Options
P_FLAGS_OPT	Optional command line flags passed to the impulse_porky program.	Generate Options
A_FLAGS_OPT	Optional command line flags passed to the impulse_arch program.	Generate Options
SM_FLAGS_OPT	Optional command-line flags passed to the impulse_sm program.	Generate Options
GENRTL_EXE	Name of the CoBuilder executable used to generate RTL ("impulse_genvhdl.exe" or "impulse_genvlog.exe").	Derived from the choice of PSP
HDL_FILE_EXT	Filename extension for generated RTL files ("vhd" or "v").	Derived from the choice of PSP
HW_SRC	A list of C source files, delimited by spaces, defining the files to be processed by CoBuilder for the purpose of RTL generation.	Build Options
HW_INC	A list of C header (include) files required for building the above source files.	Build Options
SW_SRC	A list of C source files, delimited by spaces, defining the files that are to be copied to the software build directory for later compilation on the target embedded microprocessor.	Build Options
SW_INC	A list of C header (include) files required for building the above source files.	Build Options
EXE_SRC	A list of C source files, delimited by spaces, defining the files to be compiled to create a desktop simulation executable file.	Build Options
EXE_INC	A list of C header (include) files required for building the above source files.	Build Options
EXE	The name of the executable file to be created as a result of compiling the application for desktop simulation (Simulation Executable).	Simulate Options
IMPULSE_LIB	Name of the Impulse C library to be linked when building a desktop simulation (default: "ImpulseC"). If HDL testbench generation is enabled, the name of the tracing library will be given.	Simulate Options
SMD_EXE	The name of the executable file to be created as a result of building the application for hardware simulation using Stage Master Debugger.	Simulate Options
HW_TARGET_DIR	The directory to use when generating hardware files.	Generate Options
SW_TARGET_DIR	The directory to use when generating software files.	Generate Options

If you are writing your own Makefiles for use with CoDeveloper, you can access these variables by including the following statement at the beginning of your Makefile:

```
include _Makefile.defs
```

See Also

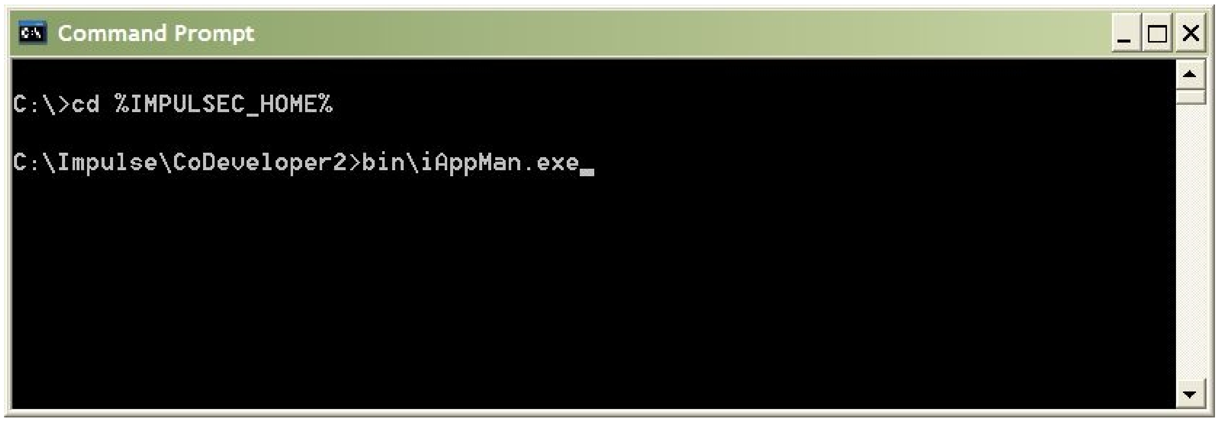
Build Options
Simulate Options
Generate Options
System Options
CoBuilder Command Line Tools

1.2.4 Command Line Operation

Overview

The CoDeveloper Application Manager can be started from a command-line shell, for example from the Windows Command Prompt. When started from the command line, CoDeveloper is run either in interactive mode, or in command mode.

To start CoDeveloper from the command line in interactive mode, enter the path to the "iAppMan.exe" program at the prompt. (If "%IMPULSEC_HOME%\bin" is in the PATH, simply run "iAppMan.exe".)



```
C:\>cd %IMPULSEC_HOME%

C:\Impulse\CoDeveloper2>bin\iAppMan.exe_
```

The iAppMan.exe command returns immediately, whether run in interactive mode or in command mode.

Interactive Mode

When CoDeveloper is started in interactive mode, the window remains open and no commands are run unless they are chosen using the menus, buttons, or hotkeys. This is how CoDeveloper opens when its icon is double-clicked or it is started from the Windows Start menu.

Interactive Mode Options

Only one option is available when launching CoDeveloper in interactive mode: the path to an Impulse C project file (.icProj). For example:

```
        C:\projects>C:\Impulse\CoDeveloper2\bin\iAppMan.exe MyProject.icProj
```

Command Mode

When run in command mode, CoDeveloper's window appears briefly and CoDeveloper opens a project, then executes a command from the Project menu (such as Generate HDL) and exits. Command mode is useful for automating CoDeveloper commands, for example using a scripting language.

Command Mode Options

Each of the following options causes a build task to be executed when CoDeveloper opens. Only one of these options may be given, followed by the path to the Impulse C project file you wish to run the task on:

Option	**Corresponding "Project" Menu Item**
-CLEAN	Clean Project
-BUILD_EXE	Build Software Simulation Executable
-LAUNCH_EXE	Launch Software Simulation Executable
-BUILD_SMD	Build/Launch Hardware Simulation Executable *(does not launch SMD)*
-LAUNCH_SMD	Build/Launch Hardware Simulation Executable
-BUILD_RTL	Generate HDL
-EXPORT_HW	Export Generated Hardware (HDL)
-EXPORT_SW	Export Generated Software

The output that normally would appear in the Build window is instead redirected to a log file. By default, this file is named CoDeveloper.log and appears in the project directory.

The iAppMan.exe command returns immediately after launching ("forking") the process that runs the chosen project build task. The build task may still be running in the background after the command line is ready for another command. When the task is finished, an empty file named CoDeveloper.log.done will be created in the project directory. By detecting the presence of this file, a script calling CoDeveloper can determine when the task has finished.

Additional Export Command Options

The -EXPORT_HW and -EXPORT_SW command-mode options can be combined with an additional option that overrides the export directory specified by the Impulse C project file. Each option takes as its argument a path to an export directory (<*EXPORT_PATH*>). This path must be quoted if it contains spaces.

Option	**Used With**
-export_hw_dir <*EXPORT_PATH*>	-EXPORT_HW
-export_sw_dir <*EXPORT_PATH*>	-EXPORT_SW

These options must precede the path to the project file on the command line. For example:

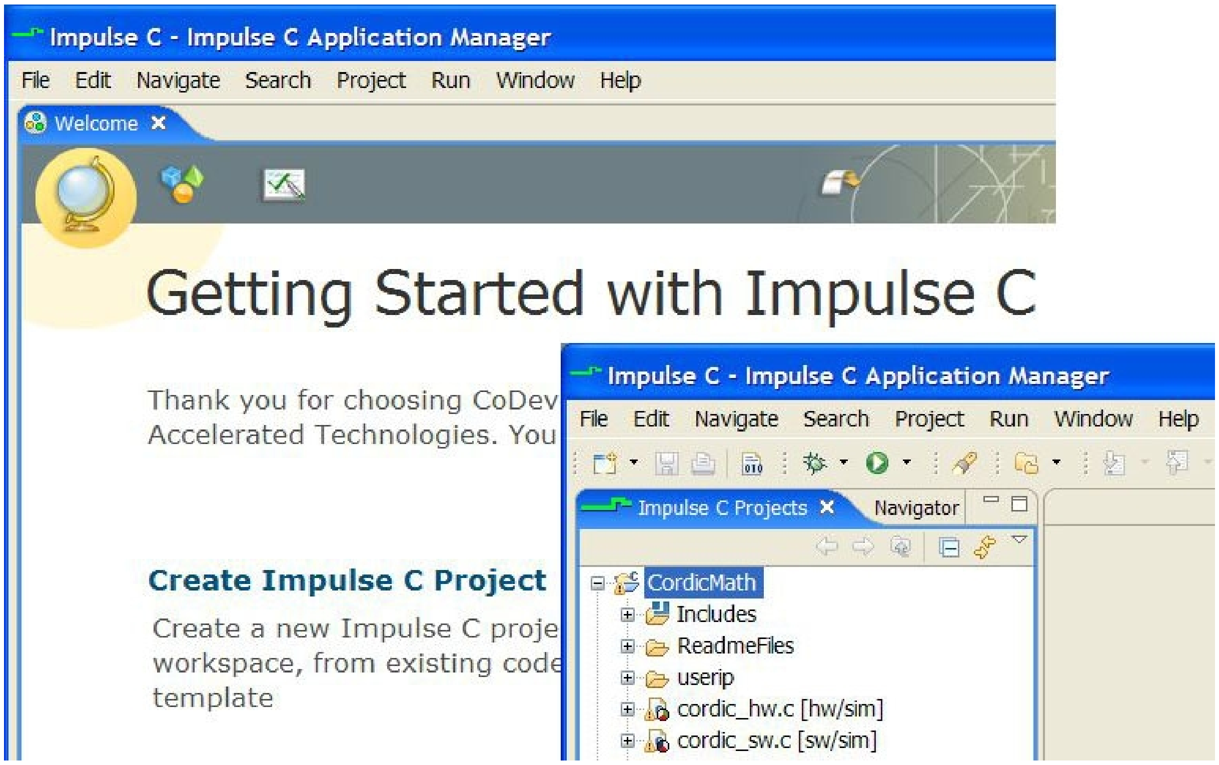
```
      C:\projects>C:\Impulse\CoDeveloper2\bin\iAppMan.exe -EXPORT_HW -export_hw_dir
SynthesisProjectDir MyProject.icProj
```

See Also

Command Line Tools

1.3 CoDeveloper Eclipse IDE

CoDeveloper Eclipse IDE


Impulse C - Impulse C Application Manager
File Edit Navigate Search Project Run Window Help
Welcome
Getting Started with Impulse C
Thank you for choosing CoDev
Accelerated Technologies. You
Create Impulse C Project
Create a new Impulse C proje
workspace, from existing code
template
Impulse C - Impulse C Application Manager
File Edit Navigate Search Project Run Window Help
Impulse C Projects
Navigator
CordicMath
Includes
ReadmeFiles
userip
cordic_hw.c [hw/sim]
cordic_sw.c [sw/sim]

Eclipse IDE Start Page and Workbench

The CoDeveloper Eclipse IDE is an Integrated Development Environment based on the widely used and robust Eclipse framework. First released with CoDeveloper Version 3, the Eclipse IDE includes an integrated source-level debugger for desktop simulation, a powerful editor, and support for generating hardware for all Impulse C Platform Support Packages (PSPs).

Getting Started

Tutorials
Before starting with CoDeveloper, we strongly suggest that you make use of the supplied tutorials, either by following the steps presented or by casual reading. Doing so will make your initial work with CoDeveloper and Impulse C much easier, and will provide you with useful insights into parallel programming and mixed hardware/software development methods.

Several Quick Start Tutorials are provided in this Help document.

Documentation
To read Impulse C documentation in the CoDeveloper Eclipse IDE, select the Help > Help Contents menu to open the help system. Expand the Impulse CoDeveloper Guide tree to view all Impulse help documentation.

Platform-Specific Documentation
Tutorials and documentation for specific FPGA boards or platforms is provided by Platform Support Package (PSP) Help files. Open the help system using the Help > Help Contents menu item in the CoDeveloper Eclipse IDE. Expand the Impulse CoDeveloper Guide item in the tree view and select the appropriate help section for your chosen PSP.

Sample Projects
A number of sample Impulse C projects are included in the CoDeveloper installation. Once you have a sample project open, you can review the Impulse C source code, build and run a desktop simulation, or invoke the Impulse C hardware compiler to generate HDL and hardware/software interface files.

To open a sample project:

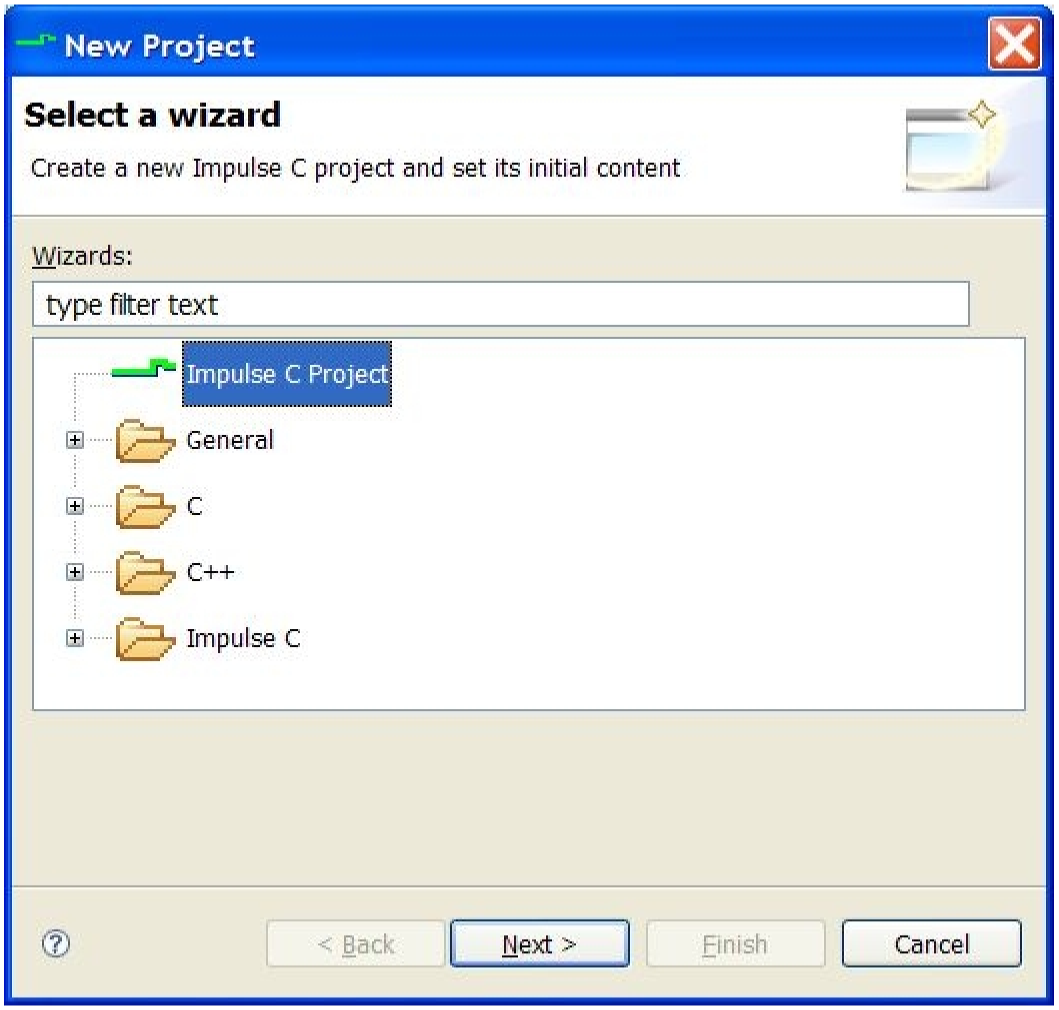
1. Start the CoDeveloper Eclipse IDE by selecting Start > Impulse Accelerated Technologies > CoDeveloper 3.70 > CoDeveloper Application Manager (Eclipse IDE).
2. Select the Help > Welcome menu item to view the Welcome page. Click the "Browse Impulse C Samples" link to view a list of sample projects included with the CoDeveloper installation.
3. Click a project link to copy the project's files and settings into your workspace.

1.3.1 Design Management

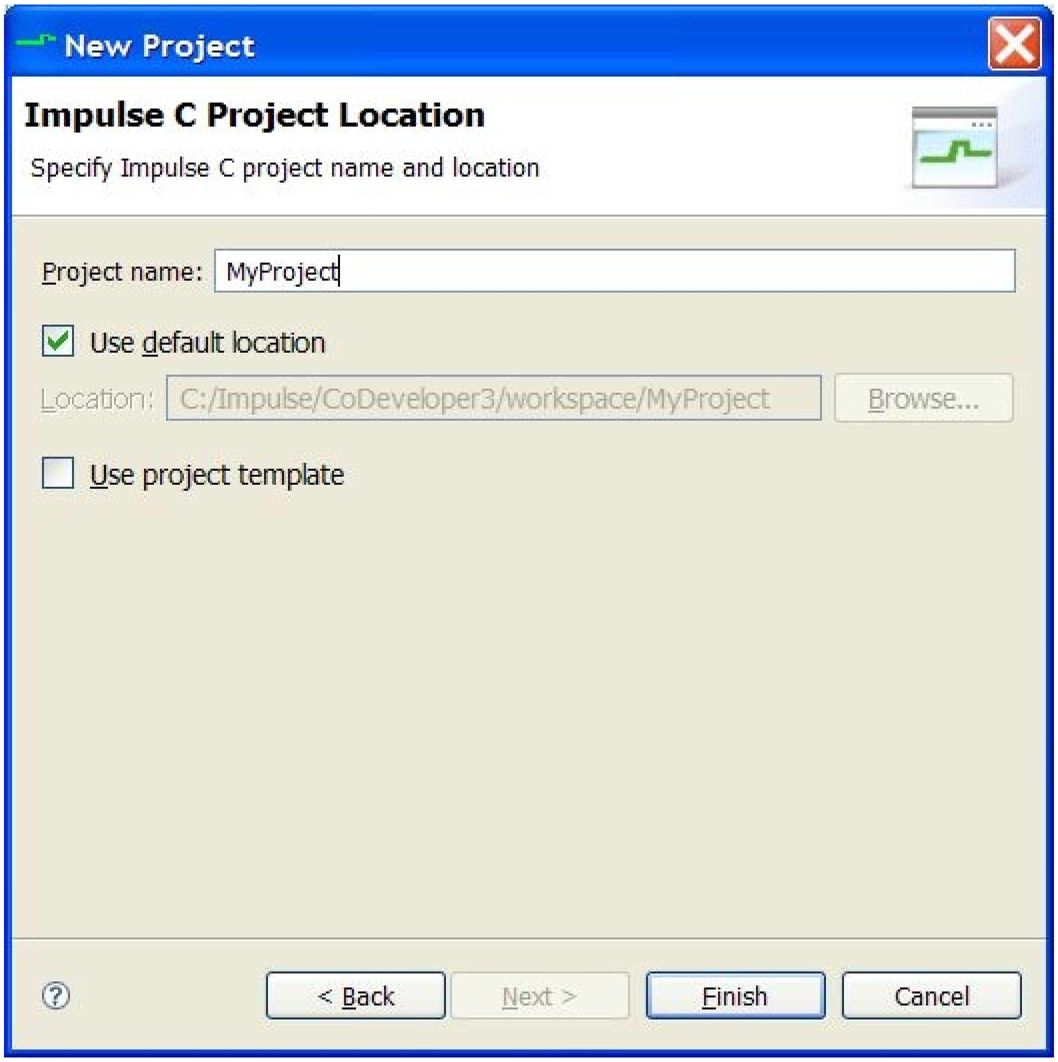
Projects are managed in the Eclipse IDE in a group, called the *workspace*. Many projects can be open simultaneously in a single workspace, and the user can switch between workspaces to work with different sets of projects.

Creating a New Project

To create a new empty Impulse C project, select the File > New Project menu item. A dialog will appear. Select "Impulse C Project", as shown:



Click Next. Enter your project's name. A new directory will be created in the current workspace by default, named after the project:


New Project
Impulse C Project Location
Specify Impulse C project name and location

Project name: MyProject

☑ Use default location

Location: C:/Impulse/CoDeveloper3/workspace/MyProject

☐ Use project template

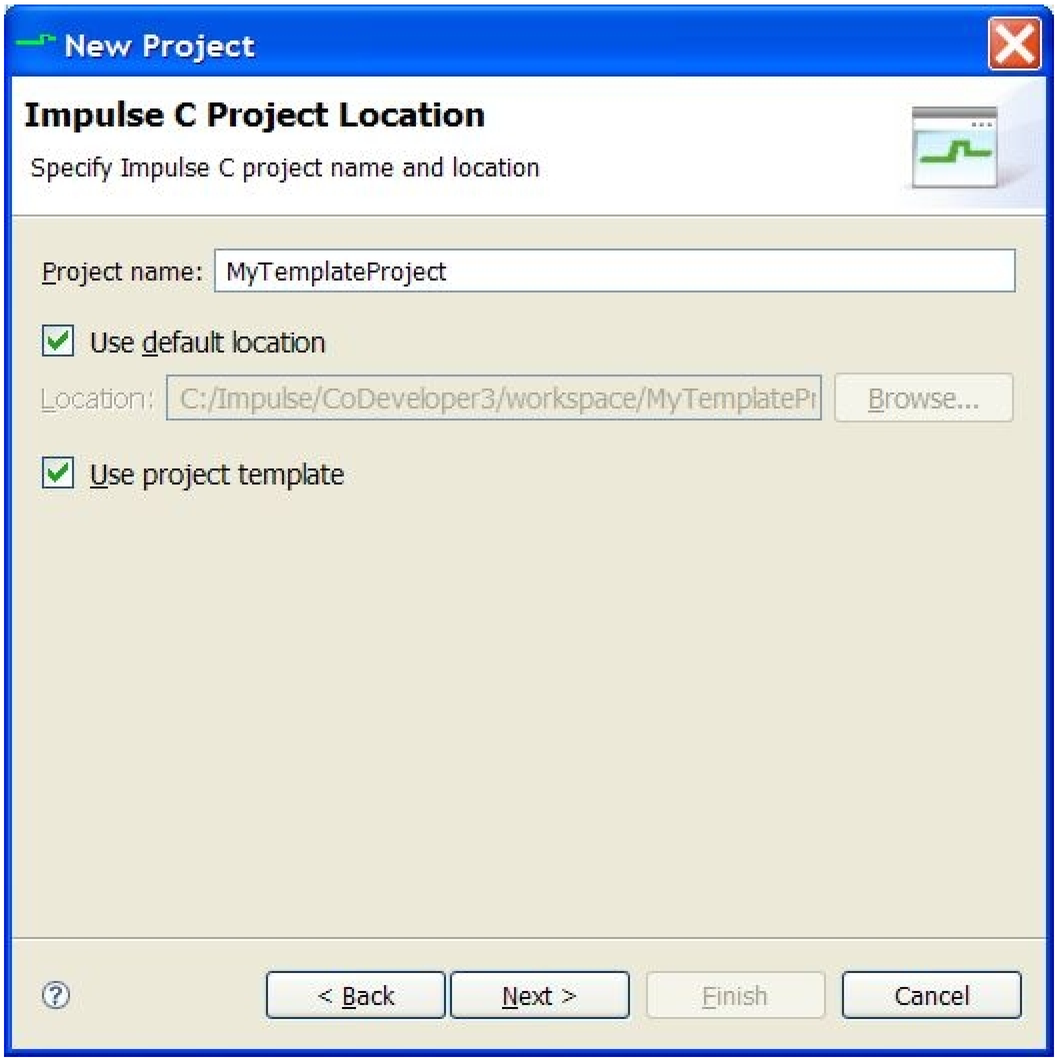
Browse...
< Back
Next >
Finish
Cancel

Click Finish to create the empty project and return to the workbench.

Creating a New Project from a Template

To create a new Impulse C project based on a design template, select the File > New Project menu item, choose "Impulse C Project" in the dialog that appears, and click Next.

Enter your project's name. A new directory will be created in the current workspace by default, named after the project. Select the "Use project template" checkbox and click Next.



