STAP Radar Processing Design Example

Altera Hardware Implementation

Agenda

- Introduction to Altera floating point in FPGAs
  - 28 nm silicon enhancements
  - “Fused Datapath”
- DSP Builder Advanced Blockset overview
- STAP Radar processing design example

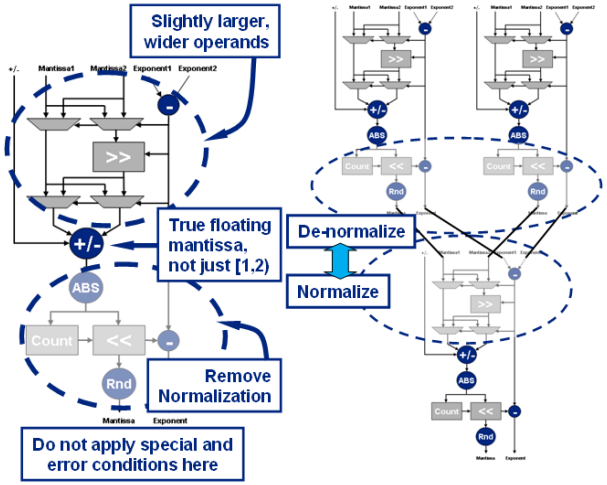
Floating Point in FPGA - Technology

Hardware	Software & IP
Stratix® V	Fused Data-path Technology Patented Algorithms & IP

Variable Precision DSP Blocks

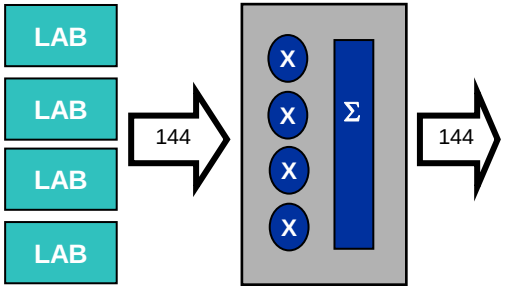
18x18, 27x27, 36x36 seamless trade-off

Greatly increased multiplier density



28 nm Variable Precision

Stratix II

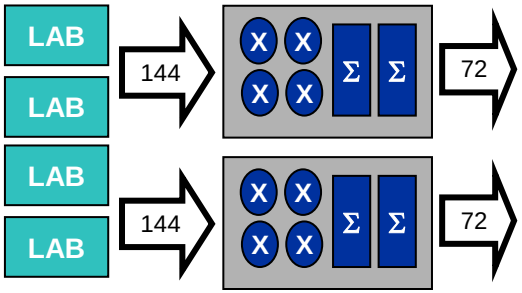


Four 18x18 (Independent)

Eight 9x9 (Independent)

One 36x36

Stratix III / IV, Arria II



Eight 18x18 Multipliers (Sum)

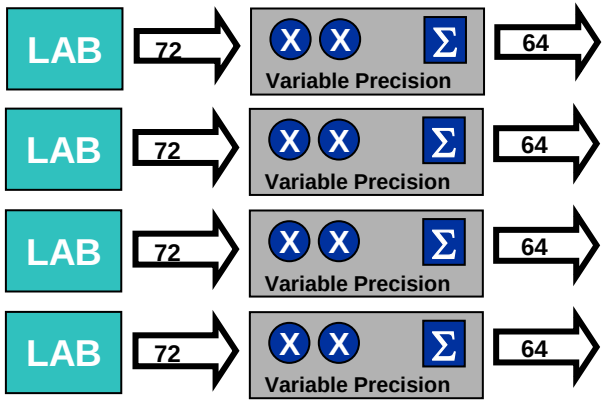
Four 18x18 (Independent)

Six 12x12 (Independent)

Two 36x36 (Independent)

Eight 9x9 (Independent)

Stratix V



Eight 18x18 (Sum)

Four 27x27 High Precision

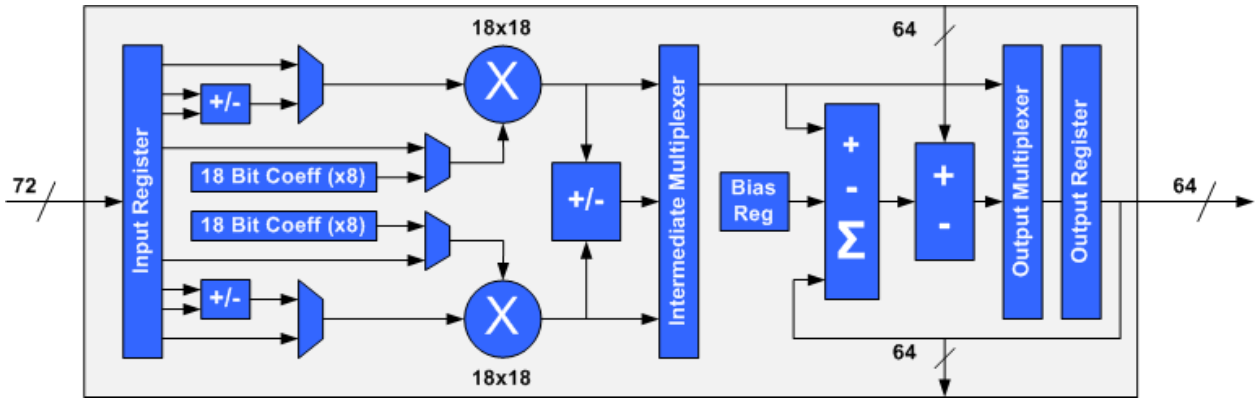
Four 18x36 High Precision

Plus more…..

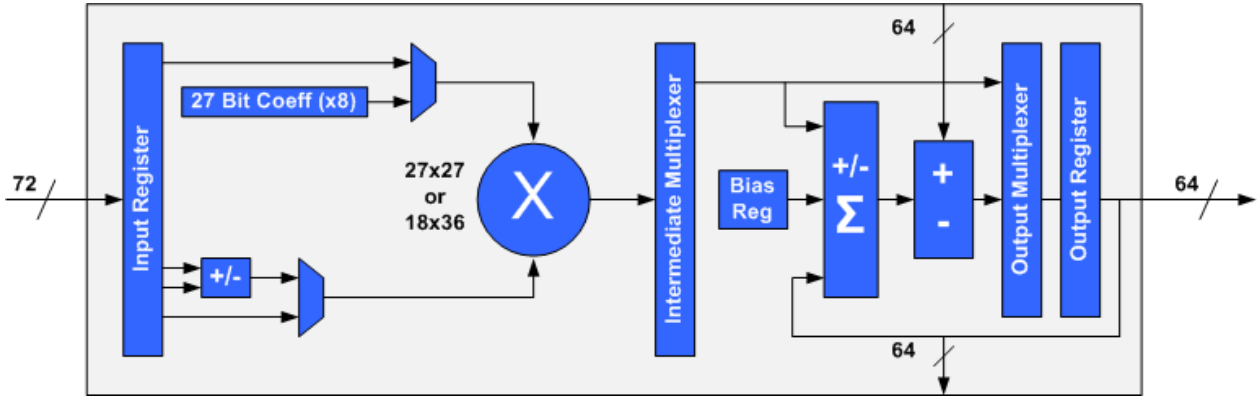
Highest performance, highest precision DSP at 28 nm

2000 GMACs or 1000 GFLOPs in single device

Variable Precision Stratix V DSP Block



18-bit Precision Mode

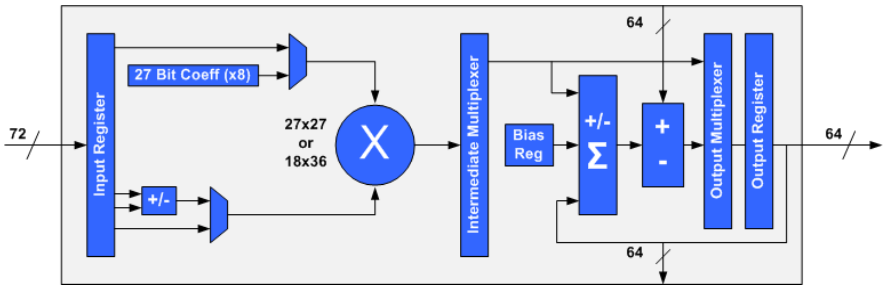


High Precision Mode

Support for Floating Point

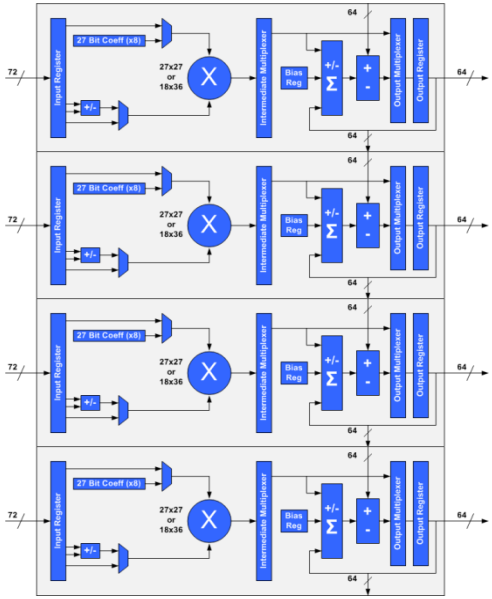
- Both single and double precision

Single-Precision Mantissa Multiplication



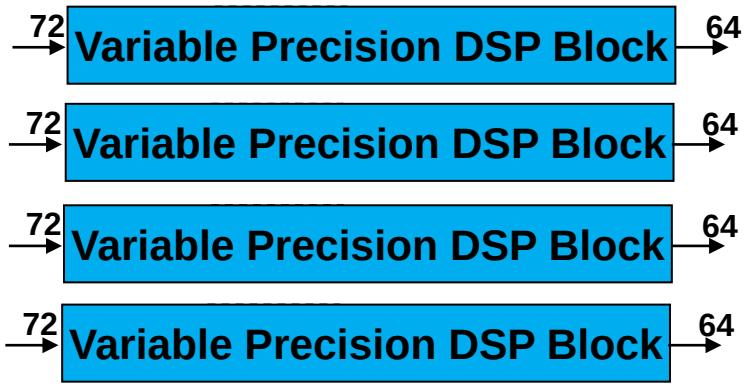
27x27

Double-Precision Mantissa Multiplication



54x54

Stratix V DSP Block Modes

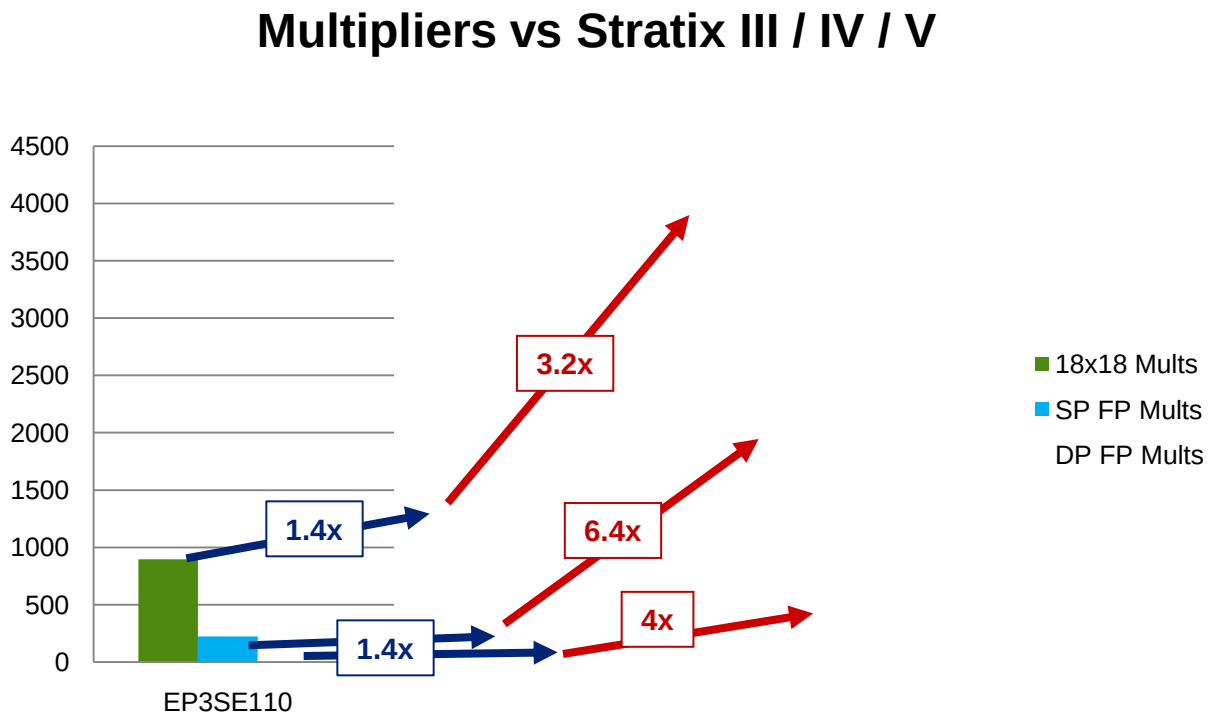


- **Floating Point**
  - (4) SPFP Mantissa Multiplier
  - (1) DPFP Mantissa Multiplier
- **Other Modes**
  - (4) A *B +/- C
  - (4) $(A - B)^2 + (C - D)^2$
- **Other Features**
  - Pre Adder
  - Rounder
  - Coefficient Banks
  - Cascade Adder

- **Independent Multiplier Modes**
  - (12) 9x9
  - (8) 16x16
  - (6) 18x18
  - (4) 18x25
  - (4) 27x27
  - (4) 18x36
  - (2) 36x36
  - (1) 54x54
- **MAC Mode (64 bit Acc)**
  - (4) Sum or Two 18x18 MAC
  - (4) 27x27 MAC
  - (4) 18x36 MAC (18x25 MAC)
- **Sum of Multiply**
  - (4) Sum of Two 18x18
  - (2) Sum of Four 18x18
  - (2) Sum of Two 18x36
  - (2) Sum of Two 27x27
- **Complex Multiply**
  - (2) 18x18
  - (1) 27x27
  - (1) 18x36

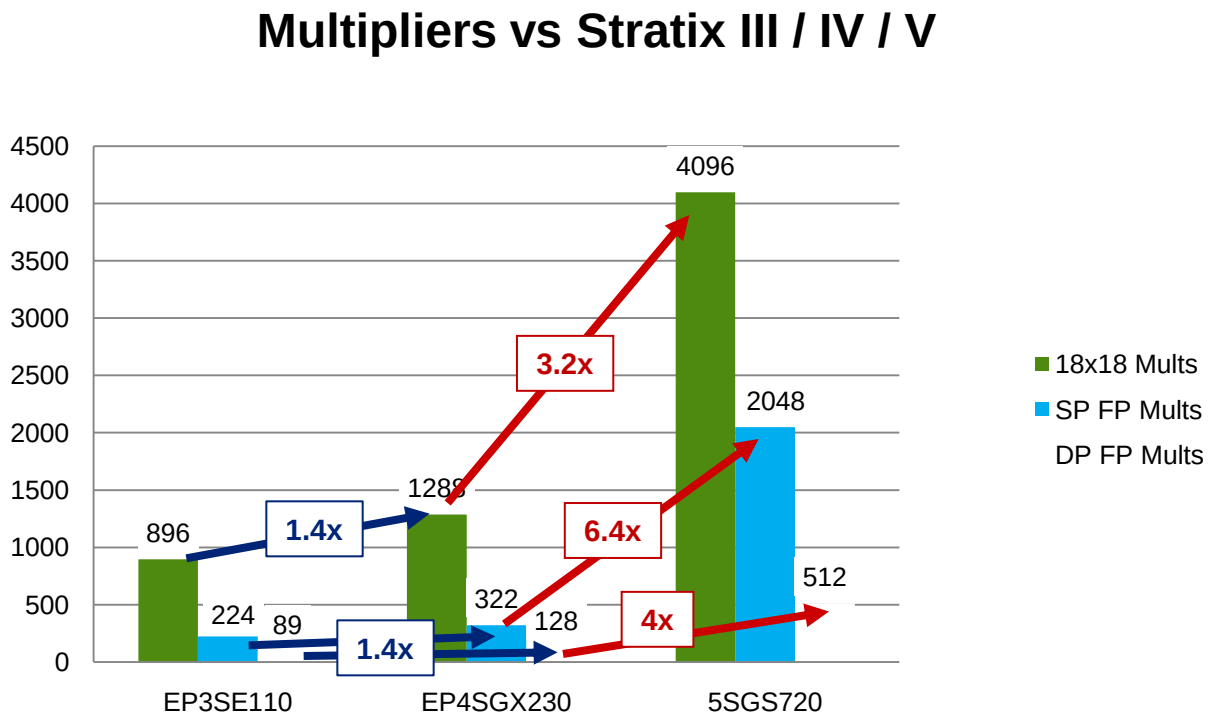
Floating Point Multiplier Capabilities

- **Floating point density largely determined by hard multiplier density**
  - Multipliers must efficiently support floating point mantissa sizes



Floating Point Multiplier Capabilities

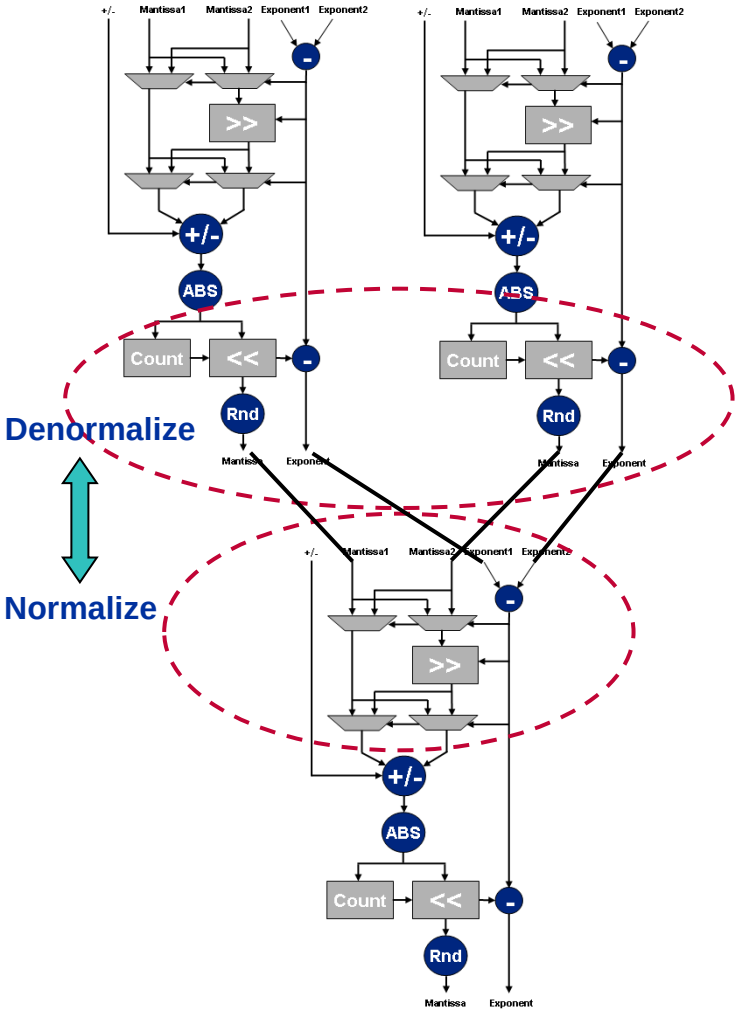
- **Floating point density largely determined by hard multiplier density**
  - Multipliers must efficiently support floating point mantissa sizes



“Fused Datapath” for Floating Point Functions

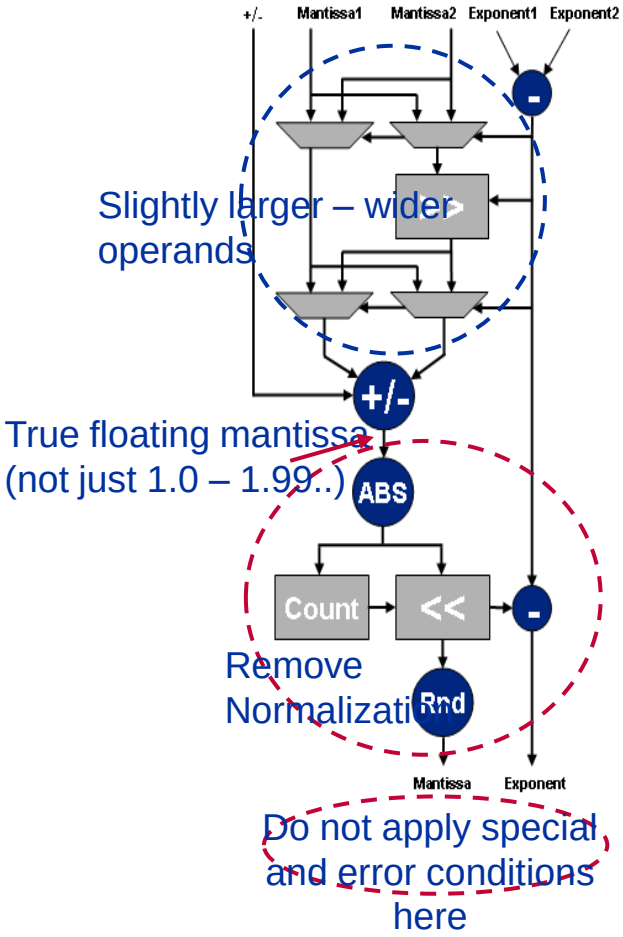
Floating Point Methodology

- Processors – each FP operation has data I/O in standardized IEEE754 format
- This can be done but not optimized in FPGAs
  - Excessive logic usage
  - Unsustainable routing requirements
  - Sub 100 MHz performance
  - This penalty discourages use of FP compared to fixed



New Floating Point Methodology

- Processors – each FP operation has data I/O in standardized IEEE754 format
- This can be done but not optimized in FPGAs
  - Excessive logic usage
  - Unsustainable routing requirements
  - Sub 100 MHz performance
  - This penalty discourages use of FP compared to fixed
- Altera has novel approach: “Fused Datapath”
  - IEEE754 interface only at algorithm boundaries
  - Large Reduction in logic and routing
  - Optimize algorithms to use hard multipliers
  - Single and Double Precision Floating Point support
  - Based upon internal C to datapath tool



Optimized "Fused Datapath" Cores

- IEEE754 interface only at algorithm boundaries
  - Large Reduction in logic and routing
  - Optimize algorithms to use hard multipliers

