MATLAB EXPO 2021

Improve FPGA, ASIC and SoC Quality with Early Architecture Modeling

Jack Erickson

MathWorks®

Video Series: Range-Doppler Radar System Deployment

MathWorks® Products Solutions Academia Support Community Events

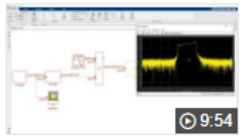
Video and Webinar Series

Search Videos

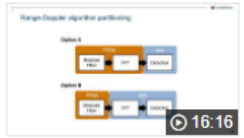
Videos Home | Search

Developing Radio Applications for RFSoC with MATLAB & Simulink

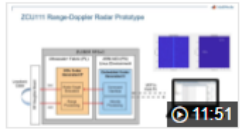
Part 1: Hardware/Software Co-Design Workflow
Target SoC architectures like Xilinx UltraScale+ RFSoC devices using Model-Based Design. Build Simulink models of hardware/software platforms to make design decisions.



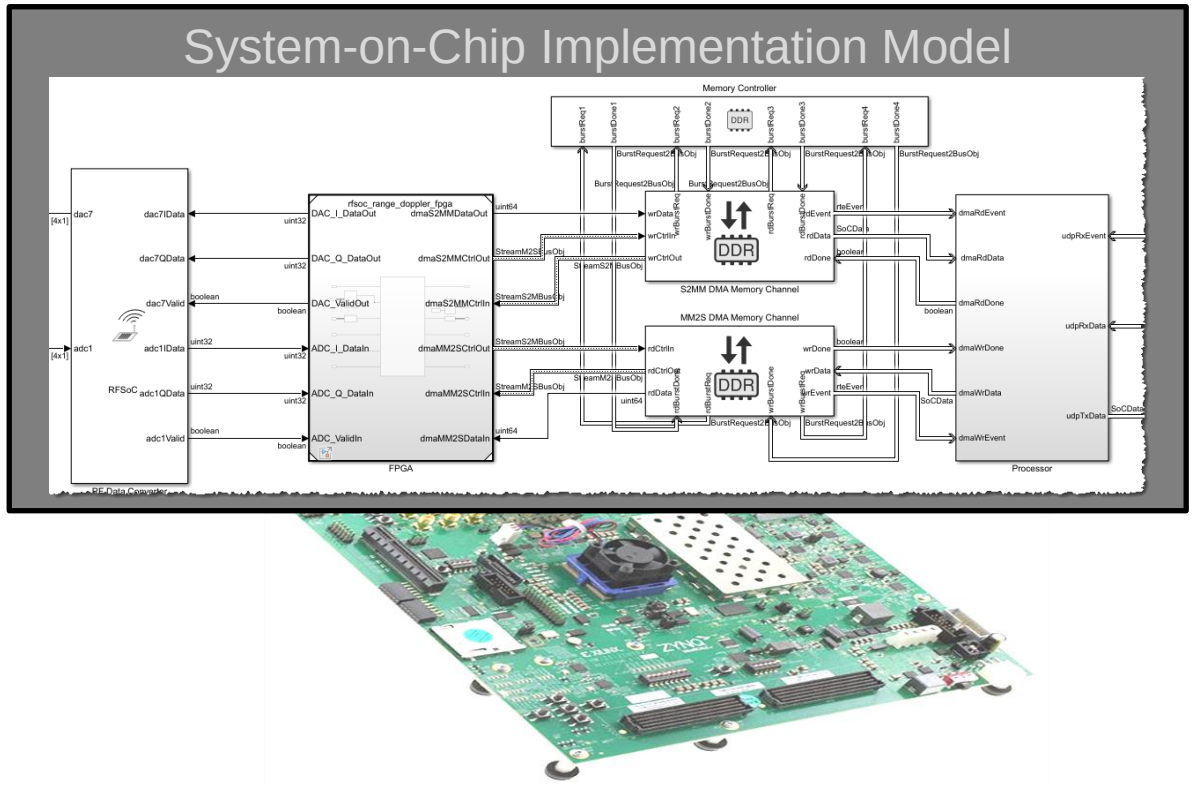
Part 2: System Specification and Design
System specifications for a range-Doppler radar are the driver for hardware/software implementation decisions when targeting SoC architectures like Xilinx RFSoC devices.



Part 3: Hardware/Software Partitioning
Perform simulation and analysis of the SoC architecture of the Xilinx RFSoC to investigate hardware/software partitioning of the range-Doppler radar algorithm.

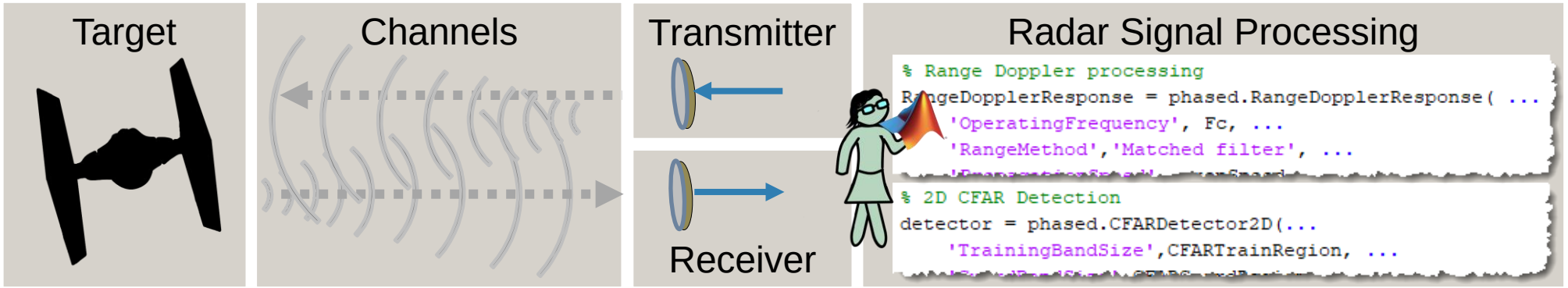


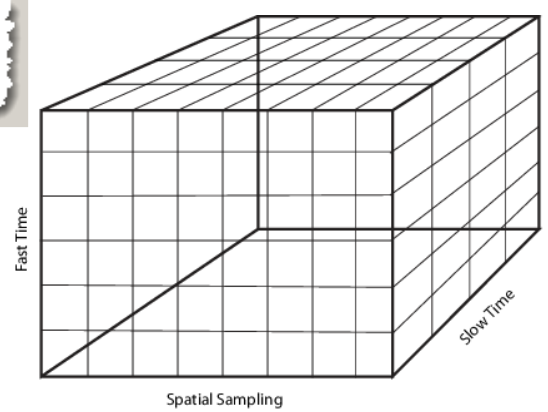
Part 4: Code Generation and Deployment
Use SoC Blockset to automate the process of C and HDL code generation from Simulink models, and to automatically deploy the range-Doppler radar algorithm to a Xilinx ZCU111 development kit.



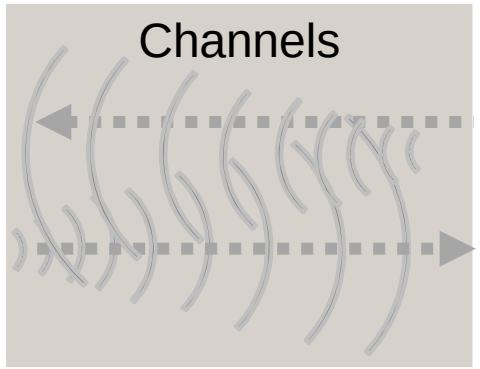
www.mathworks.com/videos/series/developing-radio-applications-for-rfsoc-with-matlab-and-simulink.html

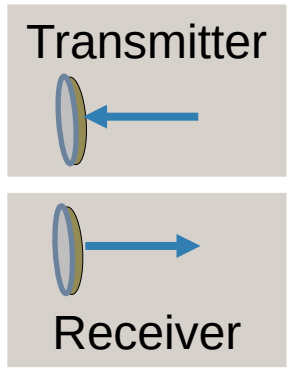
Range-Doppler Radar: System & Algorithm vs. Implementation





