



DSP Design Flow and Design Techniques using RTAX-DSP FPGAs

Mir Sayed Ali
Application Engineering
August 2011

Topics

- RTAX-DSP Overview
- Microsemi DSP Design Flow
 - Traditional DSP Design Flow
 - DSP Design Flow using Symphony Model Compiler (SMC)
- Synthesis Strategy for RTAX-DSP Design
- Place and Route Recommendation
- Conclusion



RTAX-DSP Overview

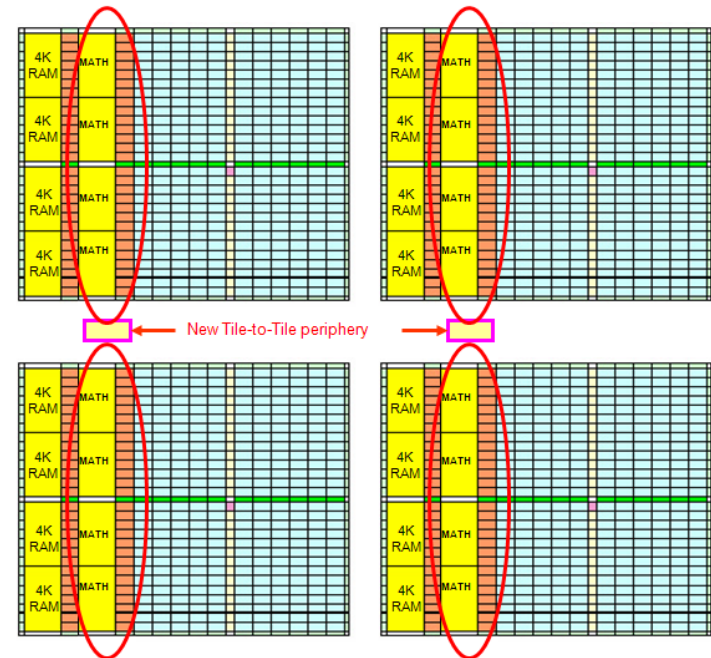
RTAX-DSP FPGAs

■ Enhancement to Existing RTAX-S/SL Devices

- Same 0.15 μ UMC process, same antifuse programming technology
- Enhanced R-cell improves Single Event Transient (SET) by 16x
- DSP mathblocks run 18-bit x 18-bit multiply-accumulate at 125 MHz
 - Embedded DSP blocks protected against heavy ion radiation effects
- Routing architecture remains unchanged: 8.3% fewer logic modules

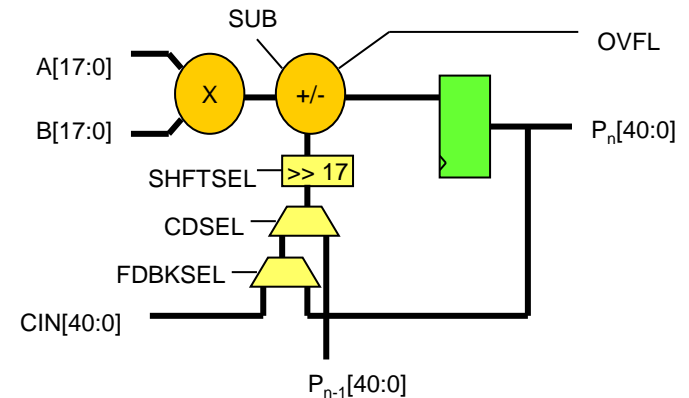


	RTAX2000	RTAX4000	RTAX2000D	RTAX4000D
R-cells	10,752	20,160	9856	18,480
C-cells	21,504	40,320	19,712	36,960
RAM Blocks	64	120	64	120
Math Blocks	0	0	64	120
Clocks	8	8	8	8
IO	684	840	684	840



RTAX-DSP MATH Block Overview

- Takes two 18-bit signed signals and multiplies them for a 36-bit result and then extended to 41 bits
 - Can do multiplication followed by addition and multiplication followed by subtraction
 - Can accumulate the current multiplication product with a previous result, a constant, a dynamic value or a result from another block
 - Can be fractured to implement two instances of signed 9x9
- All the signals of the MATH block (except CIN, CDIN and CDOUT) have optional registers to allow higher performance
- Shift and cascade inputs allows creation of precise and complex functions like wide multipliers