ADD/SUB	EXPONENT	ABS	MATRIX MULT
DIVIDE	INVERSE	COMPARE	MATRIX INVERT
MULTIPLY	LOG	CONVERT	Sine
SQ ROOT	INV SQ ROOT	FFT	Cosine

Largest Portfolio of floating-point cores

Pick “multiplier intense” algorithms to build floating point functions

Function	ALUTs	Register	Multipliers (27x27)	Latency	Performance
ALU	541	611	n/a	14	497 MHz
Multiplier	150	391	1	11	431 MHz
Divider	254	288	4	14	316 MHz
Inverse	470	683	4	20	401 MHz
SQRT	503	932	n/a	28	478 MHz
Inverse SQRT	435	705	6	26	401 MHz
EXP	626	533	5	17	279 MHz
LOG	1889	1821	2	21	394 MHz

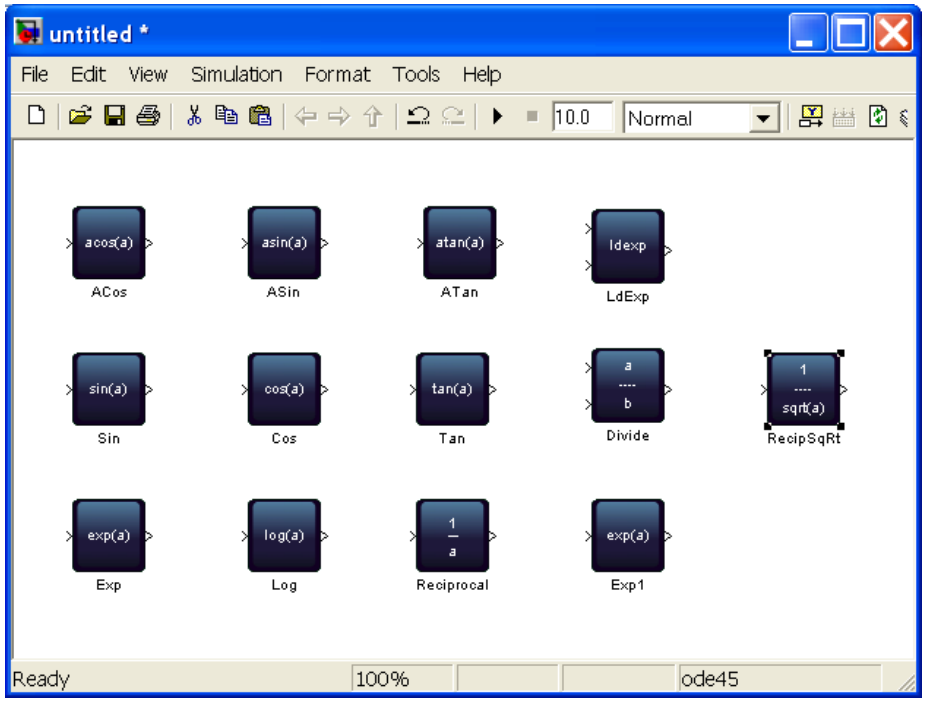
Little difference between add/subtract and common MATH.H functions
CPU can have orders of magnitude cost difference: GOPS ≠ GFLOPS
Stratix Series FPGAs: GOPS ≈ GFLOPS

Fused Data-path Technology: Floating Point FFT MegaCore

One FFT 1024pt Core	FPGA Resources	Actual Resources	Percentage
ALUTs	17,668	8,080	31%
Registers	16,068	58,080	28%
Logic utilization	24,104	58,080	42%
Block memory bits	140,340	6,617,088	2%
M9K Blocks	89	462	19%
Multipliers	64	384	17%
Fmax	**320.62 MHz**		
Fourteen FFT 1024pt Cores			
ALUTs	237,235	424,960	56%
Registers	254,671	424,960	60%
Logic utilization	343,787	424,960	81%
Block memory bits	1,942,304	21,233,664	9%
M9K Blocks	1162	1280	91%
Multipliers	896	1024	88%
Fmax	**299.13 MHz**		

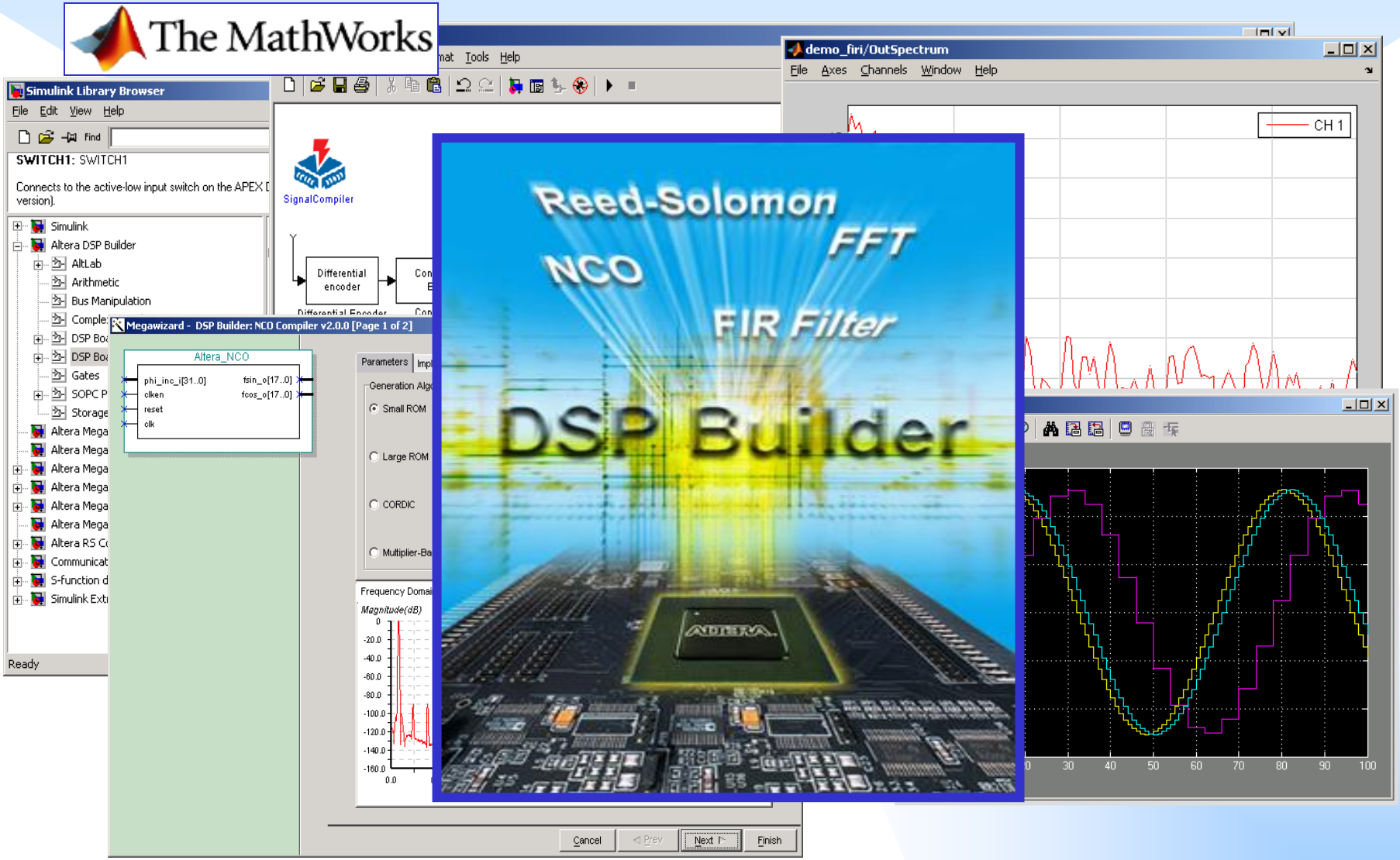
“Fused Datapath” integrated into DSP Builder

- MATH.H
  - SIN
  - COS
  - TAN
  - ASIN
  - ACOS
  - ATAN
  - EXP
  - LOG
  - SQRT
  - DIVIDE
  - SQRT



What is DSP Builder ?

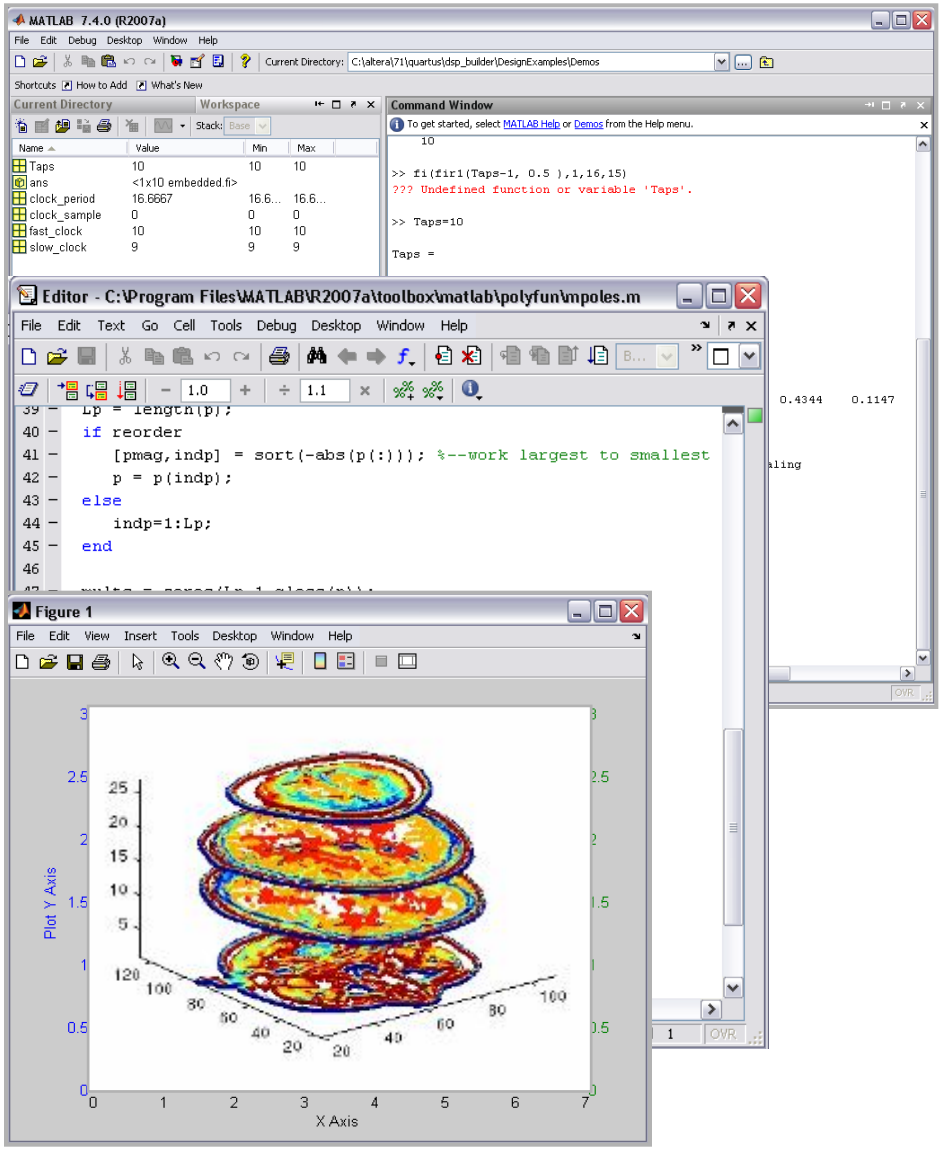
Simulink Model Based Algorithm Development



Mathworks' Matlab

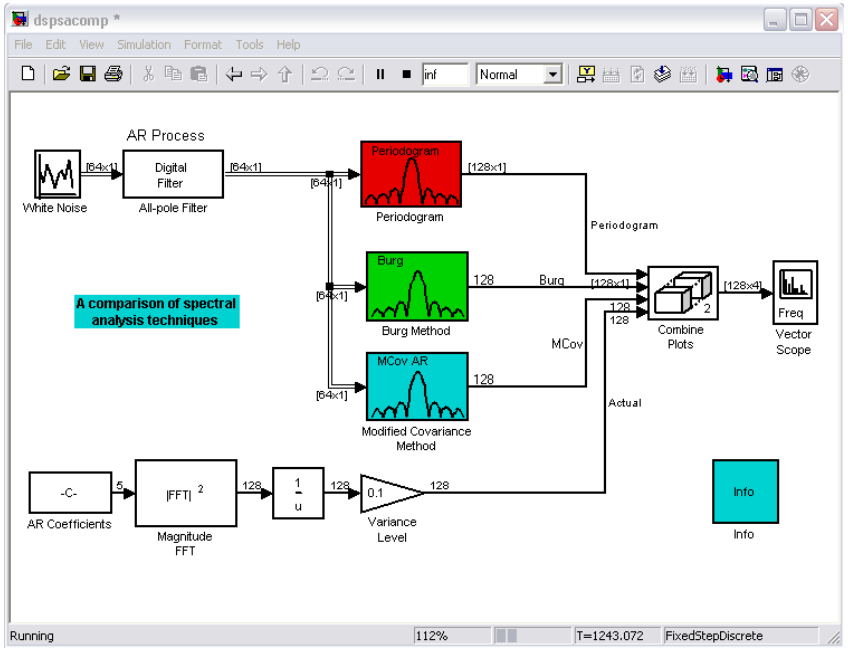
- Computer language of DSP engineers
- Matlab
  - Develop algorithms and applications
  - Analyze and access data
  - Visualize data
  - Perform numeric calculations

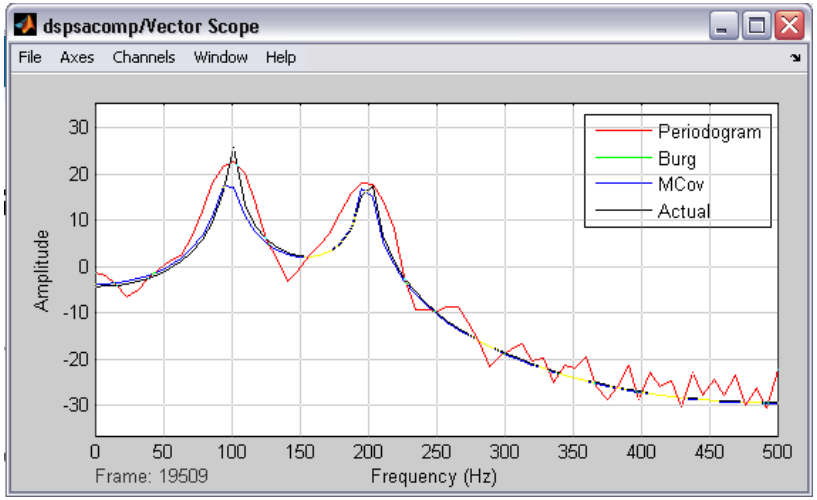
Matlab/Simulink must be purchased separately from The Mathworks



Why Simulink ?

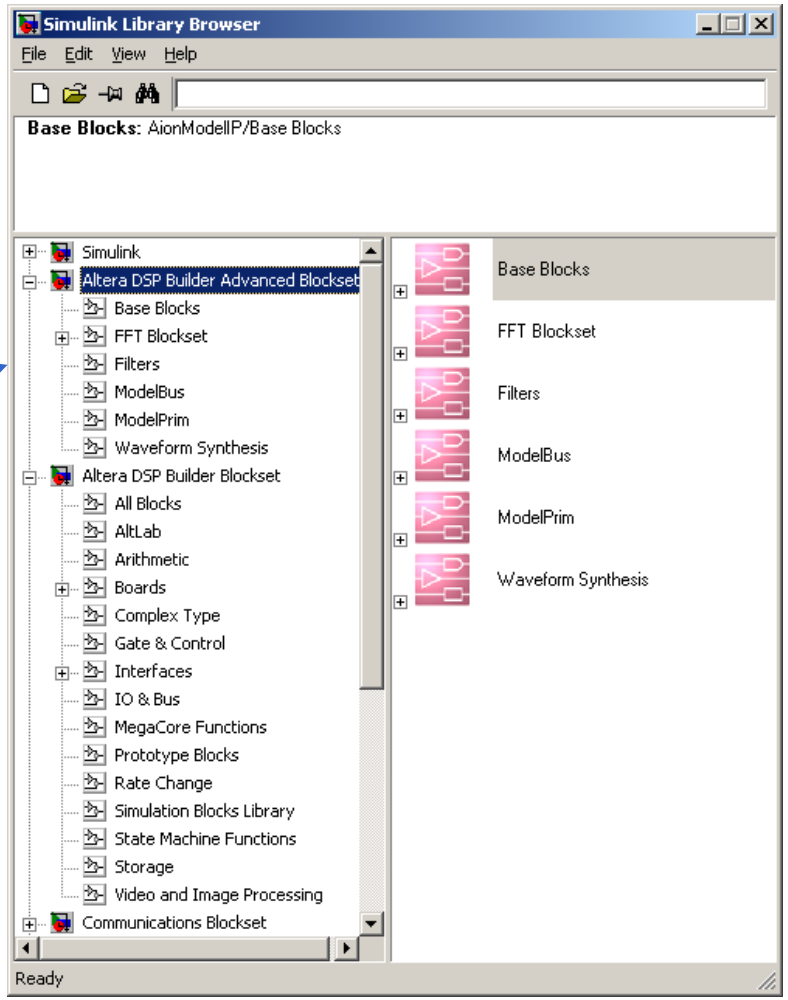
- Dynamic graphical modeling environment
- Simulink
  - Dynamically develop entire systems
  - Simulate and interact with the system
  - Explore different architectures
  - Analyze results
- Simulink adds the concept of cycles (clocks)
  - Hardware realizable



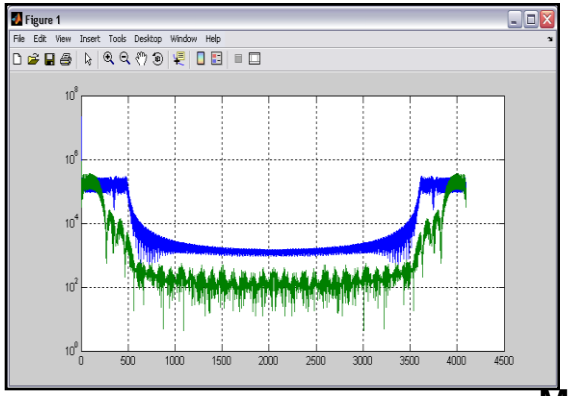


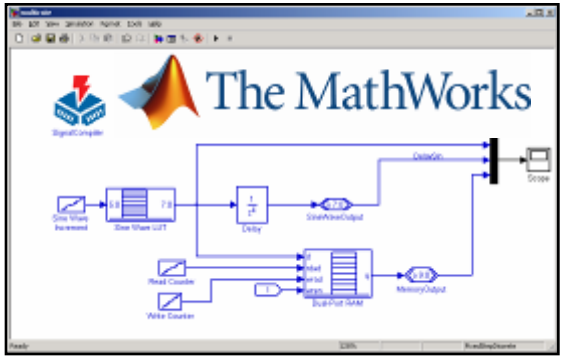
Simulink Blockset Libraries

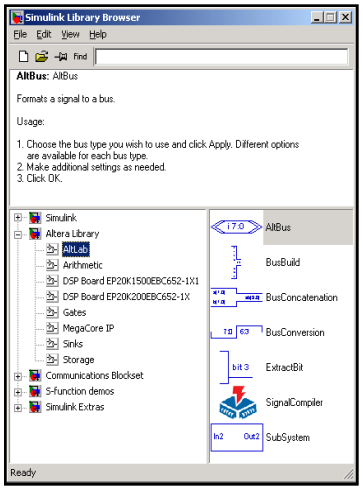
- Simulink
  - Sources
  - Sinks
  - Continuous
  - Discrete
  - Non-linear
  - Math
- *Altera DSP Builder Advanced Blockset*
- *Altera DSP Builder Blockset*
- Fixed-point blockset
- DSP blockset
- Real-time workshop
- Communications blockset
- Image acquisition toolbox
- *Many others…*



DSP Builder Design Flow

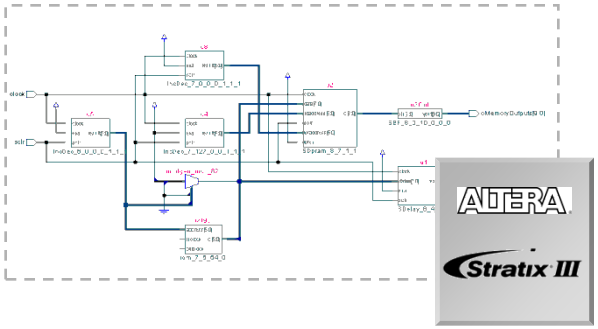






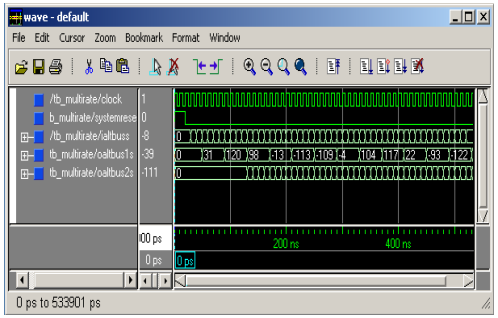
Matlab/Simulink domain
(System simulation and verification)

HDL/hardware domain
(Hardware implementation/RTL simulation)