Range-Doppler Radar: System & Algorithm vs. Implementation

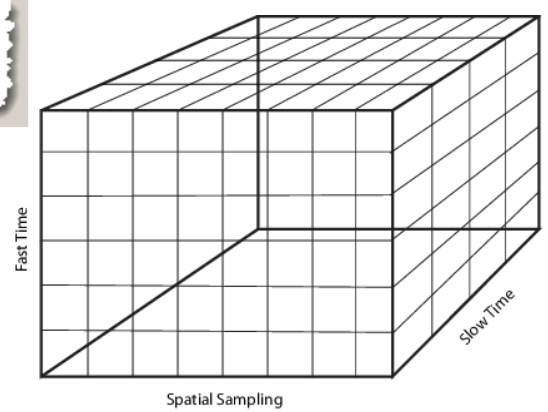


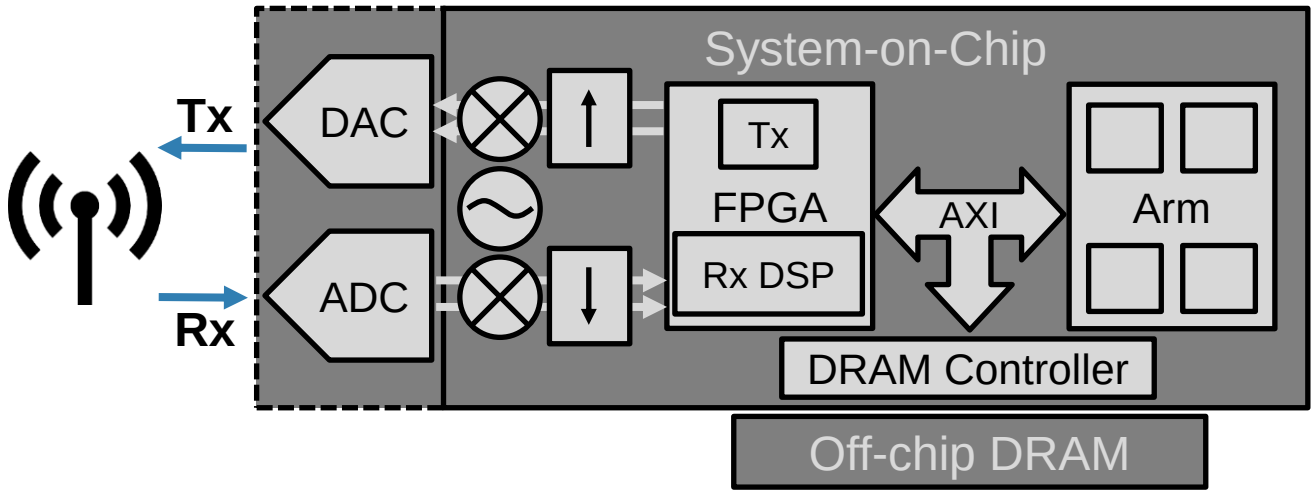


Radar Signal Processing

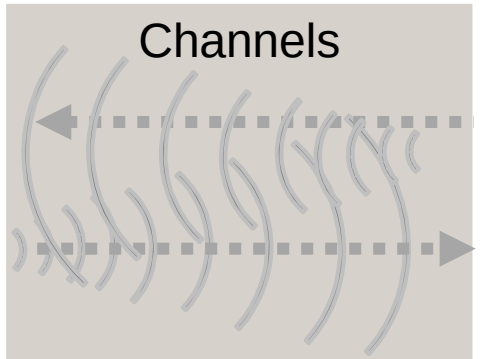
```
% Range Doppler processing
RangeDopplerResponse = phased.RangeDopplerResponse( ...
    'OperatingFrequency', Fc, ...
    'RangeMethod','Matched filter', ...
```

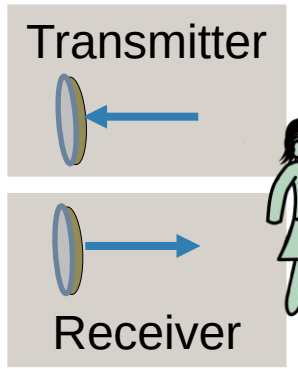
```
% 2D CFAR Detection
detector = phased.CFARDetector2D(...
    'TrainingBandSize',CFARTrainRegion, ...
```

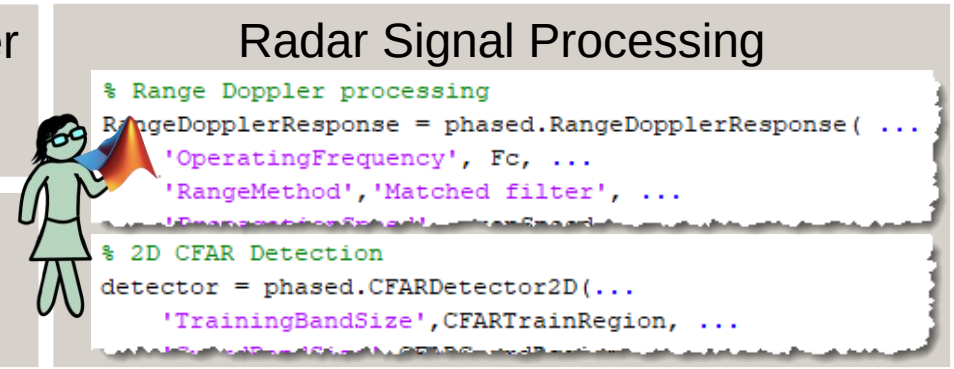


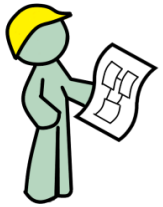


Range-Doppler Radar: System & Algorithm vs. Implementation

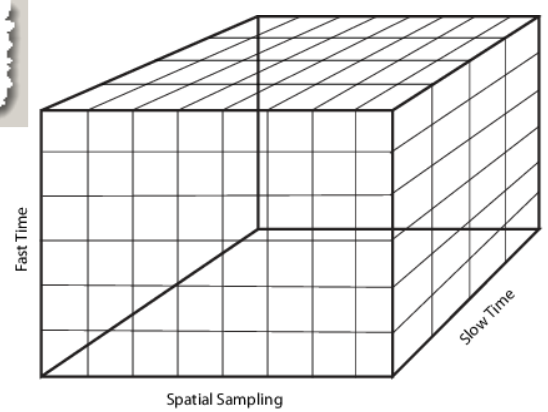


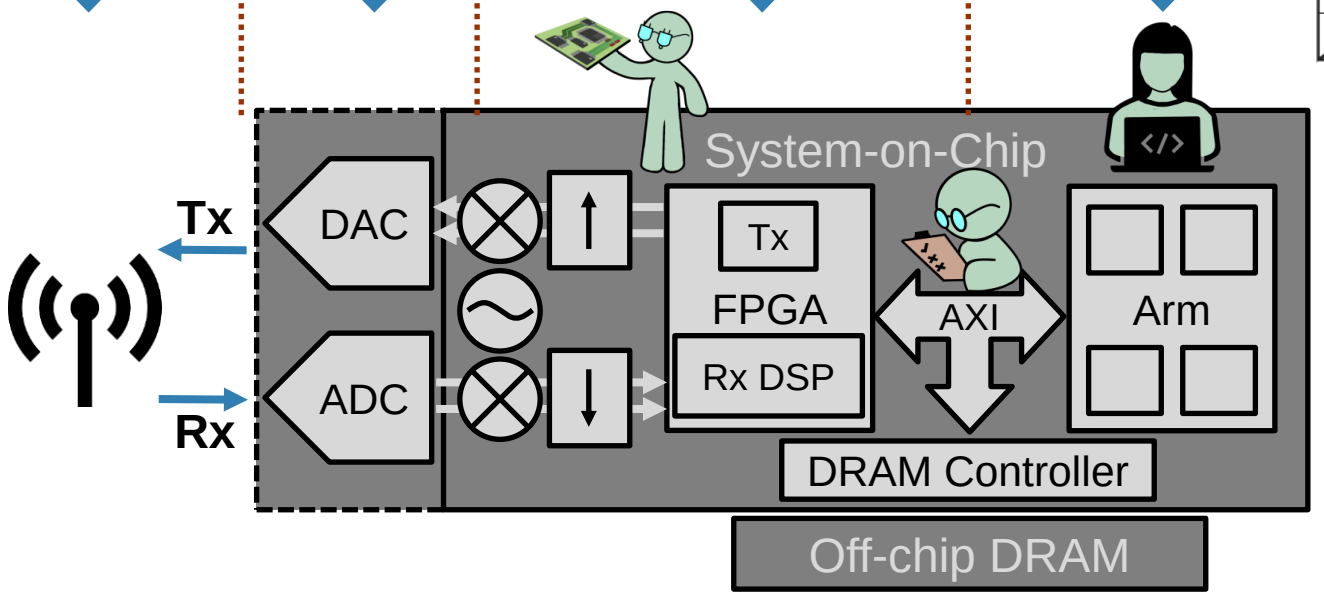




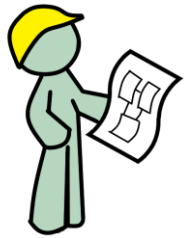


SPECIFICATIONS

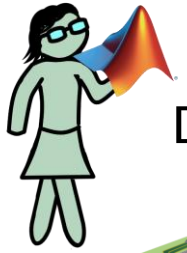




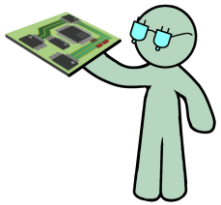
Poll: Which Role Do You Play? (select all that apply)