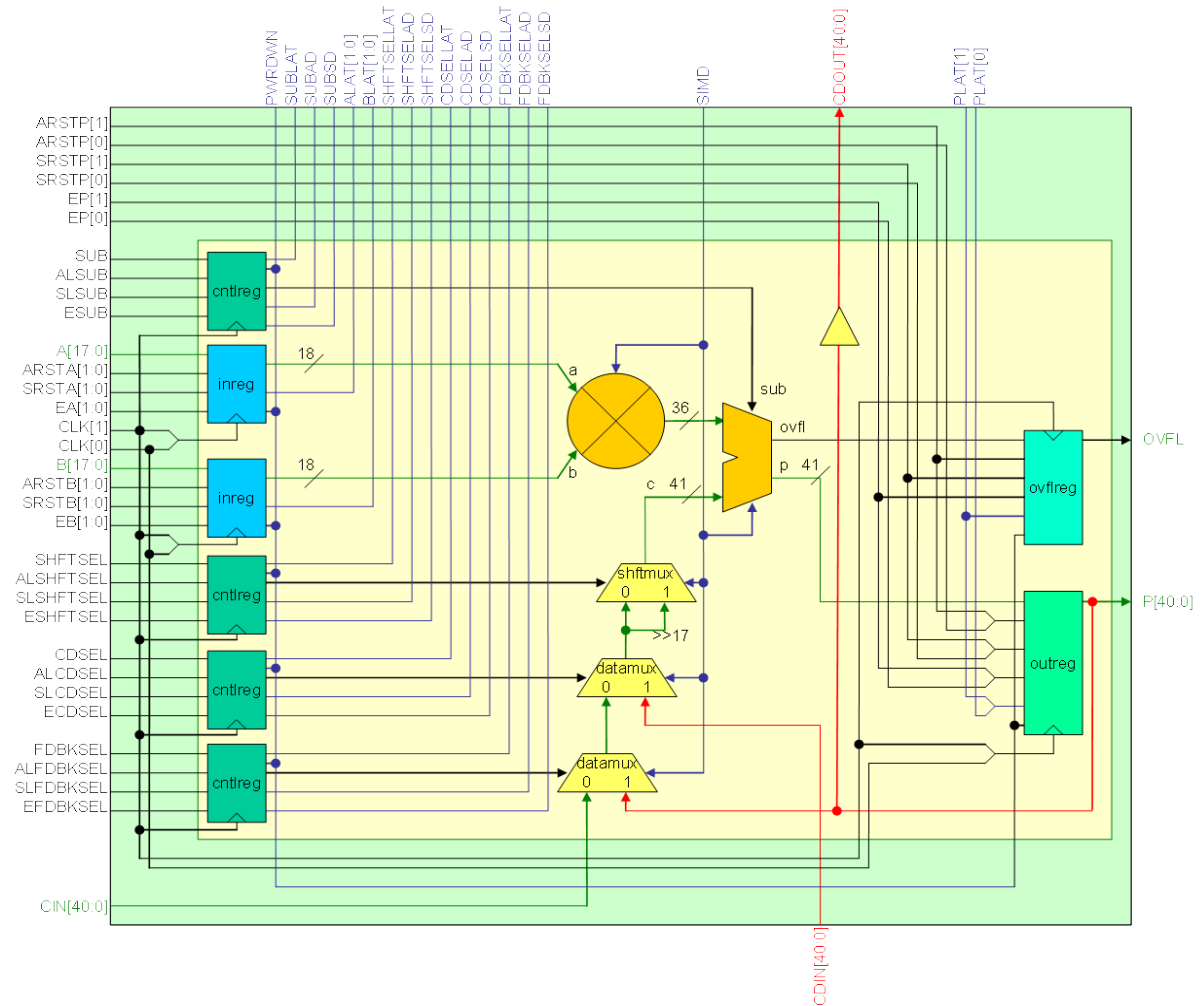


Basic Math Block

Note: optional input registers not shown

MATH18x18 Macro

- MATH Block macro name is MATH18X18
 - When SIMD = 1, MATH block is fractured into two 9 bit x 9 bit multipliers
 - When SIMD = 0, it is called the normal mode
- During cascading CDIN is driven by CDOUT of the previous block



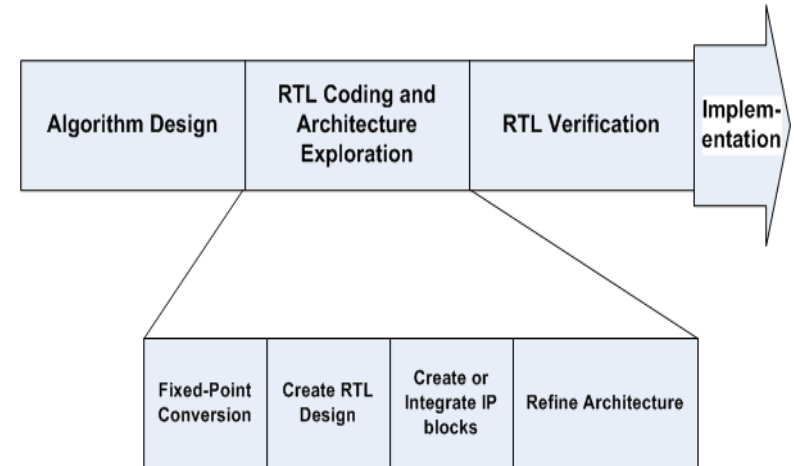


Microsemi DSP Design Flow

Microsemi DSP Design Flow

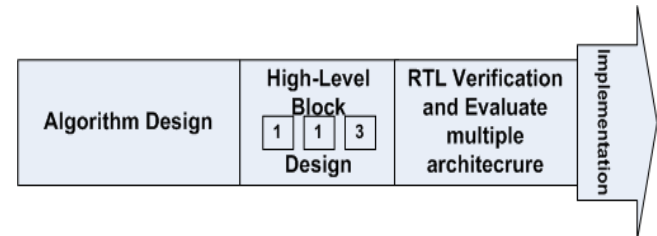
■ Traditional DSP design flow

- DSP designer or system architect creates high-level models using MATLAB/Simulink or similar tool and creates a spec
- RTL designer analyzes the specification and starts coding in RTL and verifies against the algorithm specification
- Standard FPGA flow is used for RTL creation, verification and implementation