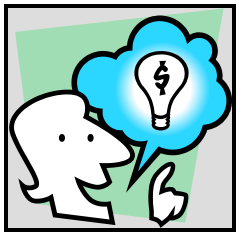
QUARTUS® II

SOPC Builder

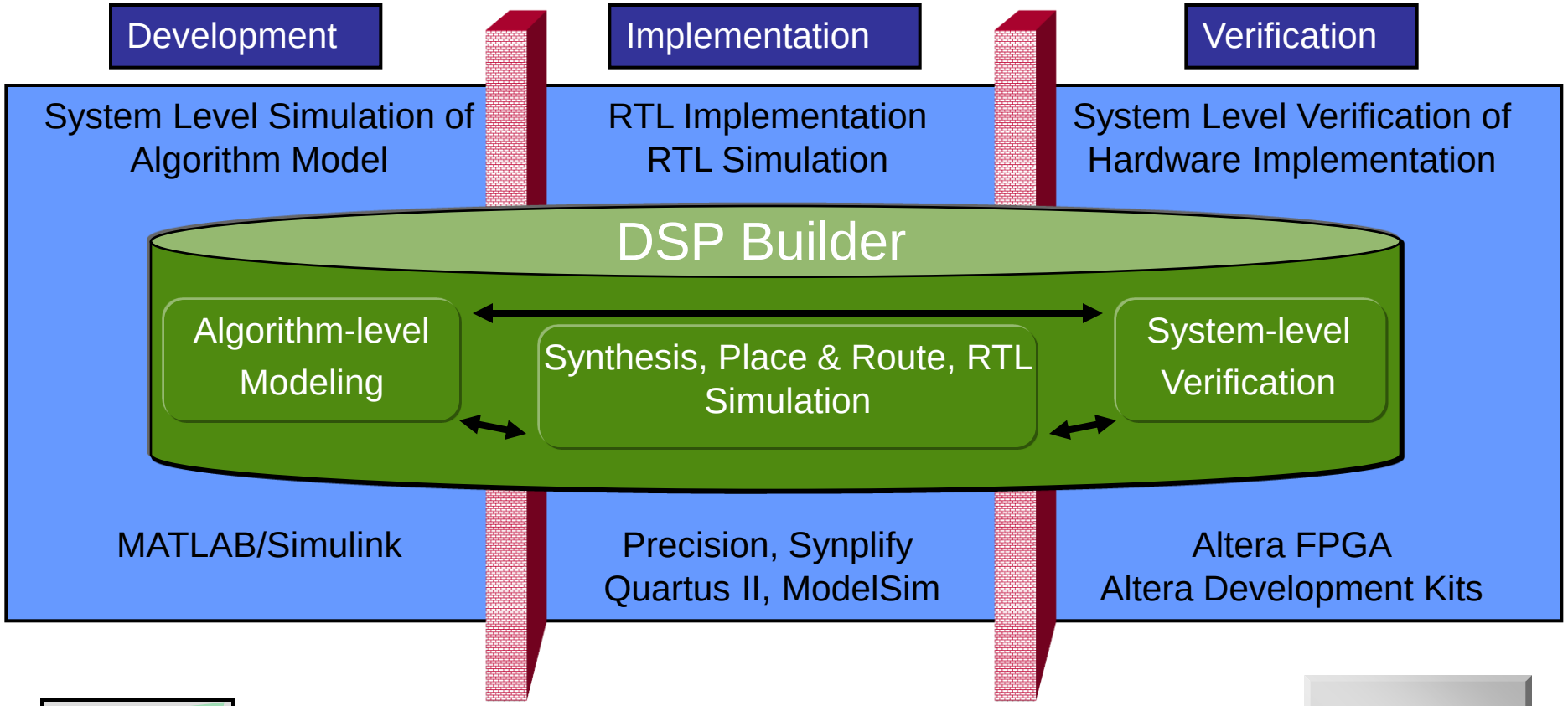


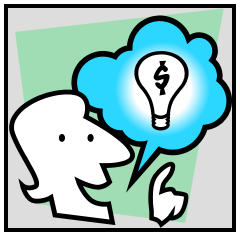
DSP System Level Design Flow

Development	Implementation	Verification
System Level Simulation of Algorithm Model	RTL Implementation RTL Simulation	System Level Verification of Hardware Implementation
Algorithm-level Modeling	Synthesis, Place & Route, RTL Simulation	System-level Verification
MATLAB/Simulink	Precision, Synplify Quartus II, ModelSim	Altera FPGA Altera Development Kits



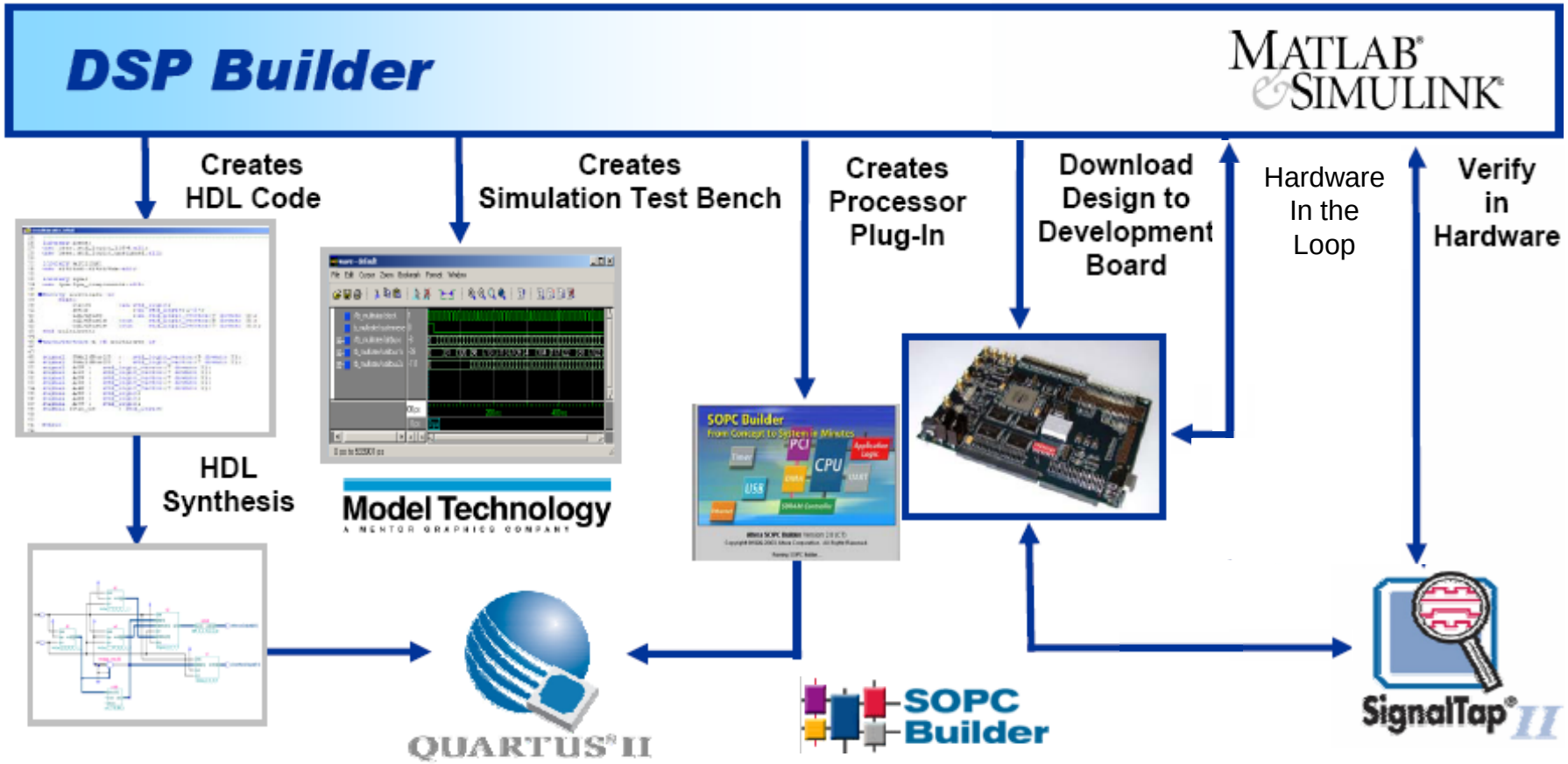
DSP Builder System Level Design Flow





ALTERA

DSP Builder Overview



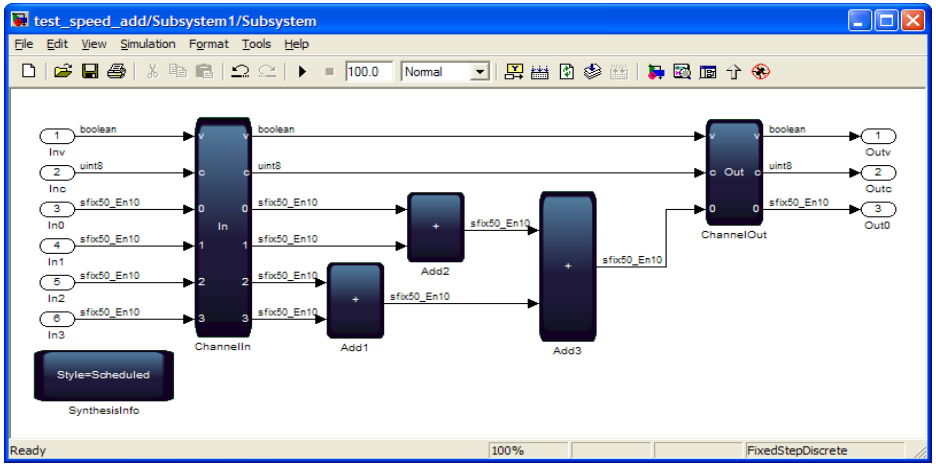
DSP Builder Advanced Blockset

DSP Builder Advanced Blockset

- Constraint-driven behavioral design
  - Set desired clock frequency
- Models silicon speeds when translating to HDL
  - Different devices families / speed grades result in different HDL
- Automated resource sharing
  - Tool analyzes clock rate in relation to sample rate, # of channels, interpolation/decimation factor and develops efficient HDL
- Automated pipelining
  - To meet desired clock rate
  - Enable timing closure at high clock rates of 400-500 MHz

Increased Productivity by Closing Timing Faster

Unique HDL for Different fmax Requirements

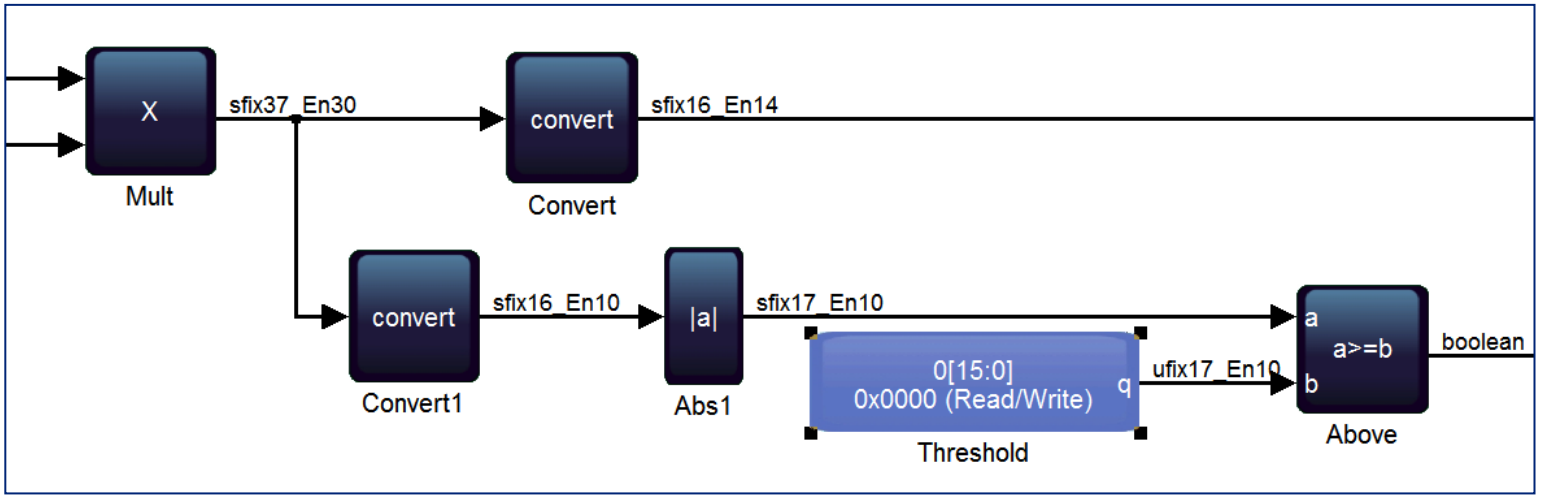


- Simply enter desired System Clock Frequency,
- No need to change model
- Simple 50-bit 4-input adder tree
  - 100 MHz Target => 118 LUT4s, 121 MHz, No pipeline
  - 200 MHz Target => 175 LUT4s, 286MHz, 1 stage pipeline
  - 400 MHz Target => 350 LUT4s, 581 MHz, 5 stage pipeline

Timing driven synthesis produces small or fast RTL from same model

ModelPrim: Zero Latency Blocks

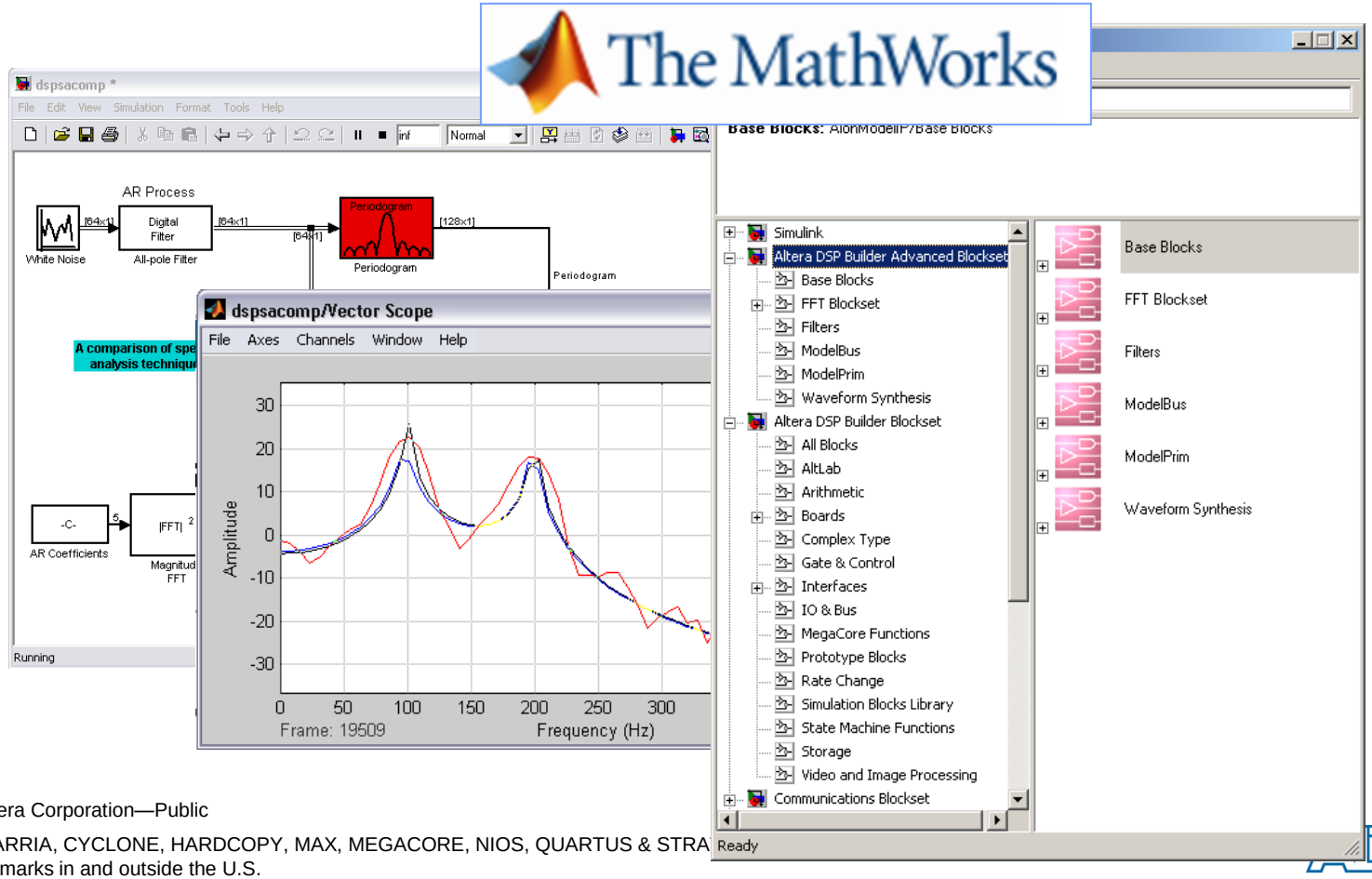
- Blocks are behavioural in nature
  - *What* to do, not *When* to do it
  - Focus on signal flow representation
- Much easier debug and modify without pipeline



Behavioural input enables Optimizations

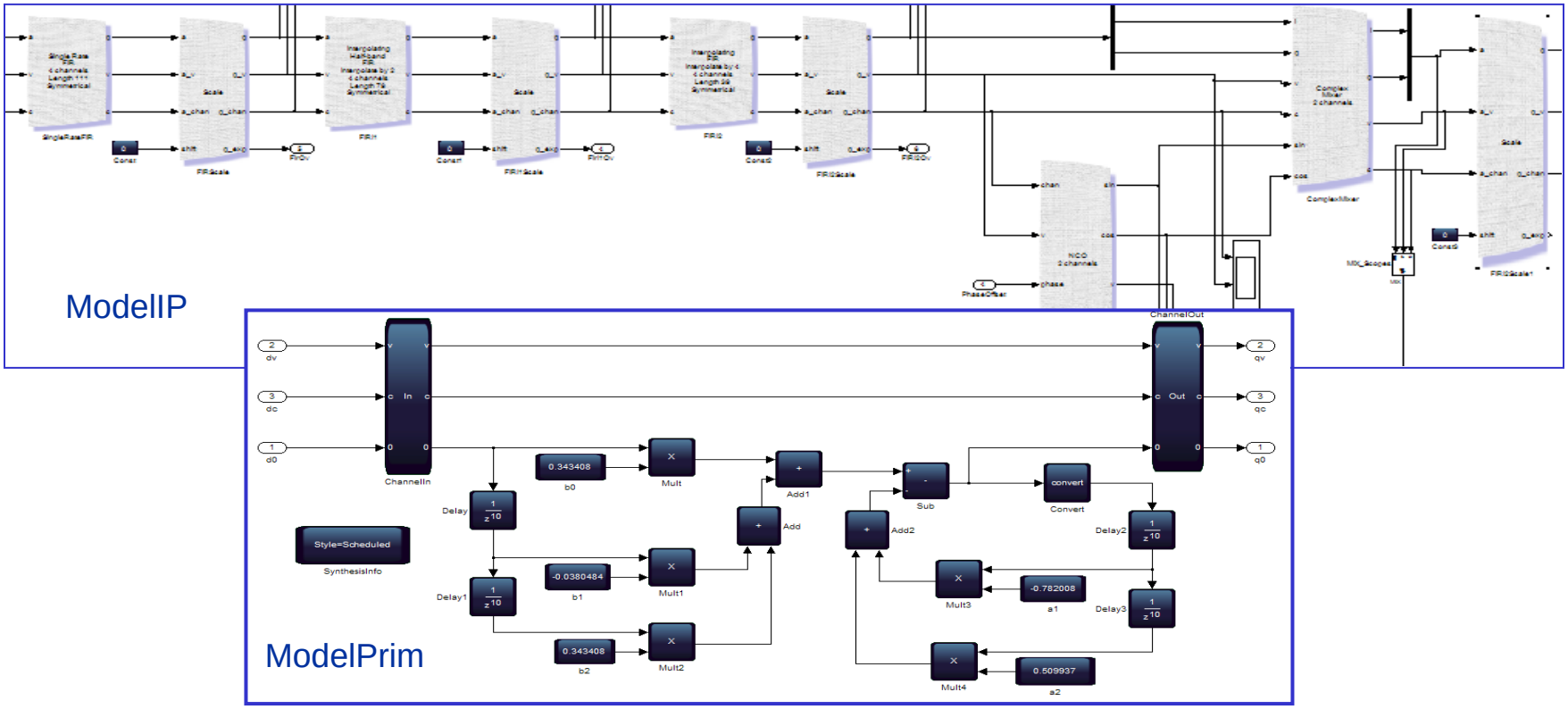
DSP Builder Advanced Blockset Feature Overview