System Designer/Architect



Domain Expert / Algorithms (DSP, comms, radar, controls, video/image, etc)



Hardware Designer (FPGA or ASIC)

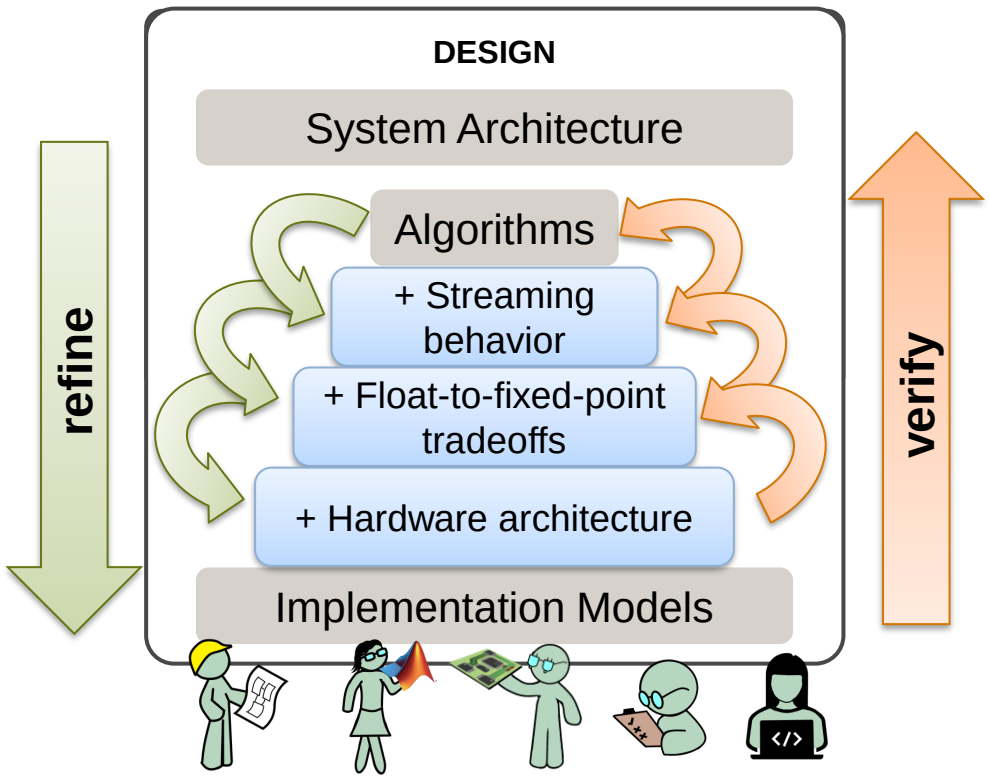
FPGA/ASIC Verification Engineer

Embedded Software Developer

Top-Down Refinement and Verification

MATLAB

- ✓ Large data sets
- ✓ Explore mathematics
- ✓ Data visualization
- ✓ Control logic

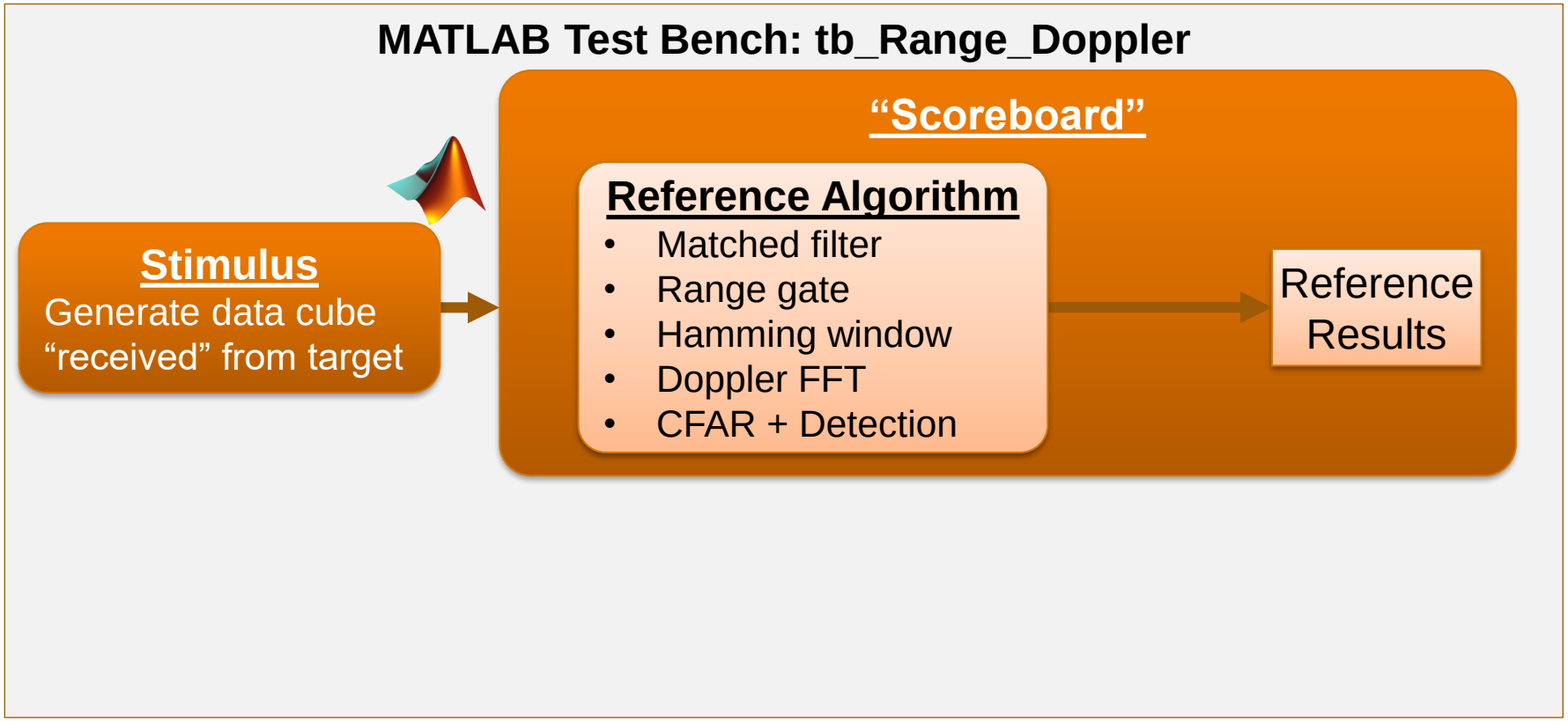


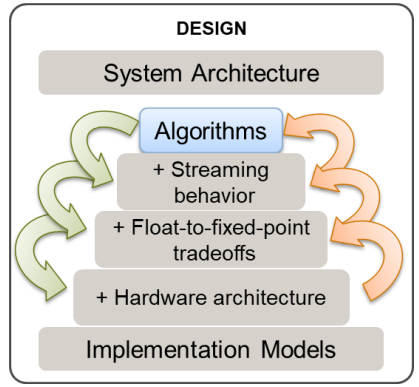
Simulink

- ✓ Parallel architectures
- ✓ Timing
- ✓ Data type propagation
- ✓ Mixed-signal modeling

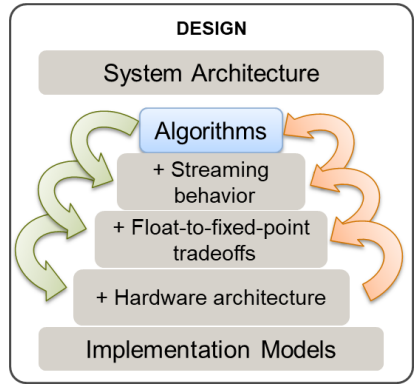
All roles contribute to key early decisions and continuously integrate

