

## Designing High-Performance DSP Hardware Using Catapult C Synthesis and the Altera Accelerated Libraries

### Introduction

Today's class of high-performance FPGAs, such as the Altera® Stratix® III device, provide design engineers with a hardware platform that is capable of addressing the computational requirements needed to implement many next-generation wireless and video algorithms. Although these devices provide dedicated hardware to implement the basic building blocks of digital signal processing (DSP) algorithms such as multiply-accumulate (MAC), designers still must meet the challenges of rapidly taking an algorithm from concept to implementation in the register transfer level (RTL).

Historically, the design flow consisted of modeling the algorithm functionality in a high-level language such as C++ and then hand-coding it in RTL. This manual method of RTL creation is not only time consuming and error prone, but often is highly sensitive to back-end routing delay problems. Catapult high-level C++ synthesis has been used to build ASIC hardware sub-systems such as extremely complex and compute-intensive applications found in wireless, video, and image processing. Combining Catapult's ASIC capabilities with Altera Accelerated Libraries provides designers with a rapid path from algorithms modeled in ANSI C++ to optimized RTL running in FPGA hardware. Furthermore, this design flow allows designers to directly target the FPGA DSP blocks from C++, easily solving back-end timing problems using high-level synthesis constraints.

### DSP Algorithm Development Using High-Level Synthesis

In a typical DSP algorithm design flow, the algorithm designer or system architect models the algorithm in a high-level language such as C++. Working at this higher level of abstraction enables the system architect to focus on functional intent, rather than worrying about implementation details such as system pipelining and how many FPGA DSP blocks are required to achieve the performance requirements. Once the algorithm functionality is proven, a specification is created detailing the required area and performance metrics and is handed off to the RTL designer.

The RTL designer determines the type and number of resources required to achieve the specification, deciding what type of hardware resources to use, such as how many RAMs, DSP blocks, shift registers, etc. Coding the RTL to be sufficiently generic can be challenging because the inference capabilities of the RTL synthesis tool often cannot take full advantage of all of the operating modes of the dedicated hardware resources. To get around this problem, RTL designers typically instantiate a DSP macro in the RTL code and configure the DSP block directly via generics/parameters. But that requires that the hardware designer understand how to interface to this block to the rest of the system.

To streamline this frequently time-consuming process, the Catapult C high-level synthesis design process starts with describing an algorithm and then choosing a target technology. The algorithm description is pure ANSI C++ source, describing only the functionality. Hardware requirements such as parallelism and interface protocols are easily applied in Catapult through constraints, which, in turn, guide the synthesis process.

For example, the algorithm below is a basic finite impulse response (FIR) filter, using the freely available Mentor Graphics Algorithmic C™ data types (add link) to define interface and internal bit widths.

```
void fir_filter (ac_int<16> *input, ac_int<16> coeffs[NUM_TAPS], ac_int<16> *output ) {
    static ac_int<16> regs[NUM_TAPS];
    ac_int<32> temp = 0;
    int i;
    SHIFT:for ( i = NUM_TAPS-1; i>=0; i-- ) {
        if ( i == 0 )
            regs[i] = *input;
        else
```

```

    regs[i] = regs[i-1];
}
MAC:for ( i = NUM_TAPS-1; i>=0; i--) {
temp += coeffs[i]*regs[i];
}
*output = temp>>16;
}

```

The C++ algorithm contains no indication about how many or what type of multipliers should be used to realize the hardware. Thus, the algorithm can be efficiently created without the system architect being burdened with the implementation details.

The next step is to determine the target technology and critical specifications. In Catapult, the target technology can be ASIC or FPGA and is independent of the source code description. Catapult C Synthesis uses technology-specific library characterization to create a highly optimized library of operators such as adders, multipliers, etc. This characterization process gathers detailed area and timing information about device specific resources and allows Catapult to create a technology-aware schedule without wasting costly runtimes through RTL synthesis during HLS exploration. The result is quick up-front area/performance estimates with technology-specific RTL output.

Once the target technology and clock frequency are specified, the designer is free to begin exploring the design space using automated high-level synthesis. Because the automated process is so much quicker than hand-coding RTL, the designer can look at a wide range of alternatives, making it possible to trade-off between area and performance easily, resulting in hardware implementation tuned to the exact design goals. The high-level synthesis tool has detailed knowledge about the target technology, choosing the appropriate operators based on the clock frequency requirements and adding system-level pipelining where needed to ensure that clock frequency constraints are never violated. The use of high-level synthesis constraints such as loop unrolling and loop pipelining allow the designer to explore a wide range of micro-architectures from the smallest sequential to fully parallel implementations (compare the implementation in [Figure 1](#) versus that of [Figure 2](#)).