Works Within the Industry's DSP Development Environment



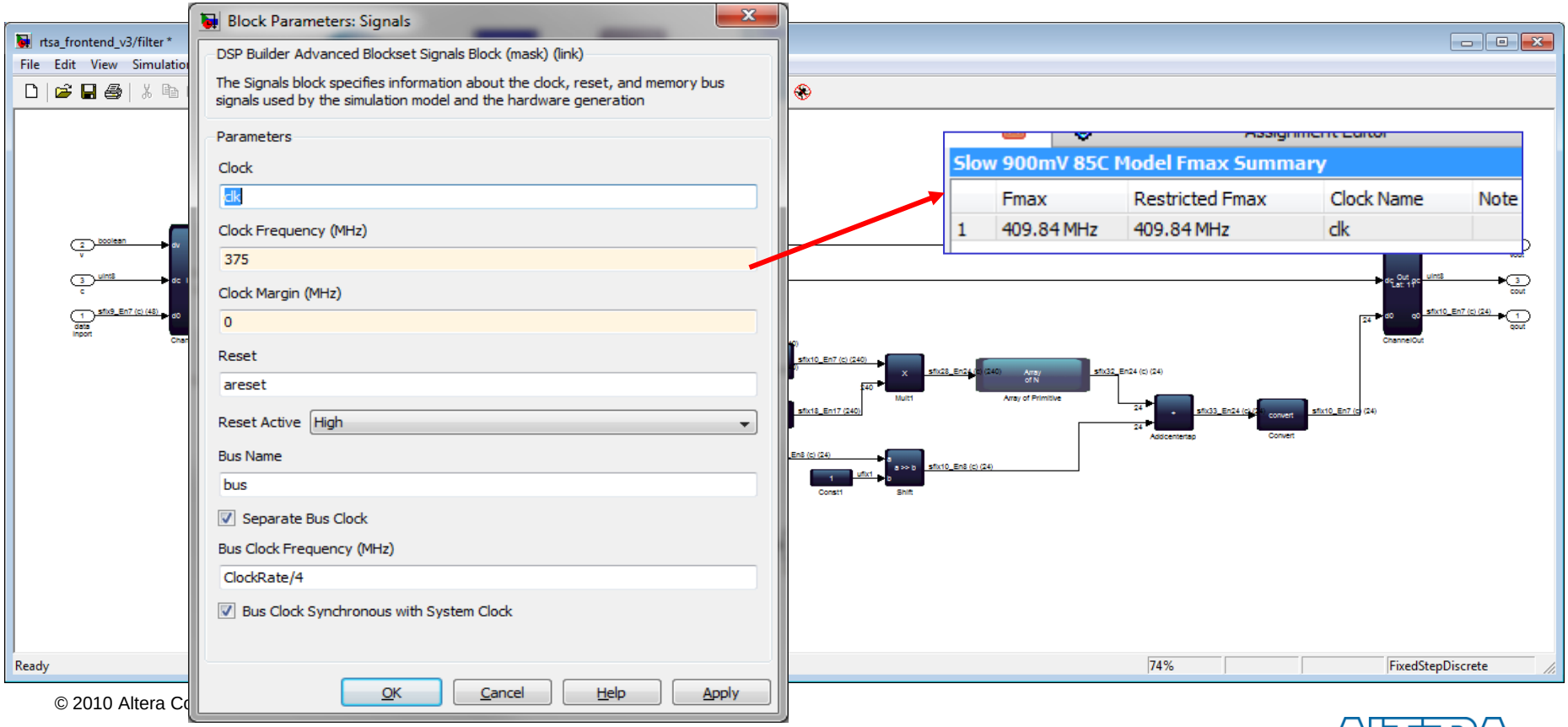
Allows You to Design Behaviorally

- No need for knowledge of silicon features



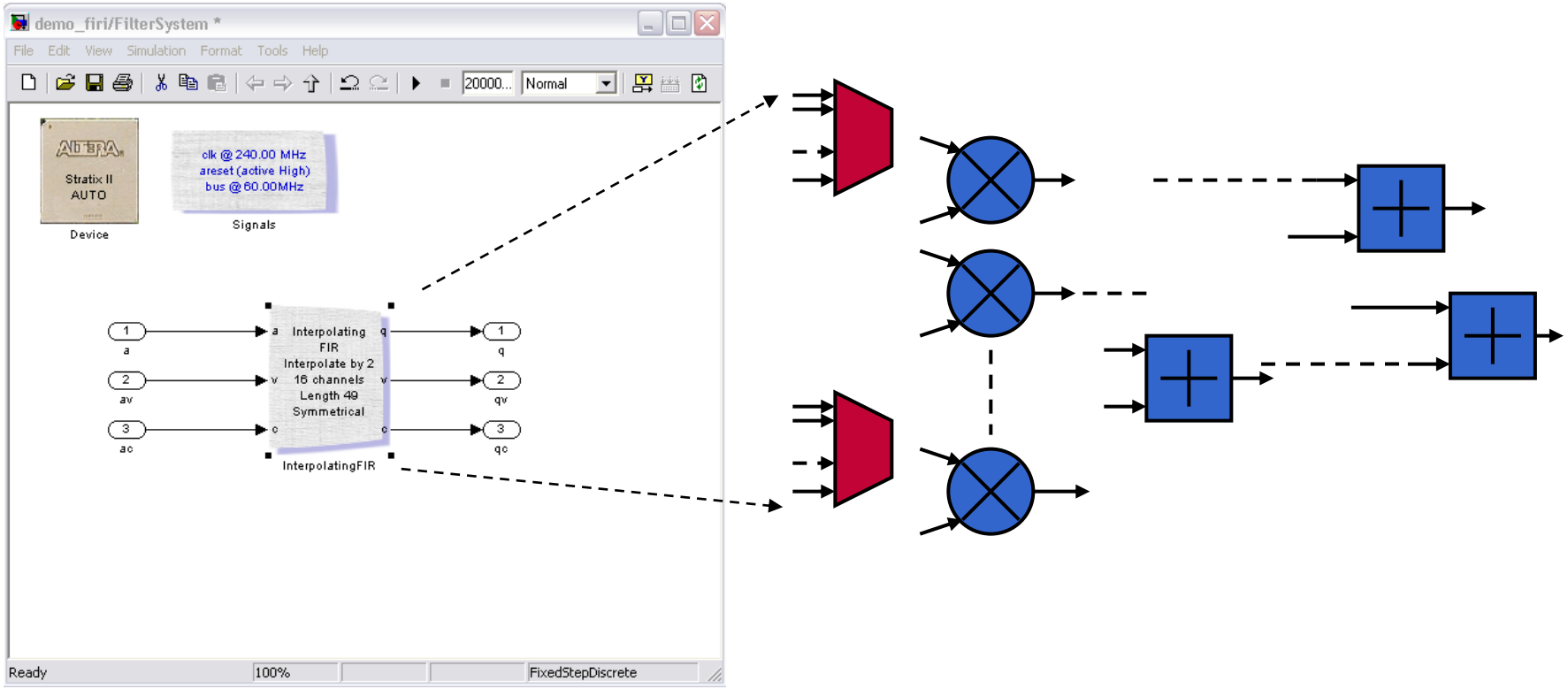
Automatically Write HDL to Meet Fmax

- Unique HDL written to meet Fmax without any intervention



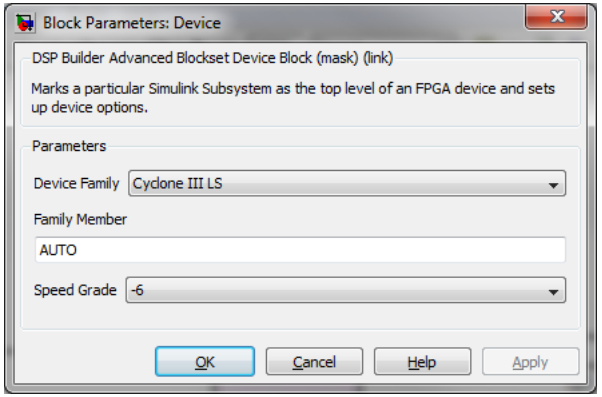
Efficiently Time Shares Resources

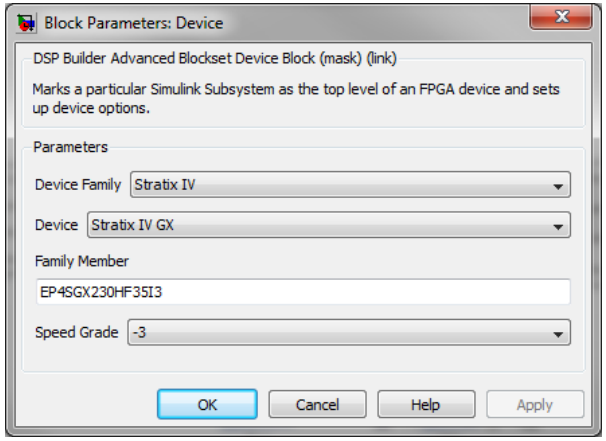
- Automatically time shares resources efficiently when clock rate > sample rate

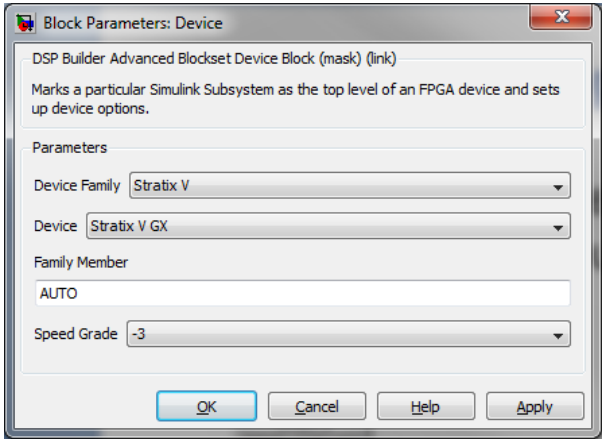


Design Space Exploration Across Families

- Lets you target different device families without changing any of your design

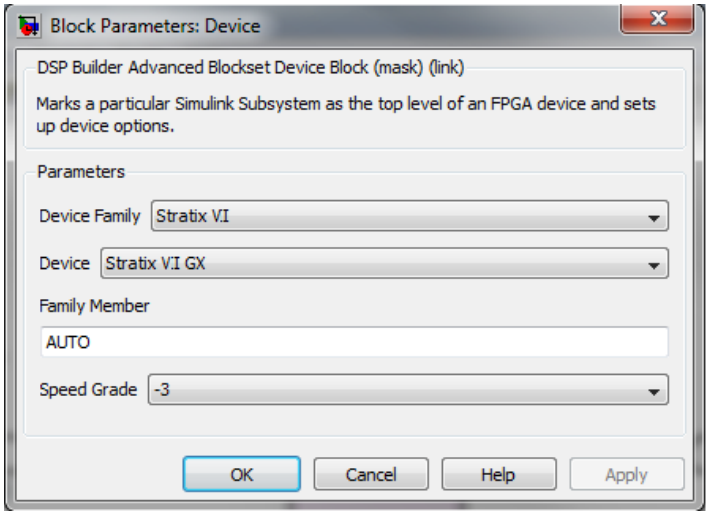


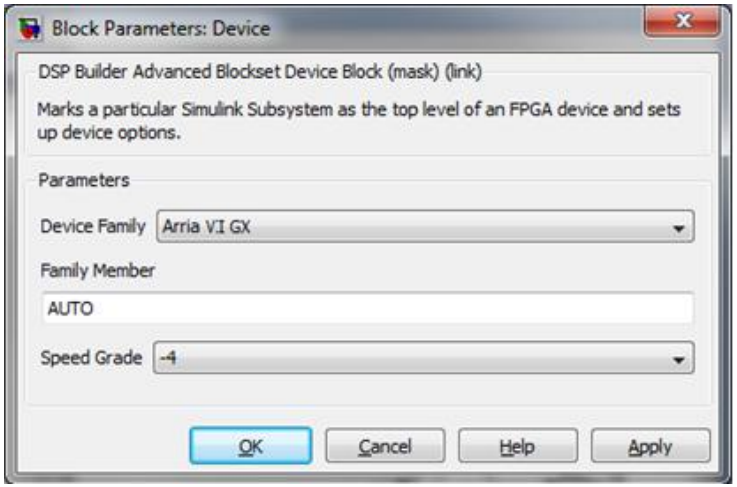




“Future Proof” Your Designs

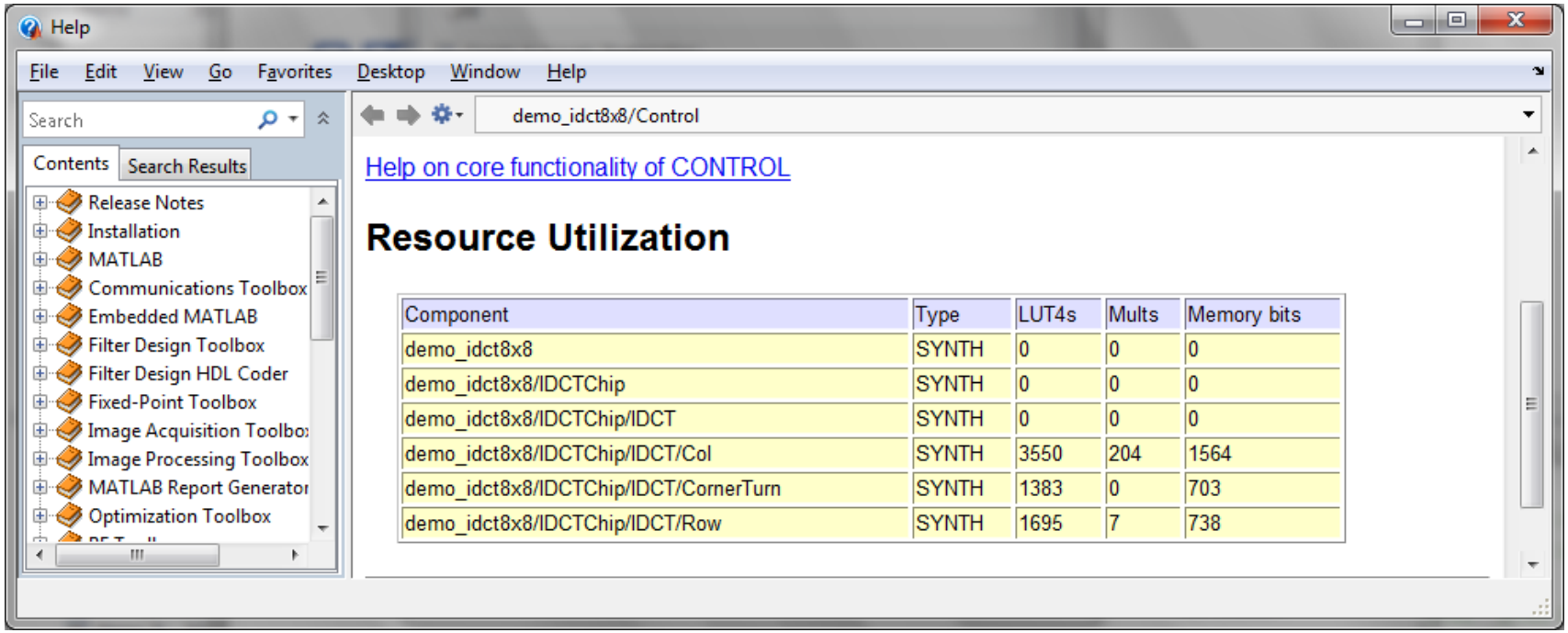
- Lets you target **future** devices families without changing any of your design





Obtain Accurate Resources

- No Need To Compile….

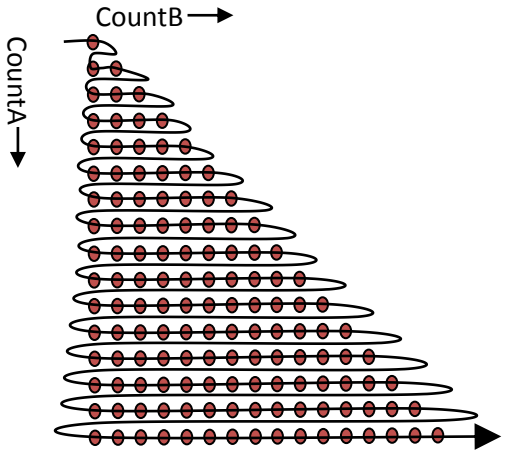


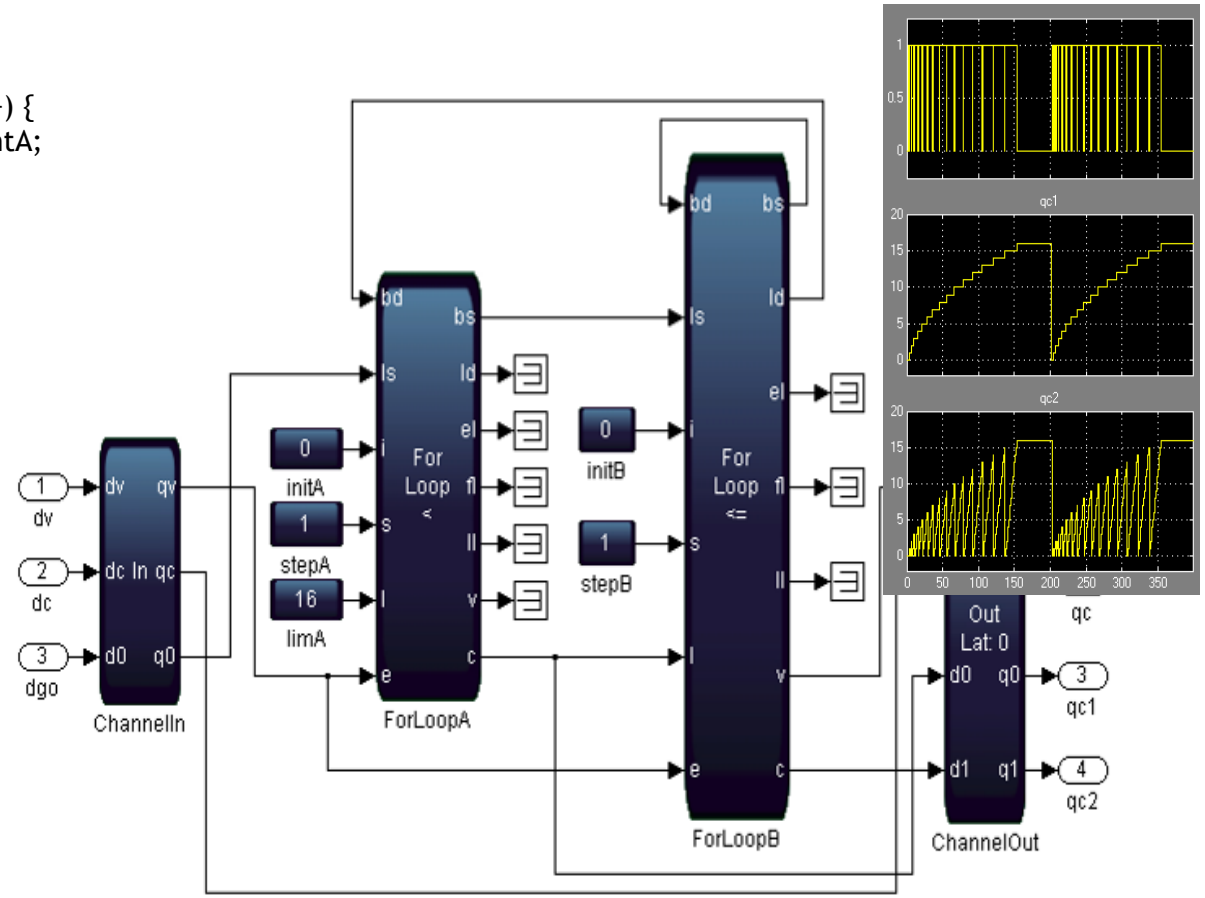
Component	Type	LUT4s	Mults	Memory bits
demo_idct8x8	SYNTH	0	0	0
demo_idct8x8/IDCTChip	SYNTH	0	0	0
demo_idct8x8/IDCTChip/IDCT	SYNTH	0	0	0
demo_idct8x8/IDCTChip/IDCT/Col	SYNTH	3550	204	1564
demo_idct8x8/IDCTChip/IDCT/CornerTurn	SYNTH	1383	0	703
demo_idct8x8/IDCTChip/IDCT/Row	SYNTH	1695	7	738

Supports Complex Control Logic

- Nested and Sequential For Loops

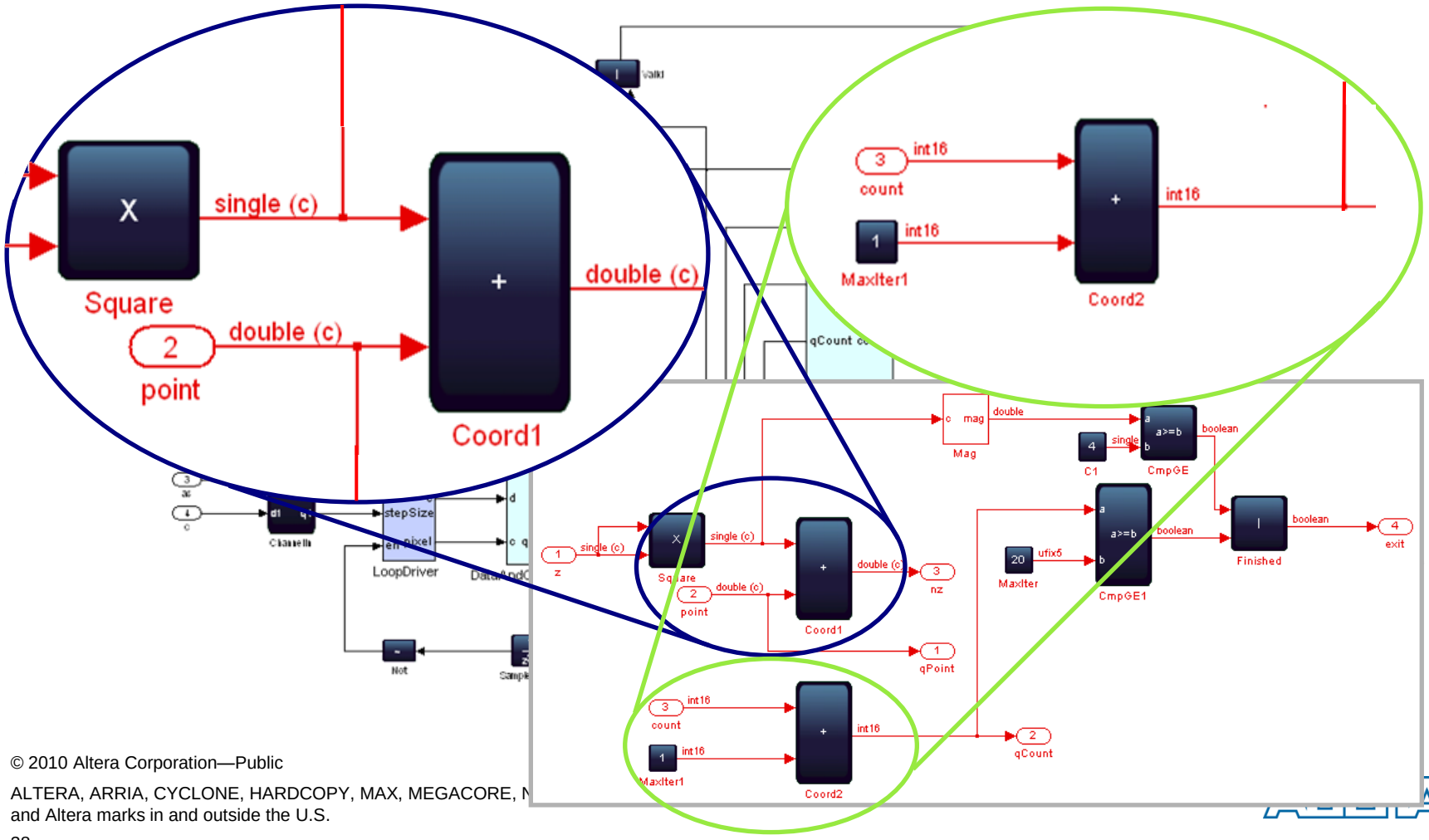
```
for (uint8 countA=0; countA<16; countA++) {
        for (uint8 countB=0; countB<=countA;
        countB++) {
                qc1 = countA;
                qc2 = countB;
        }
}
```