■ DSP design flow using Symphony

- Algorithm design is created using Symphony blockset in Simulink
 - DSP designer or RTL designer efficiently evaluate various architecture using Symphony HLS tool
- Symphony generates automatic RTL
- Auto generated RTL is used in standard FPGA flow including verification and implementation



Traditional DSP Design using Manual RTL Coding

- RTL for the DSP blocks can be created using
 - Hand-coded RTL
 - Custom IP blocks
 - MATH Block core configurator in Libero IDE
 - DSP IP from Microsemi
 - Microsemi Provides a set of highly optimized DSP IP that take advantage of the MATH block and offer outstanding performance:
 - CoreFFT: Implement forward and inverse 256-, 512- and 1,024-point complex Fast Fourier Transform (FFT)
 - CoreFIR: Implement Single rate Fully Enumerated (parallel) or Single rate Folded (semi-parallel) or Multi-rate Polyphase Interpolation FIR filter
 - CoreDDS: To generate a sine or cosine waveform as well as the complex sinusoid
 - CoreCORDIC: For calculating the trigonometric functions of sine, cosine, magnitude, and phase

MATH Block Core Configurator

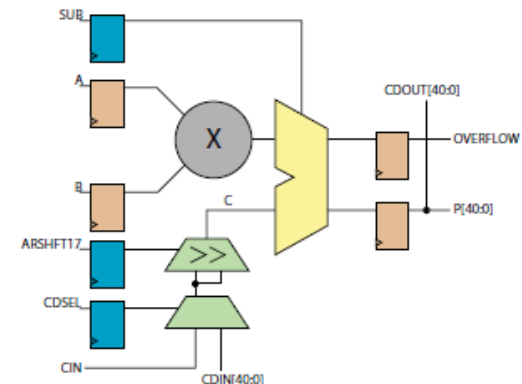
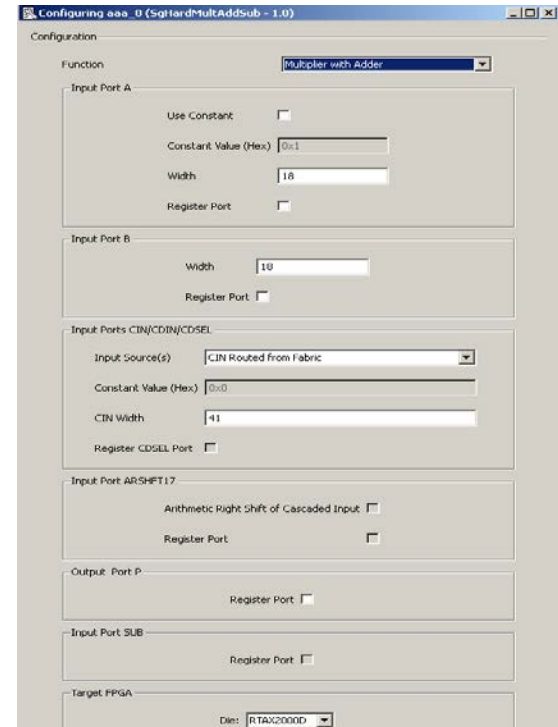
- Several MATH block core generators available in Libero IDE
 - SgHardMult (Simple Multiplier)
 - SgHardMultAddSub (Multiplier with Adder/Subtractor)
 - SgHardMultAcc (Multiplier with Accumulator)
- Each core has its own configurator GUI and can be accessed from the Catalog in Libero IDE
 - Cores are displayed under “Arithmetic” node in the Catalog
- For more info, refer to each core’s handbook:

http://www.actel.com/documents/sghardmult_HB.pdf

http://www.actel.com/documents/sghardmult_addsub_HB.pdf

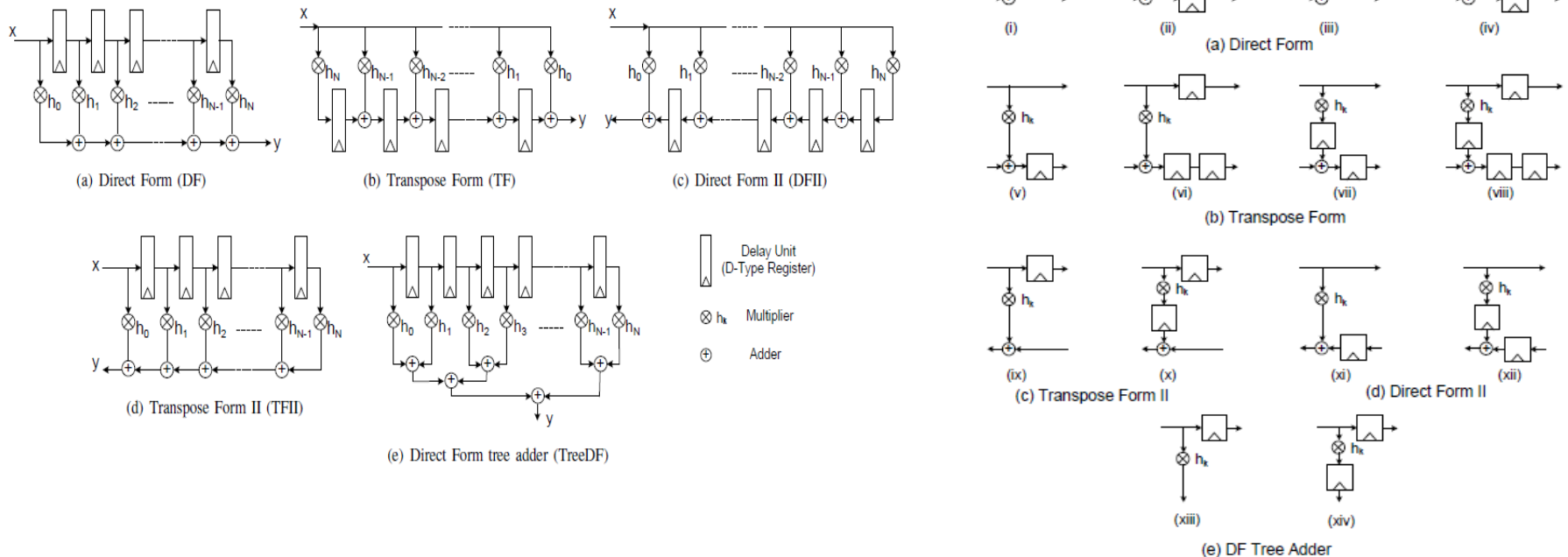
http://www.actel.com/documents/sghardmult_acc_HB.pdf

Hard Multiplier Adder/Subtractor Generator



Complexity of Analyzing DSP Block Architecture for Hand-Coded RTL

- Writing Hand-coded RTL is time consuming
- Limited ability for the designer to fully explore the design space.
 - Example: A FIR filter can be implemented in various ways and with various pipeline options
 - (Ref: Ramsey Hourani, Ravi Jenkal, W. Rhett Davis, Winsor Alexander “Automated Design Space Exploration for DSP Applications” Journal of Signal Processing Systems Volume 56, Numbers 2-3, 199-216, DOI: 10.1007/s11265-008-0226-2)



DSP Design Flow using Symphony AE

Create Design in Simulink Using Microsemi/Symphony Libraries

- Make sure that all design in-outs are defined with Port In and Port Out blocks from the Symphony blockset
- Simulate and verify the design in Simulink to ensure correct functionality

Add Signal Compiler to Model and create encrypted RTL Code and Testbench for simulation

Convert the encrypted rtl to a regular RTL Netlist (Optional)

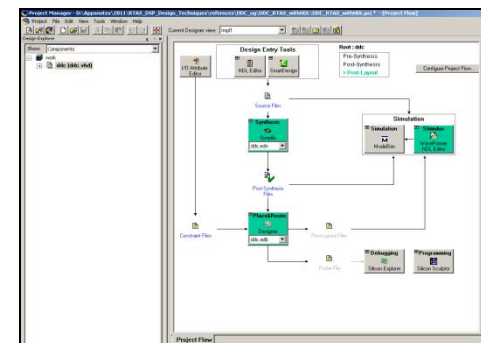
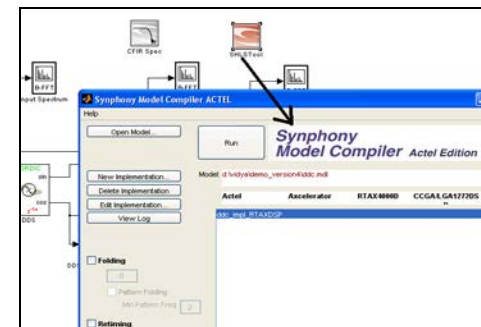
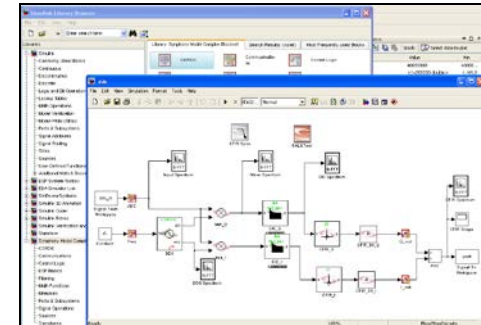
Create a Libero project and import Files and perform RTL simulation