Figure 1. Sequential FIR Implementation

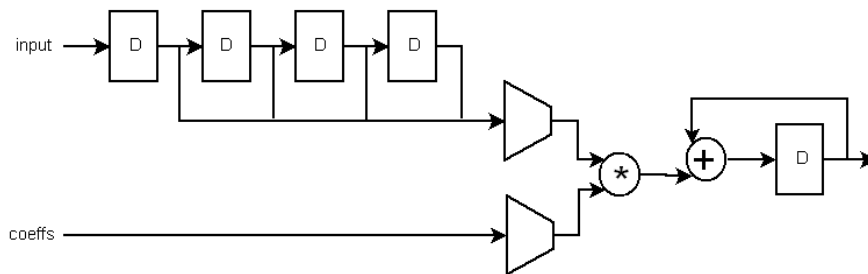
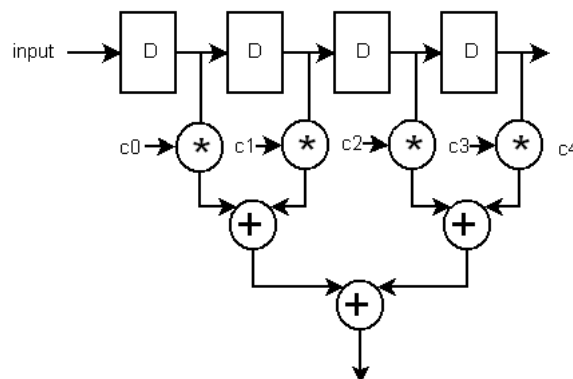


Figure 2. Parallel FIR Implementation



The operators selected for scheduling are chosen to satisfy the clock frequency constraint. Thus for high clock frequencies, pipelined components are automatically selected. In addition, high-level synthesis resource constraints can be applied to add additional pipelining stages to help reduce back-end routing delays.

Although the high-level synthesis process is technology aware, the final auto-generated RTL netlist is usually written generically, relying on the ability of downstream RTL synthesis to infer to the appropriate FPGA resources. For example, a three-stage pipelined multiplier selected during high-level synthesis is netlisted as a multiply operation followed by three registers. Downstream RTL synthesis is then used to infer this netlist into the appropriate hardware resource by automatically retiming the registers into the built-in registers of the DSP block.

One of the limitations with this flow is that the C++ language does not contain built-in operators that correspond to the “macro” operators that the FPGA DSP block is capable of implementing, such as multiply-accumulate (MAC) and multiply-add- (MULT-ADD). Furthermore, the inferencing capabilities of RTL synthesis tools are currently only capable of inferring the basic configurations of the DSP blocks. The more advanced modes that support operations like saturation and rounding, available in devices like the Stratix III FPGA, are only accessible by directly instantiating the DSP macro in the RTL.

The Catapult C Synthesis hardware accelerated flow bypasses these limitations by using the Altera Accelerated Libraries, giving designers the ability to target the FPGA DSP blocks directly from a set of pure ANSI C++ functions. All of the details of instantiating, configuring, and interfacing to the DSP macros takes place “under the hood,” resulting in optimal designs with a minimal amount of effort.

### Stratix III Family of Devices

A library is only as good as the silicon it runs on and Altera’s new Stratix III device family of high-end FPGAs provides a configurable logic fabric that combines the world’s highest performance and highest density with the lowest possible power consumption. Many complex systems such as WiMAX, 3GPP WCDMA, H.264 video compression, medical imaging, and HDTV, to name a few, use sophisticated DSP techniques that require a large number of mathematical computations. Stratix III devices are ideally suited to these systems, as their DSP blocks consist of a combination of dedicated elements that perform multiplication, addition, subtraction, accumulation, summation, and dynamic shift operations of which designers can configure to build sophisticated fixed-point and floating-point arithmetic functions. These can then be manipulated easily to implement larger computationally intensive subsystems such as FIR filters, complex FIR filters, infinite impulse response (IIR) filters, fast Fourier transform (FFT) functions, and discrete cosine transform (DCT) functions commonly used in the aforementioned systems.