Impulse C Project Location

Specify Impulse C project name and location

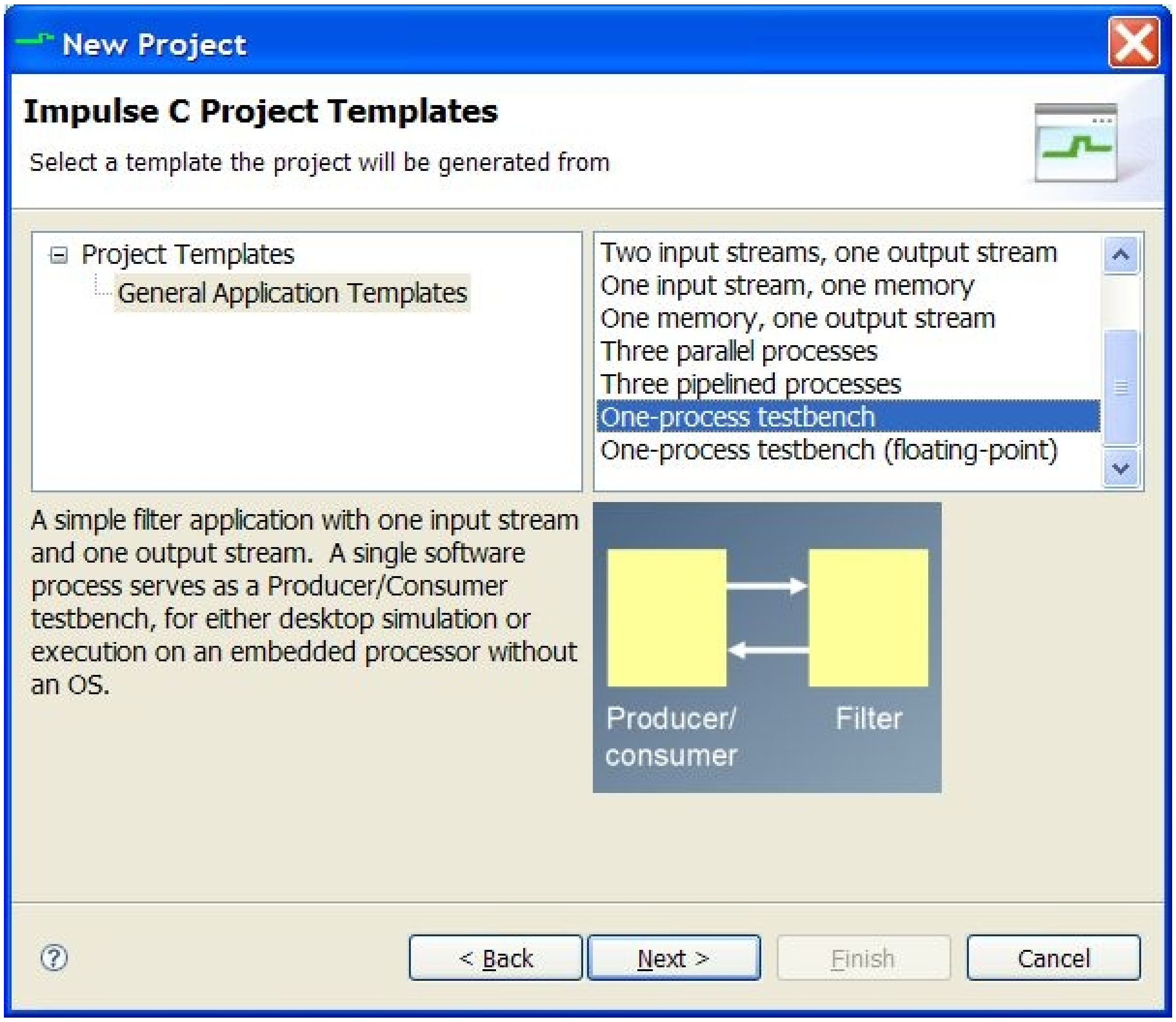
Project name: MyTemplateProject

[✓] Use default location

Location: C:/Impulse/CoDeveloper3/workspace/MyTemplatePr

[✓] Use project template

Select the template to base the new project on using the tree and list boxes:



New Project

Impulse C Project Templates

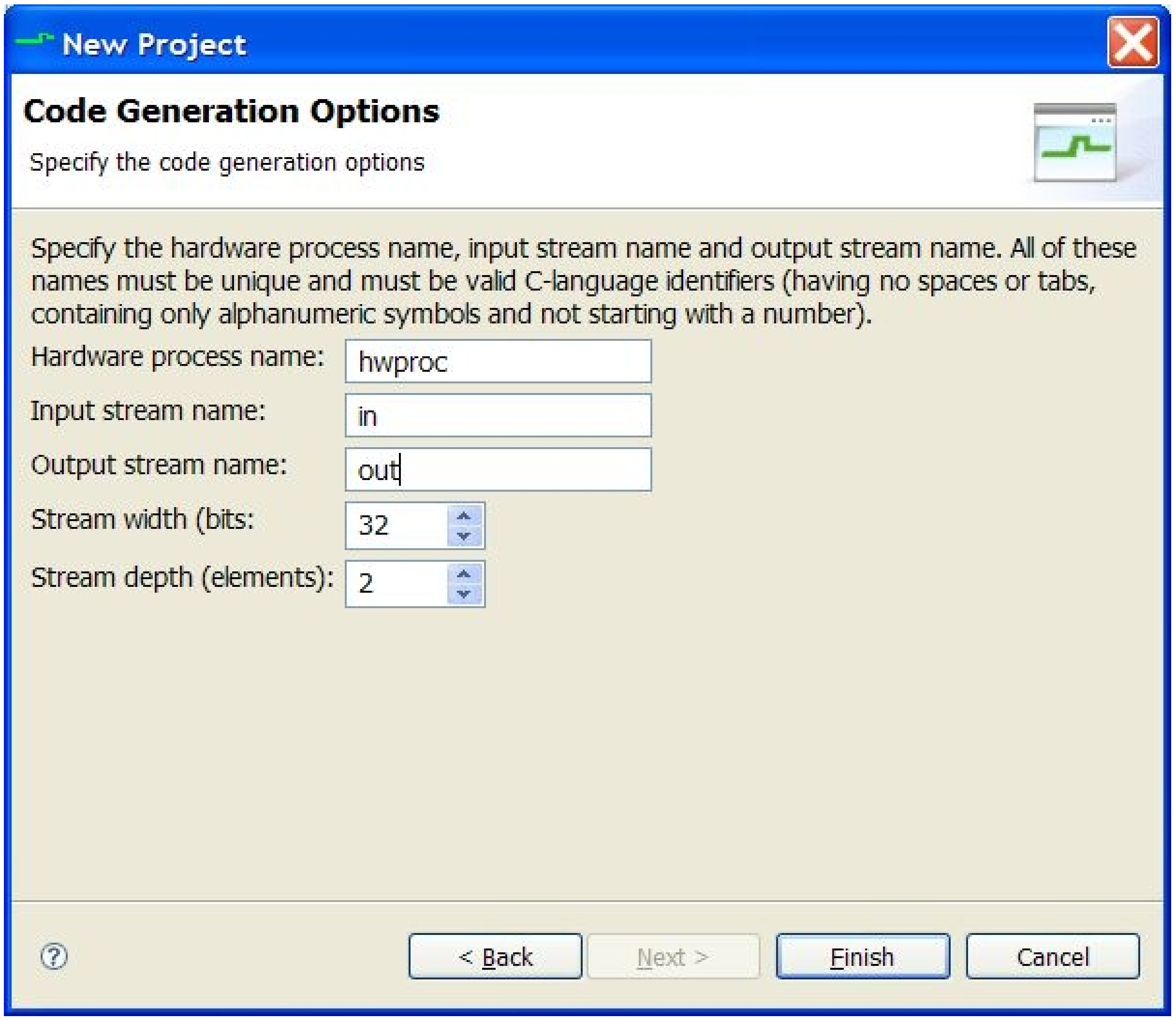
Select a template the project will be generated from

Project Templates
General Application Templates

Two input streams, one output stream
One input stream, one memory
One memory, one output stream
Three parallel processes
Three pipelined processes
One-process testbench
One-process testbench (floating-point)

A simple filter application with one input stream and one output stream. A single software process serves as a Producer/Consumer testbench, for either desktop simulation or execution on an embedded processor without an OS.

Click Next to proceed to the template-specific wizard pages, where you will enter parameters that fill in the chosen template to create a working Impulse C project, complete with source code and project properties:


New Project

Code Generation Options

Specify the code generation options

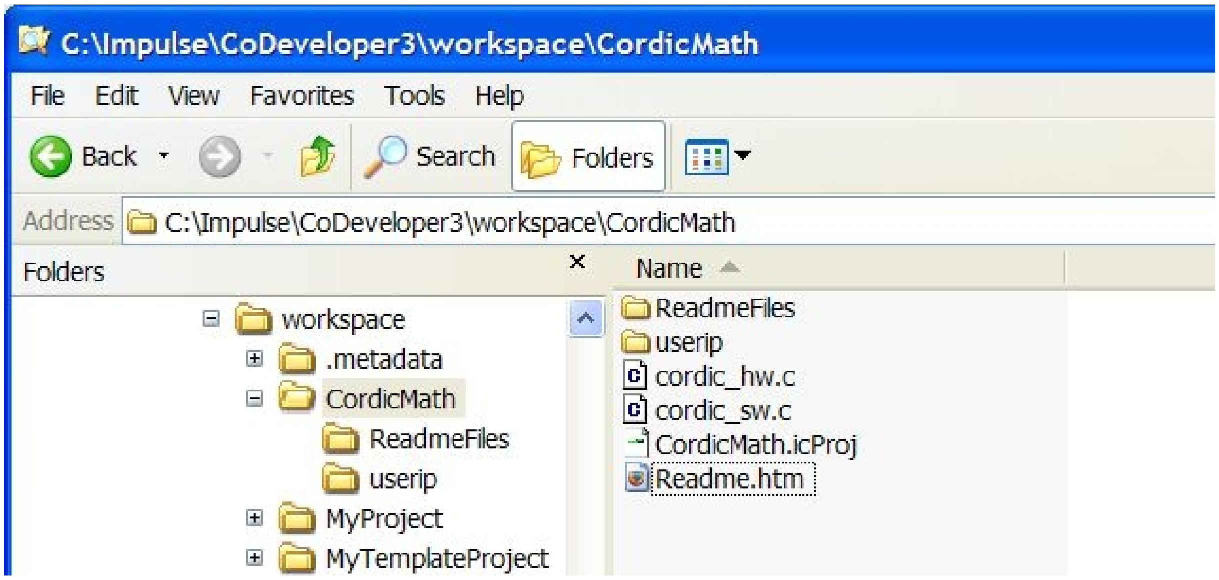
Specify the hardware process name, input stream name and output stream name. All of these names must be unique and must be valid C-language identifiers (having no spaces or tabs, containing only alphanumeric symbols and not starting with a number).

Hardware process name:	hwproc
Input stream name:	in
Output stream name:	out
Stream width (bits:	32
Stream depth (elements):	2

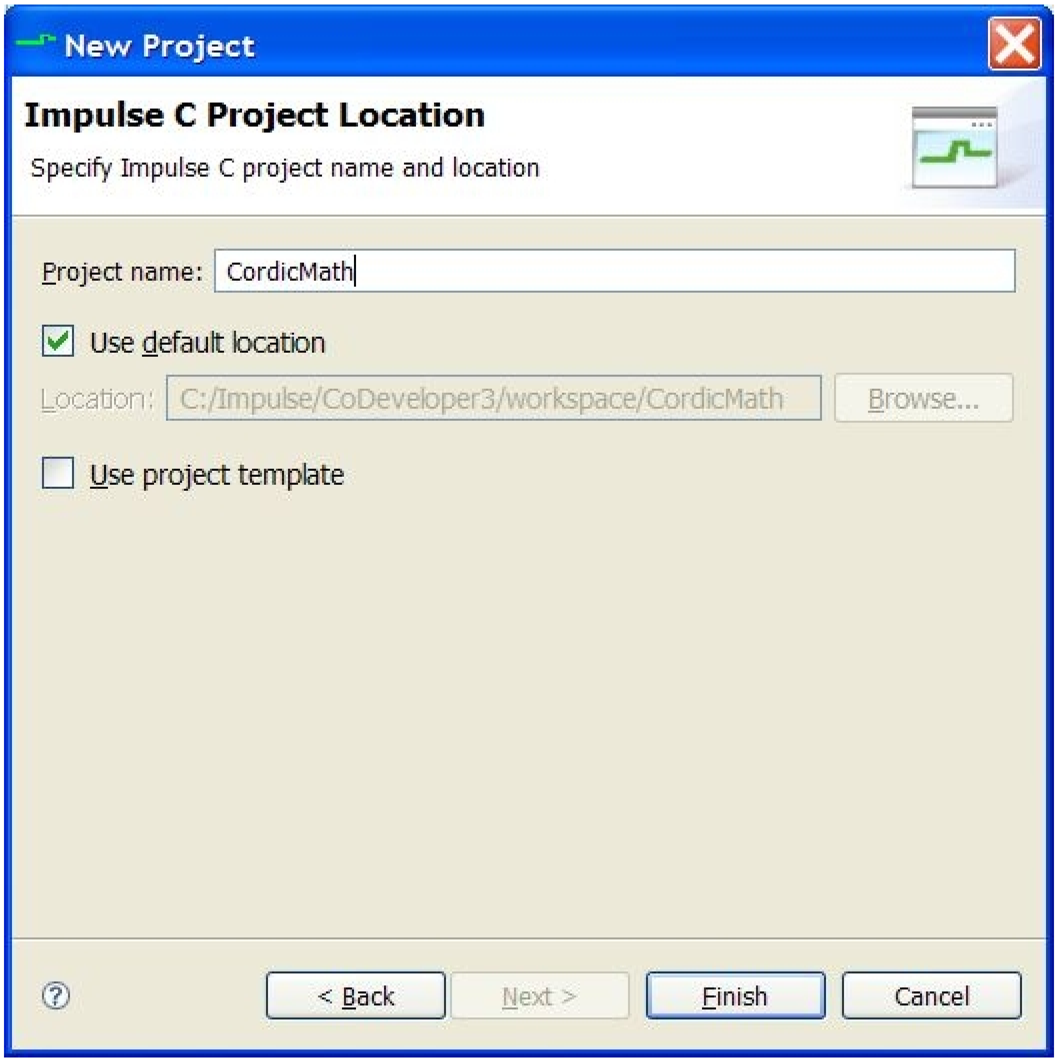
Fill in all fields (there may be more than one page to complete) and click Finish to create the new project and return to the workbench.

Importing a Legacy Project (*.icProj)

To import an existing project that uses the legacy project file format (*.icProj), first copy the project files into a directory under the workspace. Make sure to copy all source code, as well as the *.icProj file. For example, the CordicMath example, copied into the workspace for import, would appear in the filesystem as shown:

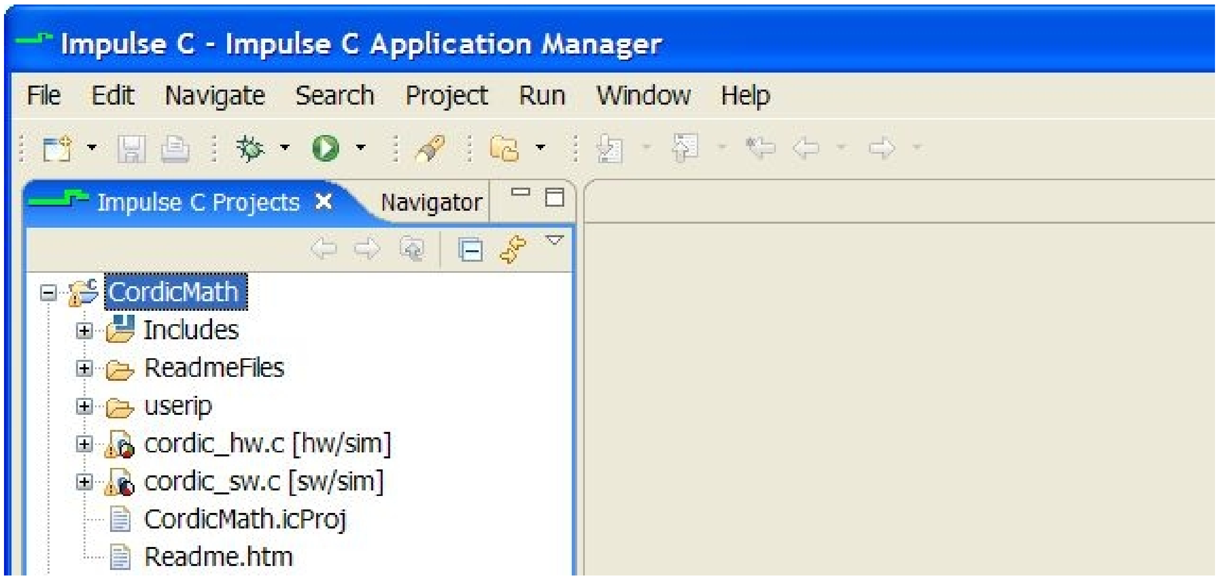


After copying the project files, select the File > New Project menu item. Select "Impulse C Project", click Next, and enter the name of the copied directory as the name of the project. These names must match for the legacy project to be properly imported. Also note that the *.icProj file must be located directly under the project directory, not in a subdirectory of the project.


New Project
Impulse C Project Location
Specify Impulse C project name and location
Project name: CordicMath
[✓] Use default location
Location: C:/Impulse/CoDeveloper3/workspace/CordicMath

[ ] Use project template
< Back
Next >
Finish
Cancel

Click Finish to import the project properties and source code, and return to the workbench. The source files should be shown under the project in the Impulse C Projects view:



Note: From this point on, changes made to the project's properties in the Eclipse IDE will diverge from the *.icProj file, since the Eclipse IDE stores project properties in a different set of files (see CoDeveloper Application Manager).

Switching Workspaces

To switch between workspaces, select the File > Switch Workspace menu item. You may specify a new or existing workspace location in the dialog that appears. Your current workspace will be saved and the Eclipse IDE will restart.

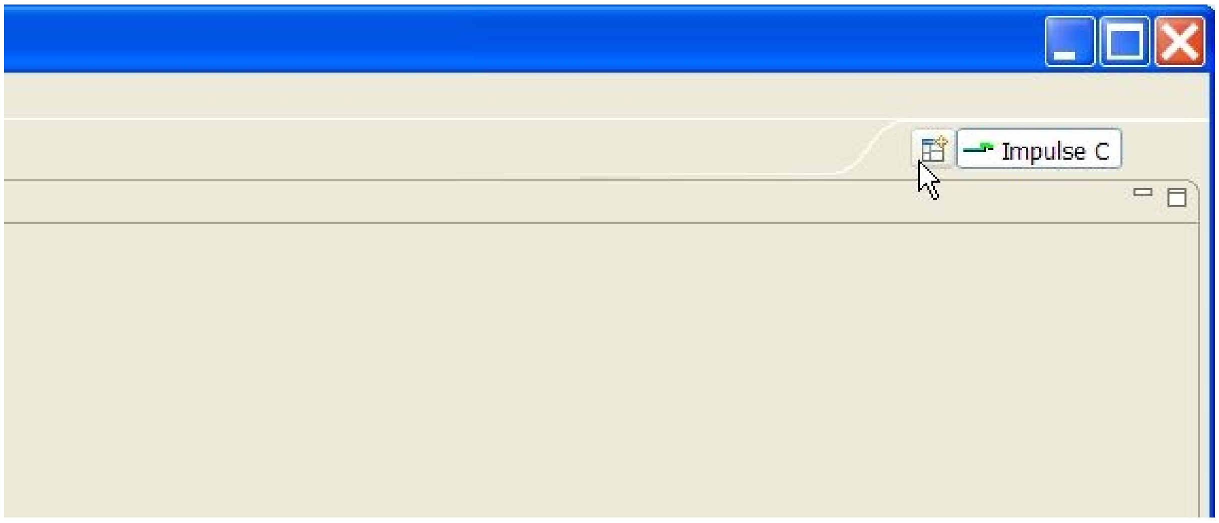
Note: For compatibility with third-party hardware design tools, workspaces should be created in directories that have no spaces in the pathnames.

Eclipse Perspectives and Views

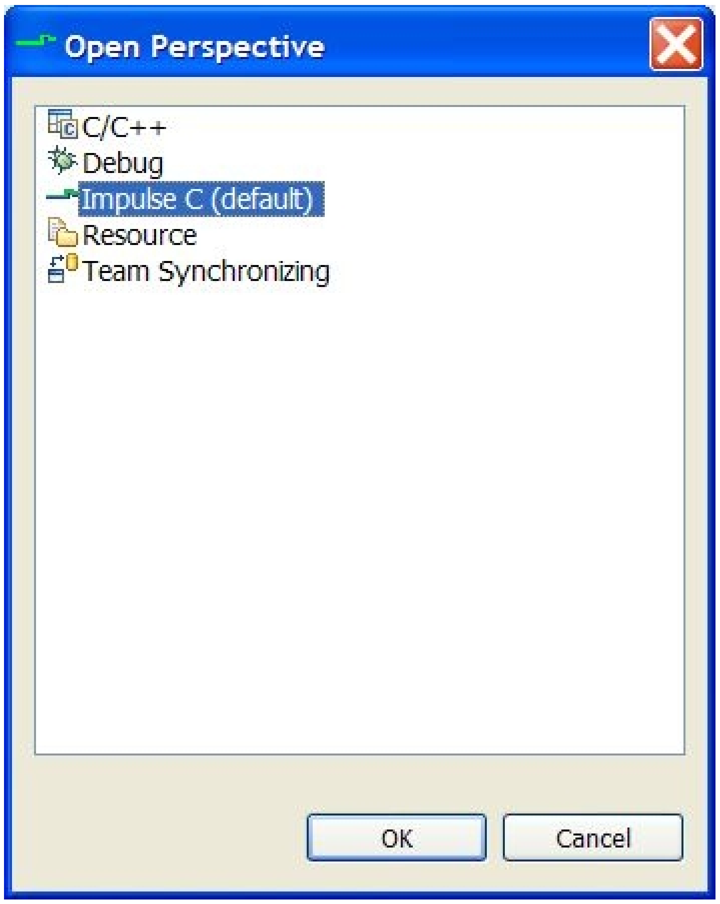
The Eclipse IDE allows several different development tasks to be managed from within a single application. These tasks, such as editing/building Impulse C code or debugging a desktop simulation, require different user interfaces to be presented. The user interface associated with a particular development task is called a *perspective* in Eclipse-speak.

Unless otherwise noted, the "Impulse C" perspective must be selected to carry out the Impulse C development tasks described in this document.