- Add other logic/block in the Libero if needed

Synthesize HDL Code
Run Place & Route
Program Device

Simulink

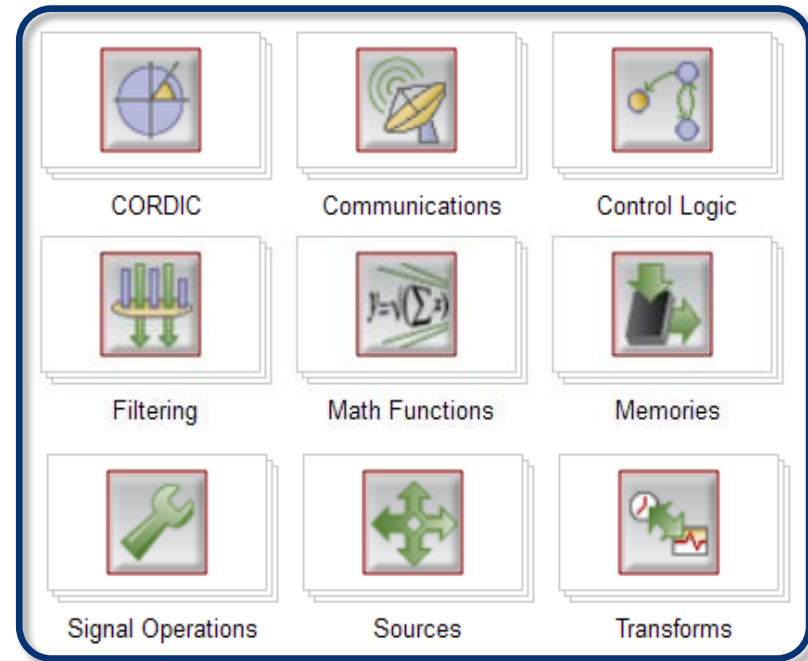
Libero



Symphony Blockset in Simulink

- Library of synthesizable fixed-point functions for math and signal processing
- High-level IP for key wireless & communications applications
- Multirate support
- Vector math support
- High-Level of abstraction

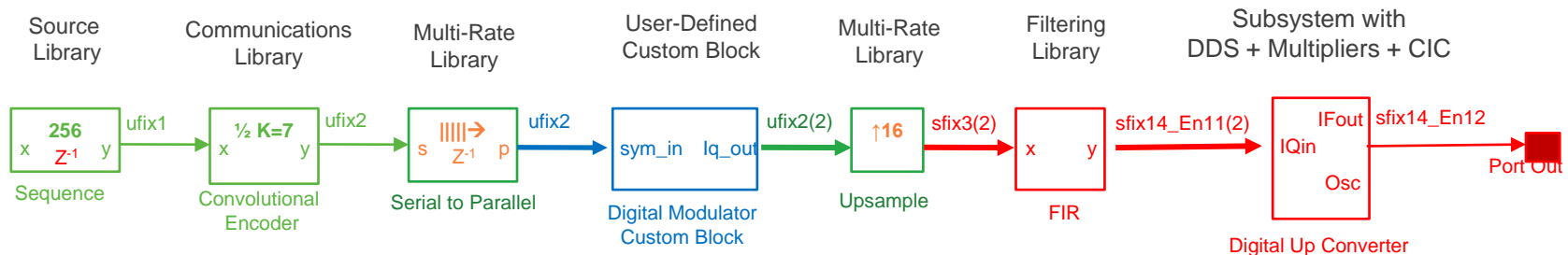
Synthesizable High-Level IP for Communications and Multimedia



Example Wireless Transceiver using Symphony Library:

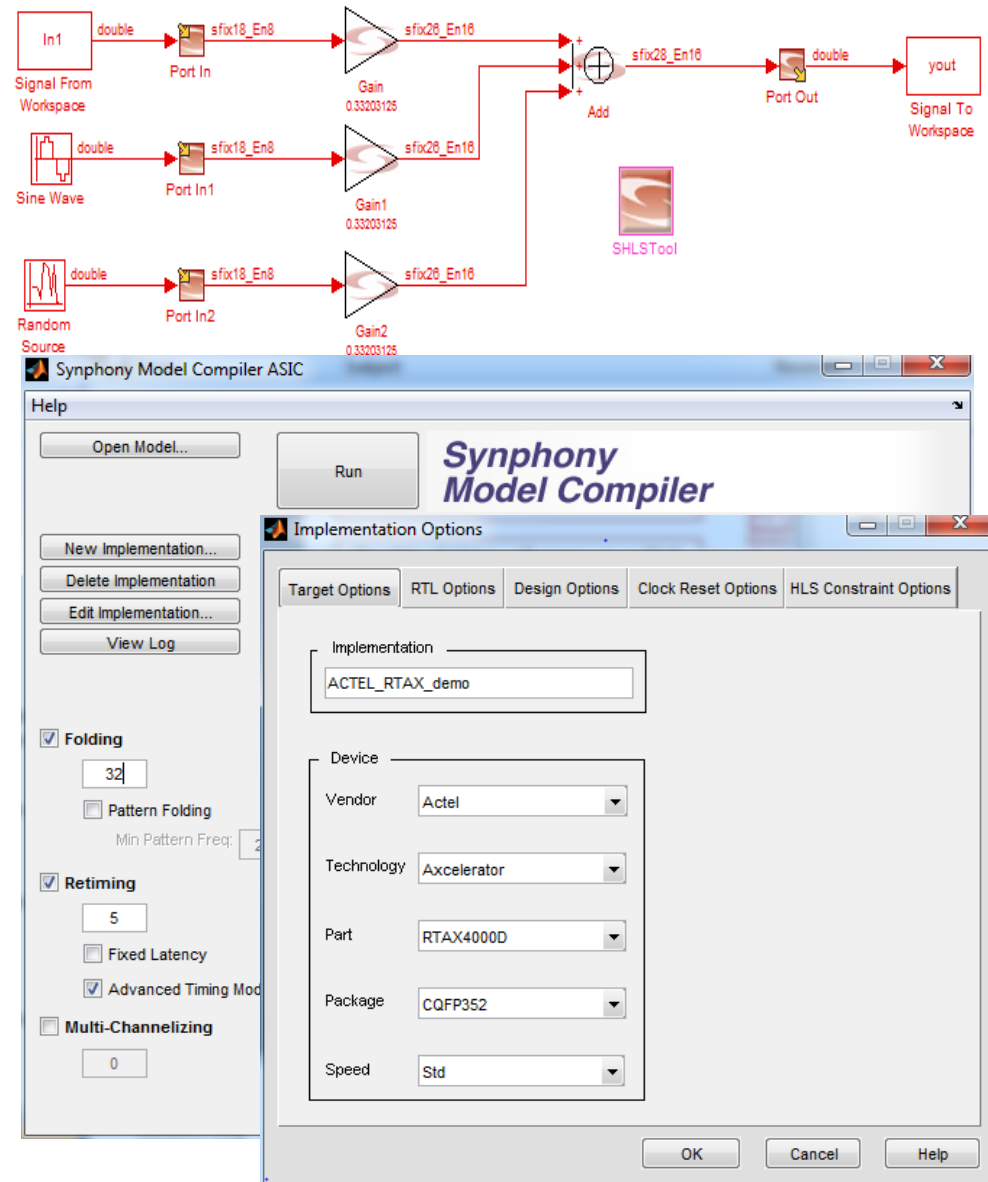
colors = different sample rates

bold = vector



Running Symphony Model Compiler

- Instantiate SHLSTool block inside the model
- Double-clicking SHLSTool block will open the GUI that points to the model file
 - Can also be opened at the command line with shlstool
- Create the implementation by clicking on **New Implementation** to bring up the Implementation dialog box.
- Specify the following:
 - Implementation Name
 - Device info
 - Output types (Verilog, VHDL)
 - Design options (Global Reset, ...)
- Click **Run**



SMC Optimization Overview

- **System-wide optimizations** are directly controlled by the constraints. They are applied globally to the entire design to create a system-wide architecture
 - Top-level optimization control is done by “constraints” in lower left panel
- **IP-level optimizations** are automatically done at the block level for more complicated IP-level functions
- All optimizations are “target-aware” based on the **technology characterization** of the selected target
- Optimizations will sometimes rely on logic synthesis **inferencing** in the downstream tool to optimize operations to device resources
- A **baseline implementation** is created when no constraints are provided, but will still reflect many optimizations for target, inferencing, and IP.
- Advanced controls are also available

Top level Optimization

■ Folding

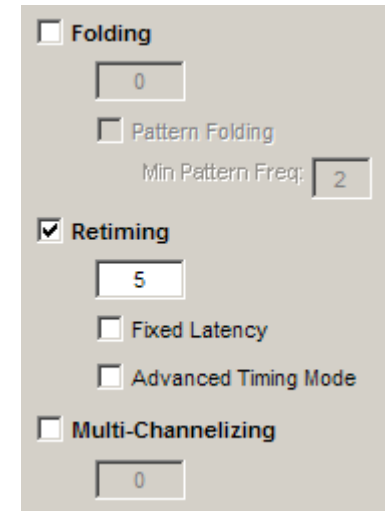
- Performs time-multiplexed resource sharing during area/speed tradeoffs within a single-channel system
 - For example, consider a FIR filter with 50 taps (stages) running at 1 MHz. Each tap has an associated multiplier and adder function. One approach would be to use 50 multipliers and 50 adders running at 1 MHz. Alternatively the architecture could comprise one multiplier and one adder running at 50 MHz, with the intermediate results being stored in the internal memory

■ Retiming

- Rearranges registers so as to optimize speed, while preserving functionality

■ Multi-channelizing

- Generates a multi-channel system from a single-channel specification to automatically optimize the entire design at multiple levels by applying pipelining, scheduling and binding optimizations across model boundaries