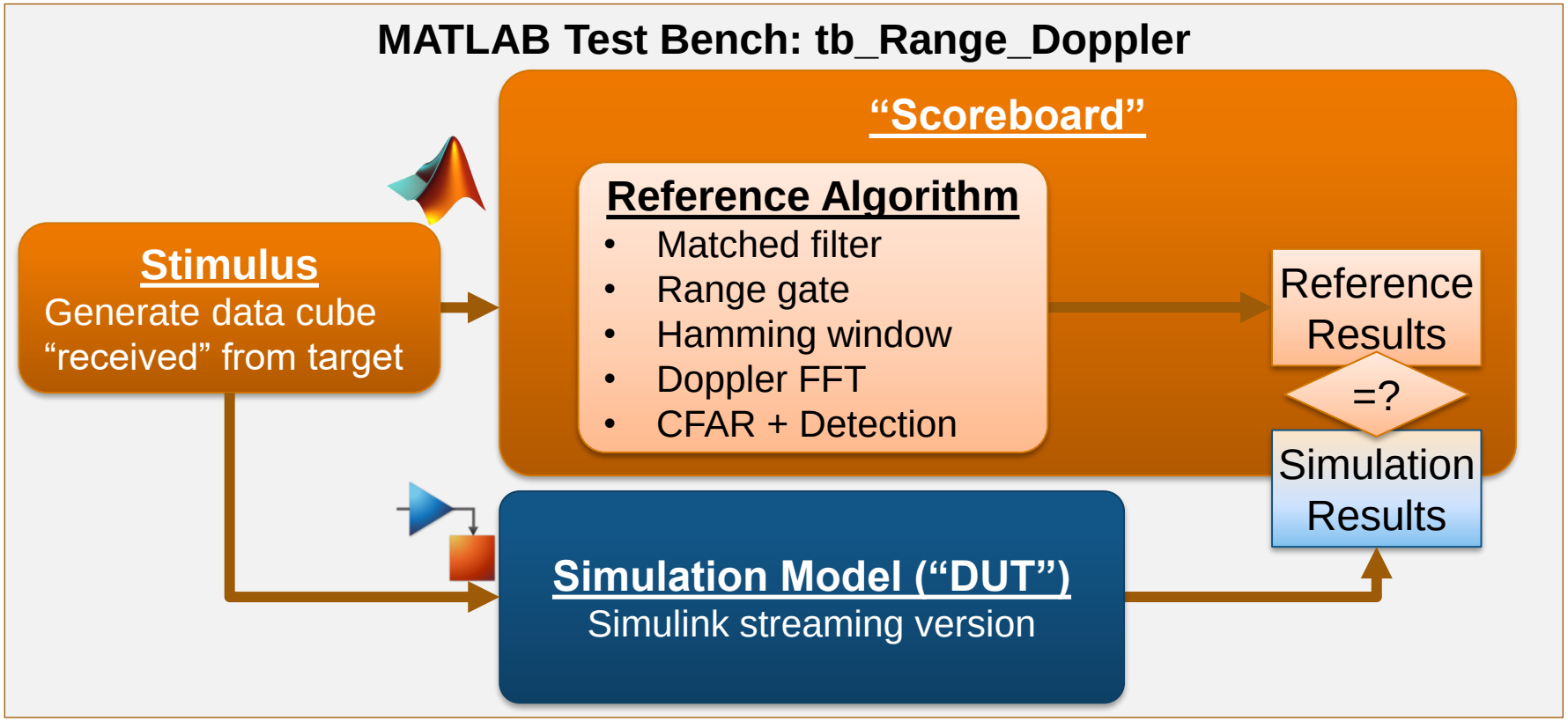
Algorithm Partitioning





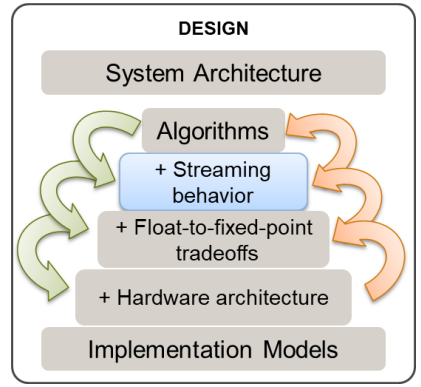
Algorithm Partitioning

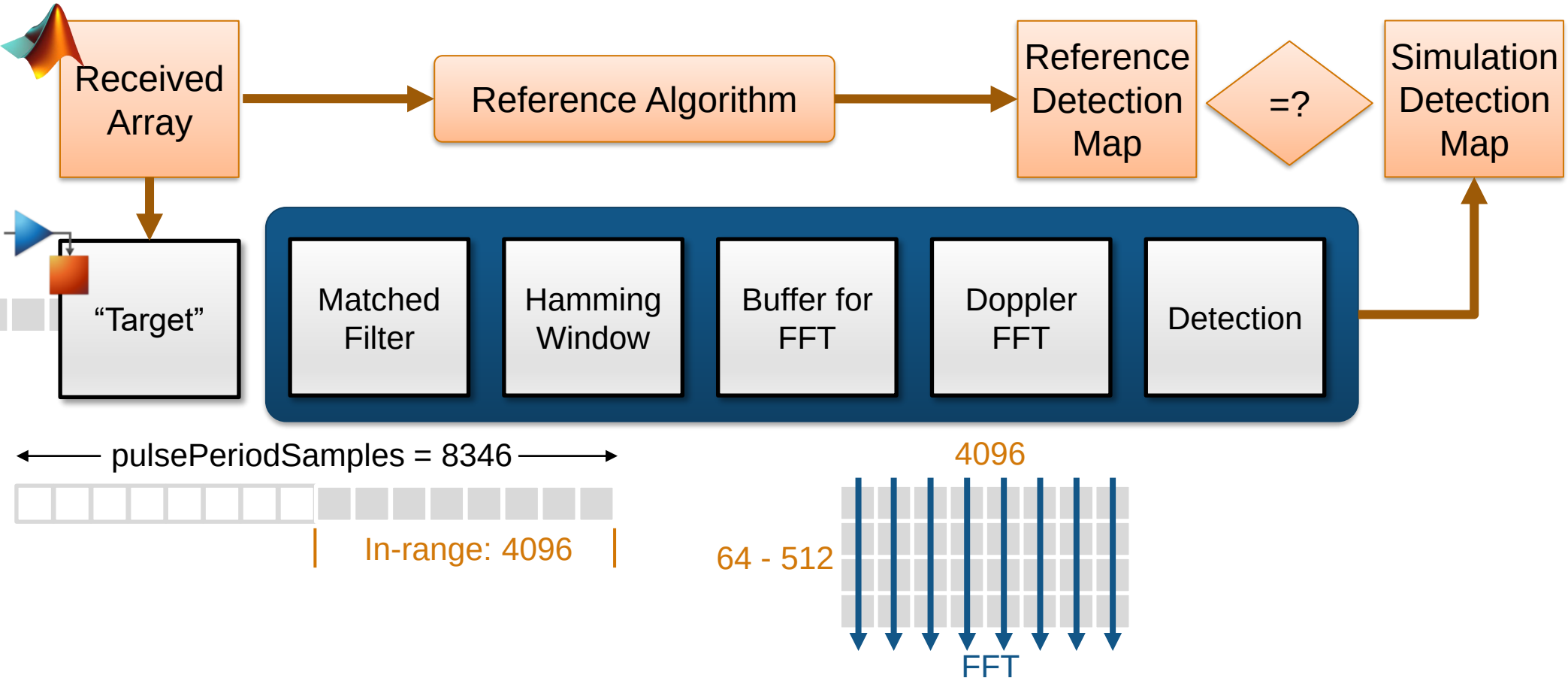




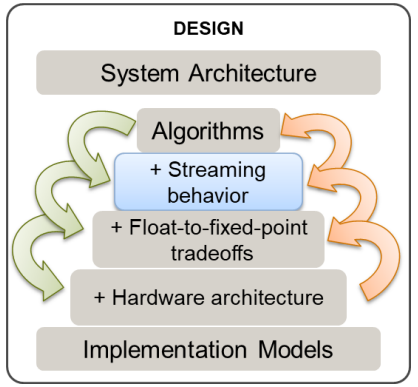
- ✓ Robust self-checking back-to-back testing
- ✓ Generate verification components
  - SystemVerilog DPI-C (via MATLAB Coder)
- ✓ Re-use functions for future projects