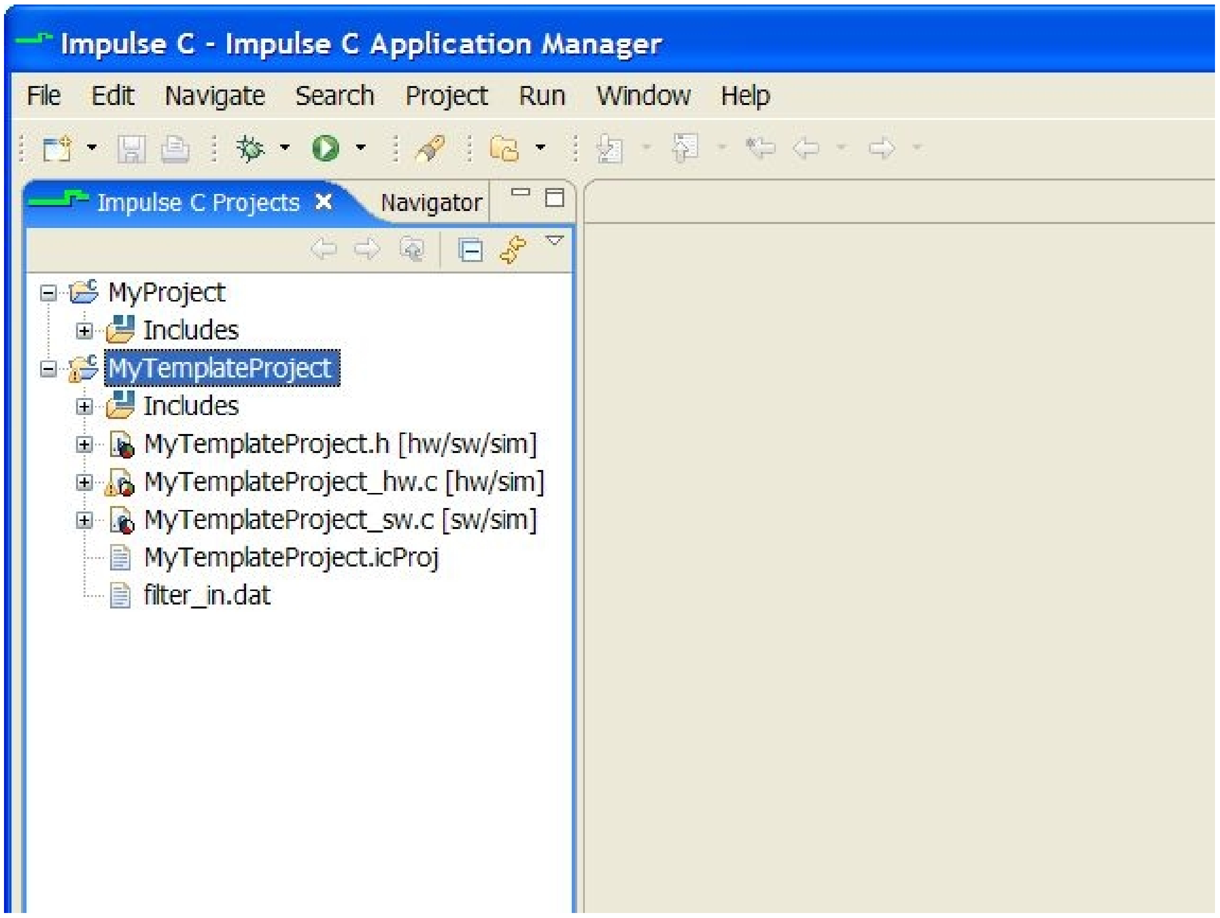
To select among perspectives, click the button in the upper-right corner with the tooltip "Open Perspective" (shown under the mouse pointer, below). The current perspective is indicated in this same area in the upper-right corner:



Alternately, select the Window > Open Perspective menu item. The Open Perspective dialog will appear:



A *view* in Eclipse is a window that displays a hierarchy of information or an editor view. For example, in the "Impulse C" perspective, the workspace's project source files are displayed in a tree hierarchy, marked as hardware/software/desktop simulation files, in the "Impulse C Projects" view. This view is open by default, appearing as a tabbed window in the upper-left corner of the workbench:

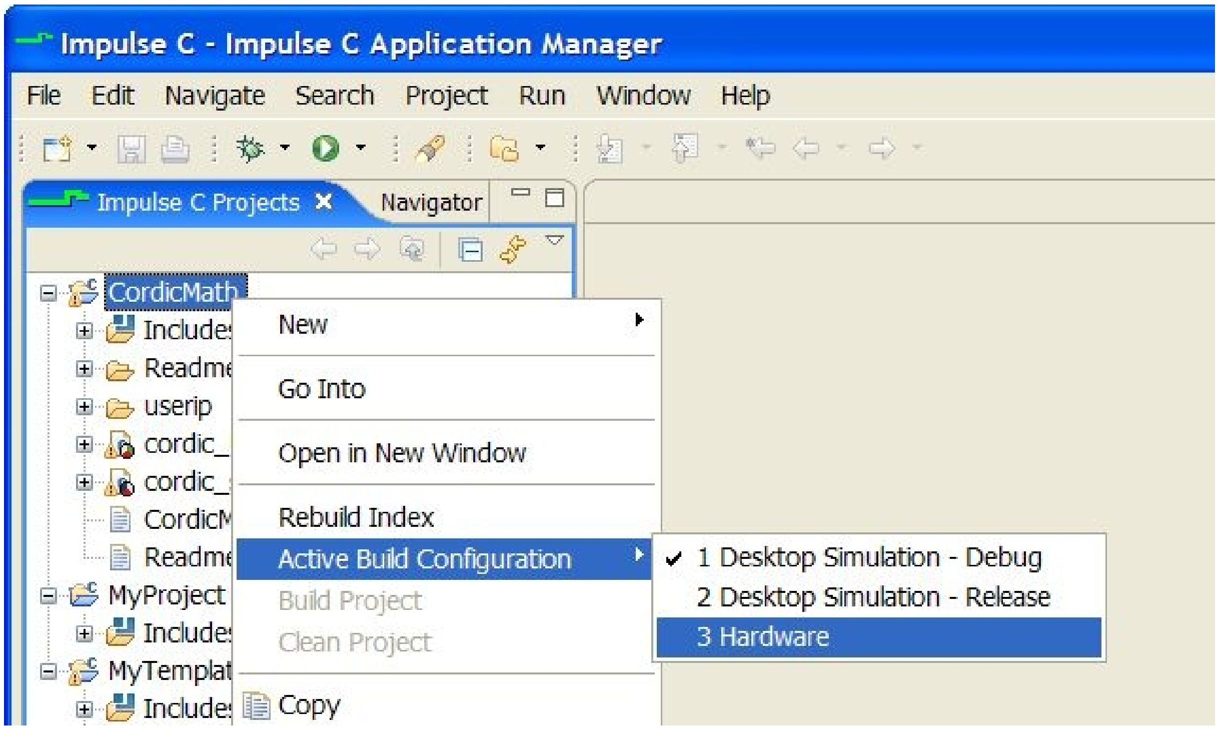

Impulse C - Impulse C Application Manager
File Edit Navigate Search Project Run Window Help
Impulse C Projects
Navigator
MyProject
Includes
MyTemplateProject
Includes
MyTemplateProject.h [hw/sw/sim]
MyTemplateProject_hw.c [hw/sim]
MyTemplateProject_sw.c [sw/sim]
MyTemplateProject.icProj
filter_in.dat

1.3.2 Build Configurations

The Eclipse IDE allows a project to be built for one of several build targets, using a different *build configuration* for each target (e.g., desktop simulation or hardware generation). Three different build configurations are supported:

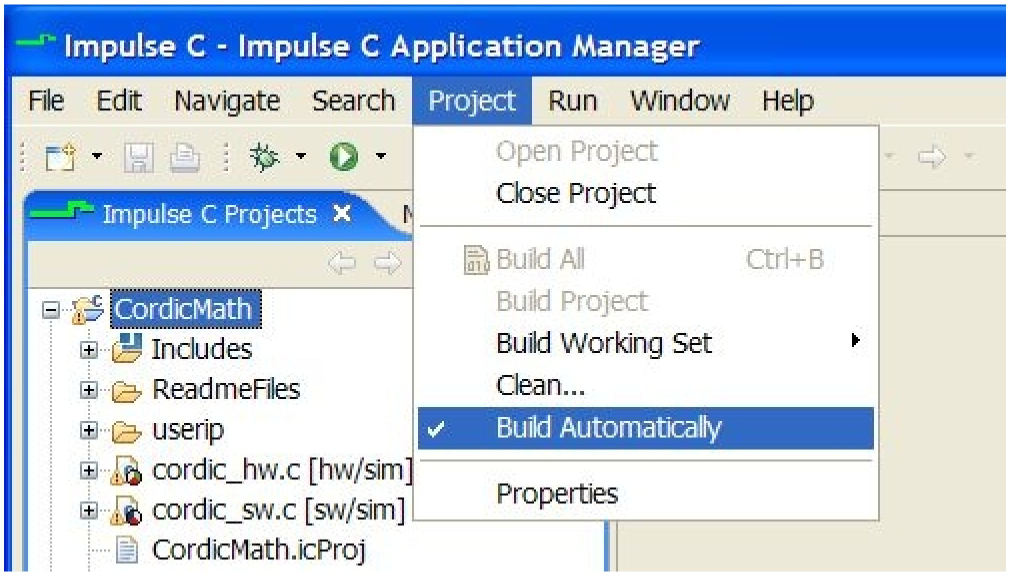
- Desktop Simulation - Debug
- Desktop Simulation - Release
- Hardware

To select a build configuration, make sure the "Impulse C" perspective is selected (see Design Management for more on Eclipse perspectives and views). Right-click the project in the "Impulse C Projects" view and select the Active Build Configuration menu item, as shown:



Automatic Builds

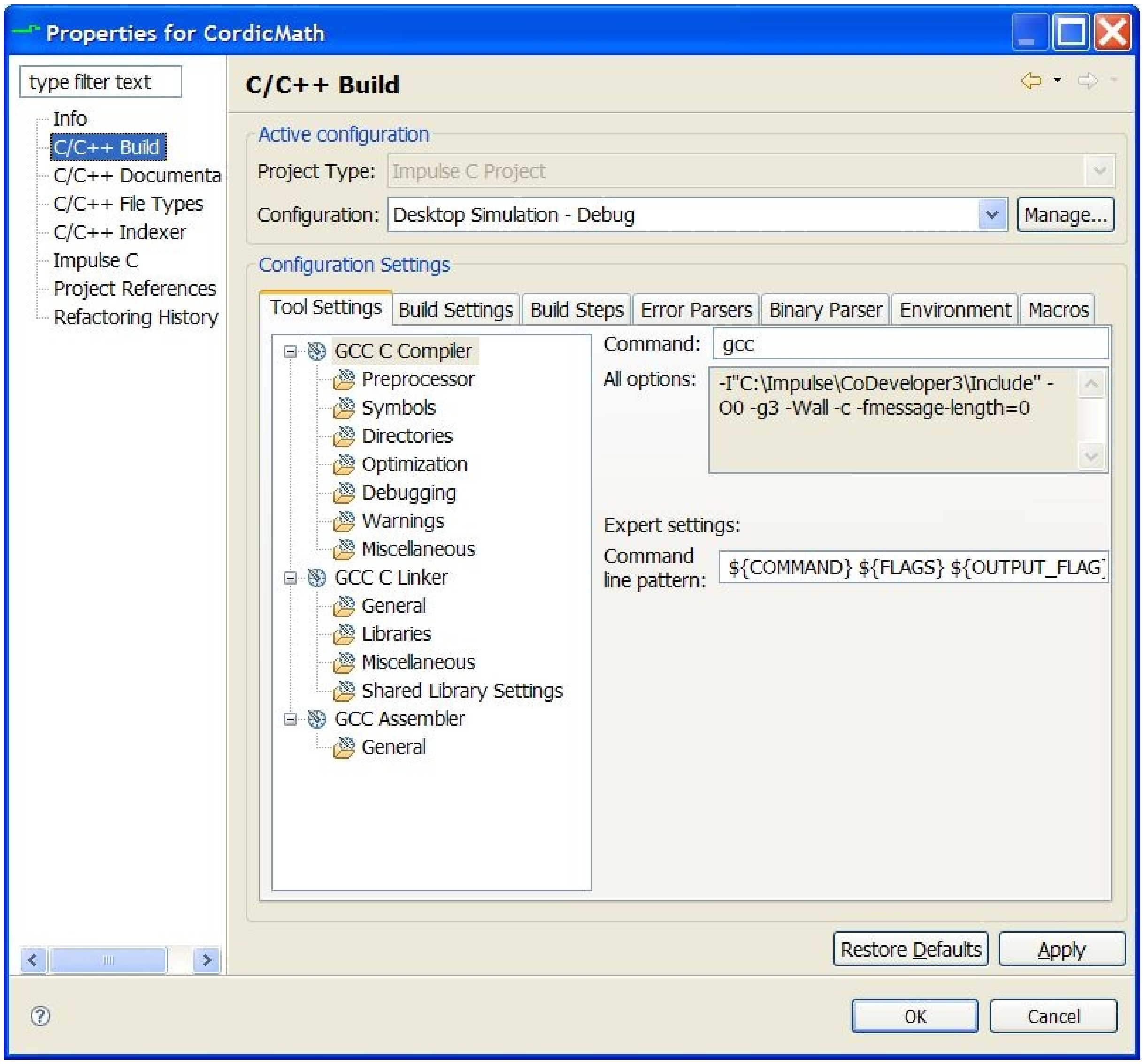
By default, the Eclipse IDE will automatically build the active build configuration whenever a change is made to a project's source code or properties. To disable this feature (recommended), deselect the Project > Build Automatically menu option:



1.3.3 Project Properties

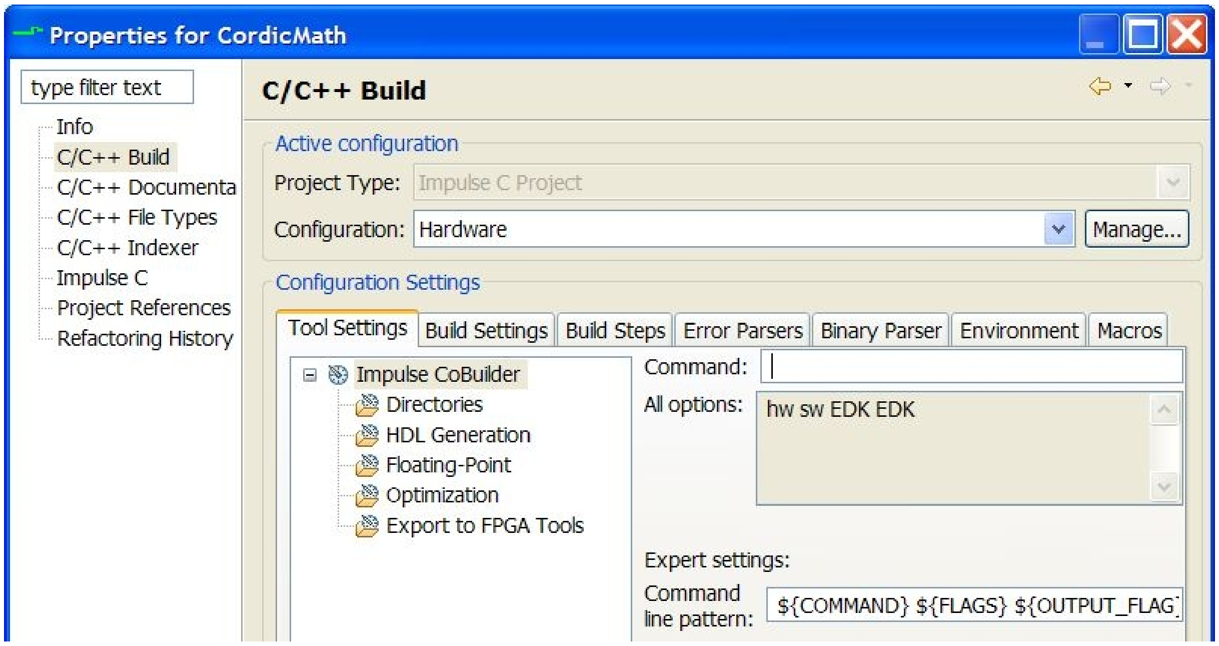
An Impulse C project's properties, including compiler options, source file assignments (hardware/software/desktop simulation), and choice of Platform Support Package, are managed in the Project Properties dialog. To use this dialog, right-click a project in the "Impulse C Projects" workbench view and select the Properties menu item.

Most relevant properties are stored in the "C/C++ Build" page (not in the "Impulse C" page), selected from the left pane of the Properties dialog, as shown:



Project Type: Impulse C Project
Configuration: Desktop Simulation - Debug
Command: gcc
All options: -I"C:\Impulse\CoDeveloper3\Include" -O0 -g3 -Wall -c -fmessage-length=0
Expert settings:
Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG

Each Build Configuration has a different set of properties. Options for the Impulse C hardware compiler are stored under the "Hardware" configuration:


Properties for CordicMath

type filter text

- Info
- C/C++ Build
- C/C++ Documenta
- C/C++ File Types
- C/C++ Indexer
- Impulse C
- Project References
- Refactoring History

C/C++ Build

Active configuration

Project Type: Impulse C Project

Configuration: Hardware

Manage...

Configuration Settings

Tool Settings | Build Settings | Build Steps | Error Parsers | Binary Parser | Environment | Macros

- Impulse CoBuilder
  - Directories
  - HDL Generation
  - Floating-Point
  - Optimization
  - Export to FPGA Tools

Command:

All options: hw sw EDK EDK

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG}

1.4 CoBuilder User Guide

Hardware Generation using CoBuilder

CoDeveloper's CoBuilder hardware generation tools are the key to successfully moving portions of your Impulse C application onto programmable hardware for acceleration. CoBuilder analyzes your application, extracts those processes that you have specified for implemenentation in hardware, and creates optimized (HDL format) hardware descriptions ready for synthesis into your chosen FPGA device.

When CoBuilder is launched from the Application Manager, only those source files that you have specified as hardware (as indicated by small "hw" icons in the project window) are analyzed by the hardware compiler. All other files in the project are ignored. In addition, any Impulse C processes that are found within the specified C source files, but that are not specified as hardware processes through the use of the co_process_config function, are also ignored.

CoBuilder includes a number of important compilation and translation passes that are invoked in sequence to generate HDL outputs corresponding to your C hardware processes. In addition, CoBuilder is capable of generating the necessary hardware/software interfaces (in the form of C libraries and HDL wrapper components), dramatically simplifying the process of moving a complete hardware/software application to selected programmable platforms.

The result of compilation by CoBuilder is a pair of HDL files representing a top-level system (possibly including references to stream components, shared memories or other interfaces) and one or more component-level modules representing the hardware processes in your application. These files are compatible with popular HDL simulation and synthesis tools.

The Impulse HDL Library

When CoBuilder generates HDL outputs representing your application and its constituent processes, the output files are created in a subdirectory of your project that is typically named "hw". When creating these files, CoBuilder makes reference to additional HDL components that are provided in a special library, called "impulse", that must be combined with the generated system-level and component-level HDL files for the purpose of simulation and/or synthesis.

For example, if you are using a VHDL synthesis or simulation tool and wish to import the generated HDL files, you must be sure to also import all of the files that appear in your project's "hw\lib" subdirectory, and assign these files to the "impulse" named library. Refer to your simulator or synthesis documentation for information about how to create and add files to named libraries.

For More Information

For additional information about CoBuilder and about the Platform Support Packages that are required to make use of CoBuilder, refer to your Platform Support Package online documentation.

Important information about CoBuilder optimization strategies and the CoBuilder-related C pragmas can be found in the section titled Programming for Hardware.

See Also

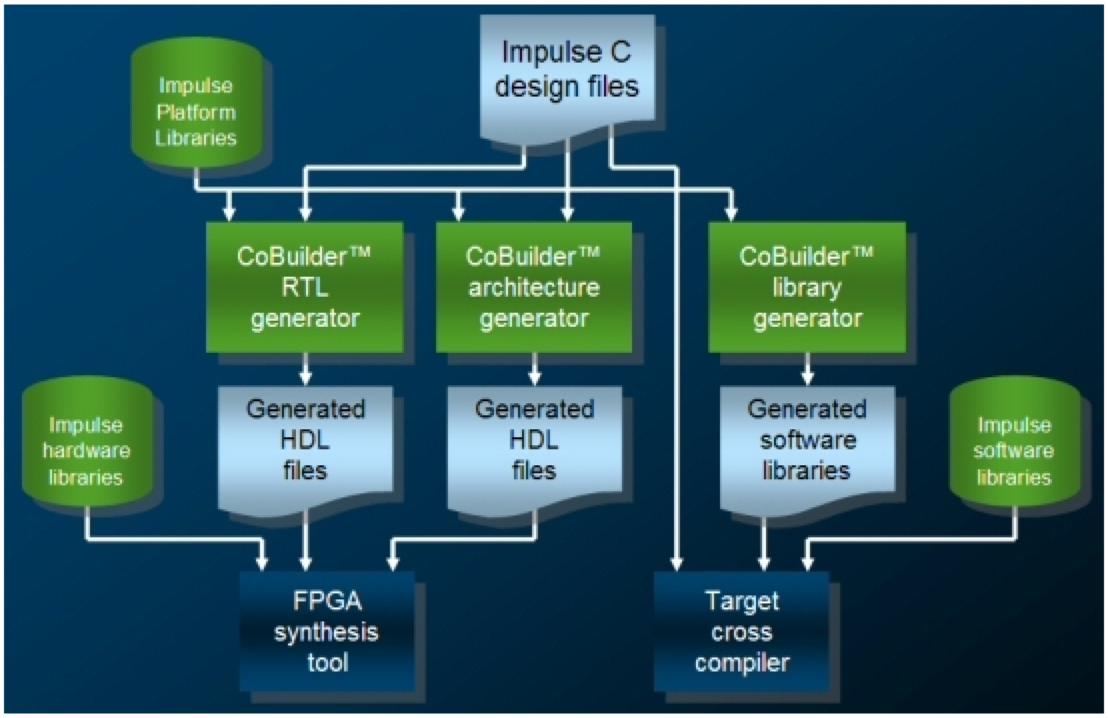
Build Features
Build Options
Generate Options
Generate HDL
Creating System Interfaces
CoBuilder Command Line Tools
Exporting Generated Files to Synthesis Tools

1.4.1 Build Features

Overview

CoBuilder consists of a sequence of translation and compilation processes that analyze your Impulse C application, extract processes that you have configured as hardware processes and generates equivalent low-level hardware descriptions in the form of HDL output files. CoBuilder also generates (depending on information available in the Platform Support Package architecture file you have selected) various hardware/software interface files, including a C runtime library and various hardware components that enable hardware/software communication on your chosen platform target.

The following diagram is a simplified description of his this process works:


Impulse C design files
Impulse Platform Libraries
CoBuilder™ RTL generator
CoBuilder™ architecture generator
CoBuilder™ library generator
Impulse hardware libraries
Generated HDL files
Generated HDL files
Generated software libraries
Impulse software libraries
FPGA synthesis tool
Target cross compiler

As the diagram shows, there are actually three major operations performed as a part of a typical CoBuilder run:

1. Your Impulse C hardware processes are compiled to create equivalent HDL format hardware descriptions.
2. Additional HDL files are generated that set up the hardware interface to the selected platform's on-chip bus.
3. Run-time libraries and other C and assembly language components are generated that set up a software interface for efficient software/hardware communication.

While the first of these operations is relatively generic (is applicable to a wide variety of FPGA-based platform targets), the second and third operations are customized for specific platform targets as described in a Platform Support Package architecture description.

The specific programs used to process an Impulse C application for hardware and software compilation are:

cpp - The GNU C preprocessor

impulse_snoot - The Impulse C parser

impulse_prep - The Impulse C pre-optimizer

impulse_porky - The SUIF (Stanford University Intermediate Format) optimizer

impulse_s2xml - A SUIF to XML processor

impulse_sm - The Stage Master optimizer and scheduler

impulse_genvhdl - The CoBuilder VHDL generator

impulse_genvlog - The CoBuilder Verilog generator

impulse_arch - The CoBuilder hardware architecture generator

impulse_lib - The CoBuilder software interface generator impulse_export - The CoBuilder hardware/software design exporter

Launching External Applications

The launching of third-party applications (such as FPGA synthesis or place and route tools) can be controlled from within the Application Manager if desired. Consult your Platform Support Package documentation for specific details.

Platform Support Packages available with CoDeveloper allow you to generate target files appropriate for platform-specific synthesis and other lower-level compilation as appropriate. When installed, a Platform Support Package is specified in the Options/Generate dialog and appropriate compilation targets are automatically invoked when your project is built. Platform-specific sample projects and additional documentation are included with each Platform Support Package.

See Also

CoDeveloper Build Options
CoDeveloper HDL Generation Options
CoBuilder Command Line Tools
CoDeveloper System Options
Build Software Simulation Executable
Clean Project

1.4.2 Build Options

Description

This option opens the Options dialog with the current Build options displayed. Build options control how the Make program is invoked to process your Impulse C application for the purpose of HDL generation or for other compilation tasks.

What You Do

To set Build options, select Project -> Options from the menu bar to bring up the Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button. Select the Build tab, then set options as needed. The various build options are discussed below.

CoBuilder hardware source (.c) files - Use this field to specify a list of C source files that are to be analyzed by the CoBuilder RTL generator. At a minimum, you must include the name of the C source file containing your application's configuration function in this field. Multiple file names entered in this field must be delimited by space characters. The list of files specified in this field correspond to the HW_SRC Makefile variable passed via the _Makefile.defs definitions file.

CoBuilder hardware include (.h) files - Use this field to specify the list of C header files that are required by the above hardware source files. Multiple file names entered in this field must be delimited by space characters. The list of files specified in this field correspond to the HW_INC Makefile variable passed via the _Makefile.defs definitions file.

CoBuilder software source (.c) files - Use this field to specify a list of C source files that are to be copied to the software build directory by CoBuilder. This list of files normally represents those portions of the application that are to be compiled onto the embedded processor, and for which no RTL will be generated. At a minimum, you must include the name of the C source file containing your application's main() function. (If you will not be compiling any part of your application to an embedded processor then this field may be blank.) Multiple file names entered in this field must be delimited by space characters. The list of files specified in this field correspond to the SW_SRC Makefile variable passed via the _Makefile.defs definitions file.

CoBuilder software include (.h) files - Use this field to specify the list of C header files that are required by the above software source files. Multiple file names entered in this field must be delimited by space characters. The list of files specified in this field correspond to the SW_INC Makefile variable passed via the _Makefile.defs definitions file.

Desktop simulation (.exe) source (.c) files - Use this field to specify a list of C source files that are to be copied to create a simulation executable (which is an .EXE file) for the purpose of desktop simulation. This list of files normally represents both the software and hardware portions of the appliation, and may include additional source files used for test purposes. Multiple file names entered in this field must be delimited by space characters. The list of files specified in this field correspond to the EXE_SRC Makefile variable passed via the _Makefile.defs definitions file.

Desktop simulation (.exe) include (.h) files - Use this field to specify the list of C header files that are required by the above source files. Multiple file names entered in this field must be delimited by space characters. The list of files specified in this field correspond to the EXE_INC Makefile variable passed via the _Makefile.defs definitions file.

Additional compiler options - Use this field to specify additional options (such as -D declarations and other flags) to the GCC compiler.

Additional linker options - Use this field to specify additional options (such as additional libraries and other options) to the GCC linker.