Each Stratix III device has two to seven columns of DSP blocks, making them ideal for pipelined algorithms. When used with the proper tool such as Catapult C, this allows for performance ranges from 6.5 GMACs to upwards of 270 GMACs, and offering up to 60 times the performance of high-end DSP processors.

### Accelerated Libraries

The Catapult Synthesis Altera Accelerated Libraries provide a library of C++ functions that directly map to Altera FPGA dedicated hardware resources. Table 1, Table 2, and Table 3 show all the libraries provided for Altera devices.

Table 1. Functions Supported in all Stratix- and Cyclone®-Based Devices

Function	Description
Alt_sqrt	Calculates the square root
Alt_sqrt_remainder	Calculates the square root remainder
Alt_abs	Calculates the absolute value
Alt_divide_quotient	Calculates the quotient of divide operation
Alt_divide_remainder	Calculates the remainder of divide operation
Alt_shift_taps	RAM-based shift register with multiple output taps
Alt_barrel_shift	Barrel shift register

Table 2. Additional Functions Supported in Stratix and Stratix II Devices

Function	Description
Alt_4mult_add	Sum of four multipliers
Alt_4mult_sub	Sum of four multipliers with the output of the first two multipliers and the last two multipliers being subtracted from each other then added at the end
Alt_3mult_add	Sum of three multipliers
Alt_3mult_sub	Sum of three multipliers with the output of the first two multipliers being subtracted from each other then added at the end
Alt_2mult_add	Sum of two multipliers
Alt_2mult_sub	Subtraction of the output of two multipliers
Alt_1mult_add_accum	Multiplier followed by a adder accumulator
Alt_1mult_sub_accum	Multiplier followed by a subtractor accumulator

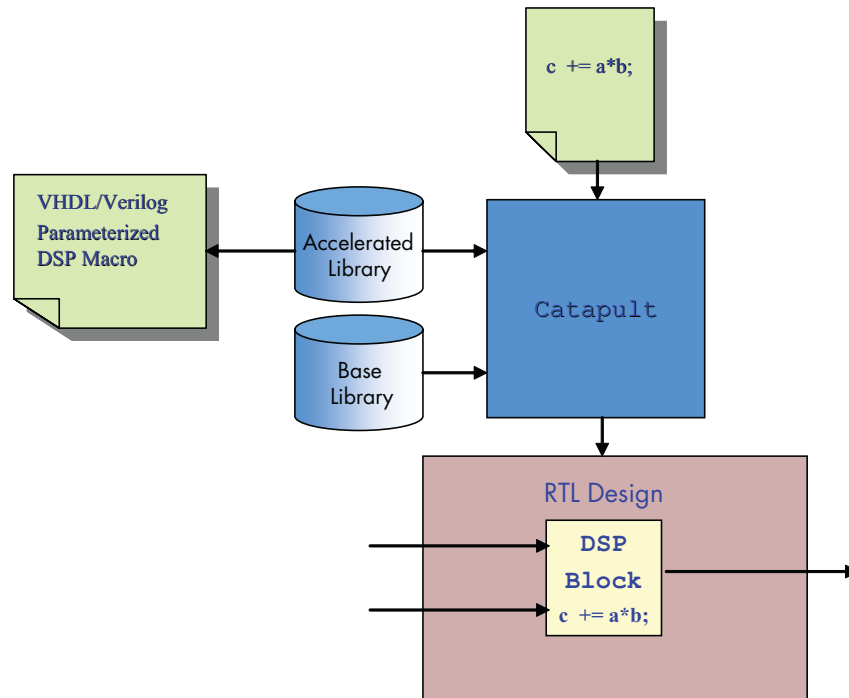
Table 3. Additional Functions Supported in Stratix III Devices

Function	Description
Alt_4mult_add_accum	Sum of four multipliers followed by an accumulator
Alt_4mult_sub_accum	Sum of four multipliers with the output of the first two multipliers and the last two multipliers being subtracted from each other then added at the end followed by an accumulator
Alt_3mult_add_accum	Sum of three multipliers followed by an accumulator
Alt_3mult_sub_accum	Sum of three multipliers with the output of the first two multipliers being subtracted from each other then added at the end followed by an accumulator
Alt_2mult_add_accum	Sum of two multipliers followed by an accumulator
Alt_2mult_sub_accum	Subtraction of the output of two multipliers followed by an accumulator

When the accelerated library is included in a Catapult project, the C++ functions are bound to a Catapult library operator, which in turn is bound to a parameterized RTL module that instantiates and configures the DSP macro for specific Altera devices. If the library is not included in the project, the C++ code is synthesized directly, allowing users to retarget the design to other devices that may not have built-in DSP hardware.