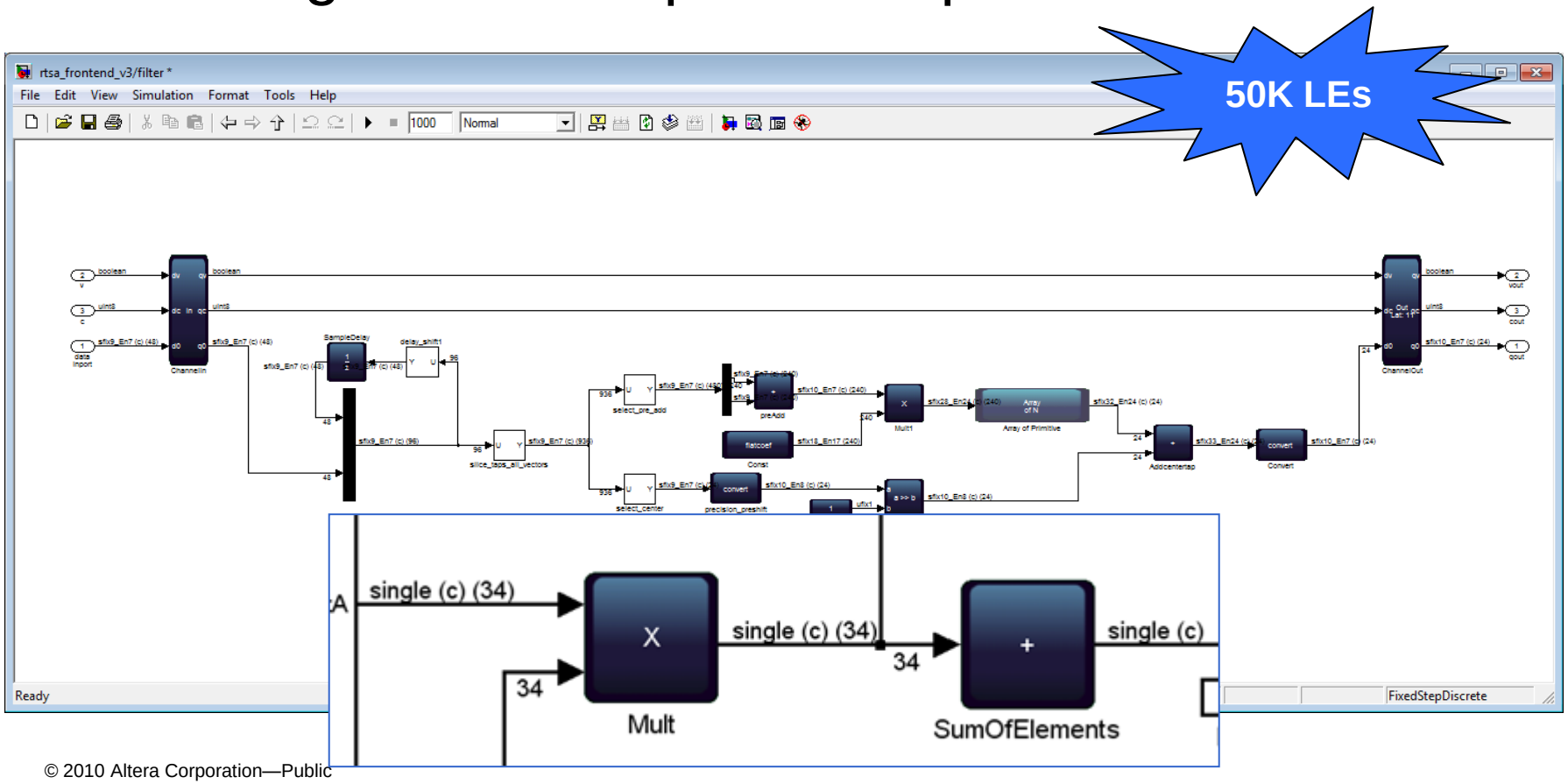
Fixed and Floating Point in Same Model

- Complex as well (c) = complex data path



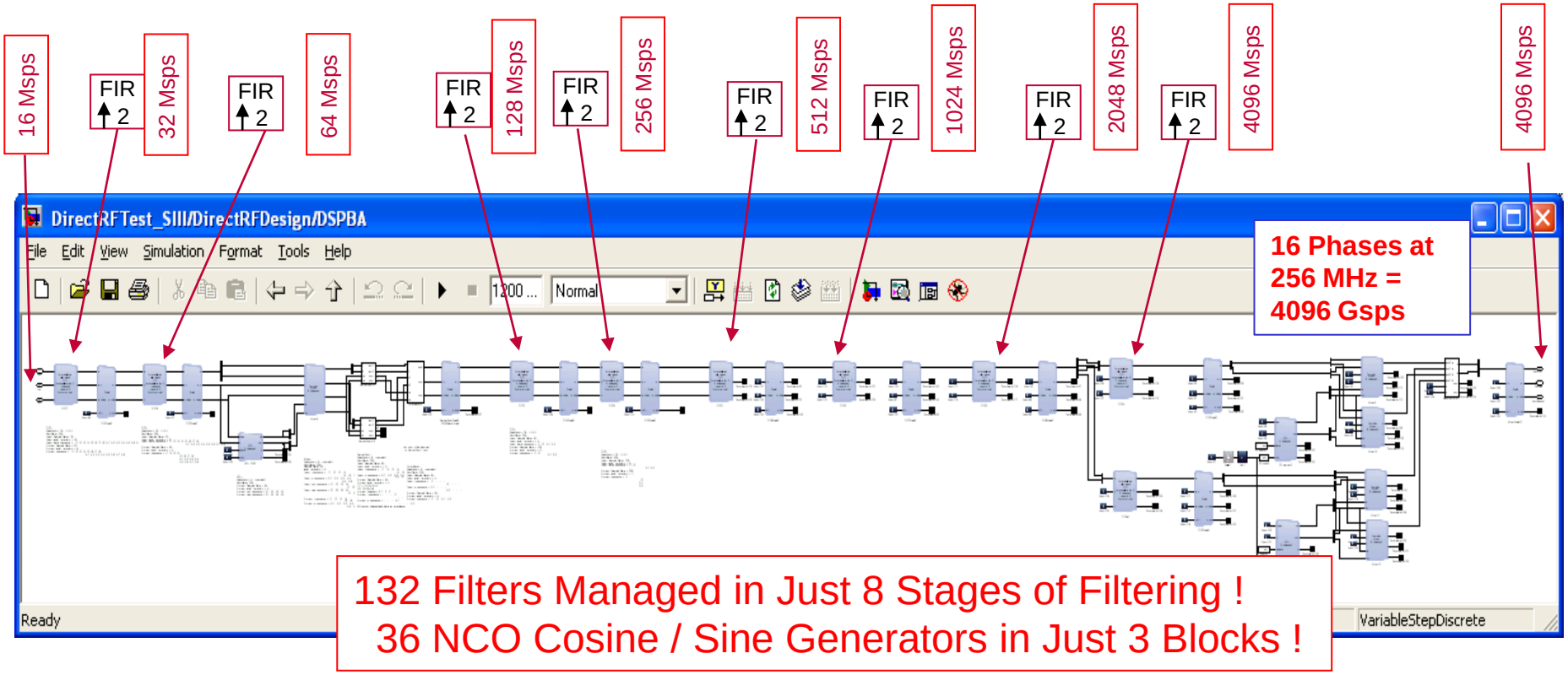
Works with Vectorized Busses

- Single Multiplier in Model = 34 Single Precision Floating Point Complex Multipliers



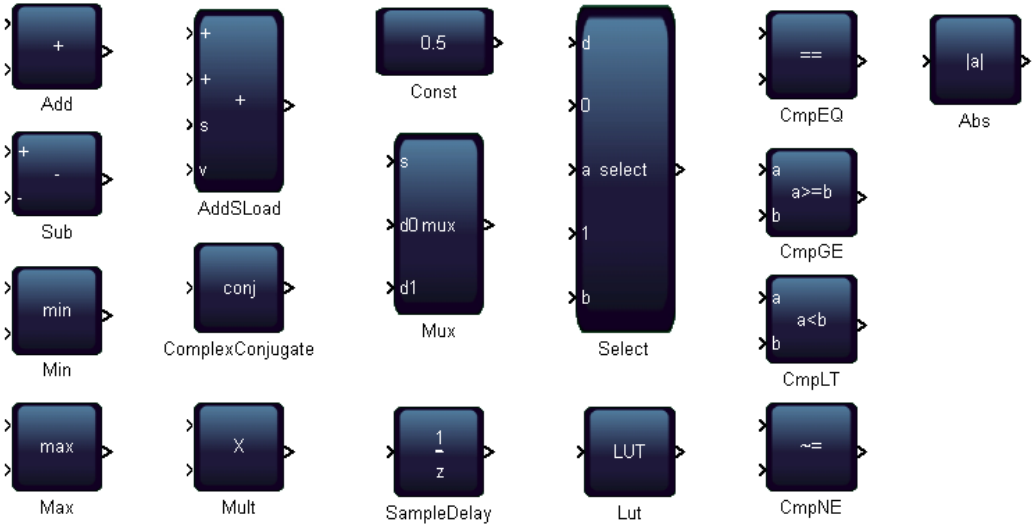
Direct RF Support

- Supports sample rates >> clock rate
- Automatically manages multiple phases



“Fused Datapath” integrated in DSPBuilder

Standard Blocks



Math.h Library

Simplifies Realization of Complex Equations

- Black-Scholes Example:

$$C = SN(d) - Le^{-rt}N(d - \sigma\sqrt{t})$$

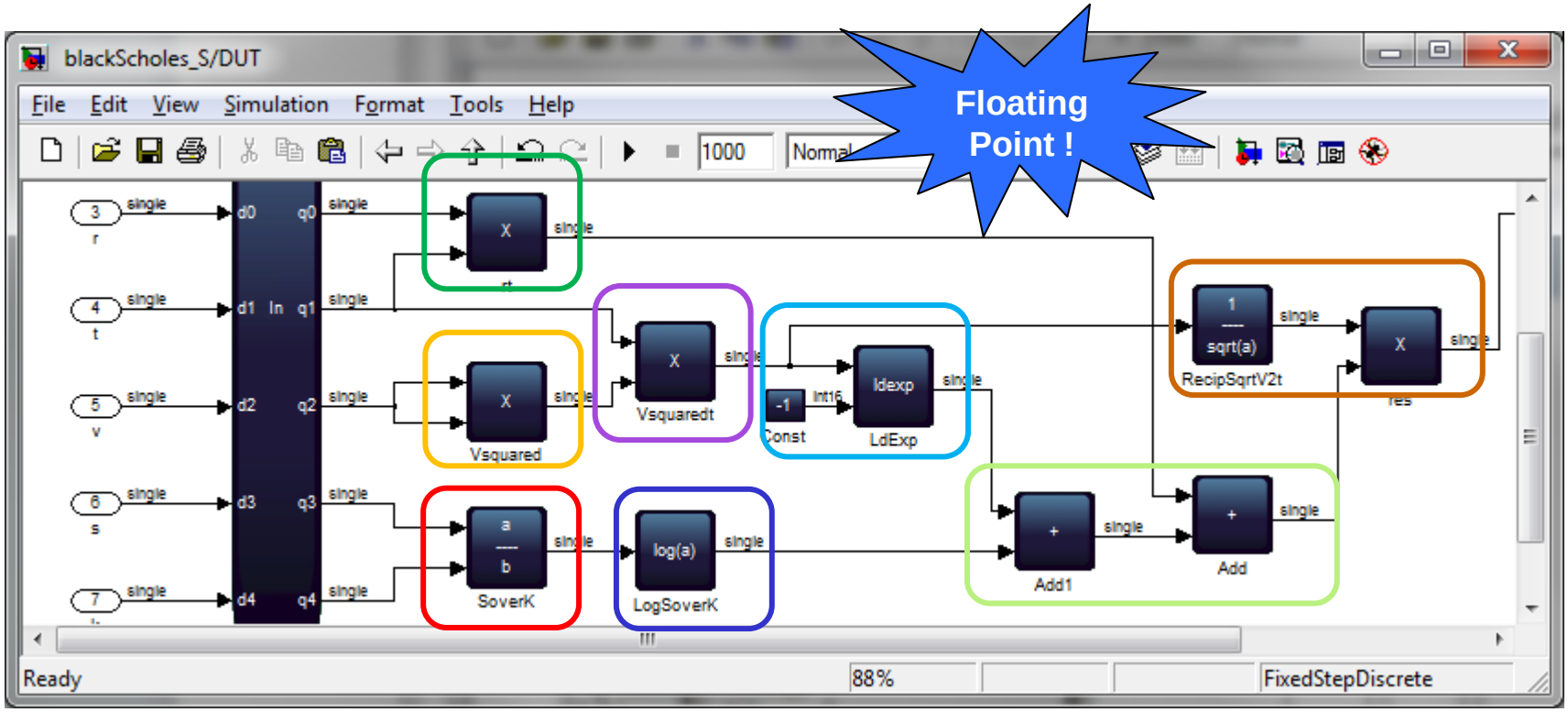
where the variable *d* is defined by:

$$d = \frac{ln\frac{S}{L} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

d = (ln(S/L) + v*v*t/2 + rt)/sqrt(v*v*t)

Simplifies Realization of Complex Equations

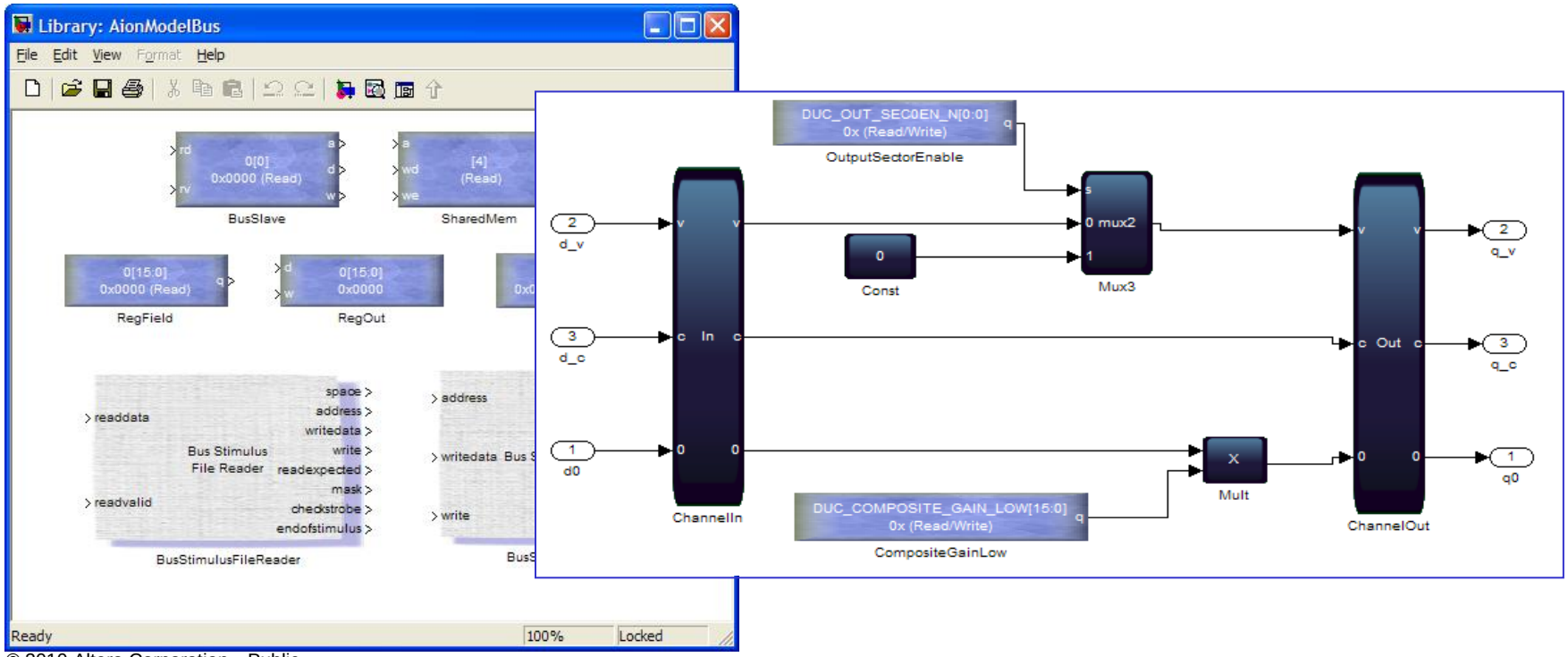
- Drop down blocks for operators…



d = (ln(S/L) + v*v*t/2 + rt)/sqrt(v*v*t)

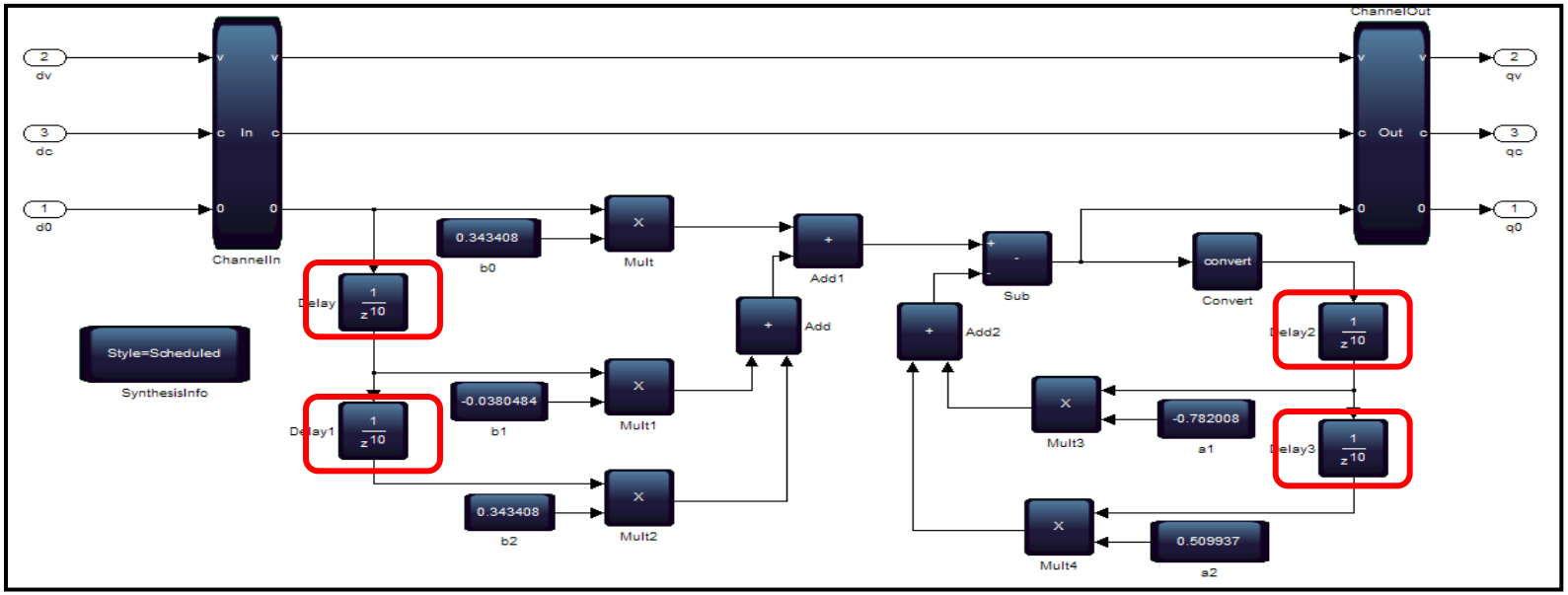
Memory Synthesis

- Automated processor ⇔ hardware interface synthesis
- Builds pipelined memory mapped interface logic for:
  - IP (eg filter coefficients)
  - Primary Systems (eg registers, shared memories)



Multi-channel Designs

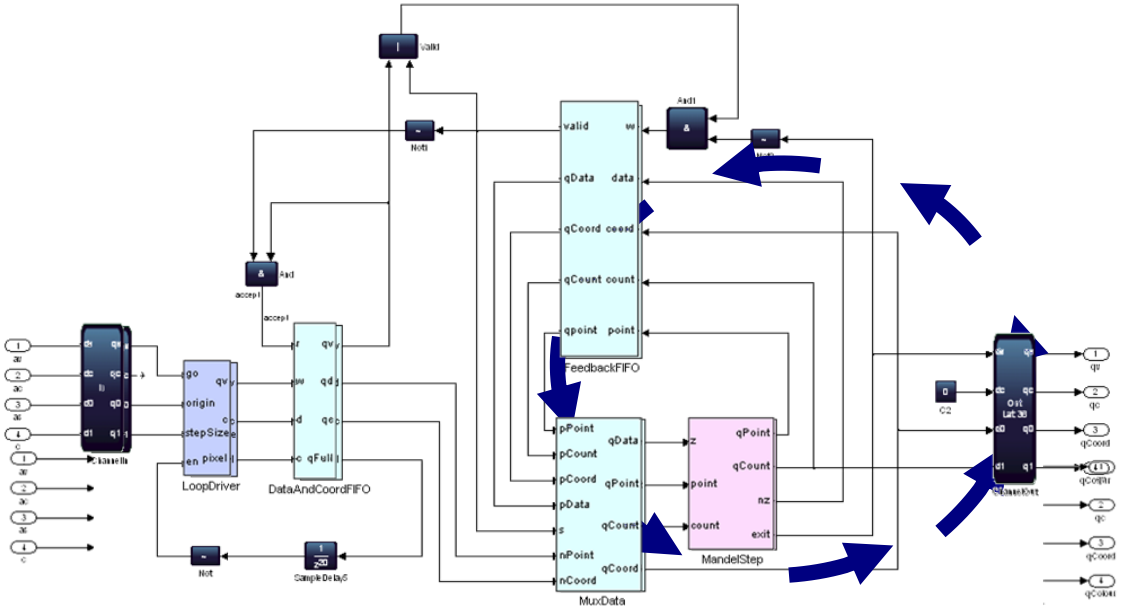
- IIR example uses ‘textbook’ lumped delays
- Replace registers with number of channels
- Delays are distributed around logic to meet fmax goal
- Processes multiple channels simultaneously