Desktop simulation EXE name - This field specifies the name of the simulation executable to be built when "Build Software Simulation Executable" is selected. This field corresponds to the EXE Makefile variable.

Build for debug (-g) - Select this option to instruct the GCC compiler to embed debugging information in the generated simulation executable. This field corresponds to the DEBUG Makefile variable.

Generate makefile - If this option is selected, CoDeveloper will automatically generate a Makefile and invoke the Make program to process your application. Other program options (listed above) are communicated to the automatically-generated Makefile via a generated _Makefile.defs file.

Custom makefile name - Use this edit field to specify the makefile associated with this project. This option does not appear if the **Generate makefile** option is selected.

Project clean target - Use this optional edit field to use specify a makefile target associated with the Clean Project function. This option does not appear if the **Generate makefile** option is selected.

CoBuilder build target - Use this optional edit field to specify the target of HDL code generation. This field should contain the names of the HDL and other format file(s) that are to be generated as a result of compiling your application for an FPGA target, or the name of a makefile target that will cause the necessary files to be generated. Your project makefile must include a corresponding target. This option does not appear if the **Generate makefile** option is selected.

When you are finished setting options, click on the **Close** button to close the dialog and update the current project options.

See Also

Build Features
Build Software Simulation Executable
Generate HDL

Project / Clean Project

1.4.3 Generate Options

Description

Generate Options are used to specify compiler flags and other options related to HDL and other output file generation. The Generate Options dialog is used to specify these options.

Compile Options

Enable constant propagation

After constant propagation is carried out, expressions that can be calculated at compile time are replaced by their actual values. Known functions of known constants are evaluated as constants and are recognized as such. Constant propagation is also able to change certain conditional branches to unconditional ones. Example:

```
b = 3;
c = b*4;
if(c>10){
  c = c-10;
}
x = c;
```

With constant propagation enabled the compiler may (depending on the circumstances) reduce this to:

```
x = 2;
```

Scalarize array variables

The scalarize option will attempt to convert local arrays into variables so that they may be implemented in registers rather than in memory. When an array is implemented in registers, any number of elements can be written or read simultaneously, allowing a higher level of parallelism. An array will only be scalarized under all of the following conditions:

- All references to the array have a constant index (after loop unrolling)
- The array contains elements of any signed or unsigned integer type (int, long, long long, short, char, co_int*, co_uint*)
- The CO NONRECURSIVE pragma has not been applied to the array
- The array is not read and written in a single C statement (workaround: insert temporaries manually)

As a consequence, scalarization often requires loop unrolling, which replaces all array references involving the loop variable with a constant expression.

Note: For the most effective use of the scalarize option, you must also select the "Enable constant propagation" option.

Relocate loop invariant expressions

This option moves certain expressions (assignments, etc.) outside of the body of loops. This can improve the ability of the Stage Master scheduler to parallelize blocks of code.

Additional optimizer options

Options given in this field will be passed to the impulse_porky optimizer.

Generate dual clocks

This option causes the process stream and signal interfaces to be generated with dual clock interfaces. Specify this option if you will operate your hardware processes at a different clock rate than the embedded processor or system bus . Refer to your Platform Support Package information for specific restrictions and additional information.

Active-low reset

When this option is selected, the global reset signal input to the generated top-level hardware module is assumed to be active low. Otherwise, the global reset signal is considered active high. (Resets internal to Impulse C-generated modules are active high.)

Use std_logic types for VHDL interfaces

This option causes top-level ports to be generated as std_logic (resolved) types rather than std_ulogic (unresolved) types. This option can be useful if you are interfacing your generated logic to other VHDL components that have resolved types. This option is only applicable to VHDL output formats. When specified, the "Use std_logic..." option results in the generation of an additional wrapper VHDL entity with the suffix "_sl". This std_logic wrapper is located at the end of the generated top_xxxx.vhd file, and can be used in place of the std_ulogic entity generated by the compiler.

Do not include co_ports in bus interface

This option specifies that input and output ports declared using the co_port type are not to be included in the generated bus interface wrapper. Un-select this option if you want co_port types to be included in the generated bus interface wrappers.

Library options

Use this field to pass '-l' (library) options to the Stage Master scheduler and the impulse_arch tool.

Include floating point library

This option specifies that FPGA-specific floating-point libraries should be included in the generated HDL, and that floating-point types are to be allowed during compilation. This option requires support for floating-point in the selected Platform Support Package. Selecting this option is the equivalent of passing '-lfloat' in the "Library options" field.

Include math library

This option enables the optional CoDeveloper math library, which includes many of the functions commonly found in math.h. Selecting this option is the equivalent of passing '-lfloat_math_fast' in the "Library options" field. (Note: This feature is not supported in all Platform Support Packages. Contact Impulse for details.)

Enable floating point optimization

This option enables additional optimizations on floating point operations to be performed by the compiler. These optimizations include floating point register allocation, operator sharing, and constant multiplier optimizations. This option may increase throughput of certain types of computations as well as reducing resource usage.

Enable floating point accumulator

This option enables the automatic generation of floating point accumulators and multiply/accumulators.

This option may increase throughput of certain types of computations as well as reducing resource usage.

Use extended floating point accumulator

This option enables the generation of extended precision floating point accumulators and multiply/accumulators. This option may increase throughput of certain types of computations.

Use higher latency, faster clock operators

This option specifies that a floating-point library tuned for faster clock rates (with the tradeoff of higher latency) should be included in the generated HDL. This option requires support for controlling floating-point operators' latency in the selected Platform Support Package. Selecting this option is the equivalent of passing '-lfloat_fast' in the "Library options" field.

Allow double-precision types and operators

This option specifies that double-precision (64-bit) floating-point operations are used by the application. This option requires support for double-precision floating-point in the selected Platform Support Package. Selecting this option is the equivalent of passing the '-with_double' option to impulse_s2xml.

Output Options

The following Generate Options are useful for integration with FPGA synthesis and routing tools:

Hardware Build Directory - This directory specification instructs the CoBuilder hardware generator where to place the generated output files. These files include (at a minimum) two HDL files describing the hardware generated for one or more Impulse C processes. A top-level HDL file describing the bus interface, if an embedded processor-capable Platform Support Package is selected, will also appear here. These files will be used by the FPGA synthesis tool to generate FPGA hardware ready for downloading. Refer to your Platform Support Package documentation for additional details.

Software Build Directory - This directory specification instructs the CoBuilder software generator where to place the generated software interface files. These files include a variety of C-language (and other) source files as well as script files that define the software side of the hardware/software interface. Refer to your Platform Support Package documentation for additional details.

Hardware Export Directory - This directory specification is used by the Export Generated Hardware function (located in the CoDeveloper Project menu) to determine where to copy files previously generated by CoBuilder. The use of this directory specification (and of the Export Generated Hardware menu item) is optional. Refer to your Platform Support Package documentation for additional details.

Software Export Directory - This directory specification is used by the Export Generated Software function (located in the CoDeveloper Project menu) to determine where to copy software source files previously generated by CoBuilder. The use of this directory specification (and of the Export Generated Software menu item) is optional. Refer to your Platform Support Package documentation for additional details.

See Also

Build Options
Generate HDL
Exporting Generated Files to Synthesis Tools
Programming for Hardware

1.4.4 Generate HDL

Description

This menu item (and associated toolbar button) generates hardware description language (HDL) files from the Impulse C hardware processes in your application.

You may specify options that affect HDL generation in the Generate Options dialog.

After generating hardware, you may optionally use the Export Generated Hardware and Export Generate Software functions to move the generated hardware (HDL) and software files to directories where they will be processed by third-party tools into FPGA bitfile or executables for the target platform's CPU, respectively. See your Platform Support Package documentation for more details.

See Also

Generate Options
Build Features
Platform Support Package Overview

1.4.5 Exporting Generated Files to Synthesis Tools

Overview

This section briefly describes how to export the hardware files generated by CoDeveloper to a synthesis environment. This information assumes that you are not generating a complete hardware/software system, which would include an embedded processor, and are instead generating a single hardware module for integration with other FPGA hardware elements. When building an application that does include an embedded processor, the chosen Platform Support Package should be one of the "Generic" packages.

If you are using an embedded or external processor in your application, please refer to information provided with your selected Platform Support Package.

Exporting CoDeveloper-Generated Files to Synthesis Tools

Exporting files from CoDeveloper to a synthesis tool is straightforward, once you understand how to specify the location of the Impulse library files. Here's how it works:

When you generate hardware for your project, the generated HDL and related files are output to the "Hardware build directory", as defined in the project's Generate Options. This directory must be a subdirectory of the project directory and is named "hw" by default. This directory will contain the generated logic for your Impulse C application, and also contains the Impulse library files, which are copied automatically during hardware generation.

When creating a synthesis project, the Impulse library files (which are found in the "hw/lib" subdirectory) need to be assigned to the "impulse" library within your synthesis tool. When you do this, make sure the correct library files are being used. For example, don't copy the files from the CoDeveloper installation; instead, make sure the correct Platform Support Package has been selected when building and that the files being imported into the synthesis project are from the designated "Hardware build directory".

If your synthesis tool supports adding a *copy* of an HDL source file, *do not* add the exported source using this feature. If you make changes to the Impulse C project and export hardware again, the newly generated files will not be recognized by the synthesis tool.

Here are the detailed steps:

1. Create a new project in your synthesis tool. Select the target device type and other options, give the project a name and save it somewhere beneath the Impulse C project directory.
2. For VHDL applications, create a new HDL library, selecting VHDL as the source file type. Give the new library the name "impulse".
3. Add all files from the "hw/lib" subdirectory to your project. If you are using VHDL, make sure the imported library files are added to (or moved to) the "impulse" library.
4. Add the other generated HDL files from the "hw" subdirectory. These files are normally named "xxxx_top.vhd" (or "xxxx_top.v") and "xxxx_comp.vhd" (or "xxxx_comp.v"), where *xxxx* refers to your Impulse C project's name. If you using VHDL, make sure these files are added to the "work" library.

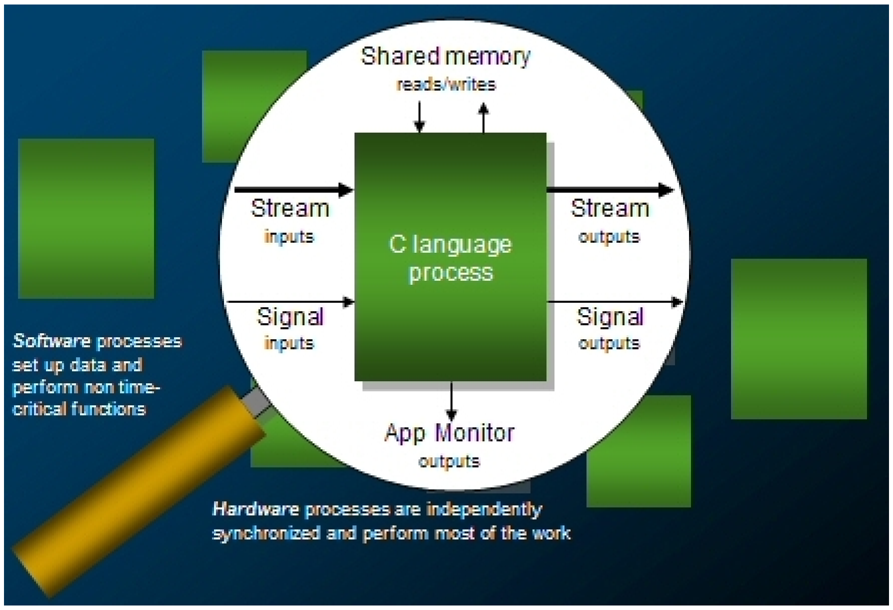
See Also

Platform Support Package Overview

1.4.6 Creating System Interfaces

Overview

The Impulse C compiler supports multiple methods of connecting your generated hardware to the rest of the system. For data movement, you have the choice of four different of I/O objects: streams, signals, shared memories and registers. As described in The Programming Model, these four types of objects make it possible for your Impulse C hardware processes to communicate with other Impulse C processes, and with external hardware or software:


Shared memory
reads/writes
Stream
inputs
C language process
Stream
outputs
Signal
inputs
Signal
outputs
Software processes set up data and perform non time-critical functions
App Monitor
outputs
Hardware processes are independently synchronized and perform most of the work

For any of these objects, you have the choice of connecting them to other Impulse C processes, or leaving them unconnected for the purpose of creating your own system or hardware interfaces. (This idea is explored in Tutorial 4.) If you want to allow CoDeveloper to generate names for your objects and generated bus interfaces, then you must supply complete connections in your configuration function. This means that every stream, signal and register object must have both a source (a "producer") and a destination (a "consumer"). If, on the other hand, you want to define your own port names and leave one end unconnected, then you must use a co_port object to complete the connection and specify the port name.

Co_ports and Bus Interfaces

By default, any input or output stream, signal or register that is declared using a **co_port** object will not be included in generated bus interface wrappers. This is the preferred behavior if you are creating a bus peripheral that requires additional direct (not bus-connected) hardware interfaces.

If you want one or more of your stream, signal or register interfaces to be connected via the generated bus interface wrapper, then you should un-select the "Do not include co_ports in bus interfaces" option in the Generate Options dialog.

See Also

Hardware/Software Interfaces

1.4.7 CoBuilder Command Line Tools

Description

CoBuilder includes a collection of programs that must be invoked in sequence to generate hardware and software output files. These programs may be invoked from within CoDeveloper or from within other IDE environments as desired. You may also wish to invoke these programs from within scripts or makefiles.

This section describes the command line options available for each program:

Impulse_snoot - Impulse C parser

Impulse_prep - Impulse C pre-optimizer

Impulse_porky - SUIF (Stanford University Intermediate Format) optimizer

Impulse_impc - Impulse C / SUIF post-processor

Impulse_s2xml - SUIF to XML processor

Impulse_sm - Stage Master optimizer and scheduler

Impulse_genvhdl - CoBuilder VHDL generator

Impulse_genvlog - CoBuilder Verilog generator

Impulse_arch - The CoBuilder hardware architecture generator

Impulse_lib - The CoBuilder software interface generator

Impulse_export - The CoBuilder hardware/software design exporter

See Also

Build Options
Build Features

1.4.7.1 impulse_snoot

impulse_snoot

Description: Impulse C parser

Command summary: impulse_snoot -Timpulse-c <input file> <output file>

Options:

none

Notes:

The input to *impulse_snoot* must be the output of the C preprocessor.

See Also

Impulse_porky

1.4.7.2 impulse_prep

impulse_prep

Description: Impulse C pre-optimizer

Command summary: impulse_prep <input file> <output file>

Options:

none

Notes:

Impulse_prep must be invoked for any application that makes use of the UNROLL pragma, and for applications that make use of C case (switch) statements. The input to *impulse_prep* must be the output of *impulse_snoot*.

See Also

Impulse_snoot
Impulse_porky
Impulse_impc

1.4.7.3 impulse_porky

impulse_porky

Description: SUIF (Stanford University Intermediate Format) optimizer

Command summary: impulse_porky [options] <input file> <output file>

Options:

-unused-syms	(Remove unused symbols. Required.)
-unused-types	(Remove unused types. Required.)
-iterate	(Iterate until complete. Required.)
-build-arefs	(Build array references from pointer usages. Required.)
-Dmemcpys	(Prevents memcpys from being used. Required.)
-cse	(Common subexpression elimination. Recommended.)
-fold	(Fold constants where possible. Recommended.)
-const-prop	(Simple constant propogation. Optional.)
-loop-invariants	(Moves certain expressions outside of loop bodies. Optional.)
-scalarize	(Scalarize local array variables to increase parallelism. Optional.)
-reduction	(Performs some reduction optimization on loops. Optional.)

Notes:

Impulse_porky may be invoked multiple times as needed to perform optimizations. Input to *impulse_porky* must be the output of *impulse_snoot* or *impulse_porky*. The -build-arefs optimization must follow other optimizations as a separate *impulse_porky* pass.

Note also that the *porky* software provided by Stanford University includes additional optimization options not listed here. The use of *porky* options not described above may result in incorrect results or errors in subsequent CoBuilder passes.

See Also

Impulse_snoot
Impulse_prep
Impulse_impc

1.4.7.4 impulse_impc

impulse_impc

Description: Impulse C / SUIF post-processor

Command summary: impulse_impc <input file> <output file>

Options:

none

Notes:

The input to *impulse_impc* must be the output of *impulse_porky*.

See Also

Impulse_porky
Impulse_s2xml

1.4.7.5 impulse_s2xml

impulse_s2xml

Description: SUIF to XML processor

Command summary: impulse_s2xml <input file> > <output file>

Options:

-with_double	(Allow use of C `double` type. Optional.)

Notes:

The input to *impulse_s2xml* must be the output of *impulse_impc*. The output of *impulse_s2xml* is printed to standard output and should be redirected to the output file.

See Also

Impulse_impc
Impulse_sm

1.4.7.6 impulse_arch

impulse_arch

Description: The CoBuilder hardware architecture generator

Command summary: impulse_arch [options] <input .xic file> <output HDL file>

Options:

Architecture generation mode only:

-files "<file1> <file2> ..."	(Filenames for both the core and top-level interface HDL files. Required.)
-swdir<software build dir>	("Software build directory" from project's Generate Options. Required for some PSPs.)

CoValidator mode only:

-tb <tool description file>	(Generate HDL testbench for the specified toolset. Required.)
-hwdir <hardware build dir>	("Hardware build directory" from project's Generate Options. Required.)

All modes of operation:

-a<architecture file>	(Architecture file; must be a PSP XML file. Required.)
-active_low_reset	(Generate an active-low global reset. Default is active-high. Optional.)
-dc	(Generate dual clock. Optional.)
-no_port_bus_connect	(Do not connect co_ports to the bus interface. Optional.)
-std_logic	(Generate std_logic interfaces at the top level. Default is std_ulogic. VHDL only. Optional.)
-l<library name>	(HDL library to include. Optional.)

Output:

The output of *impulse_arch* is an HDL file describing either the top level of the hardware module (when the -files option is given) or a testbench that instantiates the top-level module (when the -tb option is given). By convention, the output file is named <project name>_top.vhd or <project name>_top.v, or tb_<project name>_top.vhd, respectively, depending on the mode of operation and the target HDL.

For applications using an embedded processor or system bus, *impulse_arch* may output an HDL file describing a bus interface to the hardware module's top-level entity, as well as files describing the bus connection, specific to an FPGA vendor's design tools. The Platform Support Package selected with the -a option handles generating any such interface files.

Notes:

The input to *impulse_arch* must be the output of *impulse_s2xml*.

See Also

Impulse_s2xml
Impulse_lib
Impulse_export

1.4.7.7 impulse_sm

impulse_sm

Description: Stage Master optimizer and scheduler

Command summary: impulse_sm [options] <input file> <output file>

Options:

```
-a<architecture file>          (Architecture file; must be an XML file.  Required.)
-g                             (Generates a debug file for Stage Master Debugger.  Required.)
-fpopt                         (Perform additional floating point optimizations.)
-fpacc                         (Generate floating point accumulators.)
-fpaccx                        (Generate extended precision floating point accumulators.)

-l<library name>               (HDL library to include.  Optional.)
-p<library name>.<parameter name>=<parameter value>
                               (HDL library parameter specification.  Optional.)

-h                             (Displays brief architecture summary and exits. Requires -a option.)
```

Notes:

The input to *impulse_sm* must be the output of *impulse_s2xml*.

HDL Libraries:

Stage Master can schedule operations involving external HDL entities, such as a user-defined VHDL module that implements a square root function. Sets of such external HDL implementations are described in XML files called *library definition files*. For Stage Master to use such an HDL library:

1. The library's name and the location of its definition file must be given in the chosen Platform Support Package's definition file (e.g., Architectures\xilinx_generic_opb.xml) using the "library" XML element.
2. The -l option must be used to pass the library's name to Stage Master.

Libraries may support parameters. To define a parameter's value, use the -p option.

See Also

Impulse_s2xml
Impulse_genvhdl
Impulse_genvlog

1.4.7.8 impulse_genvhdl

impulse_genvhdl

Description: CoBuilder VHDL generator

Command summary: impulse_genvhdl <input file> <output file>

Options:

none

Output:

The output of *impulse_genvhdl* is a VHDL file describing the core RTL of the hardware module. By convention, it is named <project name>_comp.vhd.

Notes:

The input to *impulse_genvhdl* must be the output of *impulse_sm*.

See Also

Impulse_sm
Impulse_arch
Impulse_export

1.4.7.9 impulse_genvlog

impulse_genvlog

Description: CoBuilder Verilog generator

Command summary: impulse_genvlog <input file> <output file>

Options:

none

Output:

The output of *impulse_genvlog* is a Verilog file describing the core RTL of the hardware module. By convention, it is named <project name>_comp.v.

Notes:

The input to *impulse_genvlog* must be the output of *impulse_sm*.

See Also

Impulse_sm

Impulse_arch
Impulse_export

1.4.7.10 impulse_lib

impulse_lib

Description: The CoBuilder software interface generator

Command summary: impulse_lib [options] <input file> <output file>

Options:

-a<architecture file>	(Architecture file; must be an XML file. Required.)
-files "<file1> <file2> ..."	(Software source file names. Required.)
-hwdir<hardware build dir>	("Hardware build directory" from project's Generate Options.

Required for some PSPs.)

Output:

The output of *impulse_lib* is a C source file containing a main() function that initializes the FPGA hardware and executes the application's software processes. This code is to be compiled for the target embedded processor. By convention, the output file is named co_init.c.

For applications using an embedded processor or system bus, *impulse_lib* may create files to support building a software driver for the hardware module generated by CoBuilder. This functionality is provided by the chosen Platform Support Package.

Notes:

The input to *impulse_lib* must be the output of *impulse_s2xml.*

See Also

Impulse_s2xml
Impulse_export

1.4.7.11 impulse_export

impulse_export

Description: The CoBuilder hardware/software design exporter

Command summary: impulse_export [options] <input file> <output directory>

Options:

-a<architecture file>	(Architecture file; must be an XML file. Required.)

One of the following is required:

-software	(Export the software interface files)
-hardware	(Export the hardware interface files)

Output:

The output of *impulse_export* is a set of files ready to be imported into the next tool in the development flow. For the hardware portion of an application, this may include the hardware module's generated HDL files, Impulse C and user-defined libraries, and tool-specific definition files. For the software portion of an application, this may include software source files (generated and user-defined), Impulse C driver libraries, and Makefiles.

Notes:

The input to *impulse_export* must be the output of *impulse_s2xml.*

See Also

Impulse_s2xml
Exporting Generated Files to Synthesis Tools

1.5 CoDeveloper Pro Tools

Optional Features

The CoDeveloper Pro tools are an optional (add-on) set of utilities that enhance your ability to analyze, optimize and debug your Impulse C applications. The two primary components of these tools are Stage Master Explorer and Stage Master Debugger.