As shown in [Figure 3](#), the Altera Accelerated Libraries are accessed in the C++ by including a C++ header file of template functions. The template functions allow users to specify parameters such as data bit widths as well as hardware parameters such as pipeline registers. The template functions are supported for both integer and fixed-point Algorithmic C data types.

Figure 3. Altera Operator Flow Diagram



An example function is the `Alt_4Mult_add` function which allows us to map to the Stratix III DSP block configured for four 18x18 multiplies followed by two add stages.

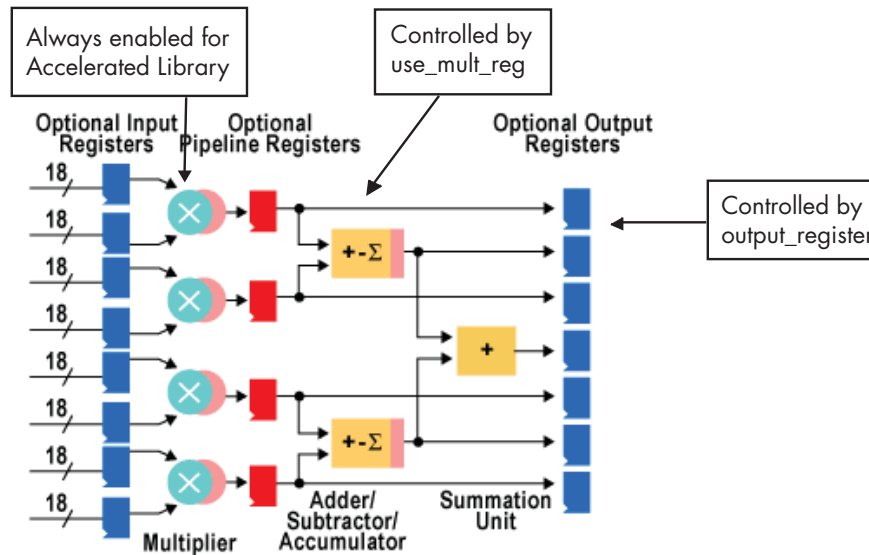
```
#pragma map_to_operator "Alt_4mult_add"
template <bool use_extra_input_reg, bool use_mult_reg, unsigned int output_register, int width_a,
         bool sign_a, int width_b, bool sign_b>
ac_int<width_a+width_b+2, sign_a || sign_b> Alt_4mult_add
(
    ac_int<width_a,sign_a> a0, ac_int<width_b,sign_b> b0,
    ac_int<width_a,sign_a> a1, ac_int<width_b,sign_b> b1,
    ac_int<width_a,sign_a> a2, ac_int<width_b,sign_b> b2,
    ac_int<width_a,sign_a> a3, ac_int<width_b,sign_b> b3
)
{
    return ((a1*b1 + a0*b0) + (a3*b3 + a2*b2));
}
```

The “`map_to_operator`” pragma indicates that this function will map directly to the hardware if the accelerated library is available. The template parameters include the following behaviors:

- `use_extra_input_reg` - Adds an additional input register outside of the DSP block to help solve routing delay timing problems.
- `use_mult_reg` - Registers the multiplier output. Valid values are 0 (no register) or 1 (one register).
- `output_register` - Sets the number of output registers. Valid values are 0 (no register), 1 (one register), or 2 (two registers).
- `Width_a` - *A* input bit width. Bit widths of up to 18-bits wide are supported.
- `Sign_a` - Sign of a input
- `Width_b` - *B* input bit width. Bit widths of up to 18-bits wide are supported.

Figure 4 shows the Altera Stratix III DSP block and how it can be configured based on the C++ function template parameters.