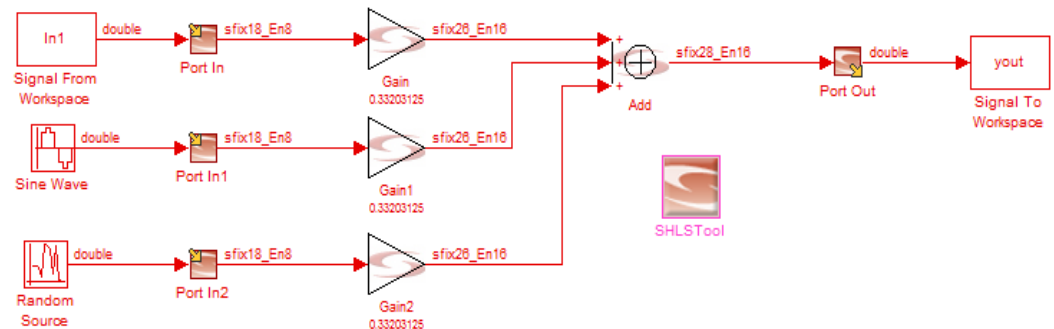
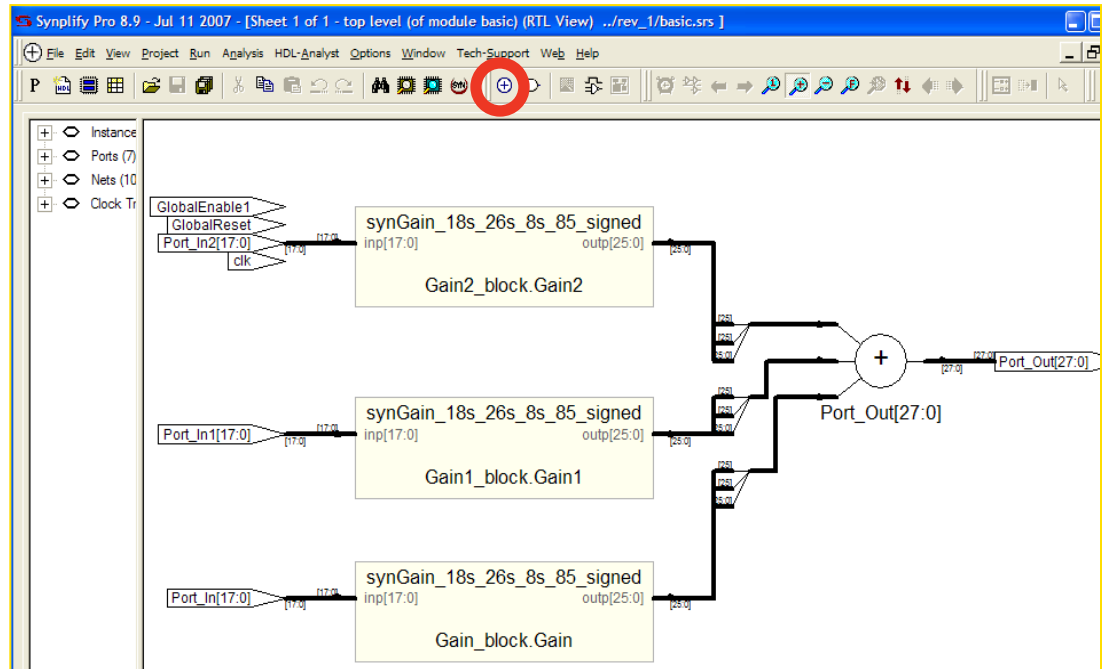


The screenshot shows a configuration window with the following settings:

- ☐ Folding
 - 0
- ☐ Pattern Folding
 - Min Pattern Freq: 2
- ☒ Retiming
 - 5
- ☐ Fixed Latency
- ☐ Advanced Timing Mode
- ☐ Multi-Channelizing
 - 0

Analyzing Blocks Using Synplify HDL Analyst

- After compiling the RTL in Synplify PRO, user can bring up the HDL Analyst RTL
 - The HDL Analyst tool enables graphical browsing and search of the design RTL
- Use the RTL View to inspect the structure of the generated VHDL/Verilog





Synthesis Strategy for RTAX-DSP Design



MATH Block Mapping in Synplify

- Synplify Pro tool extracts the following logic structures from the RTL and maps them to RTAX-DSP MATH Blocks
 - Multipliers
 - Mult-adds (multiplier followed by an adder)
 - Mult-subs (multiplier followed by a subtractor)
- By default, multipliers with input widths of 3 or greater are mapped to MATH Block and splits the multipliers that exceed these limits of the basic blocks
 - Default mapping behavior can be controlled through an attribute:
 - `syn_multstyle` = “logic” or “DSP”
- Synplify packs multiplier, input registers, output registers, and subtractor/adders into the same RTAX-DSP MATH Block, even if they are in different hierarchies
 - Packs registers at inputs and outputs as long as all the registers use the same clock
 - If the registers have different clocks, the clock that drives the output register gets priority, and all registers driven by that clock are packed into the block.
 - If the outputs are unregistered and the inputs are registered with different clocks, the input registers with input that has a larger width get priority, and is packed in the RTAX-DSP MATH block
 - Infer pipelined multipliers to achieve max performance