See Also

Stage Master Explorer
Stage Master Debugger
Tutorial 5: The CoDeveloper Pro Tools

1.5.1 Stage Master Explorer

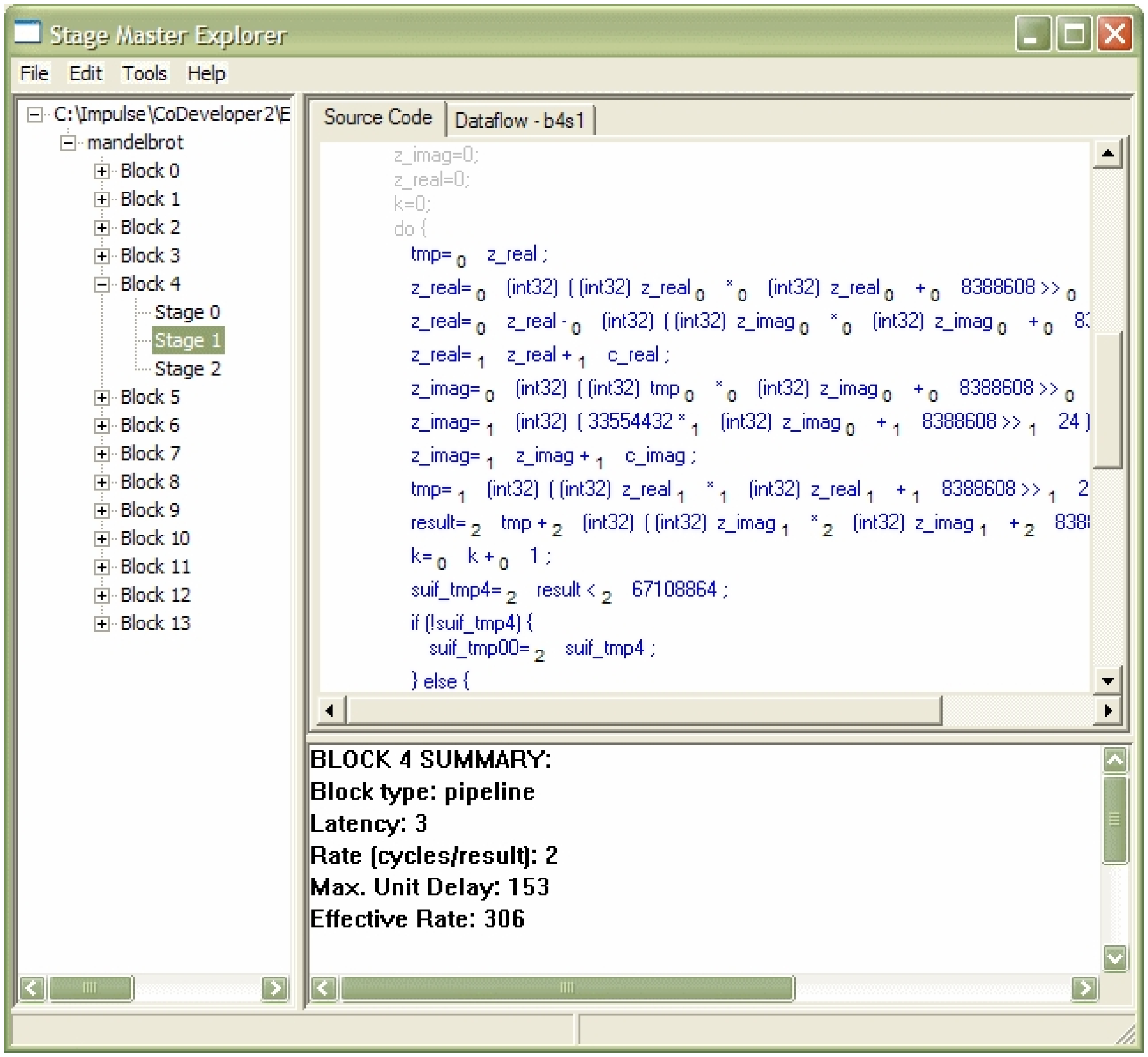
Overview

Stage Master Explorer is a graphical tool that allows you to analyze your Impulse C application and determine, on a process-by-process basis, how effectively the compiler was able to parallelize your C language statements. The tool presents you with information about each block of C code, and is capable of generating a dataflow graph showing the relationships between different operations that result from C language compilation. The tool can also provide you with a pipeline graph, showing you the estimated impact of various pipelining and compilation strategies.

The following screen images provide a summary of Stage Master Explorer features. Additional details can be found in Tutorial 5: The CoDeveloper Pro Tools.

Expanded Source Display

The Stage Master Explorer tool provides you with detailed, statement by statement analysis of parallel optimization results. Each expanded C source code line is annotated with stage information, and a block optimization summary provides you with estimated path delays, loop latencies and effective pipeline rates as appropriate:

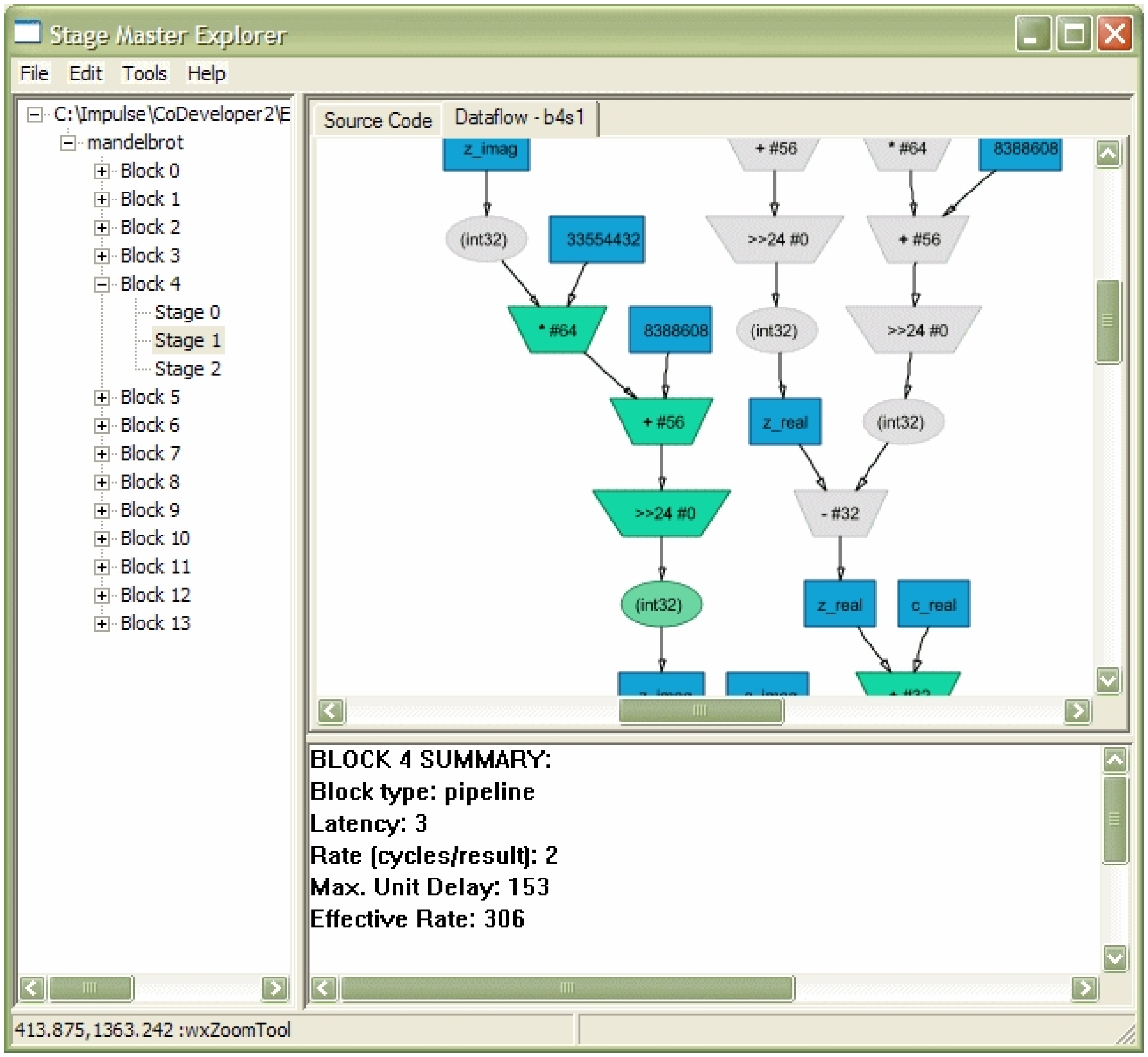


```
        z_imag=0;
        z_real=0;
        k=0;
        do {
          tmp= 0  z_real ;
          z_real= 0  (int32) ( (int32) z_real 0  * 0  (int32) z_real 0  + 0  8388608 >> 0
          z_real= 0  z_real - 0  (int32) ( (int32) z_imag 0  * 0  (int32) z_imag 0  + 0  8
          z_real= 1  z_real + 1  c_real ;
          z_imag= 0  (int32) ( (int32) tmp 0  * 0  (int32) z_imag 0  + 0  8388608 >> 0
          z_imag= 1  (int32) ( 33554432 * 1  (int32) z_imag 0  + 1  8388608 >> 1  24 )
          z_imag= 1  z_imag + 1  c_imag ;
          tmp= 1  (int32) ( (int32) z_real 1  * 1  (int32) z_real 1  + 1  8388608 >> 1  2
          result= 2  tmp + 2  (int32) ( (int32) z_imag 1  * 2  (int32) z_imag 1  + 2  838
          k= 0  k + 0  1 ;
          suif_tmp4= 2  result < 2  67108864 ;
          if (!suif_tmp4) {
            suif_tmp00= 2  suif_tmp4 ;
          } else {
```

```
BLOCK 4 SUMMARY:
Block type: pipeline
Latency: 3
Rate (cycles/result): 2
Max. Unit Delay: 153
Effective Rate: 306
```

Dataflow Graph

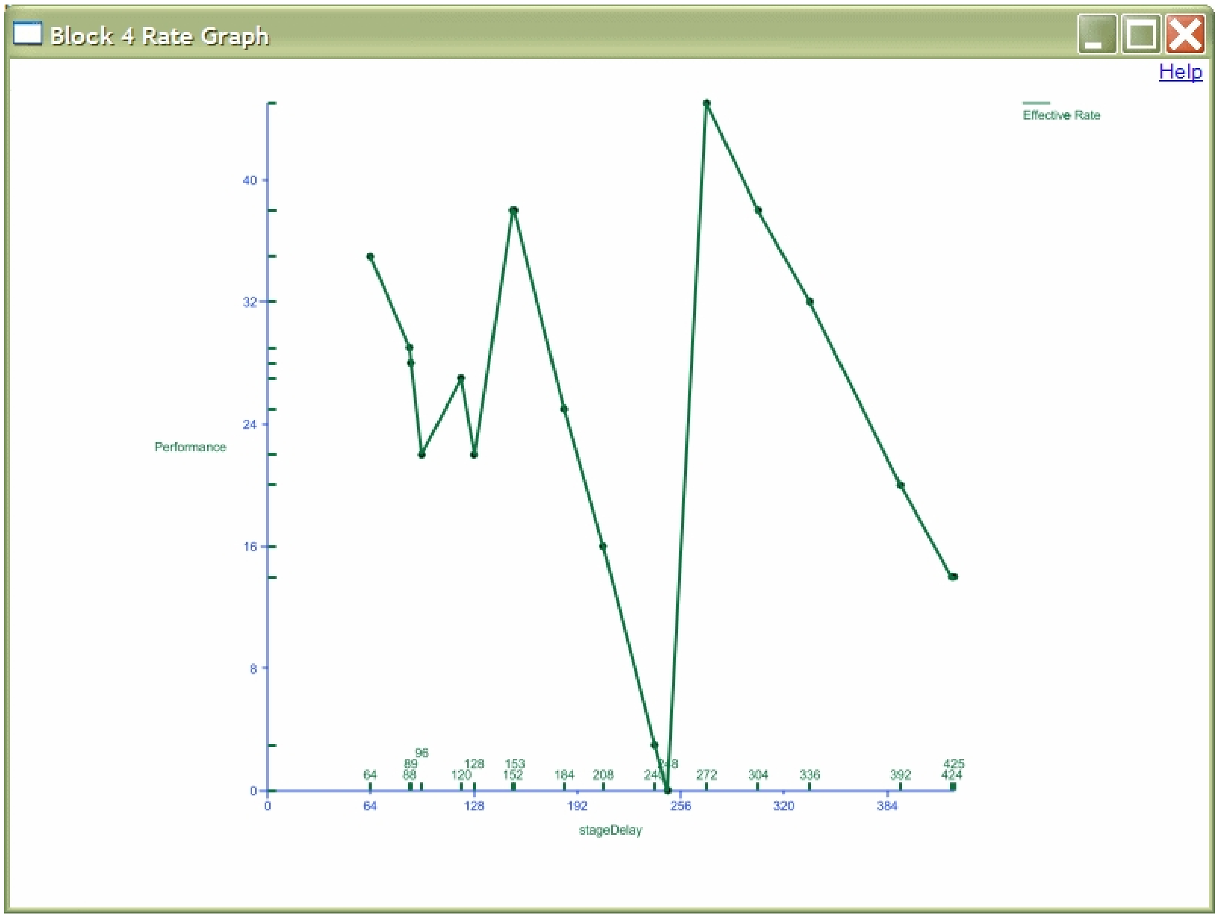
For more detailed analysis of the generated parallel hardware, select the Dataflow tab to observe a complete graph representing the generated parallel structure. This graph allows you to observe, stage-by-stage, how your C language statements have been decomposed unto parallel operations:



BLOCK 4 SUMMARY:
Block type: pipeline
Latency: 3
Rate (cycles/result): 2
Max. Unit Delay: 153
Effective Rate: 306

Pipeline Effective Rate Graph

For processes that include pipelined loops, use the Pipeline Graph to determine the StageDelay setting that results in the highest effective rate. By using this tool, you may discover that your application can actually achieve its best effective data rate at a reduced clock speed:



See Also

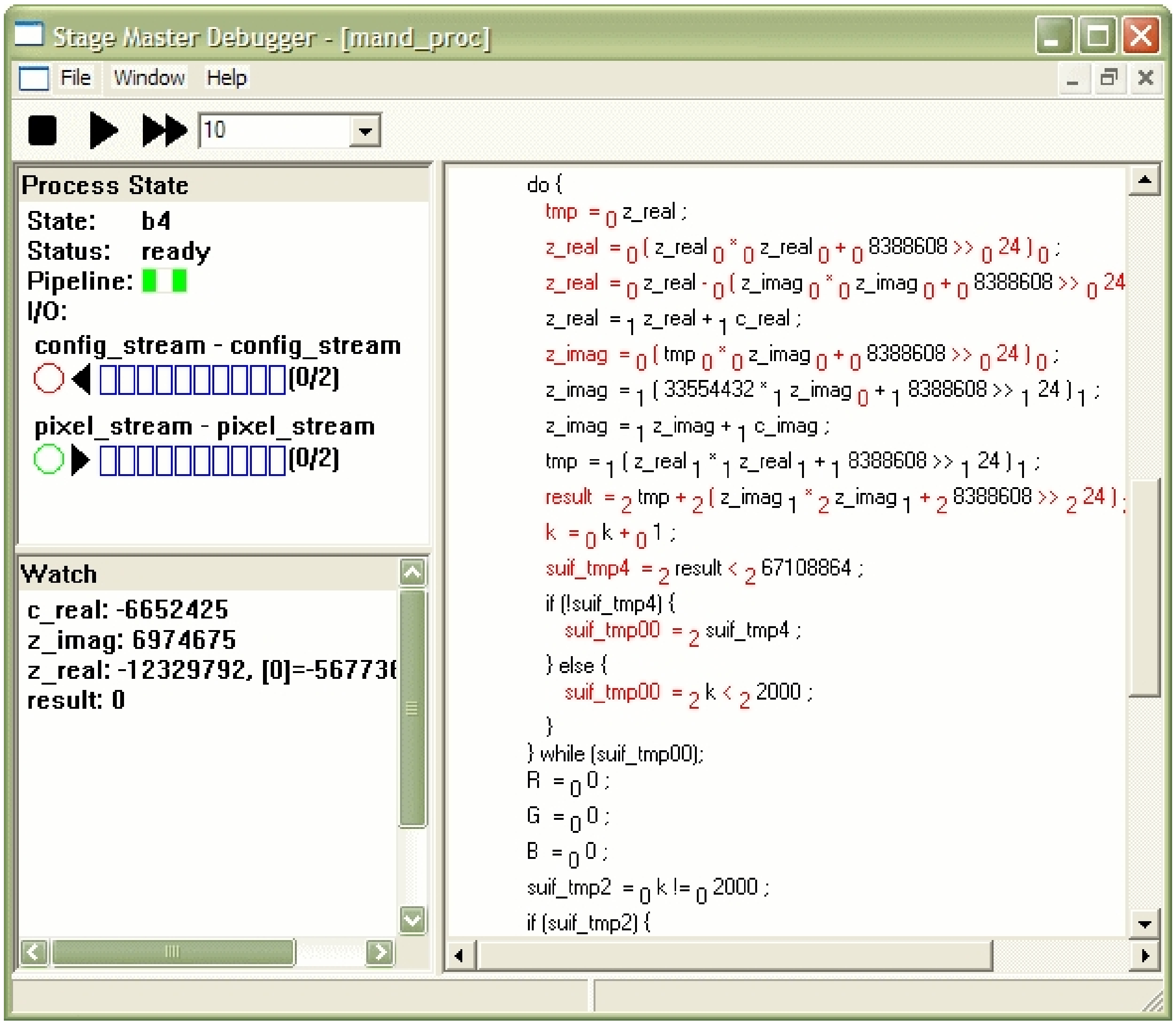
Stage Master Debugger
Tutorial 5: The CoDeveloper Pro Tools

1.5.2 Stage Master Debugger

Overview

Stage Master Debugger is a graphical debugging tool that allows you to observe the cycle-by-cycle behavior of your compiled C code. This is particularly useful when used in combination with the Stage Master Explorer tool, as it allows you to precisely determine what C language statements and operations are being executed in each clock cycle. This in turn helps you to understand how to write better, more efficient C code for parallel optimization, and to find logic errors that may not have been caught using traditional C-language debugging tools.

The following screen image illustrates a typical debugging session:



```
Process State
State:    b4
Status:   ready
Pipeline:
I/O:
config_stream - config_stream
(0/2)
pixel_stream - pixel_stream
(0/2)

Watch
c_real: -6652425
z_imag: 6974675
z_real: -12329792, [0]=-567736
result: 0
```

```
do {
  tmp  = 0 z_real ;
  z_real  = 0 ( z_real 0 * 0 z_real 0 + 0 8388608 >> 0 24 ) 0 ;
  z_real  = 0 z_real - 0 ( z_imag 0 * 0 z_imag 0 + 0 8388608 >> 0 24
  z_real  = 1 z_real + 1 c_real ;
  z_imag  = 0 ( tmp 0 * 0 z_imag 0 + 0 8388608 >> 0 24 ) 0 ;
  z_imag  = 1 ( 33554432 * 1 z_imag 0 + 1 8388608 >> 1 24 ) 1 ;
  z_imag  = 1 z_imag + 1 c_imag ;
  tmp  = 1 ( z_real 1 * 1 z_real 1 + 1 8388608 >> 1 24 ) 1 ;
  result  = 2 tmp + 2 ( z_imag 1 * 2 z_imag 1 + 2 8388608 >> 2 24 ) ;
  k  = 0 k + 0 1 ;
  suif_tmp4  = 2 result < 2 67108864 ;
  if (!suif_tmp4) {
    suif_tmp00  = 2 suif_tmp4 ;
  } else {
    suif_tmp00  = 2 k < 2 2000 ;
  }
} while (suif_tmp00);
R  = 0 0 ;
G  = 0 0 ;
B  = 0 0 ;
suif_tmp2  = 0 k != 0 2000 ;
if (suif_tmp2) {
```

Unlike a traditional source-level debugger, the Stage Master Debugger executes your application on a cycle-by-cycle basis. This means that single-stepping is accomplished by advancing the system clock, rather than by advancing by one source code line. Because C language statements are parallelized by the Impulse C compiler, you will often observe multiple lines of C code being executed simultaneously during the same clock cycle (or *instruction stage*).

In this example, notice that the inner code loop (a "do" loop) includes a three-stage pipeline. (In the original C code, this pipeline would have been enabled using the PIPELINE pragma.) During debugging using Stage Master Debugger, the lines of C code that are active in the current stage (in this case representing the first and third stages of the pipeline, numbered stage 0 and stage 2, respectively) are highlighted red. Notice also that stream graphs are presented, allowing you to observe the movement of data into and out of your process streams. A watch window also allows you to observe the values of specific variables in your C code during debugging.

How Does It Work?

To make the most effective use of the debugger, it's important to understand how your application is built for cycle-accurate debugging. These are the important steps:

1. Your application is compiled for hardware generation, resulting in a set of HDL source files and

related software test files (typically described as "producer" and "consumer" processes that write and read data on streams, respectively).

2. The hardware portion of the application (the process or processes to be tested) are converted from their HDL representation to a cycle-accurate, instrumented C-language model.
3. The generated C-language model is compiled (using the GCC compiler) and linked with your original software producer and consumer processes to create a hardware simulation executable.
4. The hardware simulation executable is launched and runs under the control of the Stage Master Debugger.

In support of these steps, it is important that you properly identify (in the CoDeveloper Application Manager) which files are associated with hardware (the "hw" icon) and which are associated with software (the "sw" icon). If you do not include the correct software and hardware source files in the "hw" and "sw" categories, the debugger will not be capable of loading and running the hardware simulation.

Note: you must have a complete application, including producer/consumer processes, in order to run a cycle-accurate debugging session.

More information about cycle-accurate debugging can be found in Tutorial 5: The CoDeveloper Pro Tools.

See Also

Stage Master Explorer
Tutorial 5: The CoDeveloper Pro Tools

1.6 CoValidator Testbench Generator

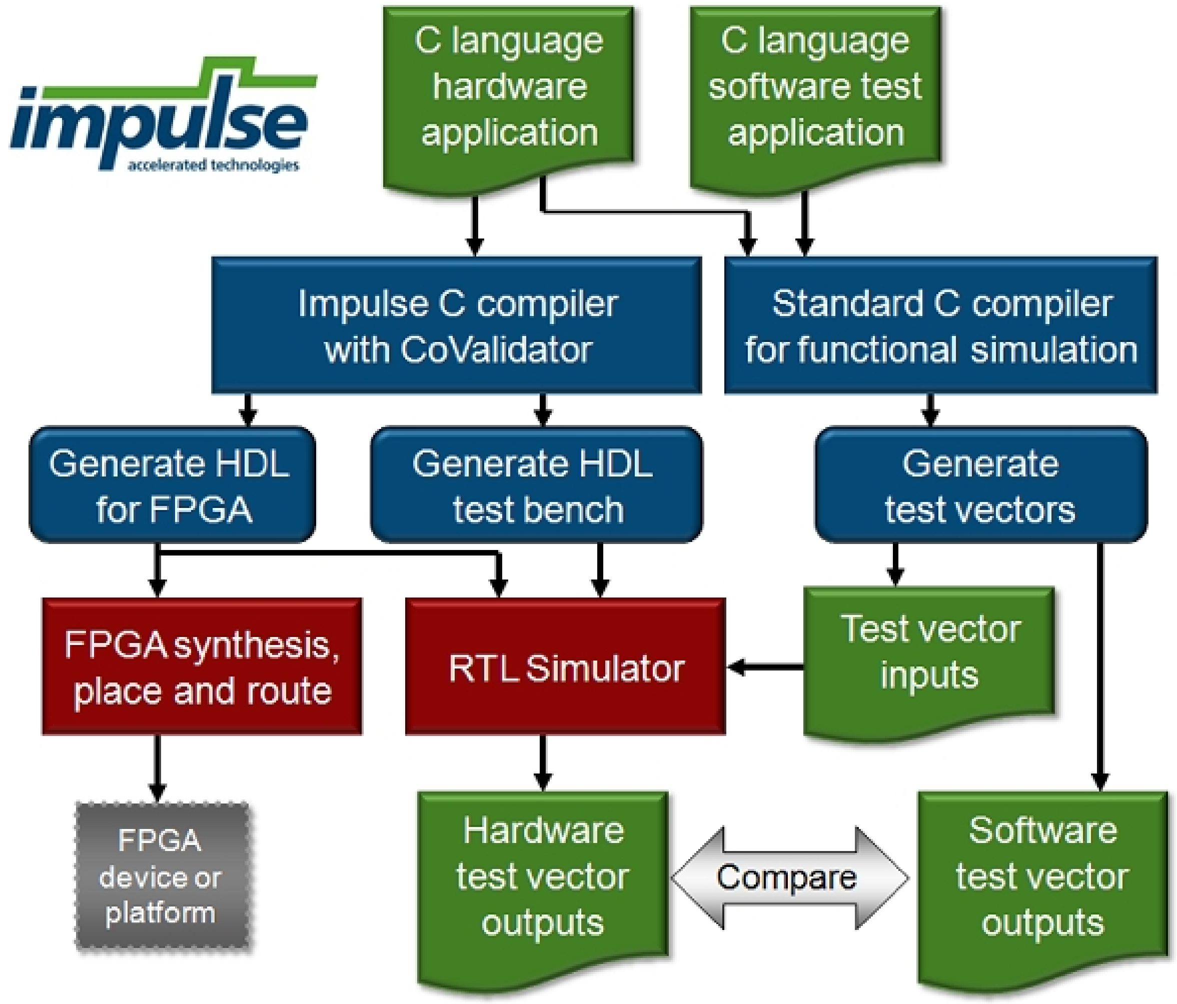
Generate HDL Testbenches from Impulse C

The CoValidator Testbench Generator provides a rapid path to verifying the output of the Impulse C hardware compiler. FPGA design teams commonly use HDL simulation tools to verify the correct behavior and desired performance of their hardware designs across a range of test scenarios. However, writing the "testbench", the HDL code that stimulates the various signals on the tested hardware module, can be nearly as time-consuming as creating the hardware module itself. CoValidator greatly speeds the path to HDL simulation by automatically generating a testbench for an Impulse C application.