Simulate Streaming Behavior

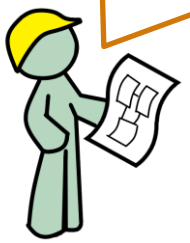


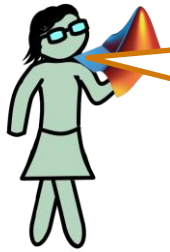


Hardware-Software Partitioning Requires Collaboration

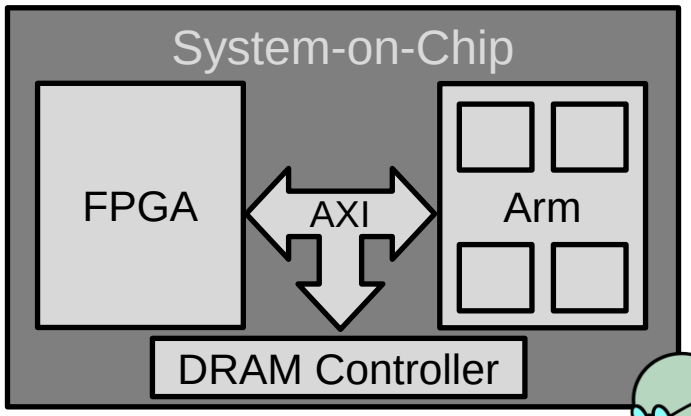


Size of data matrix:
Up to 4096 x 512
24 bits/sample (I+Q)
= 48 Mb



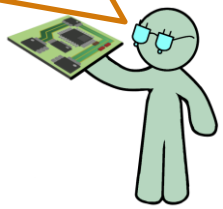


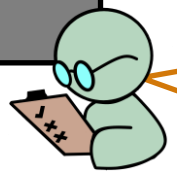
We need to match-filter the pulses then FFT across the pulses



The application processor has 4 cores we can use in parallel

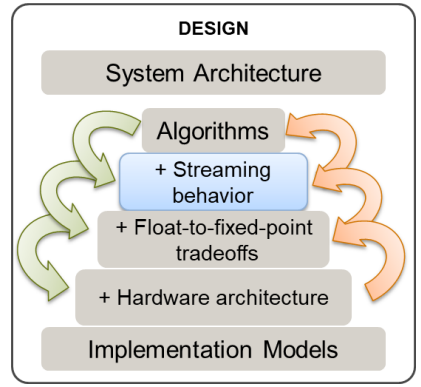
This device only has 38 Mb of RAM in the FPGA fabric



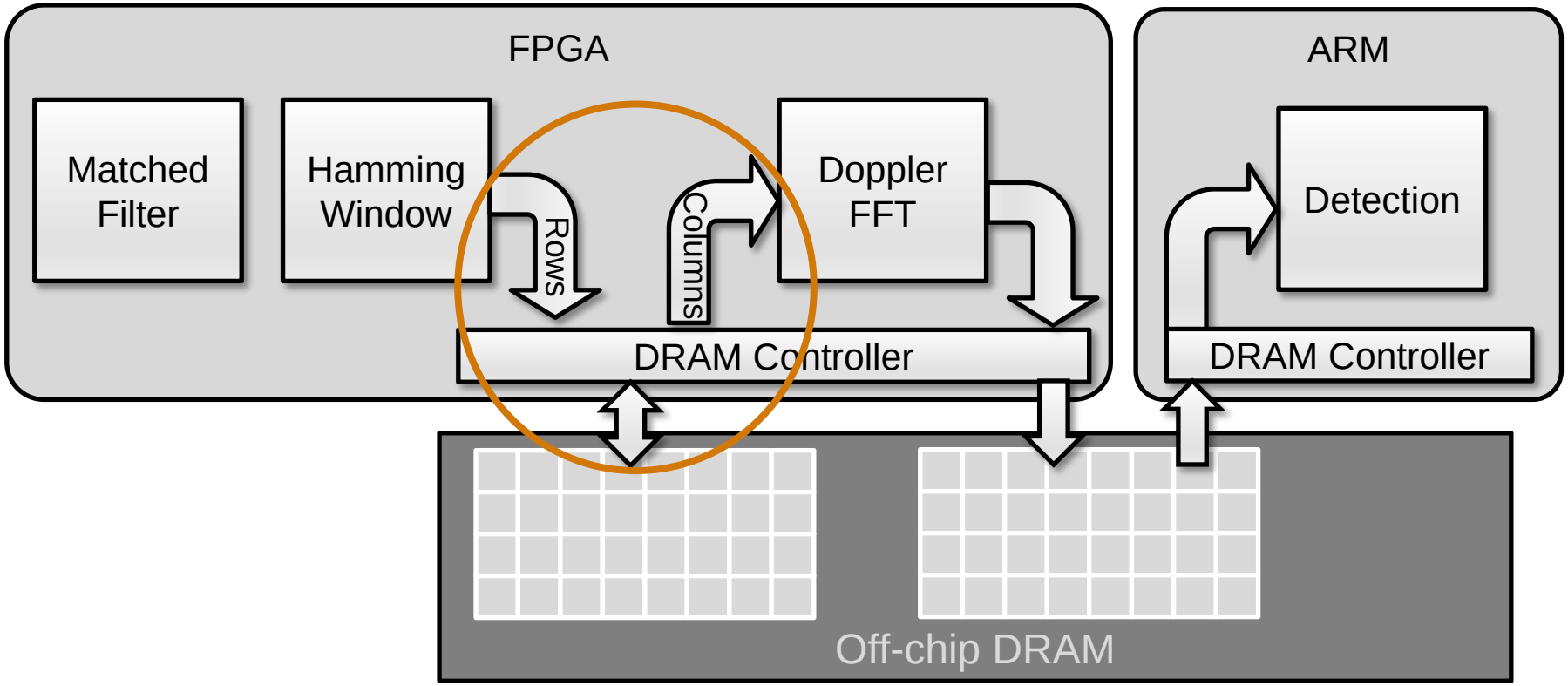


How complex is this going to be?

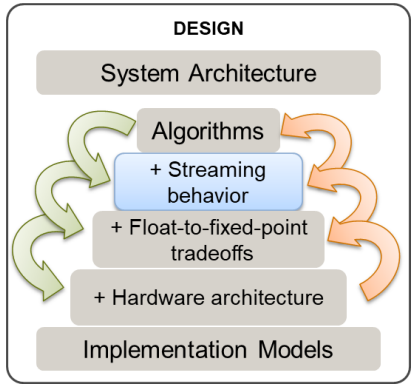
Option A: Implement FFT in Hardware



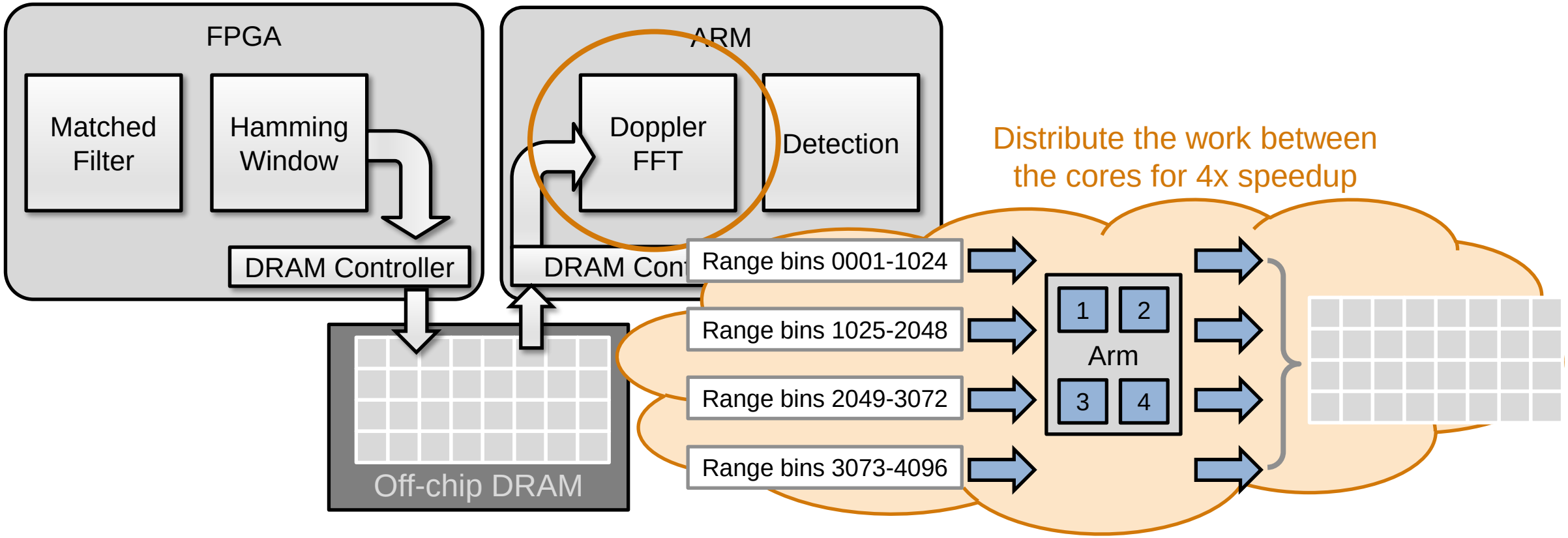
What is the latency overhead?