Example: Wide Multiplier Mapping

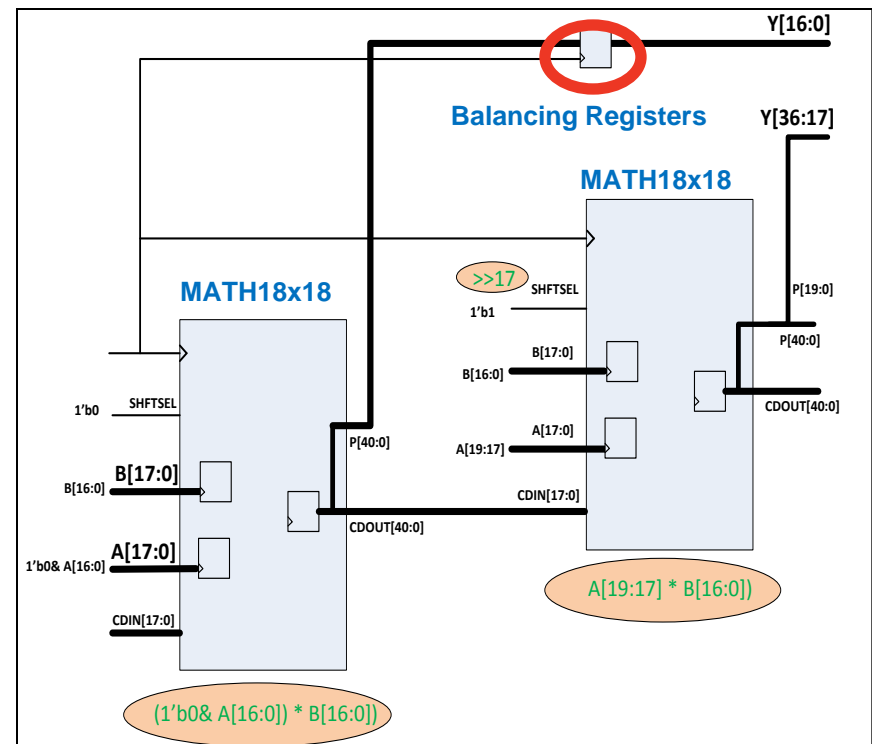
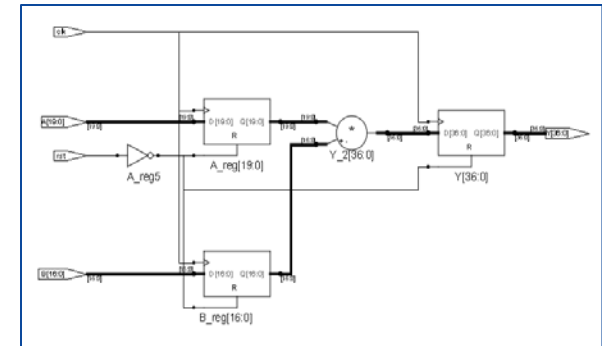
- Simple 20x17-bit unsigned registered multiplier

$$\begin{aligned}
 Y &= A[19:0] * B[16:0] \\
 &= ((A[19:17] \ll 17) + (1'b0 \& A[16:0])) * B[16:0] \\
 &= ((A[19:17] * B[16:0]) \ll 17) + ((1'b0 \& A[16:0]) * B[16:0])
 \end{aligned}$$

$$P1 = ((1'b0 \& A[16:0]) * B[16:0])$$

$$P2 = (A[19:17] * B[16:0]) + (P1 \gg 17)$$

$$Y = \{P2[19:0], P1[16:0]\}$$





Place and Route Recommendation

Place and Route

- Designer is the Place and Route that performs the Layout for RTAX-DSP design
 - Supports Timing Driven Place and Route (TDPR) and Power Driven Place and Route (PDPR)
 - Various options available including multiple seeds sessions, high effort level, and “Hold Time Fix”
 - Designer supports the standard SDC (Synopsys Design Constraints)
 - Apply clock exceptions such as multi-cycle and false paths to avoid several unnecessary iterations
 - Specify appropriate timing constraints and the physical constraints to map clocks and high fanout before running the design

Recommended TDPR Flow

- Most of the DSP design timing challenges may come from one or two DSP blocks
 - Use block flow to deal with these timing critical blocks
 - Please refer to www.actel.com/documents/designer_ug.pdf for block flow
- Recommended TDPR Flow
 1. Run the regular flow using default setting in Symphony and Synplify
 2. Check/modify HCLK/RCLK assignment and run Layout with the timing constraint including timing exception
 3. If timing constraint are not met and if the slack is less than 10%, use floor planning and various Layout options
 4. If timing constraint are not met and if the slack is more than 10%, identify the bottleneck block and try to optimize
 - If the bottleneck block is from DSP block generated from Symphony, apply the Symphony optimization setting and run the flow
 - If the bottleneck block is from DSP block created by the user RTL, try to modify the RTL to use the timing optimized coding style and run the flow
 - May need to use block flow for the critical block
 5. Finally use floor planning and various Layout options to meet timing if needed



Conclusion

Conclusion

- RTAX-DSP MATH Blocks can perform DSP-related operations like multiplication followed by addition, multiplication followed by subtraction, and multiplication with accumulate
 - MATH Blocks allow designers to easily parallelize the computational-intensive portions of their design and offers high performance and low resource utilization for DSP-intensive designs
- Traditional flow can be used, but can be timing consuming
- Symphony allows superior Simulink implementation flow
 - Quickly create synthesizable multi-rate algorithms
 - May need addition steps to convert encrypted RTL
 - May not always give higher performance compared to Hand-coded RTL
 - Higher capacity and superior optimization technologies for FPGA
 - Tight integration with Synplify
 - Best ease-of-use, portability and also re-use