Figure 4. Altera DSP Block Alt\_4mult\_add Structure



The original FIR code can be easily rewritten to use the Altera Accelerated Library function:

```
#include <ac_int.h>
#include "Altera_accel.h"
#include "stdio.h"
#define NUM_TAPS 32
#pragma design top
void fir_filter (ac_int<18> *input, ac_int<18> coeffs[NUM_TAPS], ac_int<18> *output ) {
    static ac_int<18> regs[NUM_TAPS];
    ac_int<38> temp = 0;
    int i;
    SHIFT:for ( i = NUM_TAPS-1; i>=0; i-- ) {
        if ( i == 0 )
            regs[i] = *input;
        else
            regs[i] = regs[i-1];
    }
    MAC:for ( i = 0; i< NUM_TAPS; i+=4 ) {
        temp += Alt_4mult_add<1,1,1>(
            coeffs[i],regs[i],
            coeffs[i+1],regs[i+1],
            coeffs[i+2],regs[i+2],
            coeffs[i+3],regs[i+3]);
    }
    *output = temp>>18;
}
```

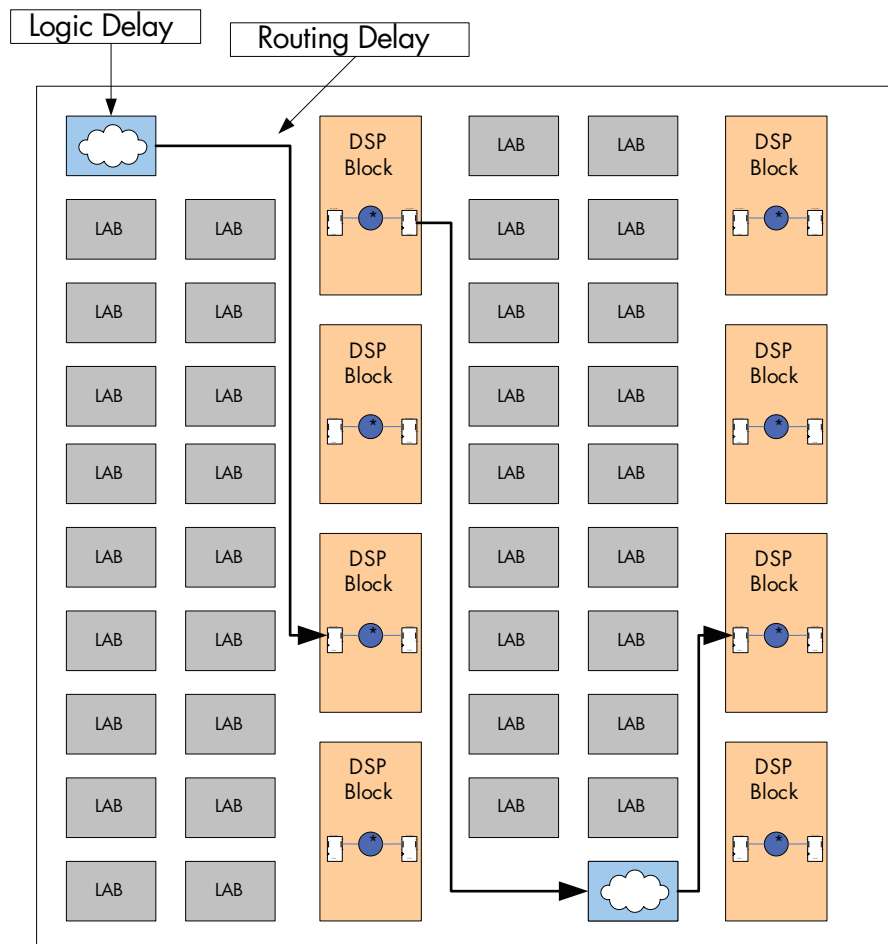
In the FIR example shown above, it is only necessary to specify the template parameters for the registers settings. The width and sign template parameters are automatically plugged in by template matching.

- Library C++ template functions
- Configuring the DSP blocks via template parameters
- Saturation and rounding using the `ac_fixed` data types