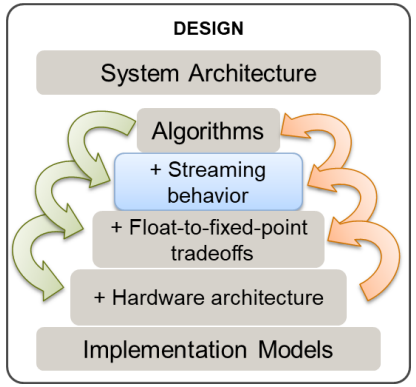
Option B: Implement FFT in Software



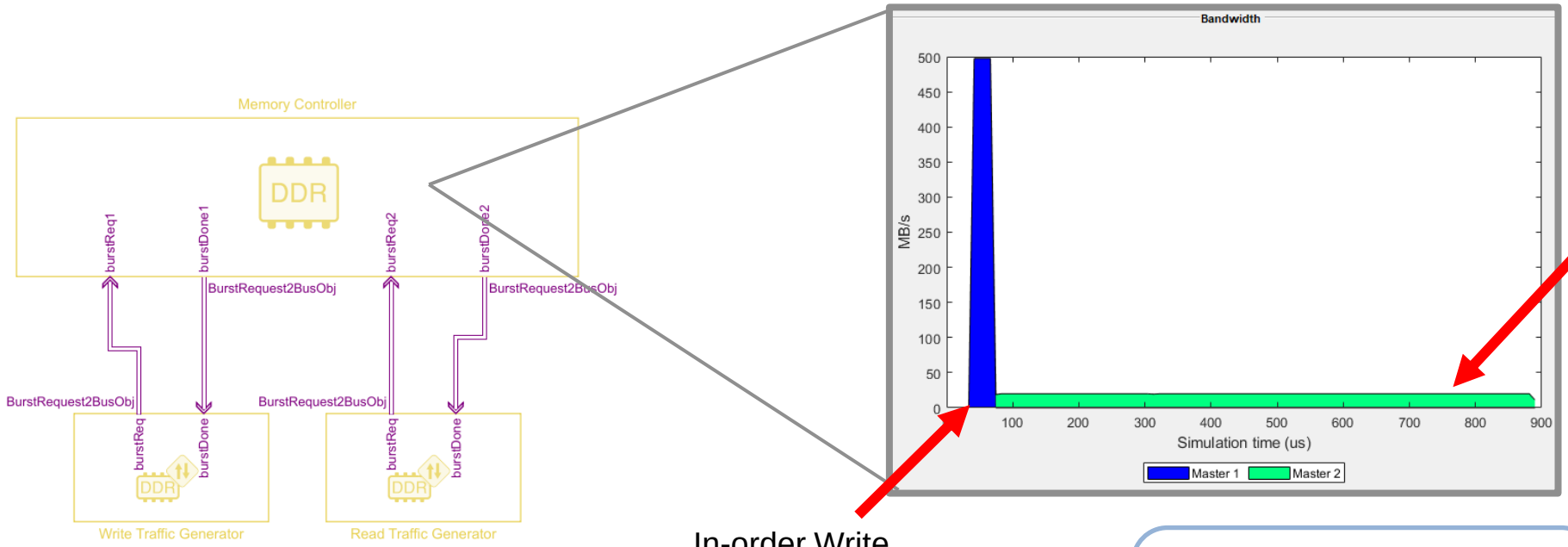
What is the latency overhead?



Analyze Option A



- Estimate DDR transpose timing with Memory Traffic Generators

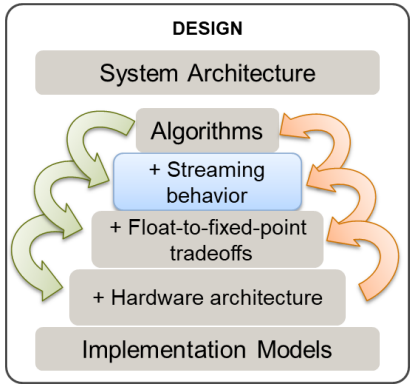


Transposed Read from DDR is slow due to 1-word bursts

In-order Write to DDR is fast due to large burst length

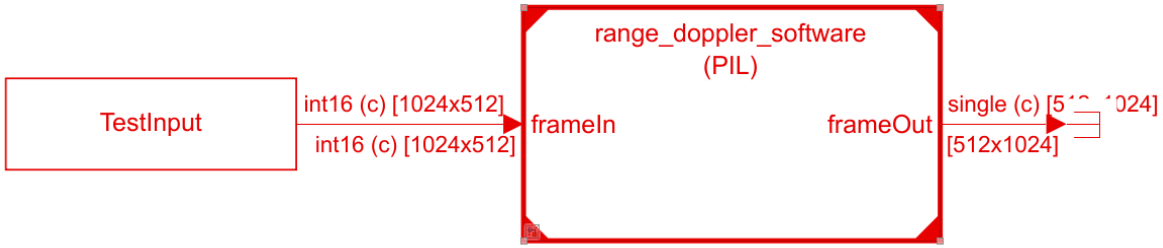
Time estimate for 4096 x 512 frame ≈ 438 ms

Analyze Option B



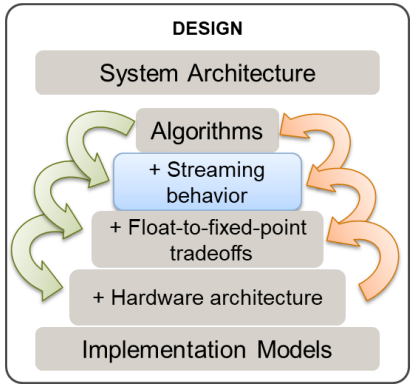
- Use Processor in the Loop (PIL) profiling to analyze timing

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls
range_doppler_softwa_initialize	860	860	860	860	1
[-] range_doppler_software [0.1 0]	478714177	475864990	2550	2236	5
Transpose	83062620	81849649	83062620	81849649	5
DTC	23243053	21569826	23243053	21569826	5
[+] FFT	372470295	372443279	1550	1426	5



Time estimate for
4096 x 512 frame
≈ 476 ms

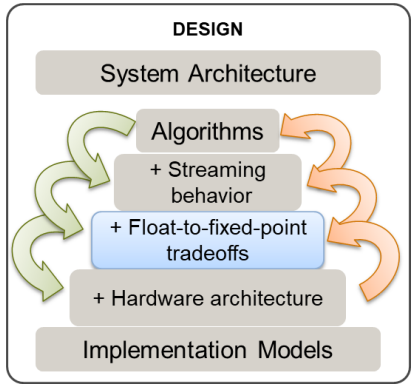
Compare Options



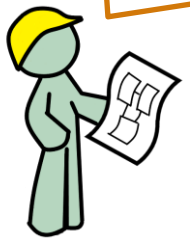
	Method	Latency	Complexity
Option A	FPGA FFT	438 ms	High
Option B	Software FFT	476 ms	Low