Supports Flow Control

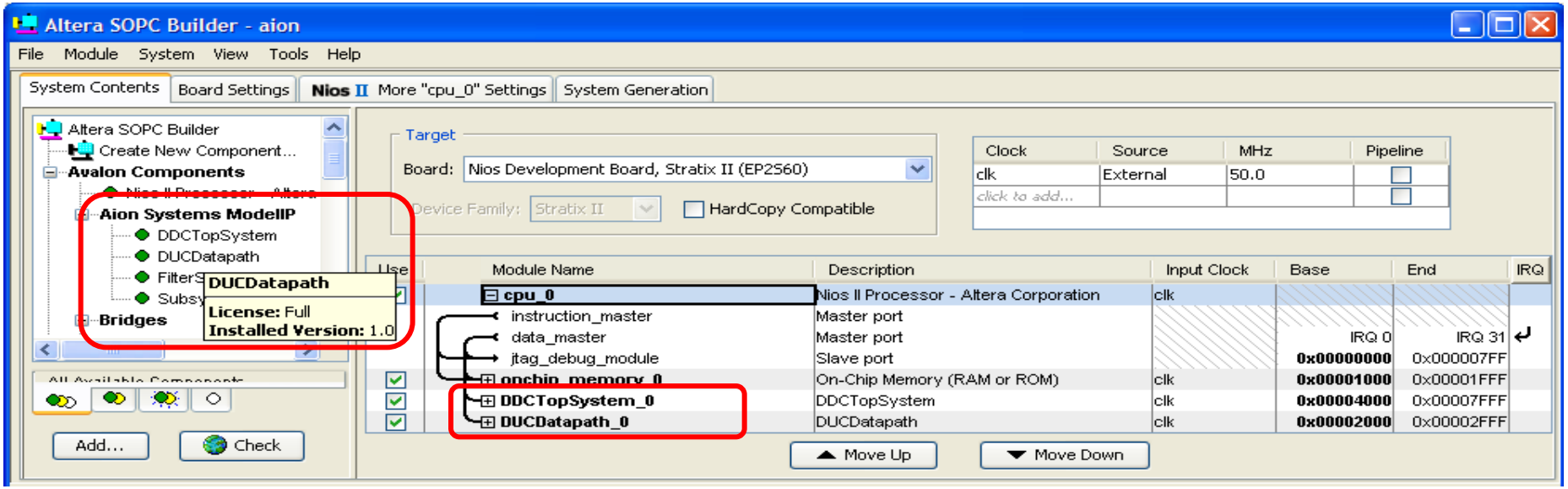
- Avalon Streaming Interfaces and FIFOs

Use flow control to stream data without stalling



SOPC Builder Integration

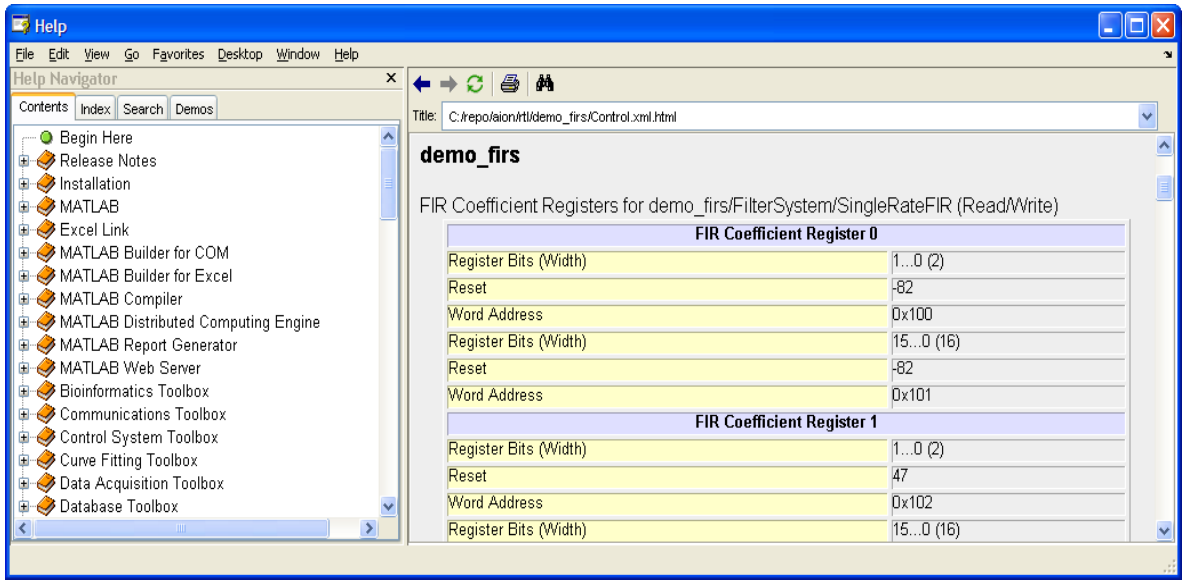
- Class.ptf files generated for each system
- Slower bus clock connects to system
- Faster signal processing clock is external
- Hierarchical address spaces, so systems can be easily integrated



Build Complete, High Performance Programmable DSP systems

Documentation Generation

- Memory mapped information is collated into XML and processed to HTML for integrated Matlab Help

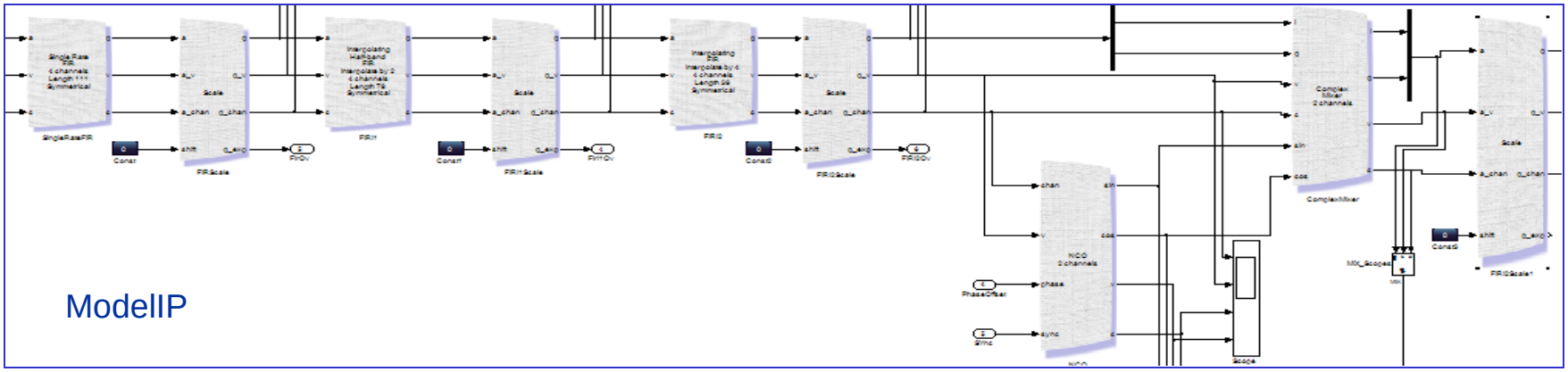


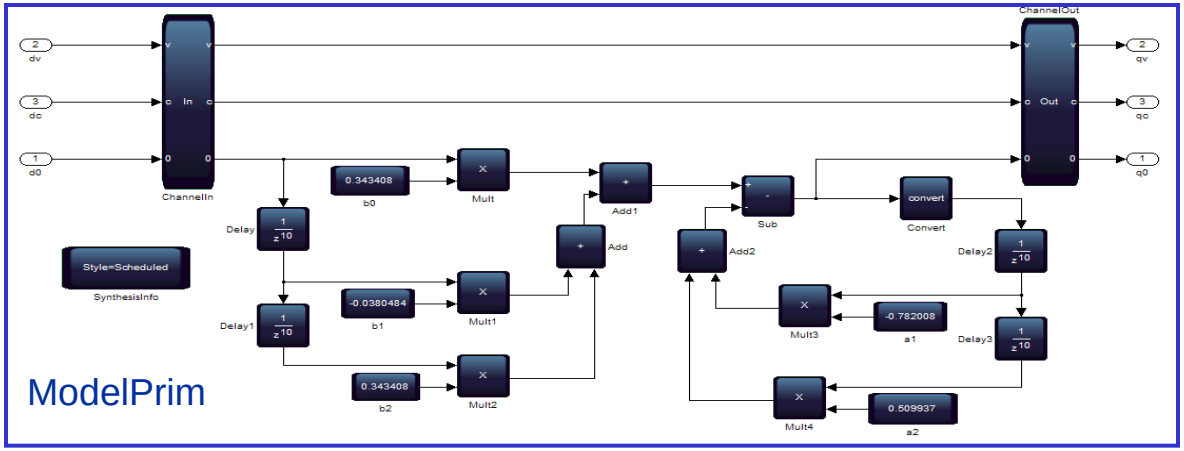
System level memory map is processed to datasheet representation

Ideal DSP Algorithm Development Tool

Constraint Driven Design: (1) Create Model

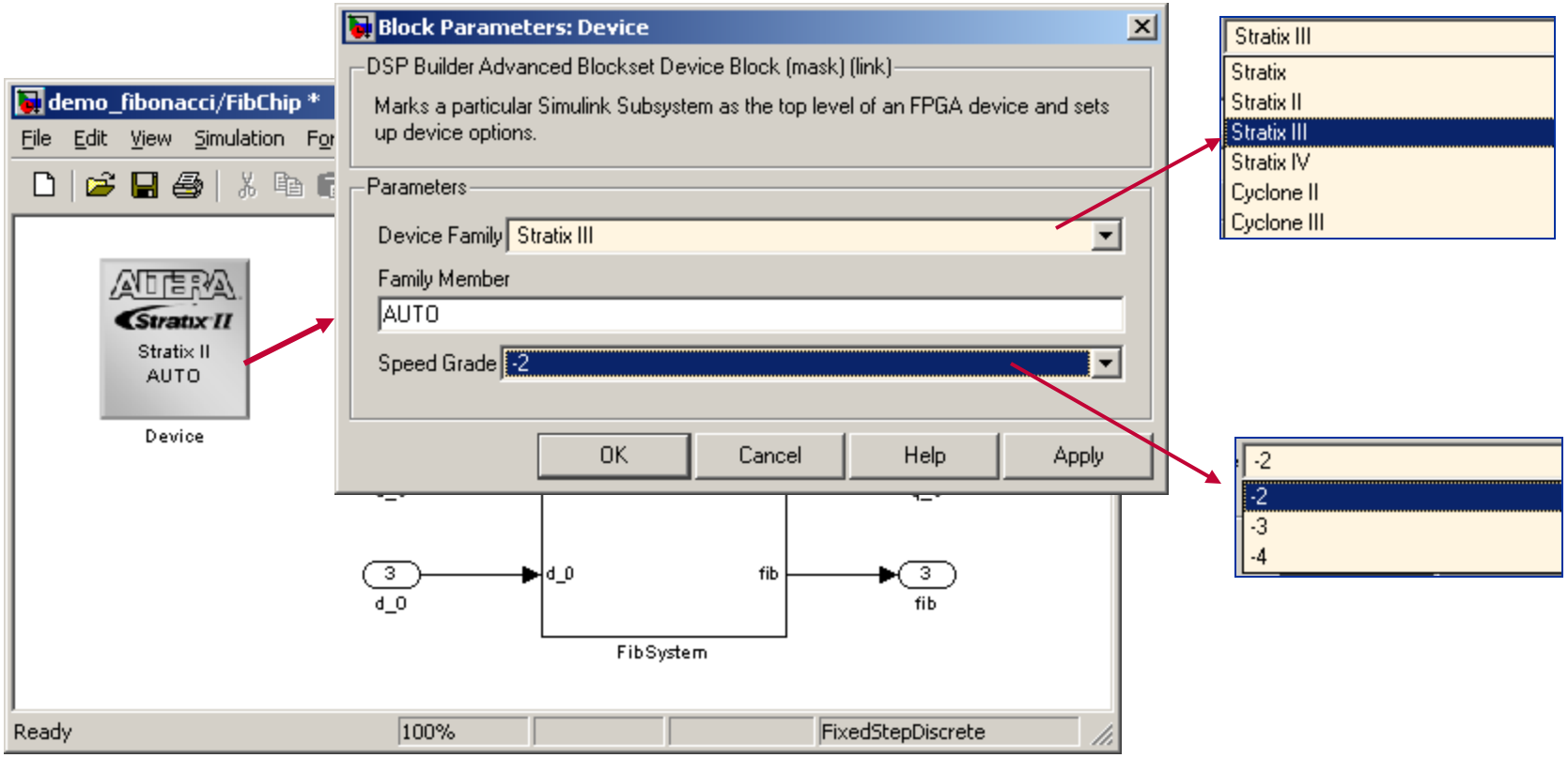
- Use ModelIP or ModelPrim Libraries





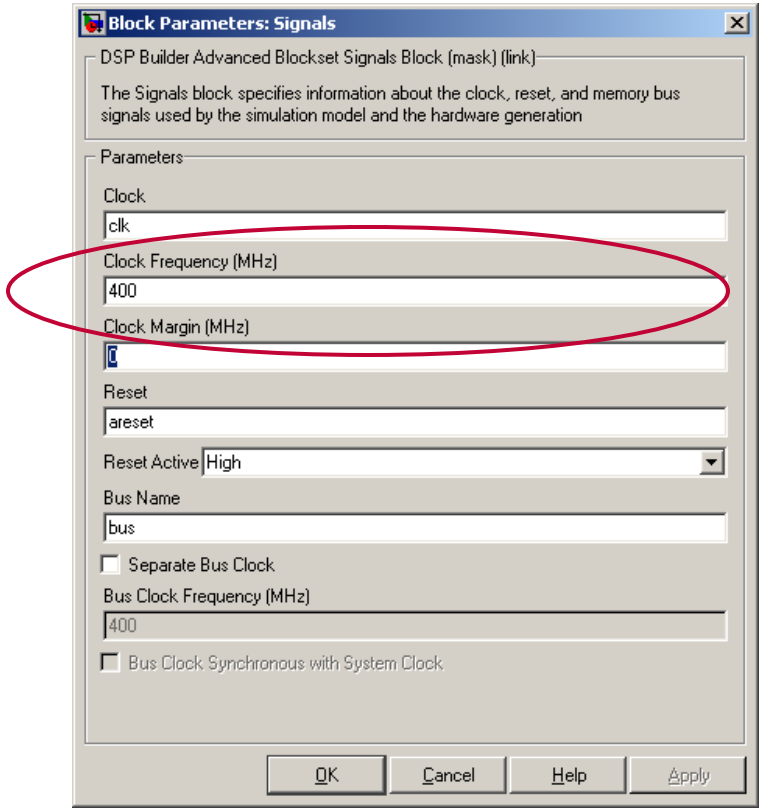
Constraint Driven Design: (2) Select Device

- Device independent modeling until this level



Constraint Driven Design: (3) Set Frequency

- Automatic Pipelining / Time Sharing (ModelIP)



Constraint Driven Design: (4) Compile

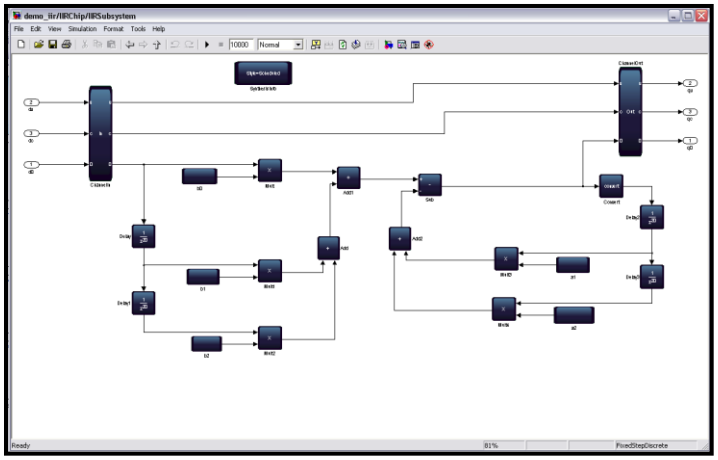
- Compilation Report
  - Legal Notice
  - Flow Summary
  - Flow Settings
  - Flow Non-Default Global Settings
  - Flow Elapsed Time
  - Flow Log
  - Analysis & Synthesis
  - Fitter
  - Assembler
  - TimeQuest Timing Analyzer
    - Summary
    - SDC File List
    - Clocks
    - Slow 900mV 85C Model
      - Fmax Summary

Slow 900mV 85C Model Fmax Summary

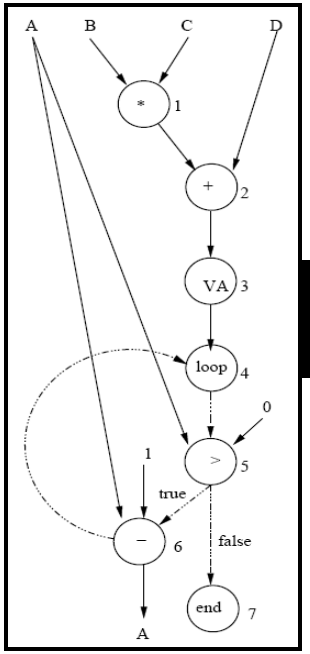
	Fmax	Restricted Fmax	Clock Name	Note
1	307.22 MHz	307.22 MHz	bus_clk	
2	408.66 MHz	408.66 MHz	clk	

Fitter Status	Successful - Fri Apr 11 11:35:26 2008
Quartus II Version	8.0 Internal Build 185 03/20/2008 SJ Full Version
Revision Name	FilterSystem
Top-level Entity Name	demo_firi_FilterSystem
Family	Stratix IV
Device	EP4SGX70DF29C2
Timing Models	Preliminary
Logic utilization	7 %
Combinational ALUTs	1,448 / 56,320 (3 %)
Memory ALUTs	770 / 28,160 (3 %)
Dedicated logic registers	3,825 / 56,320 (7 %)
Total registers	3825
Total pins	121 / 412 (29 %)
Total virtual pins	0
Total block memory bits	12,805 / 6,617,088 (< 1 %)
DSP block 18-bit elements	14 / 384 (4 %)
Total GXB Receiver Channels	0 / 16 (0 %)
Total GXB Transmitter Channels	0 / 16 (0 %)
Total PLLs	0 / 3 (0 %)
Total DLLs	0 / 4 (0 %)

Higher Level Synthesis



1



2

3

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity DSPA is port (
SEL: in STD_LOGIC_VECTOR(15 downto 0);
C, D, B: in STD_LOGIC_VECTOR(15 downto 0);
A : out STD_LOGIC);
end;

architecture BEHAVIOUR of DSPA is
begin

A := B * C + D;
......