CoValidator integrates with the complete Impulse C design flow. When it is enabled:

- Desktop simulation traces activity on the Impulse C I/O connections (streams, etc.) and records it to test vector files
- HDL generation produces an HDL testbench connected to the Impulse C hardware module
- Scripts for the supported simulators are created, so that HDL simulation can be run with a single command

When running the HDL simulation, the same inputs recorded during desktop simulation are fed into the testbench. The testbench then writes its own output test vectors to separate files, so they can be compared to the outputs from desktop simulation. Validating the behavior of the HDL generated by the Impulse C compiler is then as simple as comparing the two sets of outputs. Because the test vectors are generated during desktop simulation, creating new test scenarios is as easy as changing the C code in the application's software processes. This "closed-loop" testing practice is illustrated in the following diagram:


impulse accelerated technologies
C language hardware application
C language software test application
Impulse C compiler with CoValidator
Standard C compiler for functional simulation
Generate HDL for FPGA
Generate HDL test bench
Generate test vectors
FPGA synthesis, place and route
RTL Simulator
Test vector inputs
FPGA device or platform
Hardware test vector outputs
Compare
Software test vector outputs

CoValidator Features

- Generates test vectors and HDL testbench for stream (co_stream) interfaces only
- Supports an unlimited number of streams
- Streams may be of any data width
- VHDL output only
- Any VHDL Platform Support Package may be used
- Generated HDL testbench connects to the top-level Impulse C module (*_top.vhd)
- Generated HDL testbench compatible with any VHDL simulator
- Simulation project files and a script to launch HDL simulation are generated automatically for these supported HDL simulators:
  - Active-HDL
  - ModelSim

Prerequisites

CoValidator will not function without a license issued by Impulse or an authorized Impulse sales partner. Please contact info@ImpulseAccelerated.com to obtain a license.

To use the testbenches generated by CoValidator with a supported HDL simulator simulator, a licensed installation of Aldec Active-HDL™ or Mentor Graphics ModelSim® is required. Active-HDL and ModelSim are *not* included with Impulse CoDeveloper. Please contact Aldec or Mentor Graphics, your hardware platform vendor, or an authorized Aldec or ModelSim distributor for help obtaining software and licenses.

See Also

Using CoValidator (Linux)
Using CoValidator (Windows)

1.6.1 Using CoValidator (Linux)

Enable CoValidator
If you are creating a Makefile for your project using icProj2Make, first enable CoValidator by editing the .icProj file. Change the "SimulateGenTestbench" option's value to 1 (or add the option if it is not present), then run icProj2Make.pl.

Generate test vectors in desktop simulation
To generate input and output test vectors, first rebuild the desktop simulation to link with CoValidator's tracing library, libImpulseC_trace.so. This library is variation of the normal desktop simulation library (libImpulseC.so) that traces activity on Impulse C I/O objects to create test vector files.

If you used icProj2Make, the macro IMPULSE_LIB will be set to the value "ImpulseC_trace" in the file _Makefile.defs. Build the simulation as usual, with the "build_exe" Make target:

```
make -f _Makefile build_exe
```

If you are using your own Makefile, edit the linker options to link the desktop simulation executable with $IMPULSEC_HOME/Libraries/libImpulseC_trace.so.

Once the desktop simulation executable is built, simply run it as usual to generate test vector files. These will be created in your project directory, named after the I/O connections whose events they contain. The test vector files are overwritten every time a desktop simulation is run.

Generate HDL testbench
To generate the HDL testbench and HDL simulator scripts, generate HDL using the "build_testbench" Make target:

```
make -f _Makefile build_testbench
```

The testbench and related files will be created in a subdirectory, one each per supported HDL simulator, under your project directory. Every time you generate HDL, the contents of the subdirectory will be overwritten, so you may wish to move previously created testbenches aside for safekeeping.

See Also

Using CoValidator with Active-HDL
Using Covalidator with ModelSim
Command Line Tools
Make Integration

1.6.2 Using CoValidator (Windows)

CoValidator is integrated with desktop simulation and HDL generation in both the CoDeveloper Application Manager and the Impulse C plugin for Microsoft Visual Studio.

CoDeveloper Application Manager

Enable CoValidator
To generate an HDL testbench for your Impulse C application, you must enable CoValidator for the project. With your project open in the CoDeveloper Application Manager, open the Options dialog by selecting the Project > Options menu item, or clicking the Options toolbar button. Select the Simulate tab in the Options dialog.

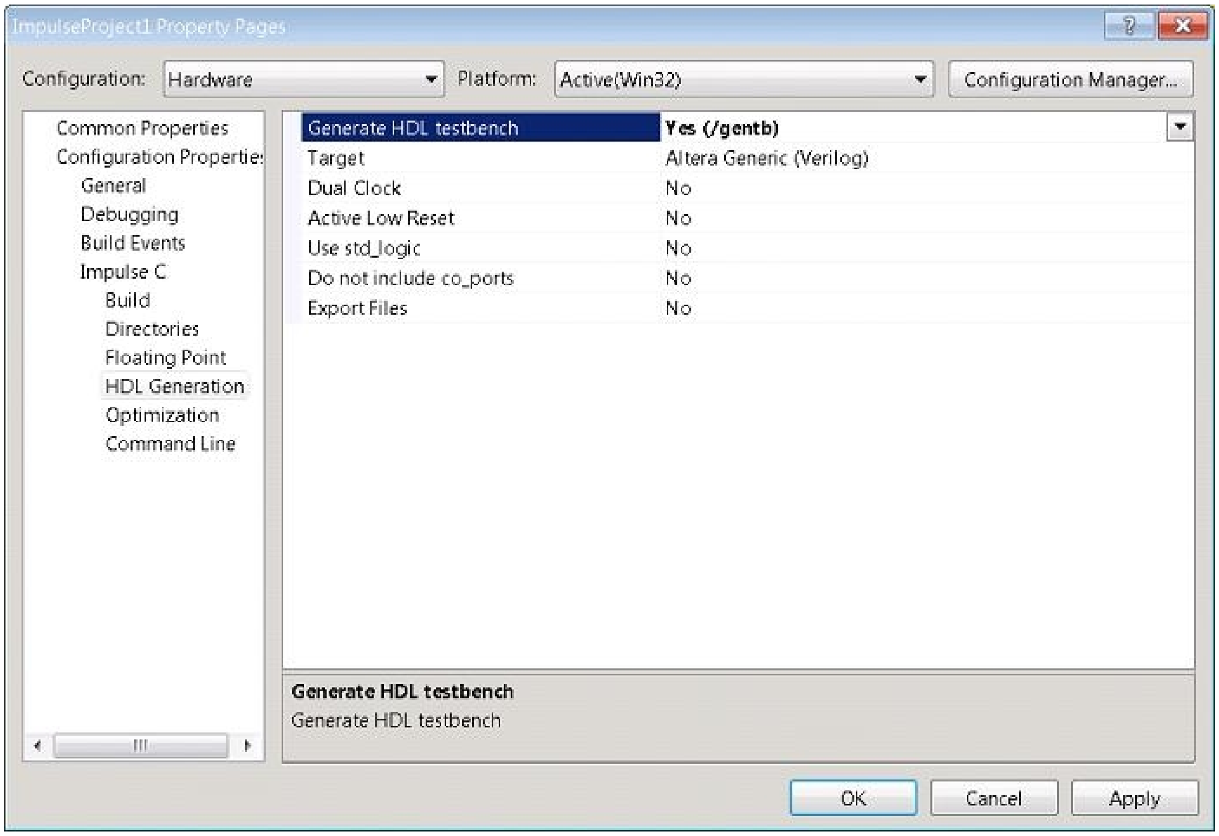
Enable CoValidator by checking the box marked "Generate HDL testbench". Click OK to save this setting and close the Options dialog.

Generate test vectors in desktop simulation
To generate input and output test vectors, simply rebuild and run desktop simulation, using the Build Software Simulation Executable and Launch Software Simulation Executable commands. Test vector files will be created in your project directory, named after the I/O connections whose events they contain. These test vector files are overwritten every time a desktop simulation is run.

Generate HDL testbench
To generate the HDL testbench and HDL simulator scripts, generate HDL using the Generate HDL command. The testbench and related files will be created in a subdirectory, one each per supported HDL simulator, under your project directory. Every time you generate HDL, the contents of the subdirectory will be overwritten, so you may wish to move previously created testbenches aside for safekeeping.

Microsoft Visual Studio Plugin

Enable CoValidator
To generate an HDL testbench for your Impulse C application, you must enable CoValidator for the project. With your project open in Visual Studio, open the Property Pages dialog by selecting the Project > Properties menu item. Make sure the "Hardware" Configuration is selected from the drop-down list. From the tree list on the left, select Configuration Properties > Impulse C > HDL Generation. The following property page will appear:



Generate HDL testbench	Yes (/gentb)
Target	Altera Generic (Verilog)
Dual Clock	No
Active Low Reset	No
Use std_logic	No
Do not include co_ports	No
Export Files	No

Generate HDL testbench
Generate HDL testbench

Enable CoValidator by selecting "Yes (/gentb)" for the option named "Generate HDL testbench". Click OK to save this setting and close the dialog.

Generate test vectors in desktop simulation
To generate input and output test vectors, first rebuild desktop simulation using either of the "Release CoValidator" or "Debug CoValidator" configurations. These build configurations link your compiled desktop simulation code with CoValidator's tracing library, ImpulseC_trace.lib.

After rebuilding the desktop simulation, run the generated executable using either the Debug > Start Debugging or Debug > Start Without Debugging menu commands. As the simulation runs, test vector files will be created in your project directory, named after the I/O connections whose events they contain. These test vector files are overwritten every time a desktop simulation is run.

Generate HDL testbench
To generate the HDL testbench and HDL simulator scripts, rebuild your project using the "Hardware" configuration. The testbench and related files will be created in a subdirectory, one each per supported HDL simulator, under your project directory. Every time you generate HDL, the contents of the subdirectory will be overwritten, so you may wish to move previously created testbenches aside for safekeeping.

See Also

Using CoValidator with Active-HDL
Using Covalidator with ModelSim
Command Line Operation
Make Integration

1.6.3 Using CoValidator with Active-HDL

CoValidator generates a ready-to-run set of Active-HDL scripts that will create a workspace and design files for the generated HDL test bench within the subdirectory named "Active-HDL".

See Also

Launching Active-HDL (Windows)
Active-HDL Commands

1.6.3.1 Launching Active-HDL (Windows)

Launching Active-HDL

After generating the HDL testbench from CoDeveloper Application Manager or Visual Studio, navigate to your project's "Active-HDL" directory and execute the "active-hdl.bat" script by double-clicking the file or running the script in a Command Prompt. If Active-HDL is installed on your development system, it will start and open a workspace and design that includes all the HDL files necessary to simulate your Impulse C hardware module using the generated testbench.

From here, you may use CoValidator shortcut commands and all the features of Active-HDL to examine the behavior of the Impulse C hardware module over time.

See Also

Active-HDL Commands

1.6.3.2 Active-HDL Commands

Several Tcl commands are predefined in the Active-HDL design generated by CoValidator and are immediately available from the Console window which is left in Tcl mode by default. These commands are shortcuts for compiling and starting the simulation. Enter any of these at the **>** prompt:

Command	**Description**
r	Recompile changed and dependent files and start simulation (same as executing 'compile.tcl')
rr	Recompile everything and start simulation
q	Quit without confirmation

For a complete listing of Active-HDL commands, please consult the Active-HDL documentation.

See Also

Launching Active-HDL (Windows)
Using CoValidator (Windows)

1.6.4 Using Covalidator with ModelSim

CoValidator generates a ready-to-run set of ModelSim scripts and project files for the generated HDL test bench within the subdirectory named "ModelSim".

See Also

Launching ModelSim (Linux)
Launching ModelSim (Windows)
ModelSim Commands

1.6.4.1 Launching ModelSim (Windows)

Launching ModelSim

After generating the HDL testbench from CoDeveloper Application Manager or Visual Studio, navigate to your project's "ModelSim" directory and execute the "modelsim.bat" script by double-clicking the file or running the script in a Command Prompt. If ModelSim is installed on your development system, it will start and open a project that includes all the HDL files necessary to simulate your Impulse C hardware module using the generated testbench.

From here, you may use CoValidator shortcut commands and all the features of ModelSim to examine the behavior of the Impulse C hardware module over time.

See Also

ModelSim Commands
Using CoValidator (Windows)

1.6.4.2 Launching ModelSim (Linux)

Launching ModelSim

After generating the HDL testbench, navigate to your project's "ModelSim" directory and execute the "modelsim.sh" script. If ModelSim is installed on your development system, it will start and open a project that includes all the HDL files necessary to simulate your Impulse C hardware module using the generated testbench.

From here, you may use CoValidator shortcut commands and all the features of ModelSim to examine the behavior of the Impulse C hardware module over time.

See Also

ModelSim Commands
Using CoValidator (Linux)

1.6.4.3 ModelSim Commands

Several Tcl commands are predefined in the ModelSim project generated by CoValidator. These commands are shortcuts for compiling and starting the simulation. Enter any of these at the `ModelSim>` or `VSIM>` prompt:

Command	Description
r	Recompile changed and dependent files, and enter simulation
rr	Recompile all files
q	Quit without confirmation

For a complete listing of ModelSim commands, please consult the ModelSim documentation.

See Also

Launching ModelSim (Linux)
Launching ModelSim (Windows)

1.7 Command Line Tools

The Impulse C development process can be driven from the command line, using a Linux shell or the Windows Command Prompt.

Linux Command Line Build Process

CoDeveloper for Linux integrates with GNU tools to compile desktop simulations and generate hardware for an Impulse C application.

Prerequisites

See the section "Before You Begin" for system requirements and installation notes.

Step 1: Generate Makefiles from Project File

Run the icProj2Make.pl program to generate Makefiles from an Impulse C project (*.icProj) file. Specifically, the files _Makefile and _Makefile.defs will be created in the current directory when icProj2Make.pl is run. These project-specific Makefiles rely on a system-wide Makefile, $IMPULSEC_HOME/MakeRules/Makefile.rules, for rules on compiling desktop simulations and generating HDL using the Impulse C compiler.

Example:

```
$ icProj2make.pl HelloWorld.icProj
$ ls
HelloWorld.c     HelloWorld.icProj  _Makefile       ReadmeFiles/
HelloWorld_hw.c  HelloWorld_sw.c    _Makefile.defs  Readme.htm
```

For help on using icProj2Make.pl, run the command without any arguments:

```
$ icProj2make.pl
Usage: perl /usr/Impulse/CoDeveloper3/bin/icProj2make.pl [Impulse C project file]
Generate Makefiles from an Impulse C project.
Make targets supported: clean, build_exe, build, export_hardware, export_software
```

Impulse C project files (*.icProj) are human-readable and may be edited by hand to change options to the Impulse C compiler flow. For more details:

- Project options: "Build Options", "Generate Options"
- Generated Makefile format (in particular, _Makefile.defs): "Make Integration"

- Compiler command line options: "CoBuilder Command Line Tools"

Step 2: Compile Desktop Simulation (build_exe)

Run make, using the generated _Makefile, to generate a desktop simulation executable for the project, linked with the Impulse C simulation library. The build target to use is "build_exe".

Example:

```
$ make -f _Makefile build_exe
gcc -g "-I/usr/Impulse/CoDeveloper3/Include"  -o HelloWorld_hw.o -c
HelloWorld_hw.c
gcc -g "-I/usr/Impulse/CoDeveloper3/Include"  -o HelloWorld.o -c HelloWorld.c
gcc -g "-I/usr/Impulse/CoDeveloper3/Include"  -o HelloWorld_sw.o -c
HelloWorld_sw.c
gcc -g HelloWorld_hw.o HelloWorld.o HelloWorld_sw.o -
L/usr/Impulse/CoDeveloper3/Libraries -lImpulseC  -o HelloWorld.exe
```

Step 3: Generate Hardware (build)

Run make on the "build" target, using the generated _Makefile, to generate HDL and hardware/software interfaces.

Example:

```
$ make -f _Makefile build
mkdir hw
HW_SRC="HelloWorld_hw.c"; \
        for i in $HW_SRC; do CPP_INCLUDES="$CPP_INCLUDES -include $i"; done; \
        echo | gcc -E -DIMPULSE_C_SYNTHESIS -Dsize_t="long unsigned int" -DNULL=0
-Dstdin=NULL -DO_RDONLY=00 -DO_WRONLY=01 -Dpthread_t=int -D_FCNTL_H -D_PTHREAD_H -
D_STDARG_H -D_STDIO_H "-I/usr/Impulse/CoDeveloper3/Include" $CPP_INCLUDES - >
HelloWorld.i
"/usr/Impulse/CoDeveloper3/bin/impulse_snoot" -Timpulse-c HelloWorld.i
HelloWorld.snt
"/usr/Impulse/CoDeveloper3/bin/impulse_prep" HelloWorld.snt HelloWorld.pk0
Impulse C Transformations
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Build Jan 28 2008.
processing HelloWorld.snt...
...
```

Step 4 (optional): Generate HDL Testbench (build_testbench)

To generate an HDL testbench using CoValidator, along with files suitable for running the with supported HDL simulators, run make on the "build_testbench" target.

Step 5: Export (export_hardware, export_software)

Run make on the "export_hardware" and "export_software" targets to export the compiler output for use in third-party tools for bitfile generation (i.e., Xilinx ISE) and host processor compilation (i.e., Altera Nios II IDE).

Windows Command Line Build Process

In Windows, the CoDeveloper Application Manager IDE can be driven from the command line to compile a desktop simulation, generate HDL, or perform other tasks normally available from the graphical development environment. See the section "Command Line Operation".

Once a build command (e.g., Project > Generate HDL) has been run in the Application Manager, Makefiles will be present in the project's directory. These Makefiles may be used to perform further build tasks using the GNU make and other utilities included with CoDeveloper. See the above section on Linux for details on using make.

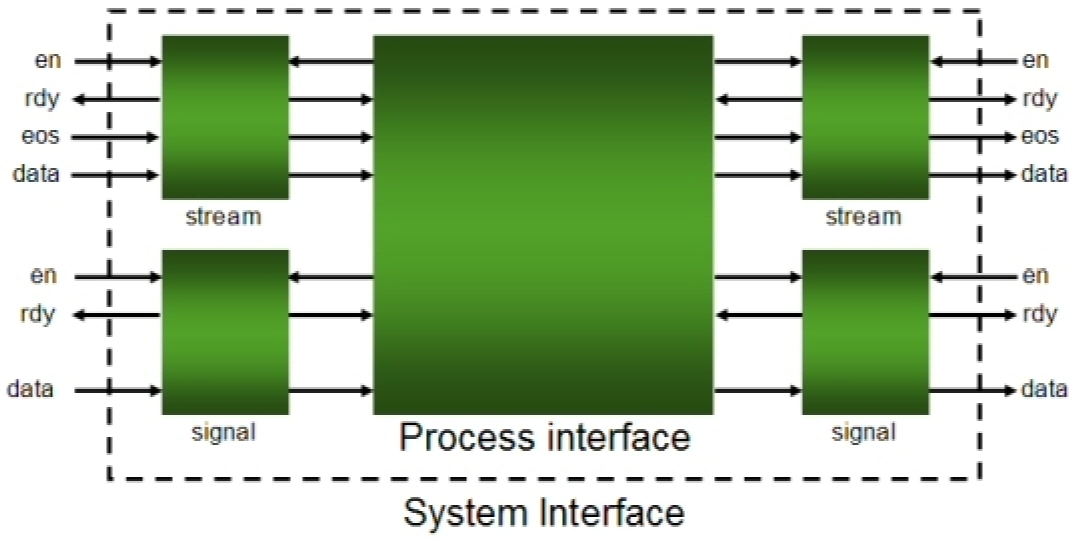
1.8 Impulse C Input/Output Interfaces

Description

Impulse C provides standardized communication channels that simplify the creation of highly parallel, multi-process applications, as well as providing a means for connecting Impulse C hardware blocks (modules) to other parts of a larger system.

The three primary methods of communication are streams, signals, and memories. This section describes the stream and signal interfaces; memory interfaces are specific to each supported platform and are described in your Platform Support Package documentation.

The following diagram summarizes the interfaces to streams and signals:



As the diagram shows, Impulse C processes are connected to other elements in the system (whether hardware or software) via buffered components that represent stream and signal interfaces. Lower-level, more direct connections are also possible using registers.

See Also

System Stream Interfaces
Process Stream Interfaces
Understanding Signal Interfaces
Understanding Register Interfaces

1.8.1 System Stream Interfaces

Overview

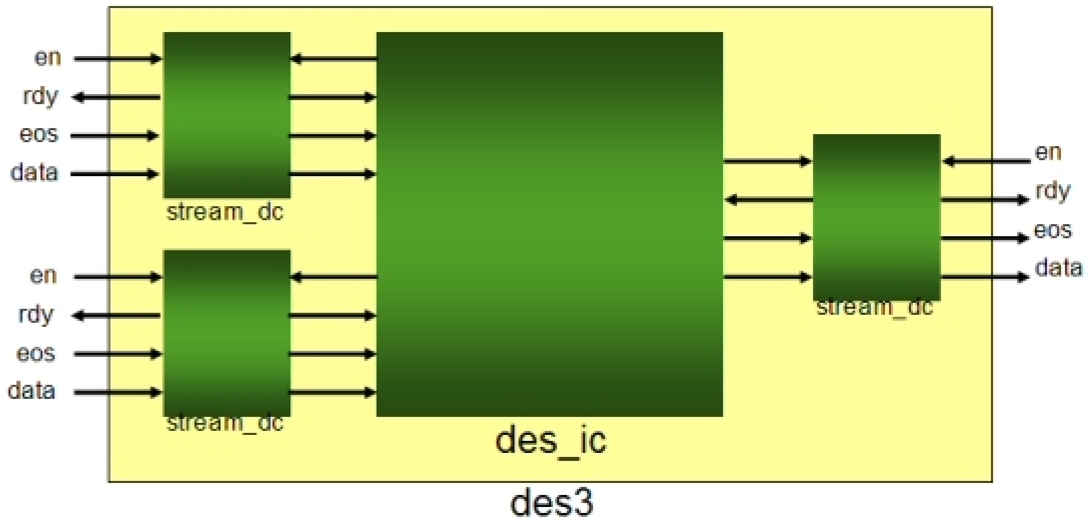
The top-level HDL file generated by CoBuilder (which is normally assigned a name ending with "_top.vhd") includes an entity declaration similar to the following:

```
entity des3 is
  port (
    reset, sclk, clk : in std_ulogic;
    p_des_producer_config_out_encrypt_en : in std_ulogic;
    p_des_producer_config_out_encrypt_eos : in std_ulogic;
    p_des_producer_config_out_encrypt_data : in std_ulogic_vector (31 downto 0);
    p_des_producer_config_out_encrypt_rdy : out std_ulogic;
    p_des_producer_blocks_out_en : in std_ulogic;
    p_des_producer_blocks_out_eos : in std_ulogic;
    p_des_producer_blocks_out_data : in std_ulogic_vector (7 downto 0);
    p_des_producer_blocks_out_rdy : out std_ulogic;
    p_des_decrypt_ic_blocks_in_en : in std_ulogic;
    p_des_decrypt_ic_blocks_in_data : out std_ulogic_vector (7 downto 0);
    p_des_decrypt_ic_blocks_in_eos : out std_ulogic;
    p_des_decrypt_ic_blocks_in_rdy : out std_ulogic);
end;
```

This entity declaration is similar to the entity declaration generated for the lower-level Impulse C process, but with a few important differences:

1. The stream interfaces have different names. The names used are based on the source and destination of the stream connections, rather than on the actual process port names.
2. The _**rdy** ports' modes are reversed: the input streams specify a mode **out** for _**rdy**, while the output stream specifies mode **in** for _**rdy**.
3. There are no generics present in the top-level entity corresponding to any **co_parameter** arguments to the underlying hardware processes.

These differences reflect that fact that the top-level interface (**des3**) describes the opposite (or outermost) ends of the data streams, as illustrated below:



The _**en** and _**rdy** signals are used to enable stream read/write operations and to determine if a stream is ready for such operations, respectively.

The **stream** component (or **stream_dc** in the case of a dual-clock stream) shown in the diagram buffers the stream data, using a FIFO whose size is determined by the arguments used in the function **co_stream_create**.