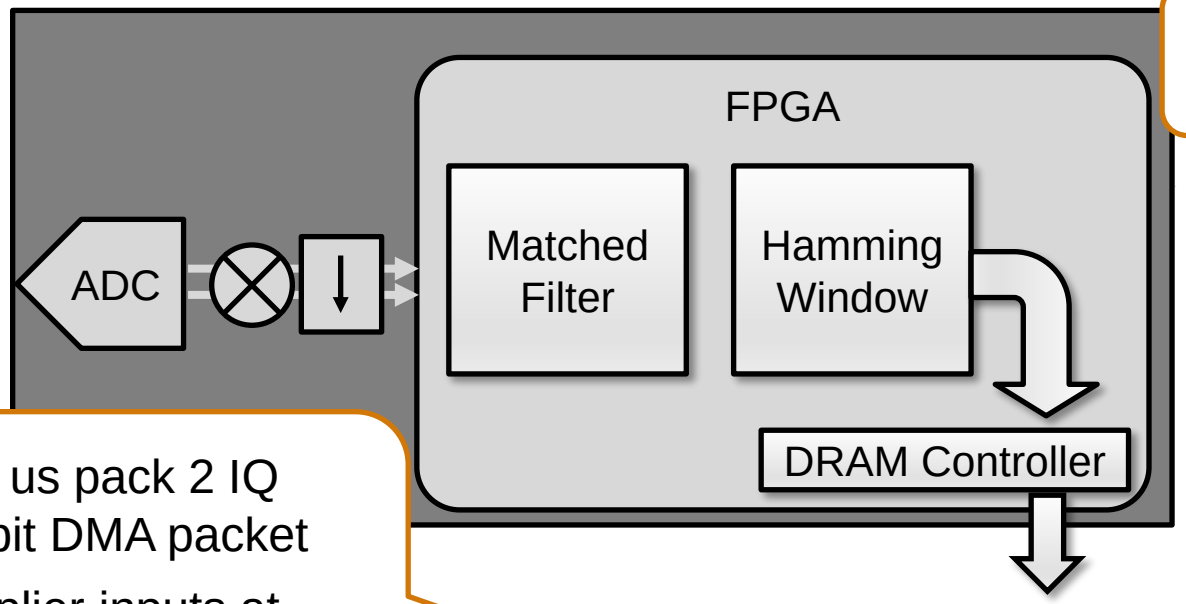
Poll: Which option would you choose?

Fixed-Point Quantization

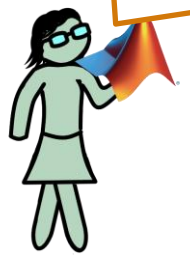


Input uses 2 ADCs
Each 24 bits/sample (I+Q)



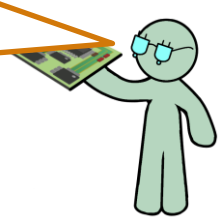


Too much noise from quantization may require adjustments to the CFAR



16-bit output lets us pack 2 IQ samples per 64-bit DMA packet

Try to keep multiplier inputs at 27x18-bits to reduce DSP usage



Need to mind our tolerances:
Range ± 10 m
Velocity ± 5 km/h

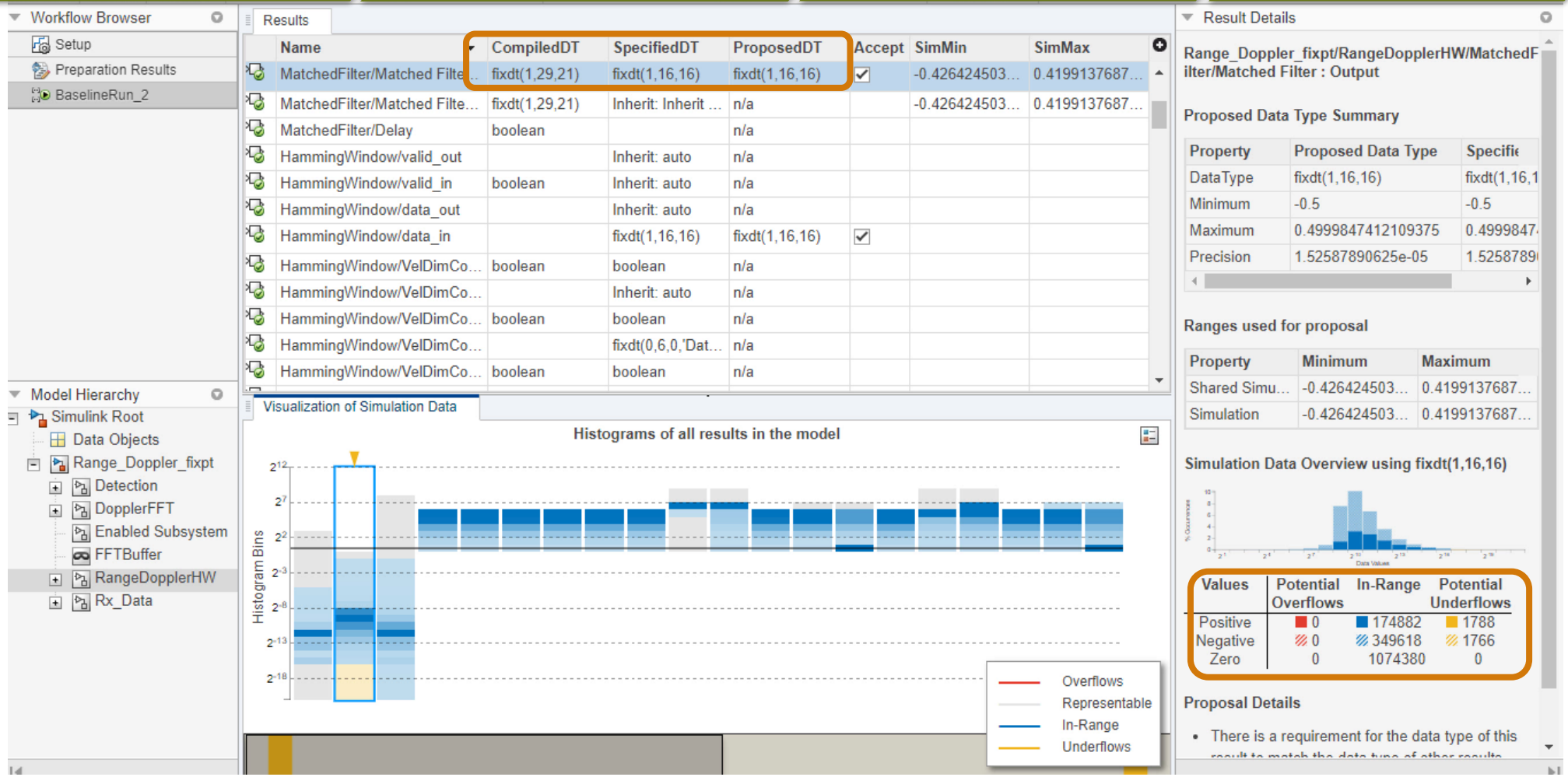
Explore and Simulate Quantization Options

Simulate with representative data to collect required ranges

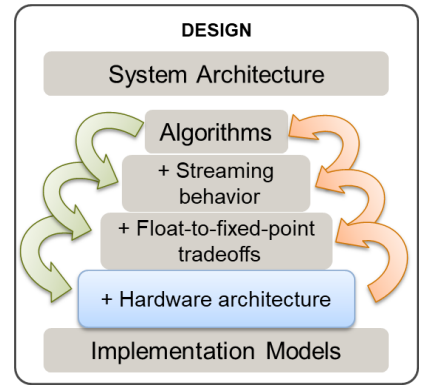
Fixed-Point Designer proposes data types

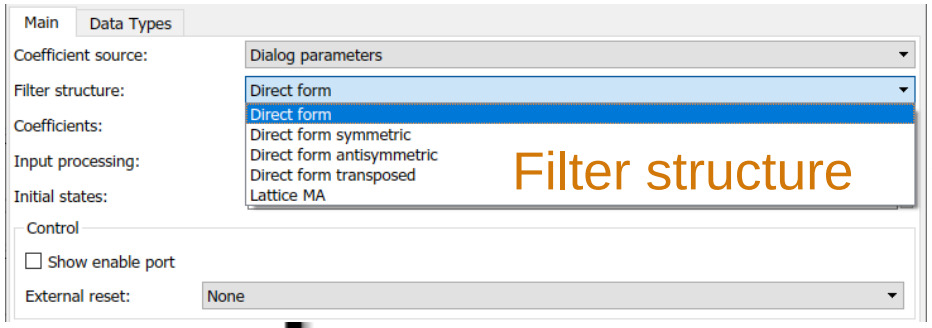
Choose to apply proposed types or set your own