end;
```

1. Covert the MDL schematic into an intermediate DFG representation

2. Apply transforms and analysis:

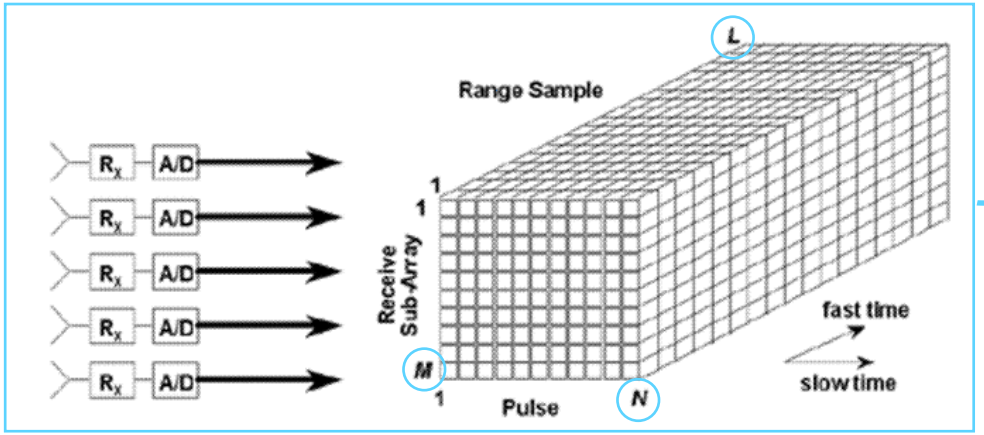
- Break apart carry chains
- DSP Block & Memory Timing
- Share multipliers
- Pipeline for:
  - required FMax performance
  - Balanced/matched delays
- ….

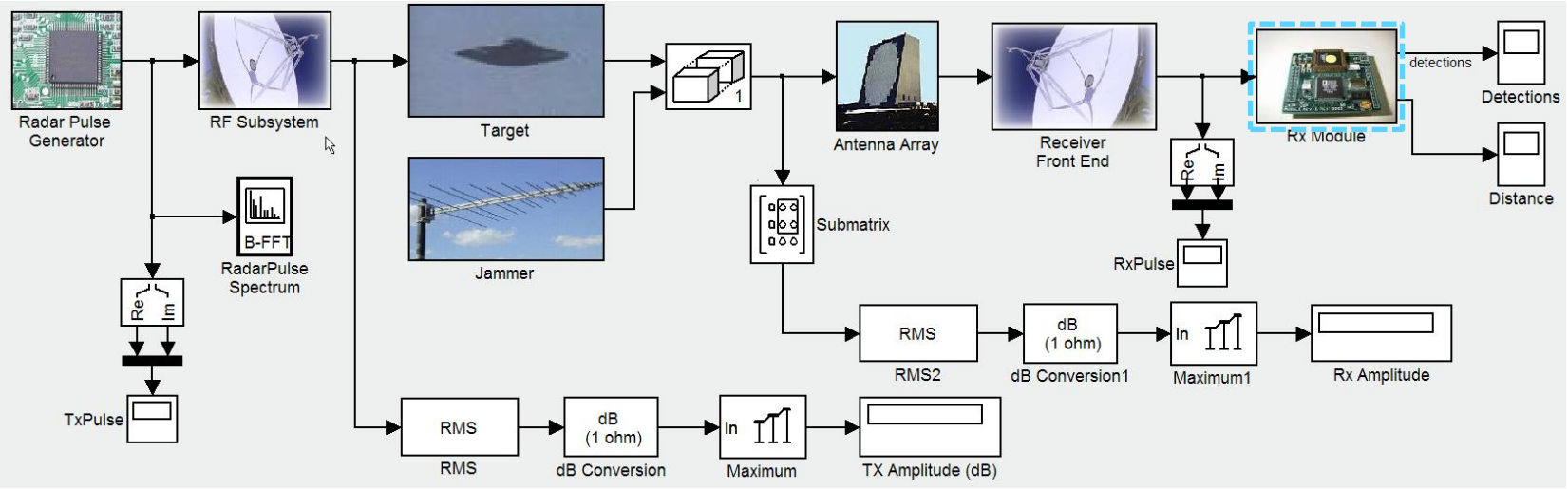
3. Generate RTL

STAP Radar Design

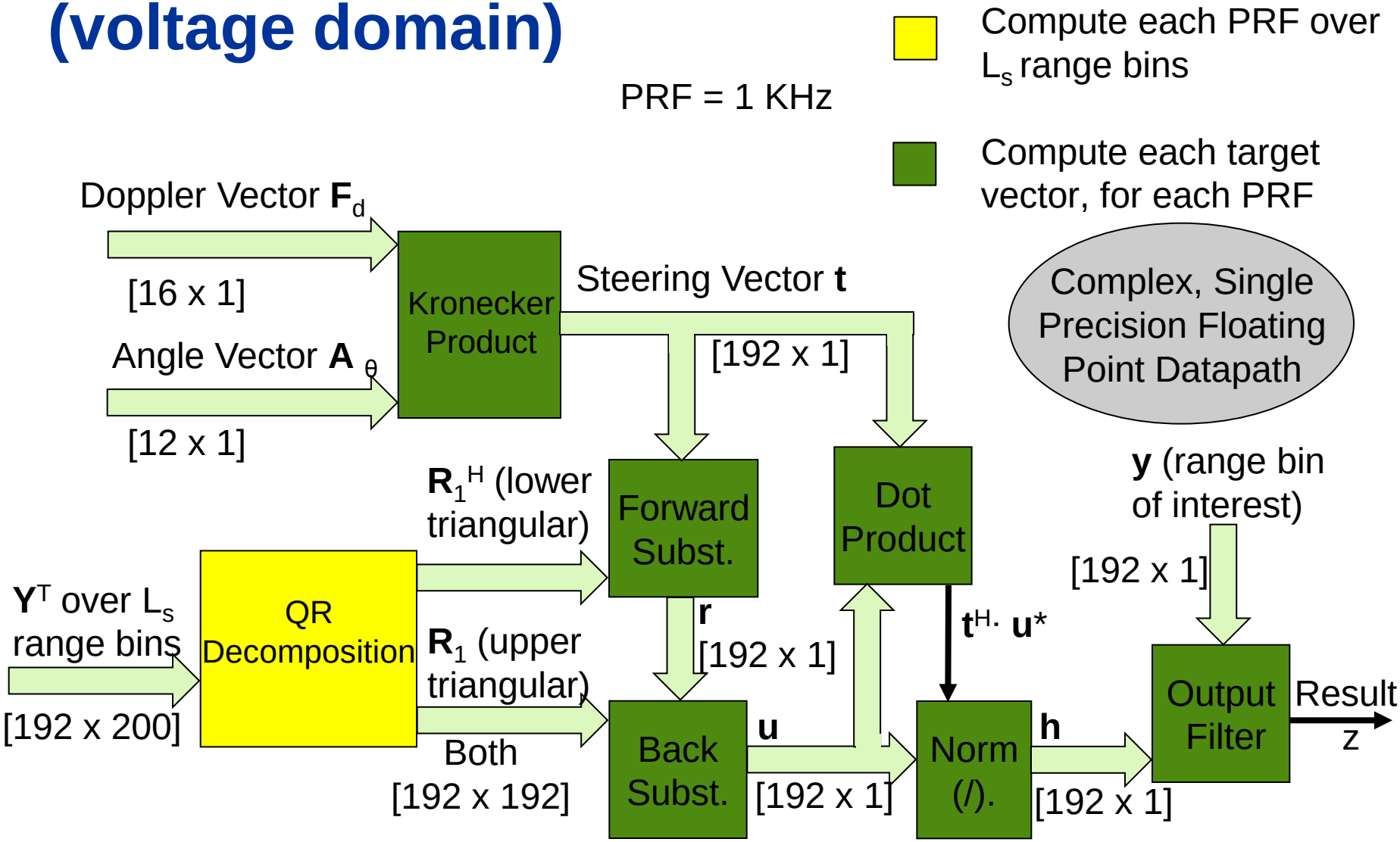
Floating Point Matrix Operations in FPGAs

STAP Processing the Radar Cube

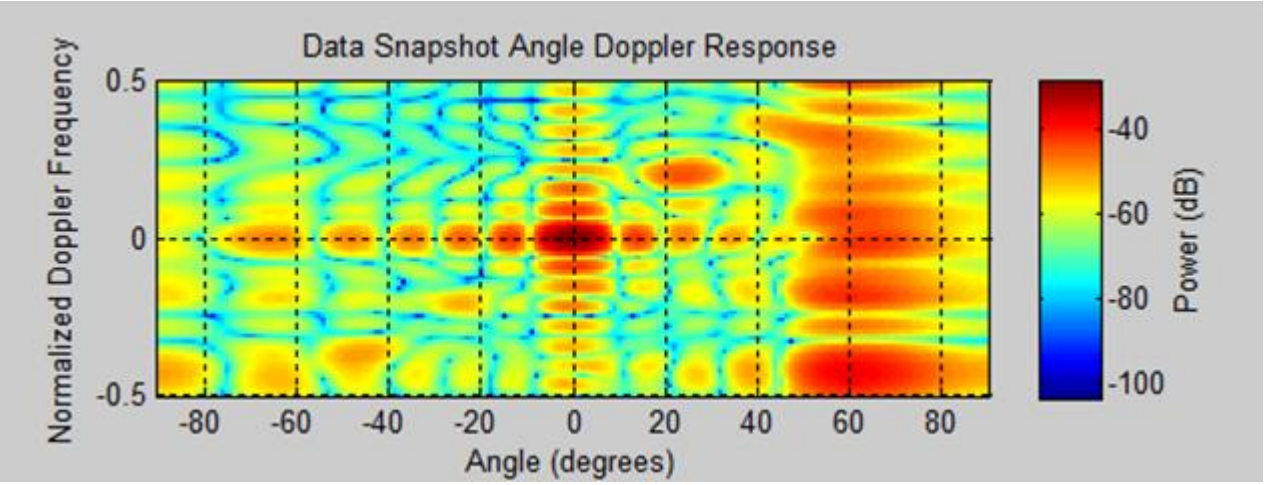


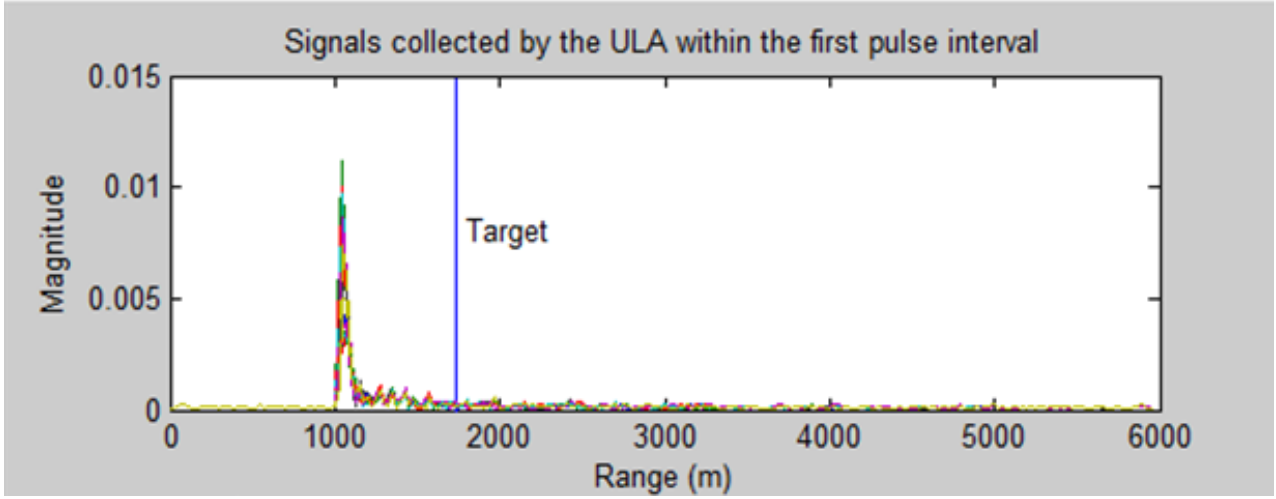


FPGA Processing Flow Implementation (voltage domain)



STAP Input Test Data





Steering Vector Generation

- Radar described in m code

Editor

File Edit Text Go Cell Tools Debug Desktop Window Help

D:\depot\vmauer_sj-perforce3_1666\depot\ssg\military\radar\STAP\reference\stapSMI.m

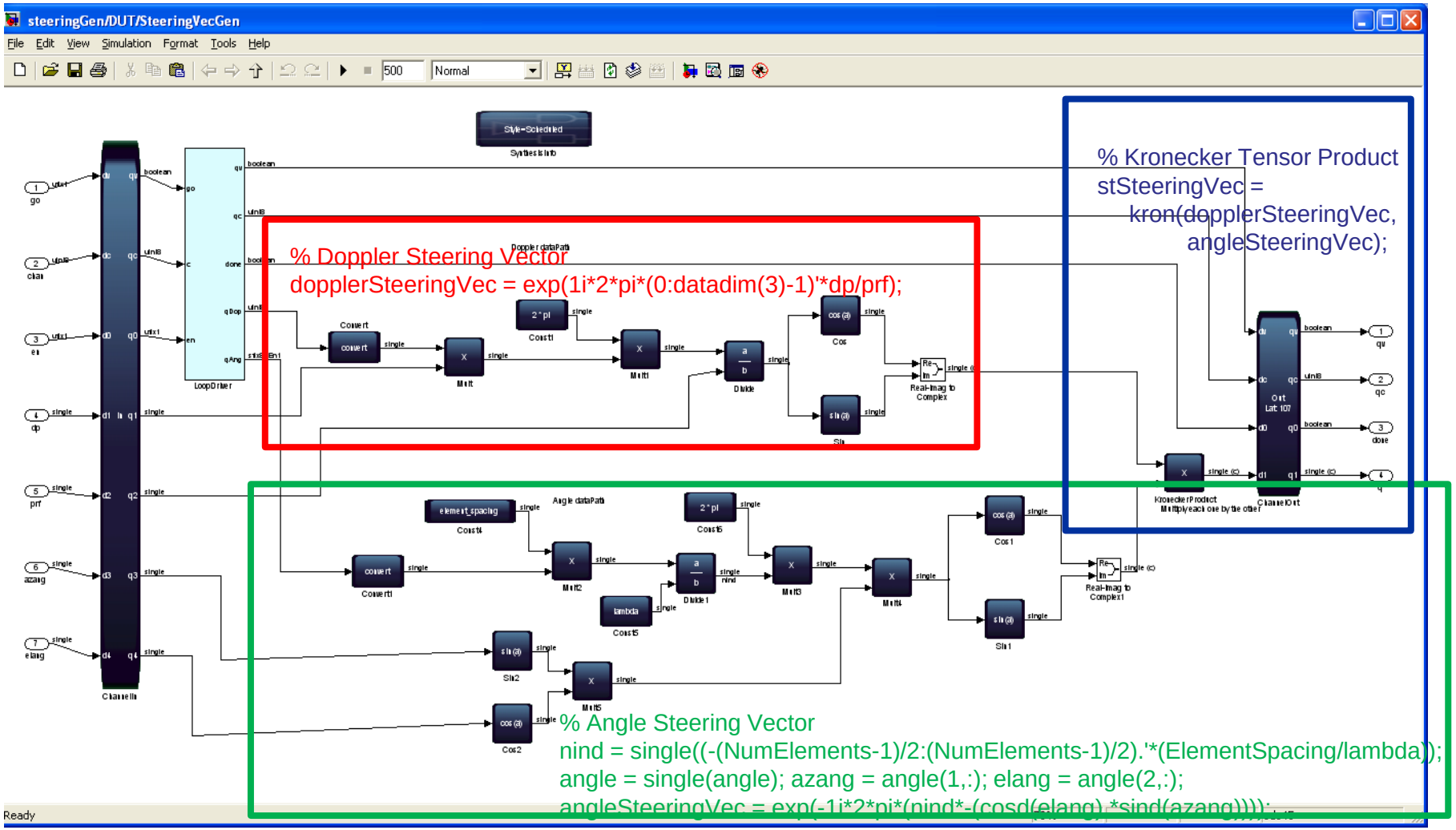
This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% Calculate Weights
% Doppler steering vector
dopplerSteeringVec = exp(1i*2*pi*(0:datadim(3)-1)'*dp/prf);
% Angle steering vector
nind = single((-(NumElements-1)/2:(NumElements-1)/2).'*(ElementSpacing/lambda));
angle = single(angle); azang = angle(1,:); elang = angle(2,:);
angleSteeringVec = exp(-1i*2*pi*(nind*-(cosd(elang).*sind(azang))));
% Kronecker tensor product
stSteeringVec = kron(dopplerSteeringVec, angleSteeringVec);

% Weights
[~, R] = qr(TrainingData',0); % R is upper triangular
temp = R\(R'\stSteeringVec);  % Forward substitution followed by backsubstitution
w = temp/(stSteeringVec'*temp);

%% Calculate output
y = w'*datavec;
y = y(:);
```

stapSMI.m | stapalgorithmdemo.m | main.m | stapSMI.m

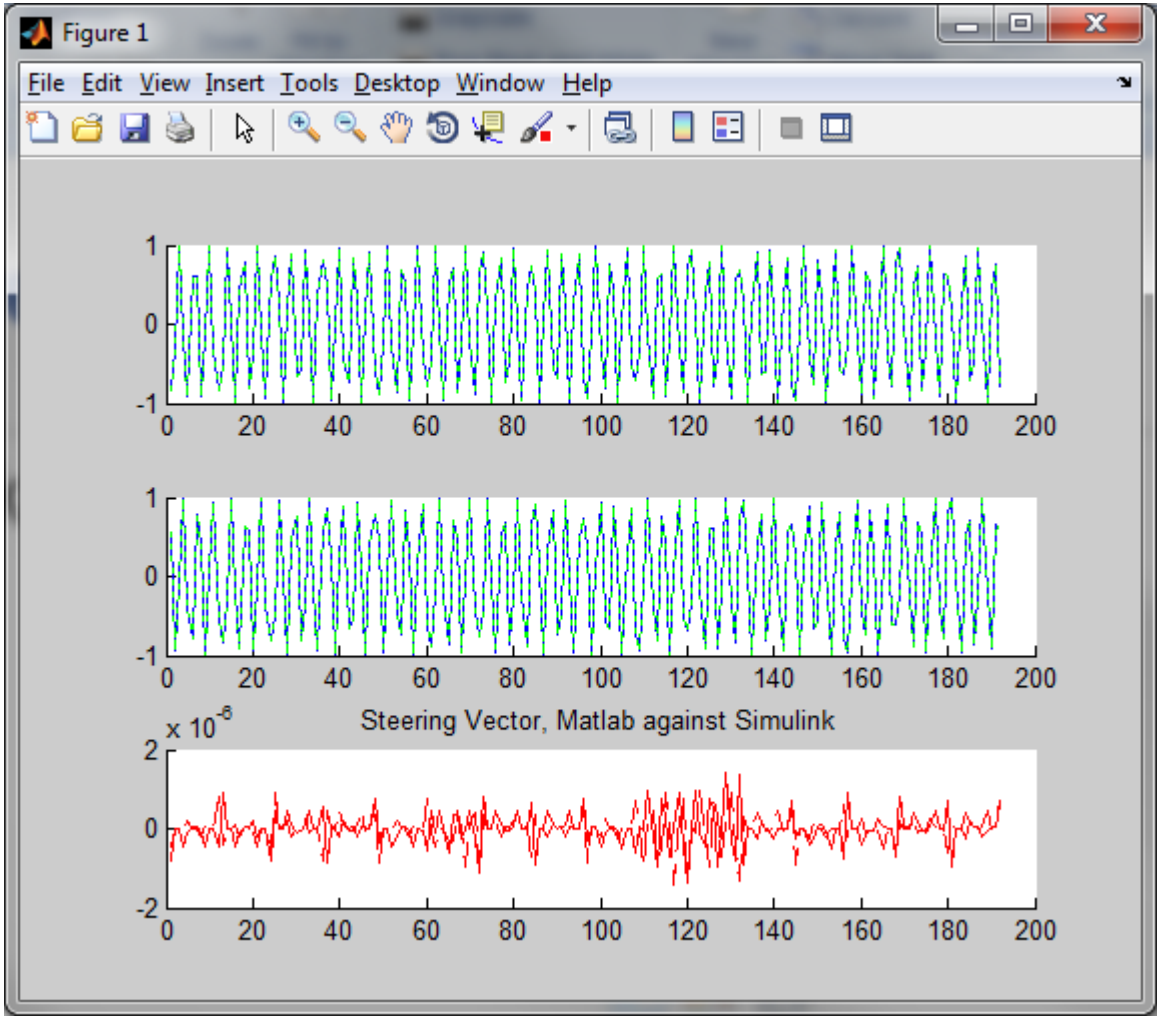
stapSMI Ln 70 Col 1 OVR

Steering Vector Generation Port



Steering Vector Results

- Difference
  > - 115dB



Steering Vector Generation

- System requirements
  - Calculate 64 vectors per ms
- Quartus results:
  - 163 18x18,
  - 16K registers, 19K ALUTs,
  - 129 MemLUTs
  - 7 KBit memory
  - 246 MHz (using seed sweep)
  - => **Processing time**: 500 cycles @ 248 MHz **= 2 us**

  - *Comment: We overachieve by a factor of 500. Timesharing would allow reducing the hardware resources*

QR Decomposition

- Radar described in m code

Editor

File Edit Text Go Cell Tools Debug Desktop Window Help

D:\depot\vmauer_sj-perforce3_1666\depot\ssg\military\radar\STAP\reference\stapSMI.m

This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% Calculate Weights
% Doppler steering vector
dopplerSteeringVec = exp(1i*2*pi*(0:datadim(3)-1)'*dp/prf);
% Angle steering vector
nind = single((-(NumElements-1)/2:(NumElements-1)/2).'*(ElementSpacing/lambda));
angle = single(angle); azang = angle(1,:); elang = angle(2,:);
angleSteeringVec = exp(-1i*2*pi*(nind*-(cosd(elang).*sind(azang))));
% Kronecker tensor product
stSteeringVec = kron(dopplerSteeringVec, angleSteeringVec);

% Weights
[~, R] = qr(TrainingData',0); % R is upper triangular
temp = R\(R'\stSteeringVec);  % Forward substitution followed by backsubstitution
w = temp/(stSteeringVec'*temp);