## Fixing Routing Delay Timing Issues

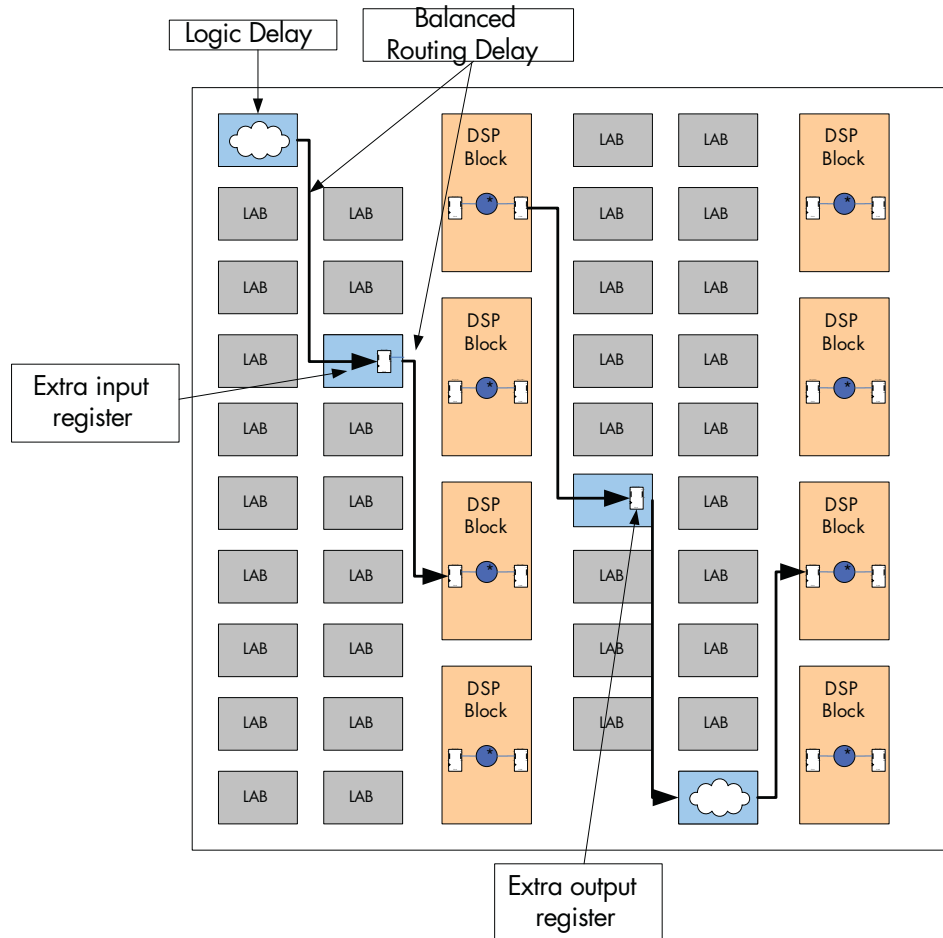
In addition to mapping directly to the Altera DSP blocks and other hardware features, the Accelerated Libraries allow the user to solve tricky back-end timing problems due to routing delays and congestion. As shown in [Figure 5](#), one of the potential sources for failing timing is due to routing to and from the DSP block. This may be true even if the DSP block is fully pipelined with both input and output registers.

Figure 5. Stratix III High-Level Architecture



The Accelerated Library functions have the ability to add an additional register to the inputs and the outputs of the hardware-mapped operators using the template parameters. These extra registers can solve the routing delay problems by providing placement and routing with the ability to balance the routing delay ([Figure 6](#)). The elegance of this approach over hand-coded RTL is that it requires no reworking of the design. Doing this in RTL could potentially require massive changes to the data path finite state machine (FSM).

Figure 6. Using Extra Input and Output Registers to Reduce Routing Delays



### Push-Button Verification of the Accelerated Libraries

One of the key requirements of a high-level synthesis design flow is the ability to automatically verify both the RTL generated from the untimed C++ source as well as the post-RTL synthesis netlist. The Catapult C automated verification flow not only provides automatic verification of the C++ against the RTL, but also automatically verifies the Accelerated Libraries C++ template functions against the RTL that instantiates the DSP macros.

Starting from pure untimed C++, along with a set of interface synthesis constraints, it is possible to automatically create a SystemC™ verification infrastructure, allowing the original C++ design and testbench to be used to test both the RTL and post-RTL synthesis netlists. This auto-generated test environment not only captures the input stimulus and design outputs directly from the C++ testbench, but also generates the SystemC transactors, allowing the RTL to be co-simulated against the C++ on a simulation platform. The RTL simulation output is also captured and automatically compared against the original C++ design to prove functionality.

### Conclusion

While high-performance FPGAs with dedicated DSP blocks have provided the hardware platform for implementing complex compute-intensive algorithms, designers are still faced with the challenges of getting their algorithms to RTL running in hardware. Catapult C, used in combination with the Altera Accelerated Libraries, allows designers not only to take algorithms rapidly from conception to implementation, but gives them much better quality of results over other C/C++ synthesis technology by allowing direct access to the Altera DSP blocks for tuning performance and timing.

---

## Acknowledgements

- Thiagaraja Gopalsamy, Supervising Member of Technical Staff, Altera Corporation
- Mike Fingeroff, Technical Marketing Engineer, Mentor Graphics



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.