*Note the the **stream_dc** component is obtained from the Impulse C architecture library, and is a dual-clock FIFO interface. Dual-clock FIFOs are generated by CoBuilder when the "Generate dual clocks" option is selected for hardware generation. Signals connecting to the instantiated **des_ic** module are synchronized to the **clk** whereas the external (system-level) ports shown above are synchronized to **sclk**. The **stream_dc** module handles the transfer of data between the two clock domains.*

Top-level Port Naming Conventions

The names used in the entity declaration are generated as a direct result of the call to **co_process_create** made in the configuration function for the application, as shown below:

```
des_encrypt_ic_process = co_process_create("des_encrypt_ic", (co_function)des_ic,
                                            4,
                                            config_encrypt_ic,
                                            blocks_plaintext_ic,
                                            blocks_encrypted_ic,
                                            DES_ENCRYPT);
```

The actual top-level signals (ports) are named according to the source or destination of the associated stream, depending on its direction. For example, the input stream declared as **config_encrypt_ic** has as its source the stream **config_out_encrypt** of process **des_producer**. As a result, the generated port names for the configuration data input stream at the top level are prefixed with **p_des_producer_config_out_encrypt_**. Similarly, the **blocks_encrypted_ic** stream declared above is connected (at the system level) to the input stream of the decrypter process, resulting in the top-level port names being assigned a prefix of **p_des_decrypt_ic_blocks_in_**.

See Also

Process Stream Interfaces
Reading and Writing Data from Streams

1.8.1.1 Reading and Writing Data from Streams

System Stream Interface Ports

The four ports that are generated at the system level (in the top-level HDL file) are summarized below:

Input Streams

Port Name	**Description**
xxx_rdy	Stream is ready to accept data
xxx_en	Enable stream write
xxx_eos	Close the stream
xxx_data	Data to be pushed onto the stream input buffer

Output Streams

Port Name	Description
xxx_rdy	Data is ready to be read from the stream
xxx_en	Enable stream read
xxx_eos	Stream is closed
xxx_data	Data on the stream output buffer

Writing Data to an Output Stream

To write data to a stream (which is in turn connected to a lower-level process), your test bench or other hardware interface must do the following:

1. The **_eos** and **_en** signals must be set low.
2. Assign the data to be written to the **_data** signal and set the **_en** signal high.
3. Wait until **_rdy** is high, indicating that the stream can accept data.
4. On the next rising clock edge, the data will be written.
5. Set **_en** low until data for the next write operation will be ready next cycle; hold **_en** high if the next data will be ready immediately.
6. Go to step 2.

The following loop in VHDL demonstrates how this sequence can be used when reading test inputs from a file. This loop will write a new value every two clock cycles:

```
-- output the characters to be encrypted
p_des_producer_blocks_out_en <= '0';
while (not endfile(text_file)) loop
    wait until p_des_producer_blocks_out_rdy = '1';
    read(text_file, ch);  -- Get character from file
    byte := conv_std_logic_vector(character'pos(ch),8);
    p_des_producer_blocks_out_data <= std_ulogic_vector(byte);
    p_des_producer_blocks_out_en <= '1';
    wait until rising_edge(clk);
    p_des_producer_blocks_out_en <= '0';
end loop;
wait until p_des_producer_blocks_out_rdy = '1';
p_des_producer_blocks_out_eos <= '1';
p_des_producer_blocks_out_en <= '1';
wait until rising_edge(clk);
p_des_producer_blocks_out_en <= '0';
```

To write a new value every cycle, assuming the reading process is ready to accept it, keep **_en** high. The following VHDL process illustrates single-cycle input, using an incrementing integer value as the data:

```
writer_proc : process
      variable data_in : integer;
begin
      -- init values
      stream_in_en <= '0';
      stream_in_eos <= '0';
      stream_in_data <= (others => '0');
      data_in := 0;

      -- wait for reset
      wait until reset = '0';

      -- write data
      stream_in_en <= '1';
      while ( data_in /= NUM_TO_SEND ) loop
```

```
                    stream_in_data <=
std_ulogic_vector(conv_std_logic_vector(data_in,DATA_WIDTH));
                    wait until rising_edge(sclk);
                    if ( stream_in_rdy = '1' ) then
                          data_in := data_in + 1;
                    end if;
              end loop;
              stream_in_en <= '0';
              wait;  -- wait forever (don't close the stream)
       end process;
```

Closing an Output Stream

When you have finished writing data to the stream, you may close the stream by performing a write operation as usual, but instead setting the **_eos** signal high. The **_data** signal is ignored when closing a stream. On the next rising clock edge after **_rdy** is high, the stream will be closed and **_en** should be set low.

Reading Data from an Input Stream

Reading from a buffered data stream is similar to the above, but uses the following sequence:

1. Wait for the **_rdy** signal to go high, indicating there is data to be read from the stream.
2. Check for an EOS (end-of-stream) condition. The **_eos** signal will be high if the writing process has closed the stream.
3. Observe (or make use of) the data appearing on signal **_data** then set **_en** high.
4. On the next rising clock edge, the current data will be removed from the stream buffer.
5. Set **_en** low until you want to read data again; hold **_en** high to consume the next data immediately.

The following loop demonstrates how this sequence can be used to read data from a stream:

```
  p_stream_blocks_out_en <= '0';
  while p_stream_blocks_out_eos != '1' loop
      wait until p_stream_blocks_out_rdy = '1';  -- Wait until data is available
      ch := character'value(conv_integer(unsigned(p_stream_blocks_out_data)));
      write(text_file, ch);
      p_stream_blocks_out_en <= '1';
      wait until rising_edge(clk);
      p_stream_blocks_out_en <= '0';
  end loop;
```

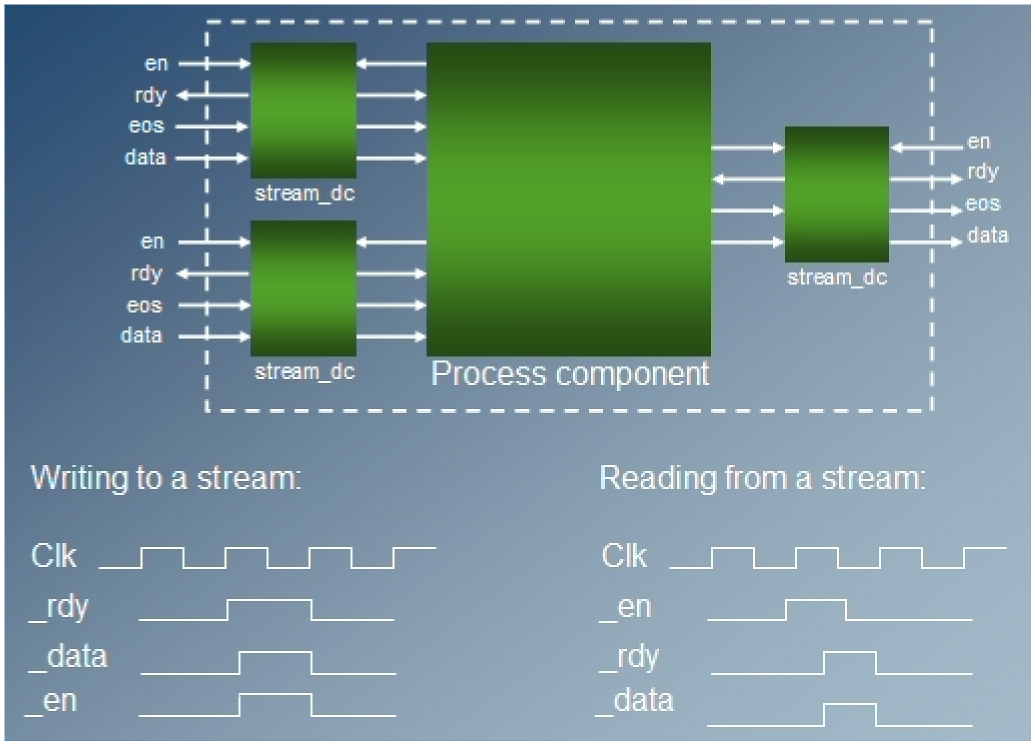
The following VHDL process illustrates single-cycle reads:

```
       reader_proc : process
              variable data_out : std_ulogic_vector (DATA_WIDTH-1 downto 0);
       begin
              -- init values
              stream_out_en       <= '0';

              -- wait for reset
              wait until reset = '0';

              -- just let data flow out capturing it into data_out
              -- EOS is not checked explicitly here
              stream_out_en       <= '1';
              while ( true ) loop  -- loop forever
                    wait until rising_edge(clk);
                    if ( stream_out_rdy = '1' ) then
                          data_out := stream_out_data;
                    else
                          data_out := (others => 'X');
```

```
                end if;
            end loop;
            wait;  -- wait forever
        end process;
```

Timing Diagram

The following diagram illustrates the stream read and write protocols:



See Also

System Stream Interfaces
Process Stream Interfaces

1.8.2 Process Stream Interfaces

Overview

The lower-level HDL file generated by CoBuilder (which is normally assigned a name ending with "_comp.vhd") includes an entity declaration similar to the following:

```
entity des_ic is
  generic (
    g_decrypt_param : std_ulogic_vector (31 downto 0)
  );
  port (
    reset, sclk, clk : std_ulogic;
    p_config_in_rdy : in std_ulogic;
    p_config_in_en : inout std_ulogic;
    p_config_in_eos : in std_ulogic;
    p_config_in_data : in std_ulogic_vector (31 downto 0);
    p_blocks_in_rdy : in std_ulogic;
    p_blocks_in_en : inout std_ulogic;
    p_blocks_in_eos : in std_ulogic;
    p_blocks_in_data : in std_ulogic_vector (7 downto 0);
    p_blocks_out_rdy : in std_ulogic;
    p_blocks_out_en : inout std_ulogic;
    p_blocks_out_eos : out std_ulogic;
    p_blocks_out_data : out std_ulogic_vector (7 downto 0)
  );
end;
```

This declaration is generated as a direct result of an Impulse C process that was previously declared as:

```
void des_ic(co_stream config_in, co_stream blocks_in, co_stream blocks_out,
co_parameter decrypt_param)
```

In particular, notice that the three streams (**config_in**, **blocks_in** and **blocks_out**) each result in the creation of four ports. These four ports are summarized below:

Input Streams

Port Name	**Description**
xxx_rdy	The stream is ready to write data
xxx_en	Enable stream write to process
xxx_eos	Stream is closed
xxx_data	Data to be read by the process from the stream

Output Streams

Port Name	**Description**
xxx_rdy	The stream is ready to accept data
xxx_en	Enable stream read from process
xxx_eos	Stream is closed
xxx_data	Data written by the process to the stream

These four signals (which have different behaviors for input and output streams) form the complete interface for one stream of an Impulse C process, as observed at the interface to the process itself.

Note that all streams of a given process are synchronized to a common clock, which appears as port **clk** in the generated entity. (The external system clock, **sclk**, is generated as a result of the "Generate dual clocks" option specified in the Generate Options dialog and is not used by the stream interface.)

See Also

System Stream Interfaces
Reading and Writing Data from Processes

1.8.2.1 Reading and Writing Data from Processes

Writing Data to a Process

While it is generally recommended that you interface Impulse C processes to other parts of the system using the stream interfaces generated by CoBuilder, it is also possible to make more direct connections, if you follow the correct sequence of operations when activating the **_rdy** and **_data** signals, and observe the correct protocols regarding the **_en** signal.

To write data to a process (emulating the behavior of a buffered stream), your test bench or other hardware interface must do the following:

1. The **_eos** signal must be held low.
2. The data to be written to the process must be assigned to the **_data** input.
3. The **_rdy** signal must be set high to indicate that data is available on input **_data**.
4. Hold the values of **_rdy** and **_data** until a rising edge of **clk** *and* **_en** is high.

The following loop of VHDL code demonstrates how this sequence can be used when reading test inputs from a file (in this case a file of characters):

```
p_blocks_in_eos <= '0';
read(text_file, ch);  -- Get first character from the file
while true loop
    if endfile(text_file) then
        exit;
    end if;
    read(text_file, ch);  -- Get next character from the file
    byte := conv_std_logic_vector(character'pos(ch),8);
    p_blocks_in_data <= std_ulogic_vector(byte);
    p_blocks_in_rdy <= '1';
    wait until rising_edge(clk) and p_blocks_in_en = '1';  -- Wait until accepted
    p_blocks_in_rdy <= '0';
end loop;
p_blocks_in_eos <= '1';
```

Note that in the case of a test bench, you can simply set the **_rdy** signal to high once, then apply the relevent data at each clock cycle. If, however, you are interfacing to an external data generator or system and cannot guarantee that the data will be available at each cycle, then you should hold the **_rdy** signal low until the data is ready and has been assigned to the **_data** input as shown.

When you have finished writing data to the stream, you may close the stream by setting the **_eos** signal high as shown.

Reading Data from a Process

Reading from a process interface (emulating the process interface of a buffered stream) is similar to the above but uses the following sequence:

1. Set **_rdy** high to indicate readiness to accept data from the process.
2. Wait for a rising edge on the clock (**clk**) when **_en** is high.
3. Sample the data appearing on signal **_data**, and check the value of **_eos**.

The following loop demonstrates how this sequence can be used to read data directly from a process:

```
p_blocks_out_rdy <= '1';
while true loop
    wait until rising_edge(clk) and p_blocks_out_en = '1';  -- Wait until data is
available
    if (p_blocks_out_eos = '1') then
        exit;
    end if;
    ch := character'value(conv_integer(unsigned(p_blocks_out_data)));
    write(text_file, ch);
end loop;
p_blocks_out_rdy <= '0';
```

Note that if you simply wish to view the results of a process during simulation (for example, as a waveform), you can hold the **_rdy** signal high during the simulation run (as shown above) and observe the data appearing on **_data** directly.

See Also

Process Stream Interfaces

1.8.3 Understanding Signal Interfaces

Overview

Signals are used to communicate status information from one process to another. The mechanism is simple, and makes use of a **co_signal_post** function call in the sending process, and a **co_signal_wait** function call in the receiving process as shown below:

Process 1:

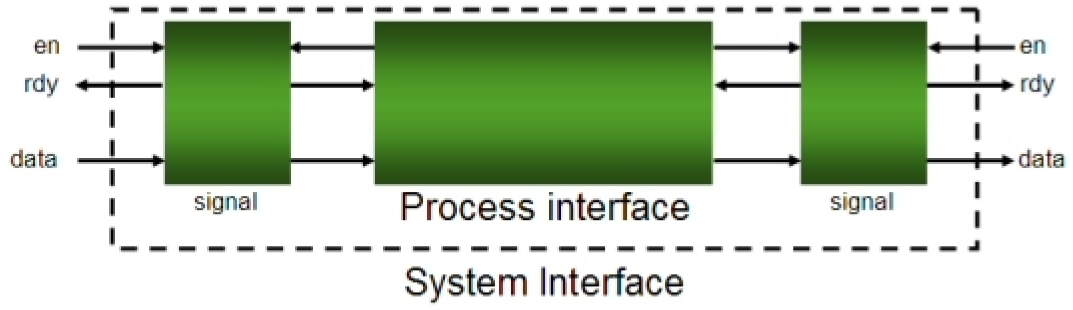
```
co_signal_post(start, value);  // Post a signal to process 2
co_signal_wait(end, &data);    // Wait for process 2 to return a signal
```

Process 2:

```
co_signal_wait(start, &data);   // Wait for the signal from process 1
// Do some thing here, such as accessing shared memory
co_signal_post(end, 1);         // Post a signal to process 1
```

As shown above, signals are often used to coordinate the operation of multiple processes that must access the same shared resource (such as a shared memory). Signals may also be used to pass values, using the second arguments to the **co_signal_post** and **co_signal_wait** functions.

The Signal Interface



As shown in the diagram above, signal interfaces include a handshaking protocol using the en and rdy ports that are associated with each signal. Data is passed on the signals using the data line, which is a 32-bit value. The generated system interface for signals appears as follows (as viewed in the top-level VHDL file):

```
p_go_en : inout std_ulogic;
p_go_rdy : in std_ulogic;
p_go_data : in std_ulogic_vector (31 downto 0);
```

In this case a signal with the name of "go" results in the generation of one data signal and three control signals.

To write data to a signal (emulating a **co_signal_post** operation) your VHDL interface must apply data to the **data** input, then wait for the **rdy** signal to be asserted high. When **rdy** is high, your interface applies signal to the **en** input indicating that data is available on the **data** line.

To read data from a signal (emulating a **co_signal_wait** operation) your VHDL interface will assert a high value to the **en** input, then wait for the **rdy** signal to be asserted high. When **rdy** is high, the signal data is available on the **data** line.

See Also

System Stream Interfaces
Process Stream Interfaces
Understanding Register Interfaces

1.8.4 Understanding Register Interfaces

Overview

Registers are used to communicate information in an unbuffered manner from one process to another, or to and from Impulse C processes and external hardware interfaces. When used in conjuction with co_port types, registers can provide a convenient method for creating direct, named connections for sending and receiving data.

Using co_register and co_port for hardware interfaces

The co_register Impulse C primitive is a non-blocking interface you can use, in conjunction with co_port, to create direct connections to external HDL components, as demonstrated in the following simple example:

```
#include "co.h"

void uc_process(co_register nwe, co_register addr, co_register data)
{
    co_uint1 nwe_r = 1;
    co_int16 addr_r = 0;
    co_int16 x;

    while ( addr_r != 0x0010 && nwe_r != 0 ) {
        co_register_read(nwe, &nwe_r, sizeof(co_uint1));
        co_register_read(addr, &addr_r, sizeof(co_int16));
    }
    co_register_read(data, &x, sizeof(co_int16));
}

void config(void * arg)
{
    co_register nwe, addr, data;
    co_port nwe_port, addr_port, data_port;
    co_process uc;

    nwe = co_register_create("nwe_reg", UINT_TYPE(1));
    addr = co_register_create("addr_reg", INT_TYPE(16));
    data = co_register_create("data_reg", INT_TYPE(16));

    nwe_port = co_port_create("nwe", co_input, nwe);
    addr_port = co_port_create("addr", co_input, addr);
    data_port = co_port_create("data", co_input, data);
    uc = co_process_create("uc", (co_function)uc_process, 3, nwe, addr, data);

    co_process_config(uc, co_loc, "pe0");
}

co_architecture co_initialize(void * arg)
{
    return co_architecture_create("uc_arch", "generic_vhdl", config, arg);
}
```

The above Impulse C program will generate an HDL component with the following ports, which you can then connect to an external hardware component device, or directly to FPGA pins:

```
entity uc_arch is
    port (
        reset : in std_ulogic;
        sclk : in std_ulogic;
        clk : in std_ulogic;
        nwe_en : in std_ulogic;
        nwe_data : in std_ulogic_vector (0 to 0);
        addr_en : in std_ulogic;
        addr_data : in std_ulogic_vector (15 downto 0);
        data_en : in std_ulogic;
        data_data : in std_ulogic_vector (15 downto 0));
end;
```

See Also

Creating and Using Registers
System Stream Interfaces
Process Stream Interfaces
Understanding Signal Interfaces

1.9 Platform Support Package Overview

Overview

Platform Support Packages (or *PSP*s) are optional packages that provide platform-specific libraries, examples and documentation for Impulse C users. Platform support packages simplify the creation of mixed software/hardware applications by providing you with the necessary hardware/software interfaces for both the hardware (FPGA) and software (microprocessor) elements of the platform.

What is a Platform?

A platform is a specific combination of hardware and software components that can be used to run an Impulse C application. A platform may be a board-level system that includes one or more external processors combined with an off-the-shelf FPGA, or it may be a single-chip system that utilizes an embedded (or "soft") processor within a larger FPGA. A given platform may also be defined by the type of bus it uses to connect the processor and FPGA, the type(s) of memories available in the system and other factors such as whether an operating system will be involved. A typical platform is defined by:

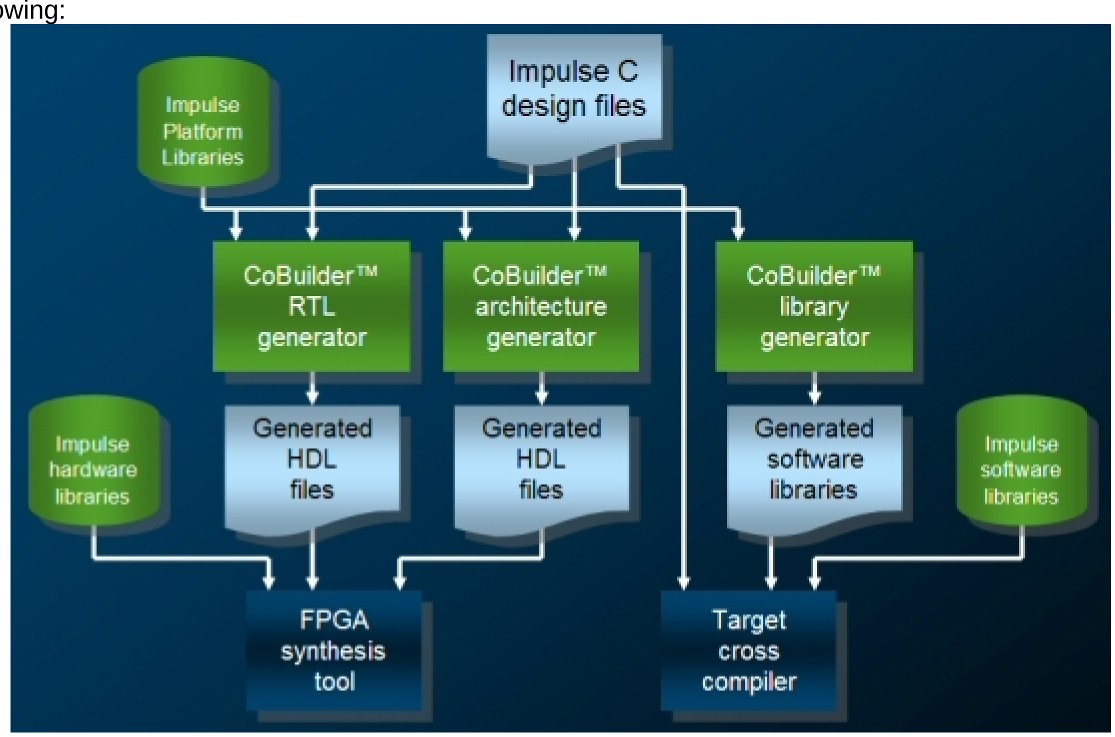
- The family of FPGA being used to implemented hardware processes
- The type of embedded microprocessor being used to implement software processes
- The type of bus used to communicate data in the hardware/software system

- The type and number of memory resources available
- The operating system (if any) that will run on the embedded processor
- Platform-specific design tools (such as EDA or embedded development tools) that may be required

In essence, any factor in the target hardware or software environment that influences how data is moved between the target processor and the target FPGA will be a factor in defining a platform target for Impulse C applications.

CoBuilder Processing Flow

When you process an Impulse C application using platform information provided in the selected Platform Support Package, the CoBuilder tools process your design using a flow similar to the following:


Impulse Platform Libraries
Impulse C design files
CoBuilder™ RTL generator
CoBuilder™ architecture generator
CoBuilder™ library generator
Impulse hardware libraries
Generated HDL files
Generated HDL files
Generated software libraries
Impulse software libraries
FPGA synthesis tool
Target cross compiler

Depending on the nature of your application, each process will be assigned either to a hardware or software resource and the necessary hardware/software interfaces will be generated for use with the target hardware synthesis and software compilation tools.

The **RTL Generator** reads your application source file(s), compiles those processes that you have identified as hardware and creates equivalent hardware descriptions in the form of HDL files.

The **Architecture Generator** also reads your application source file(s) and generates hardware "wrappers" implementing the necessary stream, signal and memory interfaces allowing hardware processes to communicate with other parts of the application.

The **Library Generator** creates software library elements needed for compilation of Impulse C software processes on the target embedded processor.

The result of processing is a set of files (representing both hardware and software) ready for use with your target platform development software and hardware.

Specific information about this design flow including documentation, tutorials and examples, are supplied with each Platform Support package.

See Also

About Platform Support Packages
Hardware/Software Interfaces
Platform Architecture Files
CoBuilder User Guide

1.9.1 About Platform Support Packages

Description

A Platform Support Package for a given platform includes some or all of the following:

- A set of CoBuilder architecture files (in XML format) describing the target platform
- A set of target-specific runtime libraries and related header files that are linked with your Impulse C software application
- A collection of hardware components (supplied as VHDL and/or Verilog libraries) in support of software/hardware interfaces
- One or more platform-specific design examples
- Platform-specific EDA interface tools and libraries
- Platform-specific documentation

Platform Factors

The primary purpose of a Platform Support Package is to simplify the creation of Impulse C applications that must communicate data (via streams, signals and shared memories) between an Impulse C application running as software on a microprocessor (which may be external to the target FPGA or embedded within the FPGA) and elements of that application running on hardware (implemented with FPGA gates by the CoBuilder compuiler) using the most efficient and reliable means.

Each platform supported by CoDeveloper is somewhat different, and may require a unique set of library files and/or design techniques for the most practical and efficient operation. Factors that influence the creation of a new Platform Support Package include:

- The type of microprocessor
- The type of bus
- The number and variety of memory resources
- The FPGA family
- The EDA tools being used
- The embedded software development tools being used

Each Platform Support Package includes documentation, tutorials and examples that are specific to the platform.