%% Calculate output
y = w'*datavec;
y = y(:);
```

stapSMI.m | stapalgorithmdemo.m | main.m | stapSMI.m

stapSMI Ln 70 Col 1 OVR

QR Decomposition Algorithm Analysis

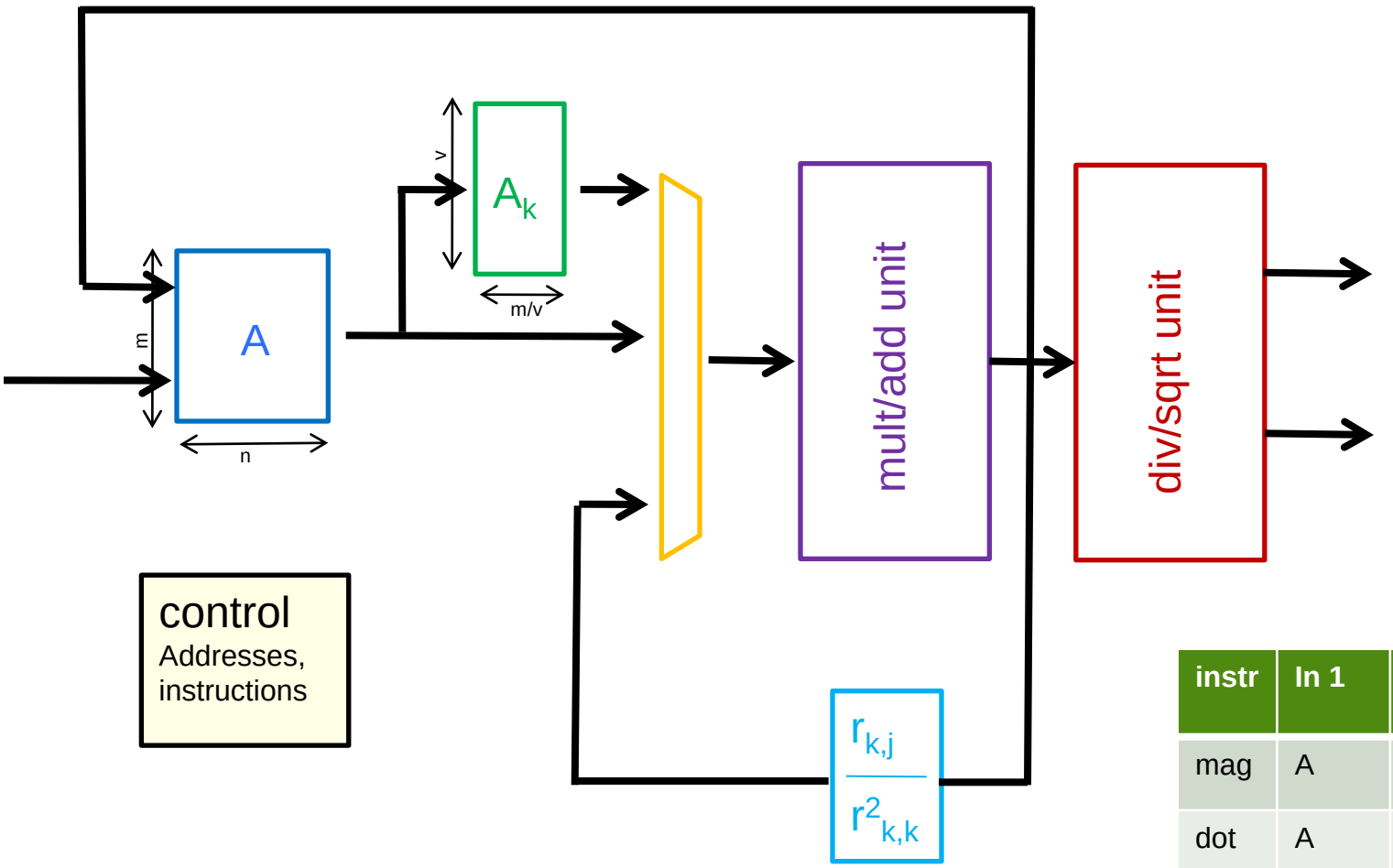
```
for k=1:n
    r(k,k) = norm(A(1:m, k));
    for j = k+1:n
        r(k, j) = dot(A(1:m, k), A(1:m, j)) / r(k,k);
    end
    q(1:m, k) = A(1:m, k) / r(k,k);
    for j = k+1:n
         A(1:m, j) = A(1:m, j) - r(k, j) * q(1:m, k);
    end
end
```

- Standard algorithm, source: Numerical Recipes in C
- Possible to implement as is, but some changes make it more FPGA friendly and increase numerical accuracy and stability

Parallelism

- Algorithmic requirements
  - $m*(k^2+k)$ cmults per QRD
  - $200 * (192^2+192) = 7.411.200$
- System requirements
  - Update rate 1 ms:
- Parallelism
  - 7.411.200 cmults / (1ms * 250 MHz) → 29.6 cmults in parallel are needed
  - Since matrix is 200: 200/32 = 6.25 clocks per sample = 7 clocks per sample
  - Our design uses 6 clocks per sample → 204/6 = 34 parallel paths
- Quartus results:
  - 550 18x18,
  - 120K registers,
  - 96K ALUTs,
  - 8MBit memory

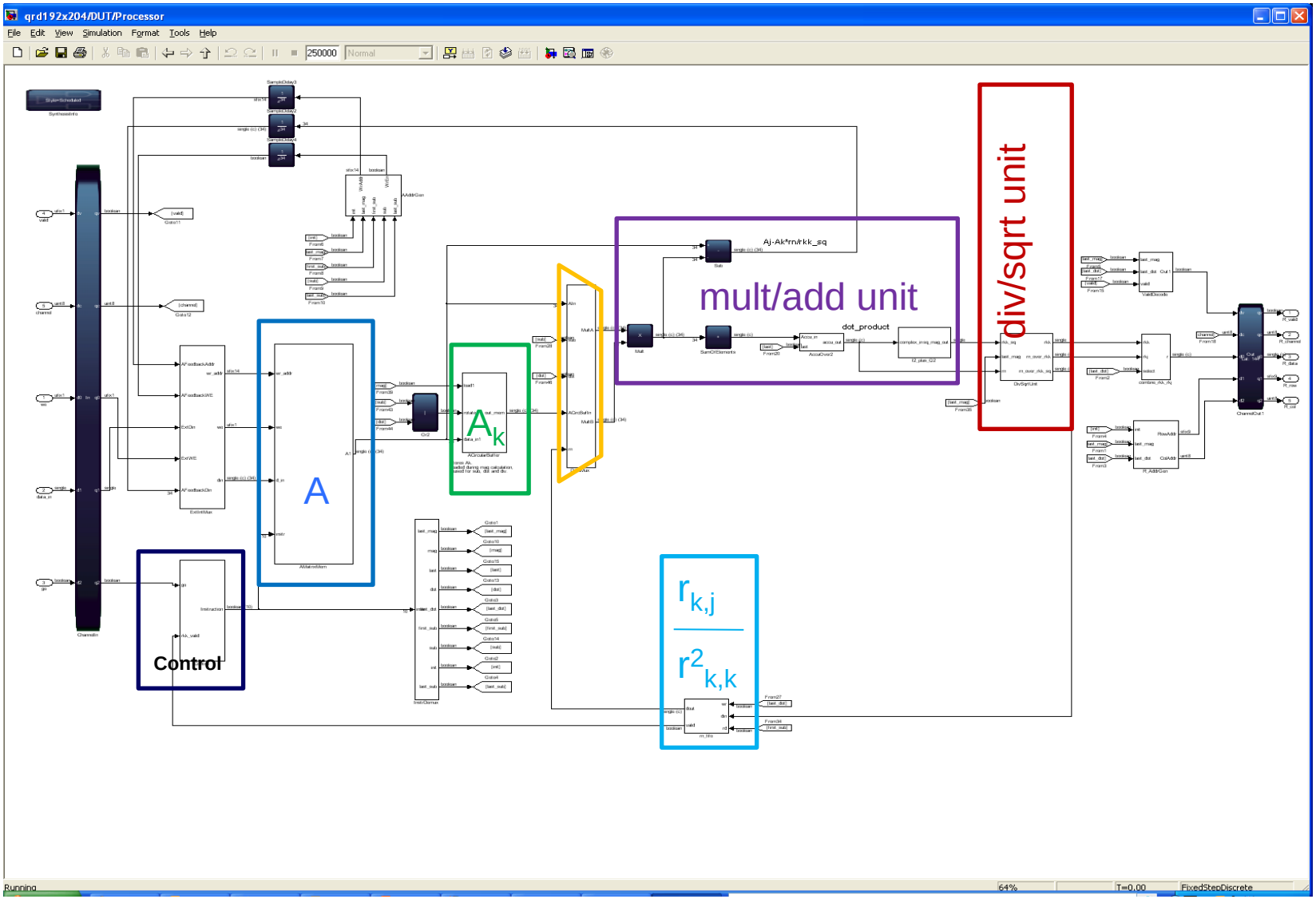
Structure



Fifo *("leaky bucket")*

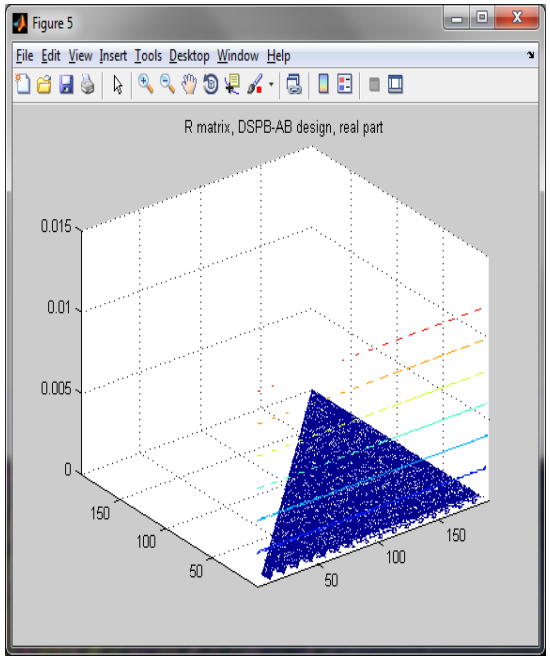
instr	In 1	In 2	In 3
mag	A	---	
dot	A	A_k	
div		A_k	rk
sub	A	A_k	$r_{k,j}/r^2_{k,k}$

QRD

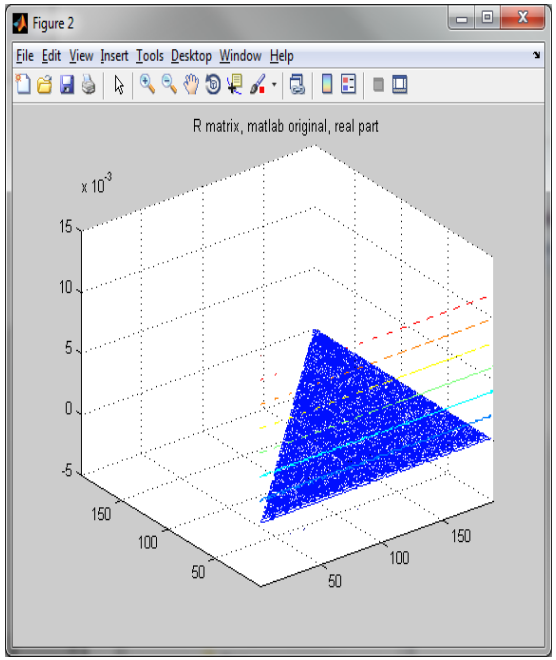


R Matrix Results

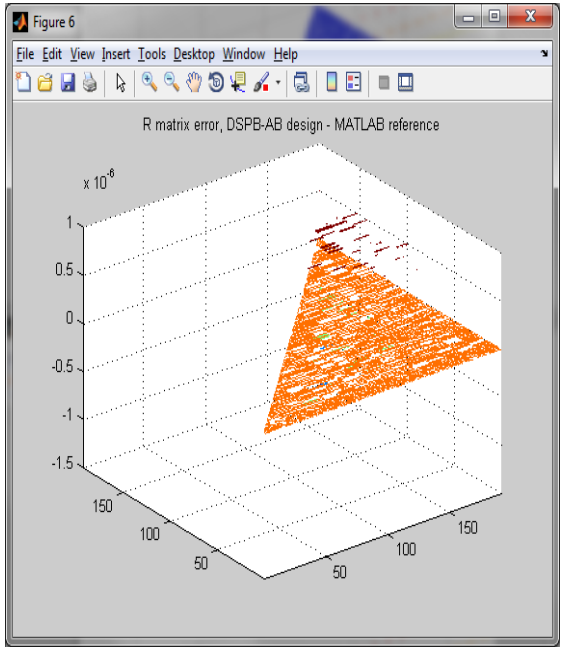
- Is the error too high ? Use double precision



—



=



Forward and Back Substitution

- Radar described in m code

Editor

File Edit Text Go Cell Tools Debug Desktop Window Help

D:\depot\vmauer_sj-perforce3_1666\depot\ssg\military\radar\STAP\reference\stapSMI.m

This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
%% Calculate Weights
% Doppler steering vector
dopplerSteeringVec = exp(1i*2*pi*(0:datadim(3)-1)'*dp/prf);
% Angle steering vector
nind = single((-(NumElements-1)/2:(NumElements-1)/2).'*(ElementSpacing/lambda));
angle = single(angle); azang = angle(1,:); elang = angle(2,:);
angleSteeringVec = exp(-1i*2*pi*(nind*-(cosd(elang).*sind(azang))));
% Kronecker tensor product
stSteeringVec = kron(dopplerSteeringVec, angleSteeringVec);

% Weights
[~, R] = qr(TrainingData',0); % R is upper triangular
temp = R\(R'\stSteeringVec);  % Forward substitution followed by backsubstitution
w = temp/(stSteeringVec'*temp);

%% Calculate output
y = w'*datavec;
y = y(:);
```

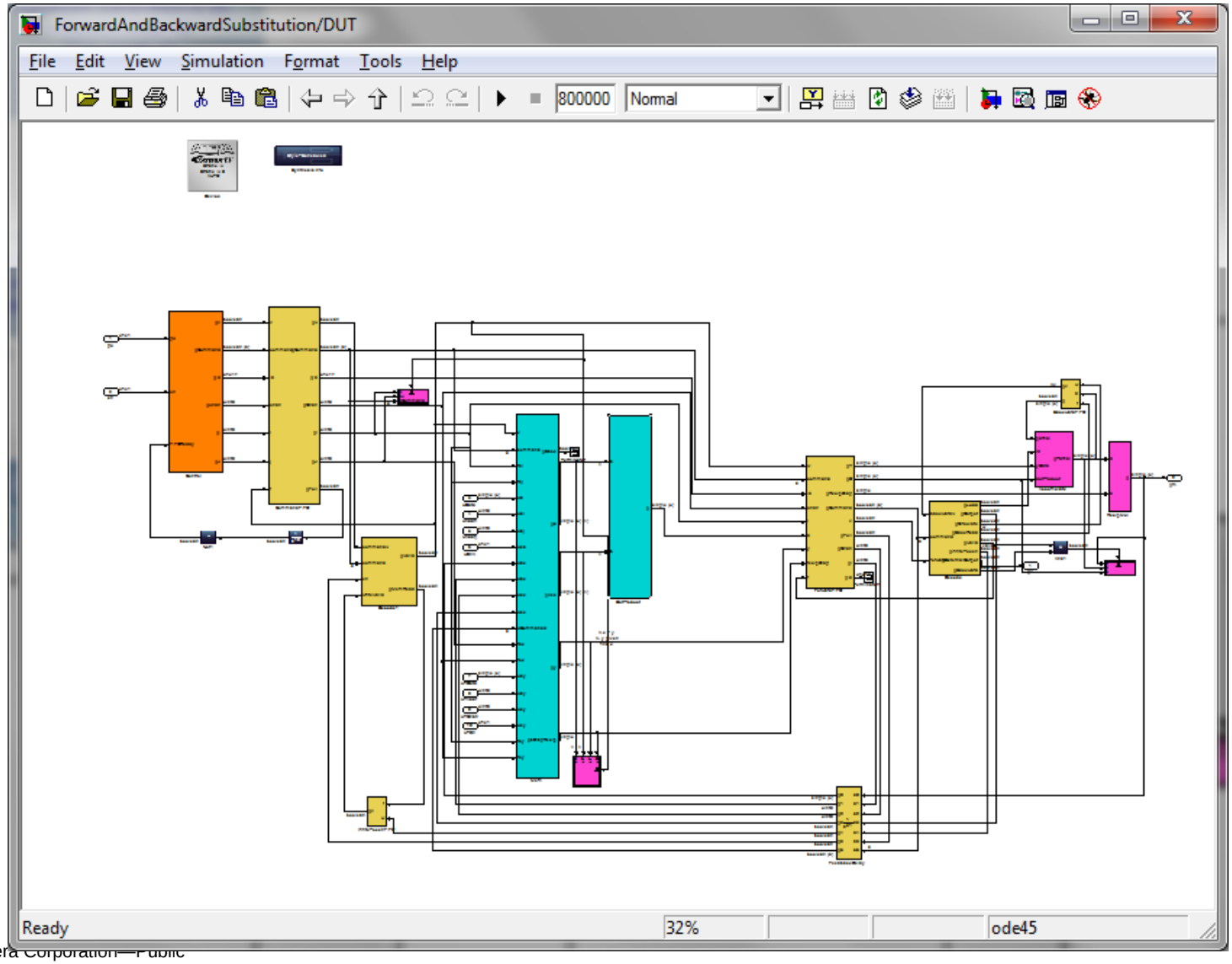
stapSMI.m | stapalgorithmdemo.m | main.m | stapSMI.m

stapSMI Ln 70 Col 1 OVR

Forward/Backward Substitution

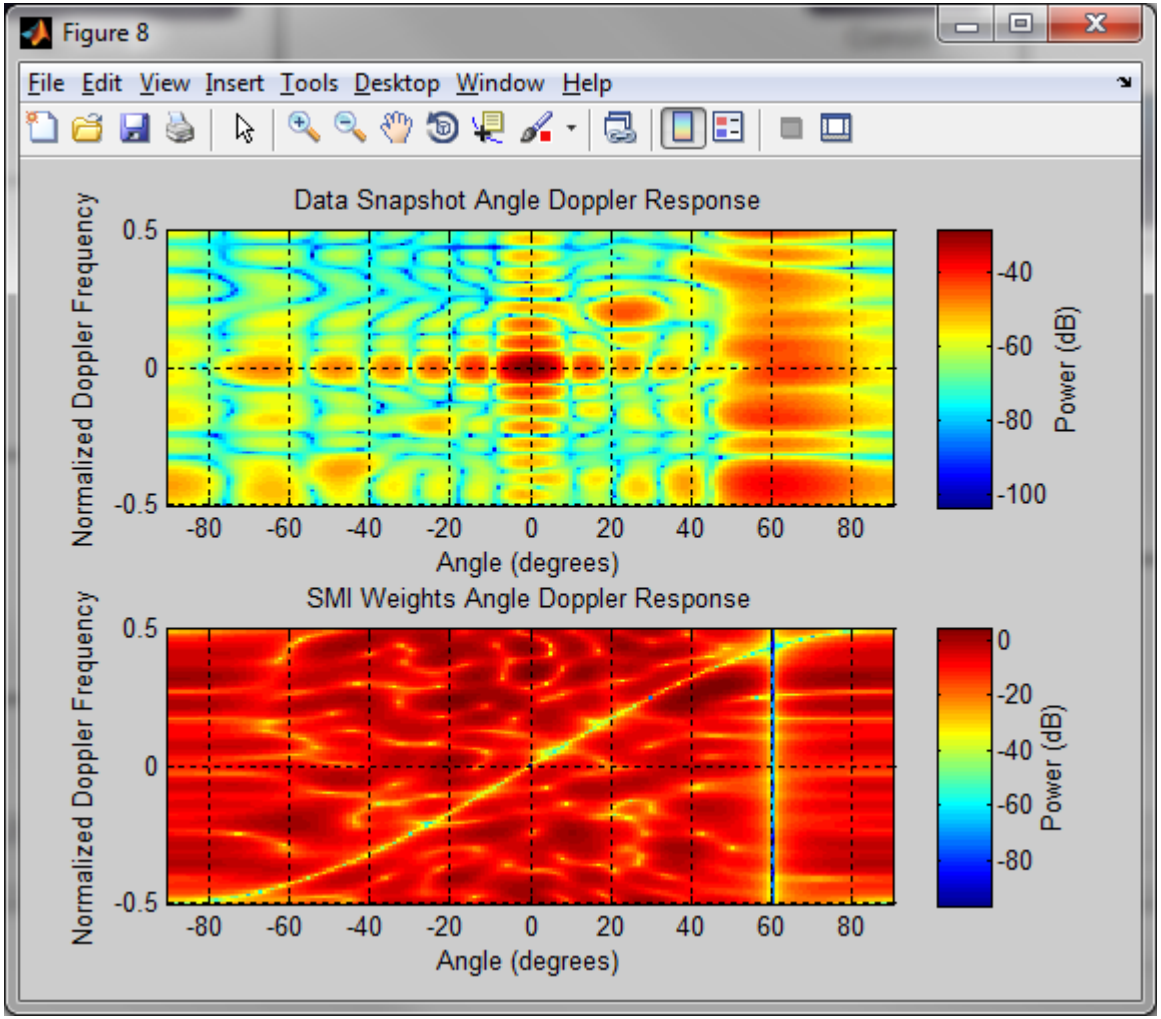
- Algorithmic requirements
  - $m*(k^2+k)$ cmults per QRD
  - $200 * (192^2+192) = 7.411.200$
- System requirements
  - Update rate 1 ms:
- Parallelism
  - 7.411.200 cmults / (1ms * 250 MHz) => 29.6 cmults in parallel are needed
  - Chosen: 32 parallel cmults = 128 mults = 512 18x18s on SIII or 128 27x27 on SV
- Quartus results:
  - 82 18x18,
  - 14K registers, 11K ALUTs,
  - 1.7K MemLUTs
  - 5.7 MBit memory
  - 246 MHz (using seed sweep)
  - => **Processing time**: 223658 cycles @ 249 MHz **= 0.91 ms**

Forward/Backward Substitution

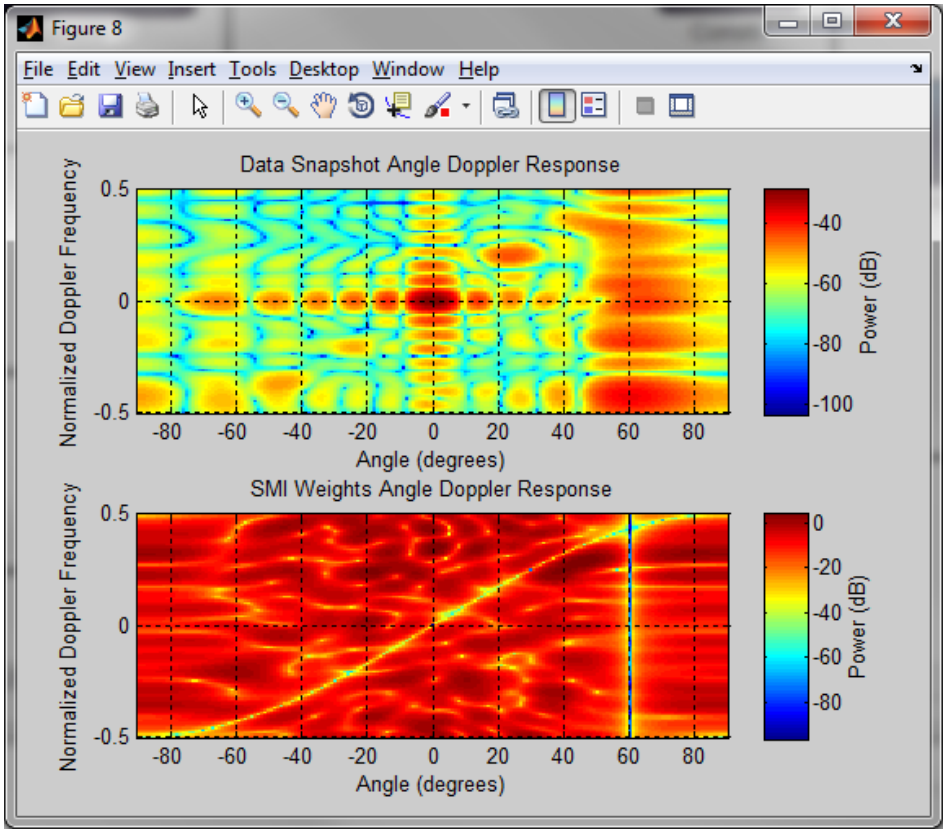


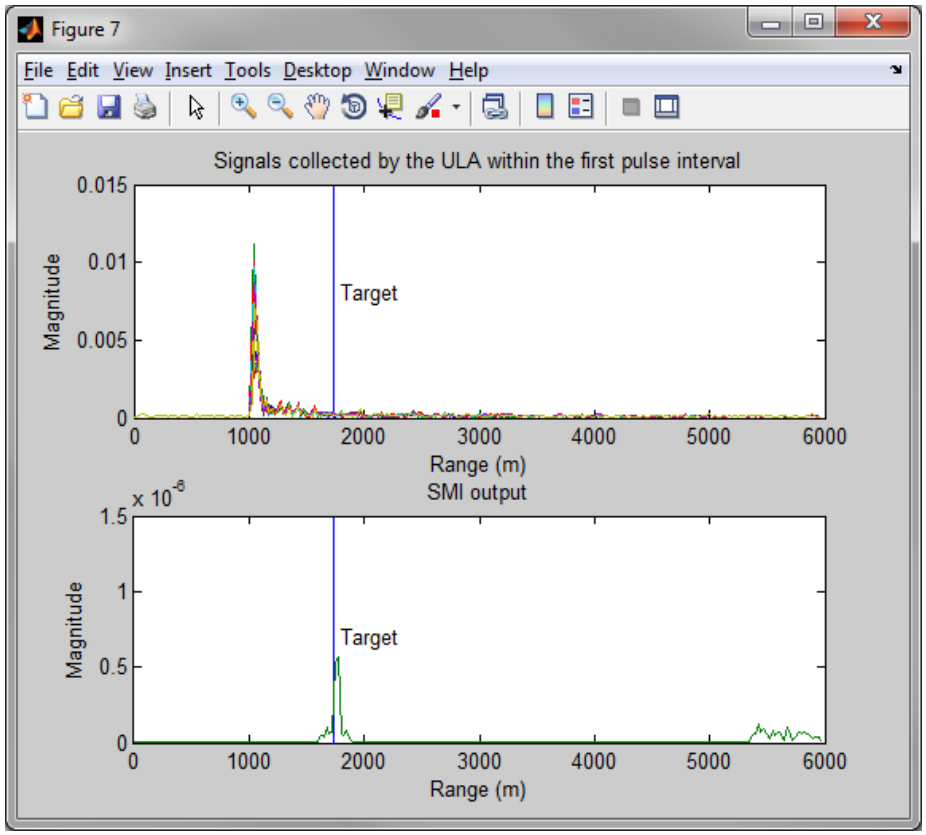
Back Substitution Results

- Top is original data set with jammer at 60 degrees
- Bottom represents the weights after back substitution. Note that at 60 degrees you apply filter of -80 dB



Single Steering Vector STAP Results





DSP Builder Advanced Blockset Simulation Results

STAP Resources

- 1 channel on EP4SGX230 @ 212 MHz
  - ALUTs: 119k (65% of total)
  - Registers: 146k (80% of total)
  - Multipliers: 827 (64% of total)
  - Memory: 8.3 Mbit (57% of total)
  - Latency: ~1 millisecond (QRD + 64 steering vec)
- 4 channels on EP5SGXD8
  - ALUTs: 476k (83% of total)
  - Registers: 584k (52% of total)
  - Multipliers: 3308 (81% of total)
  - Memory: 33 Mbit (60% of total)

STAP Radar Design Example Summary

- Advanced Blockset simplifies meeting performance
  - Design behavioral models and the tool pipelines to meet speed
  - Auto generation of HDL based upon timing requirements
  - Supports vector processing
- Floating Point Blockset extension
  - Fused datapath allows for floating point in FPGAs
  - Support for Math.h type functions
  - Integer, single and double precision in same model
- STAP Design: Matrix Operations
  - Steering vector generation
  - QR Decomposition
  - Single precision complex