See Also

Hardware/Software Interfaces

Platform Architecture Files

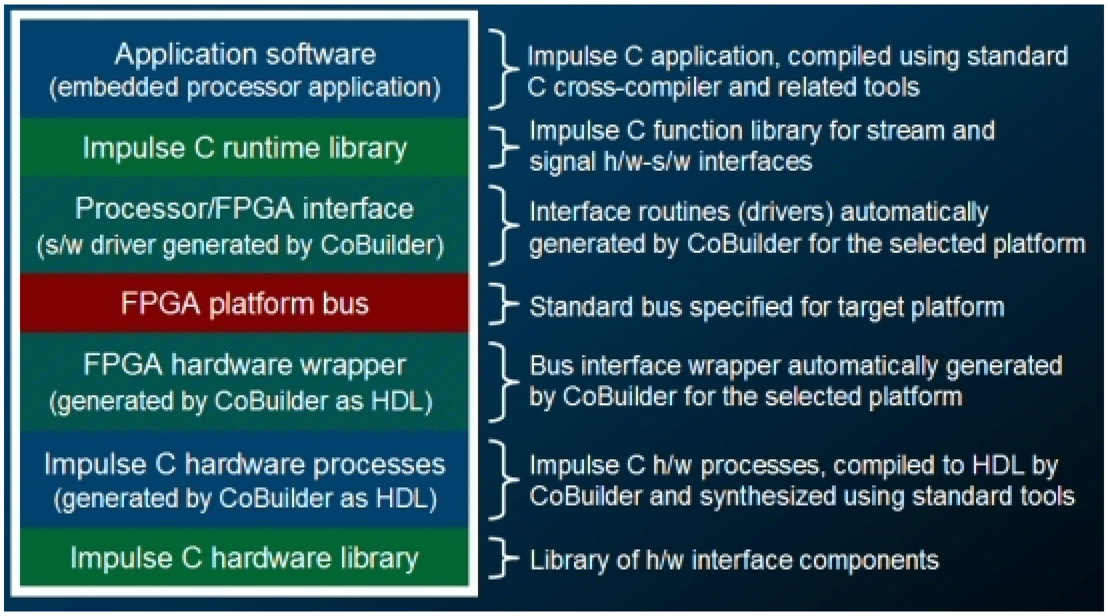
1.9.2 Hardware/Software Interfaces

Hardware/Software Interfaces

The Impulse C programming model emphasizes dataflow-oriented movement of data between processes in a mixed hardware/software application. The performance of Impulse C applications running on target platforms is therefore heavily dependent on the efficient use of hardware/software interfaces (primarily streams, which are implemented in hardware as FIFO buffers).

At a minimum, a Platform Support Package must include software libraries (containing Impulse C functions callable from C code) allowing streams and signals to be written to and read from software processes, as well as hardware components that implement corresponding read and write functions directly as FPGA hardware. Some parts of these libraries are platform-specific (but do not change for different applications), while other parts are specific to an application and must be generated by the CoBuilder tools as a part of the total compilation process.

The following diagram helps to explain how the various elements of a Platform Support Package are interrelated:


Application software
(embedded processor application)
Impulse C application, compiled using standard C cross-compiler and related tools
Impulse C runtime library
Impulse C function library for stream and signal h/w-s/w interfaces
Processor/FPGA interface
(s/w driver generated by CoBuilder)
Interface routines (drivers) automatically generated by CoBuilder for the selected platform
FPGA platform bus
Standard bus specified for target platform
FPGA hardware wrapper
(generated by CoBuilder as HDL)
Bus interface wrapper automatically generated by CoBuilder for the selected platform
Impulse C hardware processes
(generated by CoBuilder as HDL)
Impulse C h/w processes, compiled to HDL by CoBuilder and synthesized using standard tools
Impulse C hardware library
Library of h/w interface components

When CoBuilder processes your application, those streams, signals and registers that cross a hardware/software boundary are included in an automatically-generated bus interface wrapper. This wrapper, which may include both hardware and software components, provides the low-level interface between the generated hardware and the selected embedded processor, using an FPGA-specific bus interface.

The Role of the Platform Bus and Operating System

The interfaces between software and hardware processes are highly dependent on the specific bus architectures (and related memory mappings) being used to connect the target processor (whether external or embedded) and the target FPGA. For this reason, the type of bus structure provided in the target platform is an important factor in the creation of a Platform Support Package.

Another important factor in the design of a Platform Support Package is the existence (or non-existence) of an operating system on the target processor. An operating system is not required to create a mixed hardware/software application (and in fact the highest-performance embedded applications will not make use of an OS), but using an OS makes it possible to create applications that combine multiple independent Impulse C software processes (running under the control of the target OS) with multiple independent hardware processes that are mapped to the target FPGA. Please refer to your Platform Support Package documentation for additional information about operating system support.

Co_ports and Bus Interfaces

By default, any input or output stream, signal or register that is connected to a **co_port** object will not be included in generated bus interface. This is the preferred behavior if you are creating a bus peripheral that requires additional direct (not bus-connected) hardware interfaces.

If you want one or more of your co_port-connected stream, signal or register interfaces to be connected via the generated bus interface wrapper, then you should un-select the "Do not include co_ports in bus interfaces" option in the Generate Options dialog. This method is effectively useful only for renaming the ports of an I/O object as they appear in the generated HDL.

Certain PSPs include the ability to assign specific ports, for example video inputs and outputs, to platform-specific bus interfaces using the co_port_config function.

See Also

About Platform Support Packages
Creating System Interfaces

1.9.3 Platform Architecture Files

Overview

A Platform Support Package for a given platform includes some or all of the following:

- A set of CoBuilder architecture description files (in XML format) describing the target platform.
- A set of target-specific software libraries and related header files that are linked with your Impulse C software application.
- A collection of hardware library components (supplied as VHDL and/or Verilog libraries) in support of hardware interfaces.
- One or more platform-specific design examples.
- Platform-specific EDA interface tools, scripts and libraries.
- Platform-specific documentation in the form of HTML Help files.

Architecture Description Files

Architecture description files are provided with each Platform Support Package. These files are provided in XML format and define characteristics of the platform. The XML files are installed into the Architectures directory within your CoDeveloper installation area, and make reference to lower-level XML files within the Architectures directory structure.

If you wish to create a new, custom platform, you should begin by copying the XML files and related subdirectories for an existing platform (preferably a platform based on the same family of FPGA that you will be targeting), renaming the files and directories, and modifying the files.

Software Libraries

Software libraries are provide for specific platforms within subdirectories of the Architectures directory. These software libraries include C source files, header files, and (in some cases) Makefiles and related scripts allowing the libraries to be built using third-party or platform vendor-supplied compiler tools. These files are normally found within a directory structure identified by the platform name, the processor name, and the bus architecture name (for example, Architectures\Xilinx\Microblaze\FSL).

Hardware Libraries

Hardware libraries include HDL files appropriate for synthesis, as well as Tcl scripts that generate HDL. These hardware files define such things as stream components, memories, and bus interfaces. Hardware libraries are provided as HDL source code and may be examined and modified as needed for platform customization. The hardware library files are normally found within the Architectures\VHDL and Architectures\Verilog subdirectories.

See Also

Hardware/Software Interfaces

1.10 Registering Your Software

CoDeveloper is protected from unauthorized duplication and use by a software-based licensing system. You must obtain a license file from Impulse Accelerated Technologies (or an authorized reseller) and activate the product before beginning to use it.

Impulse uses a FLEXlm®-based licensing method. This licensing method requires that you provide Impulse with an Ethernet network adapter address, which is a 12-digit hexadecimal value. A license file specific to that network adapter will be provided to you. This method also supports a "floating" license model, where a license server controls a limited number of concurrent licenses shared among any number of CoDeveloper installations.

After receiving your new license file ("CoDeveloper.lic"), copy it into the "bin" subdirectory of your CoDeveloper installation (typically "C:\Impulse\CoDeveloper3\bin"). Be sure to exit the CoDeveloper application completely before copying the license file to the "bin" subdirectory.

Further instructions can be found here:
http://ImpulseAccelerated.com/ReleaseFiles/installation_notes.html

See Also

Using a Floating License
License Agreement

1.10.1 Using a Floating License

Note: For the most up-to-date instructions on using a floating license, see the Impulse website:

http://ImpulseAccelerated.com/ReleaseFiles/license_server_setup.html

Server Setup

To set up a floating license server for Impulse CoDeveloper, first read Acresso Software's FLEXnet Licensing End User Guide, available here:

http://ImpulseAccelerated.com/ReleaseFiles/FLEXnet/v11.5/LicenseAdministration.pdf

Then follow these steps (for a Windows license server):

1. Install CoDeveloper on the server machine in the default location (C:\Impulse\CoDeveloper3).
2. Copy the license file provided by Impulse into the C:\Impulse\CoDeveloper3\bin directory.
3. Run the LMTOOLS program (C:\Impulse\CoDeveloper3\bin\lmtools.exe) and read the section "Configuring the License Server Manager as a Windows Service" in the FLEXnet End User Guide. LMTOOLS will let you set up the Impulse license server to run automatically.
4. In the "Service/License File" tab of LMTOOLS, select the "Configuration using Services" radio button.
5. Click the "Config Services" tab.
6. In the "Service Name" field, enter "Impulse License Server". In the "Path to the lmgrd.exe file" field, enter "C:\Impulse\CoDeveloper3\bin\lmgrd.exe". In "Path to the license file", enter "C:\Impulse\CoDeveloper3\bin\CoDeveloper.lic".
7. Check the boxes marked "Start Server at Power Up" and "Use Services".
8. Click "Save Service".
9. Click the "Start/Stop/Reread" tab and click "Start Server" to start the Impulse licensing server.

Client Setup

Each machine on which Impulse CoDeveloper will be used must have a copy of the license file in C:\Impulse\CoDeveloper3\bin, which CoDeveloper will use to locate the license server. Make sure that any client machine running CoDeveloper is able to reach the license server over your network using the server name in the license file (see the "SERVER" line in CoDeveloper.lic).

You may change the server name in the license file without requiring a new license to be issued. For example, if the license server has a different hostname inside a firewall than it does outside the firewall, you may have two sets of license files for clients running CoDeveloper on either side of the firewall. If a different machine will run the floating license server, however, Impulse must issue you a new license file.

Troubleshooting

If you run into problems with FLEXnet licensing, please refer to Acresso Software's FLEXnet Licensing End User Guide first; it is the authoritative source for troubleshooting the licensing system.

See Also

Registering Your Software

1.11 Copyright Acknowledgements

Copyrights and Other Acknowledgements

GCC, GCC, GMAKE and the MinGW library are distributed under terms of the GNU General Public License. Source code for these components is readily available on-line at http://sourceforge.net/.

SUIF is Copyright (C) 1994,95 Stanford University. All rights reserved.

The LCC parser used in SUIF is Copyright (C) 1990,1991,1992 David R. Hanson.

Stage Master is Copyright 2003-2015, Green Mountain Computing Systems, Inc. All rights reserved.

The Scintilla text editor component is Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>. All Rights Reserved.

Eclipse components are licensed under the Eclipse Public License (EPL).

Active-HDL™ is a trademark of Aldec, Inc.

ModelSim® is a registered trademark of Mentor Graphics Corporation.

Certain hardware (HDL) libraries, bus interface wrappers, software drivers, and other files are subject to the terms of the Impulse Accelerated Techologies, Inc. Redistributable IP License Agreement ("Impulse IP License"). These files, some of which may be generated by the Impulse C compiler, are also Copyright (C) 2002-2015, Impulse Accerated Technologies. All rights reserved.

Other parts of the software, including CoDeveloper Application Manager, CoMonitor Application Monitor, Impulse_s2xml, Impulse_genvhdl, Impulse_genvlog, Impulse_arch, and other programs so identified, and this document, are Copyright (C) 2002-2015, Impulse Accerated Technologies. All rights reserved.

See Also

GNU Public License
SUIF Copyright
Eclipse Public License
http://www.mingw.org/
http://sourceforge.net/

1.11.1 Impulse IP License

IMPULSE ACCELERATED TECHNOLOGIES, INC.
REDISTRIBUTABLE IP LICENSE AGREEMENT

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE LICENSED MATERIALS. BY USING THE LICENSED MATERIALS, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, YOU ARE NOT PERMITTED TO USE THE LICENSED MATERIALS; IF YOU HAVE ALREADY PAID FOR USE OF THE LICENSED MATERIALS, PROMPTLY RETURN THE LICENSED MATERIALS TO THE PLACE WHERE YOU OBTAINED THEM AND YOUR MONEY WILL BE REFUNDED.

1. **Definitions**

(a) "Intellectual Property Rights" means any and all tangible and intangible: (i) rights associated with works of authorship, including copyrights, moral rights, neighboring rights, and derivative works thereof, (ii) trademark and trade name rights, (iii) trade secret rights, (iv) patents, design rights, and other industrial property rights, and, (v) all other intellectual property rights (of every kind and nature however designated) whether arising by operation of law, treaty, contract, license, or otherwise, together with all registrations, initial applications, renewals, extensions, continuations, divisions or reissues thereof.
(b) "Licensee" or "You" means the person, either individual or company, that is obtaining the Licensed Materials from Impulse subject to the terms herein to be completed in connection with this Agreement.
(c) "Licensed Materials" means design data and information specifically designated by Impulse as "redistributable", either in source code comments or by other written agreement between Impulse and Licensee.
(d) "Impulse Tools" means the Impulse C compiler, optimizer, FPGA hardware generator, Impulse Platform Support Packages and any other software applications or related documentation provided to Licensee by Impulse, subject to the terms of an Impulse End User License Agreement (EULA).
(e) "End Products" means any integrated circuit, computer system or software application, including FPGAs, FPGA-based boards or systems, software applications or hardware applications, manufactured by or for Licensee and programmed in whole or part using Impulse Tools.
(f) "Licensed Site" means a geographic location in which Licensee conducts business, with a radius of no more than five (5) miles.

2. **License**. Upon payment of any applicable fees and subject to the terms herein, Impulse hereby grants Licensee a nonexclusive, transferable, worldwide, royalty-free license: (i) to use the Licensed Materials for the sole purpose of creating, simulating and implementing End Products; and (ii) to sell or otherwise distribute binary (FPGA bitmap or software executable) versions of the Licensed Materials when incorporated in Licensee's End Products.

3. **Restrictions on Use**.

3.1 Use of the Licensed Materials for product development purposes by any person outside the Licensed Site is prohibited unless Licensee has entered into a separate written agreement with Impulse for such use.

3.2 Licensee may copy the Licensed Materials only to the extent necessary for its authorized use of the Licensed Materials, and for archival and back-up purposes, provided always that Licensee will at all times and in each instance, reproduce all copyright notices, source code comments and proprietary legends on each copy in the same manner as such notices and legends appeared inn the original. No other copies may be made without the prior written consent of Impulse.

3.3 Licensee may not provide Licensed Materials to a third party without prior written approval from Impulse; provided, however, Licensee may provide device programming files (such as FPGA bitstream files and software executable files) to third parties without prior approval in order to deliver End Products.

3.4 Licensee acknowledges that use of the Licensed Materials in combination with other functionality, software or protocols may require licenses from third parties and Licensee accepts sole responsibility for obtaining such licenses.

3.5 Licensee may not publish or disclose the results of any benchmarking of the Licensed Materials or of the Impulse Tools, or use such results for its own competing software development activities, without the prior written permission of Impulse.

4. **Critical Applications**. LICENSED MATERIALS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS LIFE-SUPPORT OR SAFETY DEVICES OR SYSTEMS, CLASS III MEDICAL DEVICES, NUCLEAR FACILITIES, APPLICATIONS RELATED TO THE DEPLOYMENT OF AIRBAGS, OR ANY OTHER APPLICATIONS THAT COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (INDIVIDUALLY AND COLLECTIVELY, "CRITICAL APPLICATIONS"). LICENSEE ASSUMES THE RISK OF ANY USE OF LICENSED MATERIALS IN CRITICAL APPLICATIONS, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

5. **Intellectual Property Rights**. Licensee acknowledges that all Intellectual Property Rights in the Licensed Materials are and will remain the sole property of Impulse or its licensors, if any. To protect such Intellectual Property Rights, Licensee may not decompile, reverse-engineer, disassemble, or otherwise reduce binary format Licensed Materials to a human-perceivable form. Licensee may not modify or prepare derivative works of the Licensed Materials in whole or in part, except for the purposes set forth in Section 2. Nothing contained in this Agreement will be construed as conferring by implication, estoppel or otherwise upon either party any license or other right except the licenses and rights expressly granted hereunder to a party hereto.

6. **Term; Termination.** This Agreement will commence upon the date that this

Agreement is accepted by Licensee and will remain effective until terminated. Licensee may terminate this Agreement at any time by destroying the Licensed Materials and all copies thereof. This Agreement will terminate immediately without notice from Impulse if Licensee fails to comply with any provision of this Agreement, provided that any payment obligations accruing prior to such termination will remain due and owing. Upon termination of this Agreement, the licenses, rights and covenants granted hereunder and the obligations imposed hereunder will cease, except as otherwise expressly provided for herein, and Licensee will destroy the Licensed Materials, including all copies and all relevant documentation. Sections 1, 3.3, 4 ~ 6, 8.2, 9, 10, 13 and 14 will survive the termination of this Agreement.

7. **Governmental Use**. If Licensed Materials are being acquired by the U.S. Government, the software and related documentation is commercial computer software and commercial computer software documentation developed exclusively at private expense, and (i) if acquired by or on behalf of a civilian agency, shall be subject to the terms of this computer software license as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors; or (ii) if acquired by or on behalf of units of the Department of Defense("DoD") shall be subject to the terms of this commercial computer software license as specified in 48 C.F.R. 227.7202, DoD FAR Supplement and its successors. Manufacturer is Impulse Accelerated Technologies, Inc., 550 Kirkland Way, Suite 408, Kirkland, WA 98033.

8. Limited Warranty and Disclaimer,

8.1 Impulse represents that for a period of one (1) year from acceptance of this Agreement by Licensee that the Licensed Materials will substantially conform to the specifications published by Impulse for the Licensed Materials. The sole liability of Impulse and the exclusive remedy of Licensee with respect to breach of the foregoing limited representation will be limited to error correction or replacement, or if neither is in the opinion of Impulse commercially feasible, termination of this Agreement and refund of any license fee received by Impulse from Licensee for the Licensed Materials.

8.2 EXCEPT AS SPECIFICALLY STATED ABOVE, THE LICENSED MATERIALS LICENSED HEREUNDER ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION, ANY WARRANTY WITH RESPECT TO NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Impulse does not warrant that the functions contained in any of the Licensed Materials will meet Licensee's requirements, or that the operation of the Licensed Materials will be uninterrupted or error-free, or that defects in the Licensed Materials will be corrected. Furthermore, Impulse does not warrant or make any representations regarding use or the results of the use of the Licensed Materials in terms of correctness, accuracy, reliability or otherwise.

8.3 THE LIMITED WARRANTIES CONTAINED ABOVE ARE GRANTED TO LICENSEE ONLY AND MAY NOT BE TRANSFERRED, ASSIGNED OR

EXTENDED TO ANY OTHER PERSON, INCLUDING WITHOUT LIMITATION ANY SUBLICENSEES OR USERS/LICENSEES OF THE END-PRODUCTS. THIS LIMITED WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED. IMPULSE EXPRESSLY DISCLAIMS ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR PARTICULAR PURPOSE OR NONINFRINGEMENT, AND ANY WHICH MAY ARISE FROM COURSE OF PERFORMANCE, COURSE OF DEALING, OR USAGE OF TRADE.

8.4 Licensee shall not (a) make any false or misleading representations with regard to Impulse or the Licensed Materials, (b) publish or employ, or cooperate in the publication or employment of, any misleading or deceptive advertising material with regard to Impulse or the Licensed Materials; (c) or make any representations, warranties or guarantees for or on behalf of Impulse or with respect to the specifications, features or capabilities of any Licensed Materials, without Impulse's prior written consent.

8.5 Licensee shall indemnify, defend and hold Impulse harmless from and against any claim, damages or liability (including reasonable attorneys fees) arising out of, based upon or relating to the its use of the Licensed Materials and the manufacture, marketing, distribution, service and/or support of the End-Products.

9. **Limitation of Liability**. THE ENTIRE LIABILITY OF IMPULSE IN RESPECT OF ANY BREACH OF ITS CONTRACTUAL OBLIGATIONS ARISING UNDER THIS AGREEMENT AND ANY REPRESENTATION, STATEMENT OR TORTIOUS ACT OR OMISSION INCLUDING NEGLIGENCE ARISING UNDER OR IN CONNECTION WITH THIS AGREEMENT (TOGETHER AN "EVENT OF DEFAULT") SHALL BE LIMITED TO DAMAGES IN AN AMOUNT EQUAL TO ALL LICENSE FEES PAID BY LICENSEE TO IMPULSE IN THE PRECEDING 12 MONTHS FOR THE APPLICABLE LICENSED MATERIALS. NOTWITHSTANDING THE FOREGOING, IMPULSE WILL NOT BE LIABLE TO LICENSEE IN REGARD TO ANY EVENT OF DEFAULT FOR LOSS OF DATA, PROFITS, GOODWILL OR ANY TYPE OF SPECIAL, INDIRECT OR CONSEQUENTIAL LOSS (INCLUDING LOSS OR DAMAGE SUFFERED BY LICENSEE AS A RESULT OF ANY ACTION BROUGHT BY A THIRD PARTY) EVEN IF SUCH LOSS WAS REASONABLY FORESEEABLE OR IMPULSE HAD BEEN ADVISED OF THE POSSIBILITY OF LICENSEE INCURRING THE SAME. THIS LIMITATION SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDIES HEREIN. NOTHING IN THIS SECTION WILL CONFER ANY RIGHT OR REMEDY UPON LICENSEE TO WHICH IT WOULD NOT OTHERWISE BE LEGALLY ENTITLED.

10. **Export Restriction**. Licensee agrees to obey all applicable export laws and regulations, including those administered by the U.S. Department of Commerce (U.S. Export Administration Regulations 15 CFR 730 *et seq*.), and shall not export, reexport, resell, transfer, or disclose, directly or indirectly, any Licensed Materials, or the direct product thereof, to any proscribed person, entity, or country, or foreign national thereof, unless properly authorized by the U.S. government.

11. **Third-Party Beneficiary**. Licensee understands that portions of the Licensed Materials and related documentation may have been licensed to Impulse from third parties and that such third parties are intended third-party beneficiaries of the provisions of this Agreement.

12. **Nondisclosure**. Except as otherwise expressly permitted in this Agreement, Licensee will hold in confidence the Licensed Materials and all other information received hereunder from Impulse. Licensee agrees that the Licensed Materials and documentation furnished hereunder will be treated as proprietary trade secrets of Impulse, and Licensee will not make the Licensed Materials or the documentation available in any form to any person other than to its employees and to contractors located on its premises with a need to know subject to restrictions no less stringent than those contained herein. Licensee represents to Impulse that it maintains a system of confidentiality consistent with semiconductor industry standards to protect its own confidential business information, including written agreements with employees, and that the Licensed Materials and documentation will be protected by such a system to the same extent. Licensee may not publish or disclose the results of any benchmarking of the Licensed Materials, or use such results for competing software development activities, without the prior written permission of Impulse.

13. **Governing Law**. **This Agreement shall be governed by the laws of the State of Washington, without reference to conflict of laws principles.**

14. **General**. Licensee may not assign this Agreement or transfer any of the rights or obligations under this Agreement without the prior written consent of Impulse. This Agreement shall be binding upon, and inure to the benefit of, the successors and permitted assigns of the parties. No addition or modification to this Agreement is valid unless made in writing and signed by both parties. No waiver will be implied from conduct or failure to enforce rights, nor be effective, unless in writing signed on behalf of the party against whom the waiver is asserted. Any part of this Agreement found to be unenforceable shall be enforced to the maximum extent permitted by law and the remainder of this Agreement will remain in full force. This Agreement contains the entire agreement between the parties relating to its subject matter and supersedes all prior representations, discussions and agreements.

1.11.2 SUIF Copyright

Overview

The SUIF Copyright applies only to the following portions of the CoDeveloper software:

- Impulse_snoot.exe
- Impulse_porky.exe

SUIF version 1.3.0.1 Copyright Notice

This software is Copyright (C) 1994,95 Stanford University.
All rights reserved.

1.11.3 GNU Public License

Overview

The GNU General Public License (GPL) applies only to those portions of the CoDeveloper software explicity defined as covered under the GPL. These components are limited to:

- GCC, GDB, GMAKE and related libraries and utilities
- MinGW libraries

GNU GENERAL PUBLIC LICENSE

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and

thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.
In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY
11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.11.4 Eclipse Public License

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY

CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Index