Simulate and compare results



Hardware Micro-Architecture Considerations



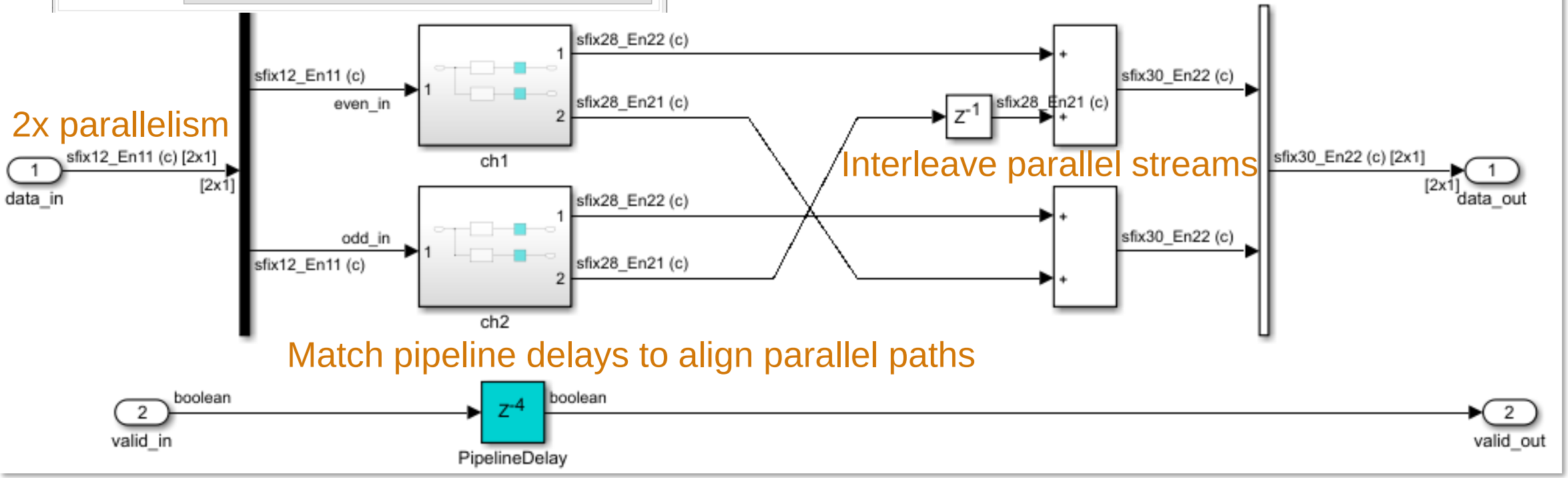


Filter structure

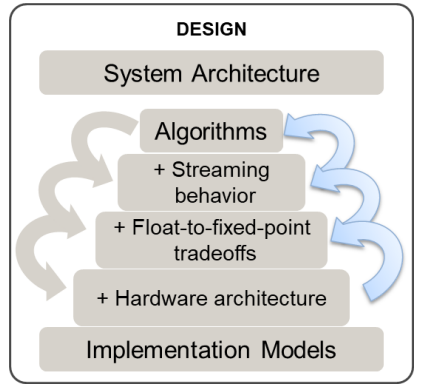
2x Vector FIR

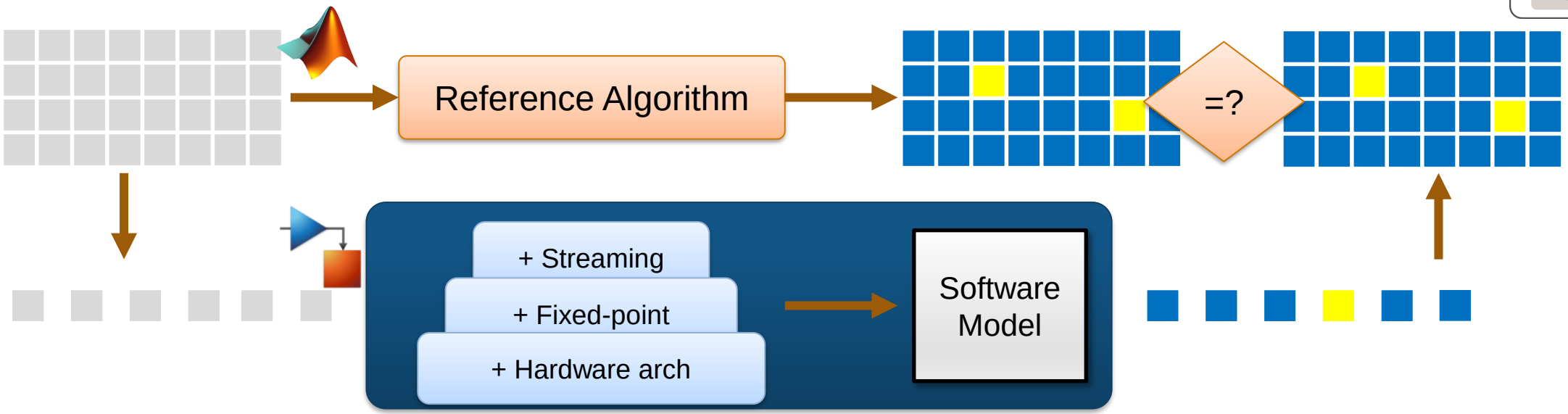
2x parallelism

Interleave parallel streams

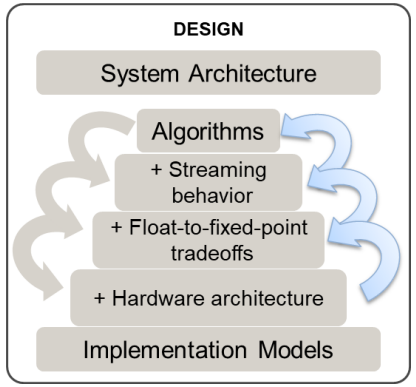


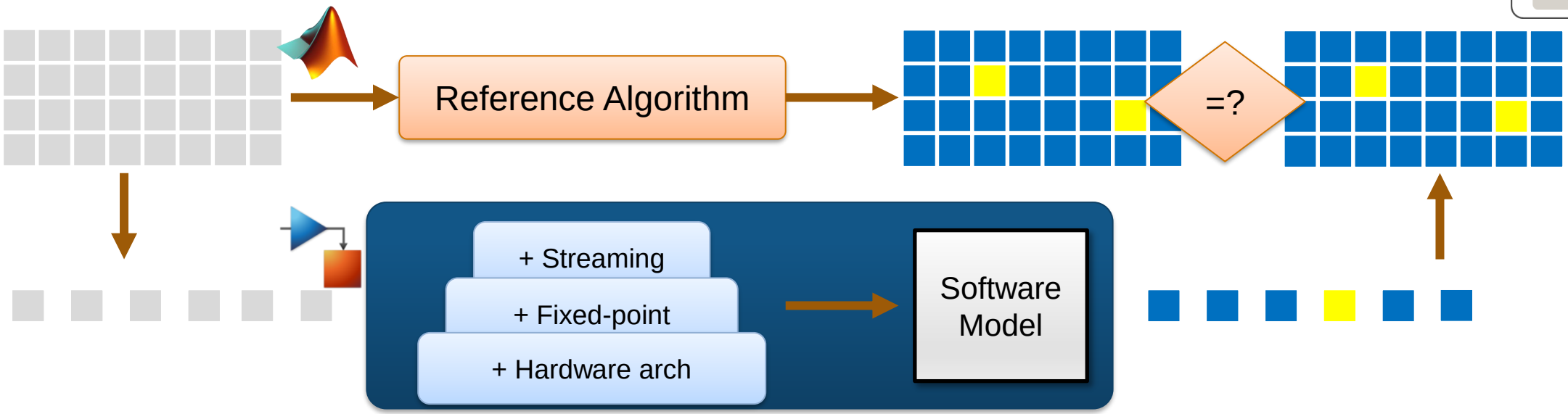
Automatically Verify Each Step
Then Generate Verification Components

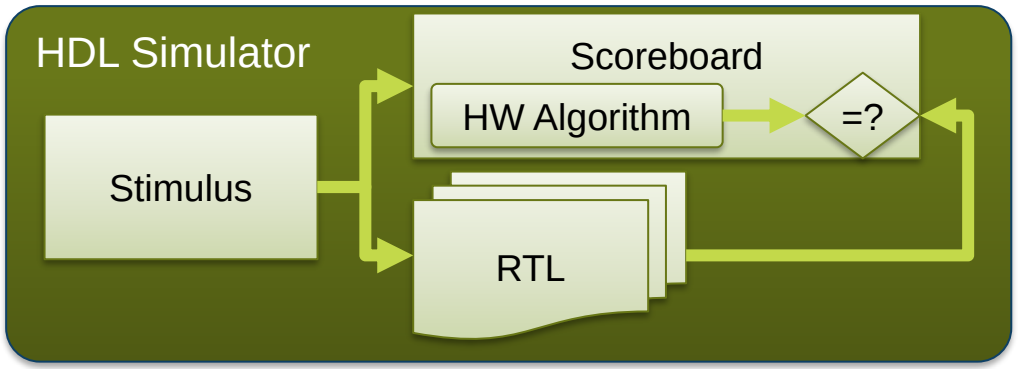




Automatically Verify Each Step
Then Generate Verification Components





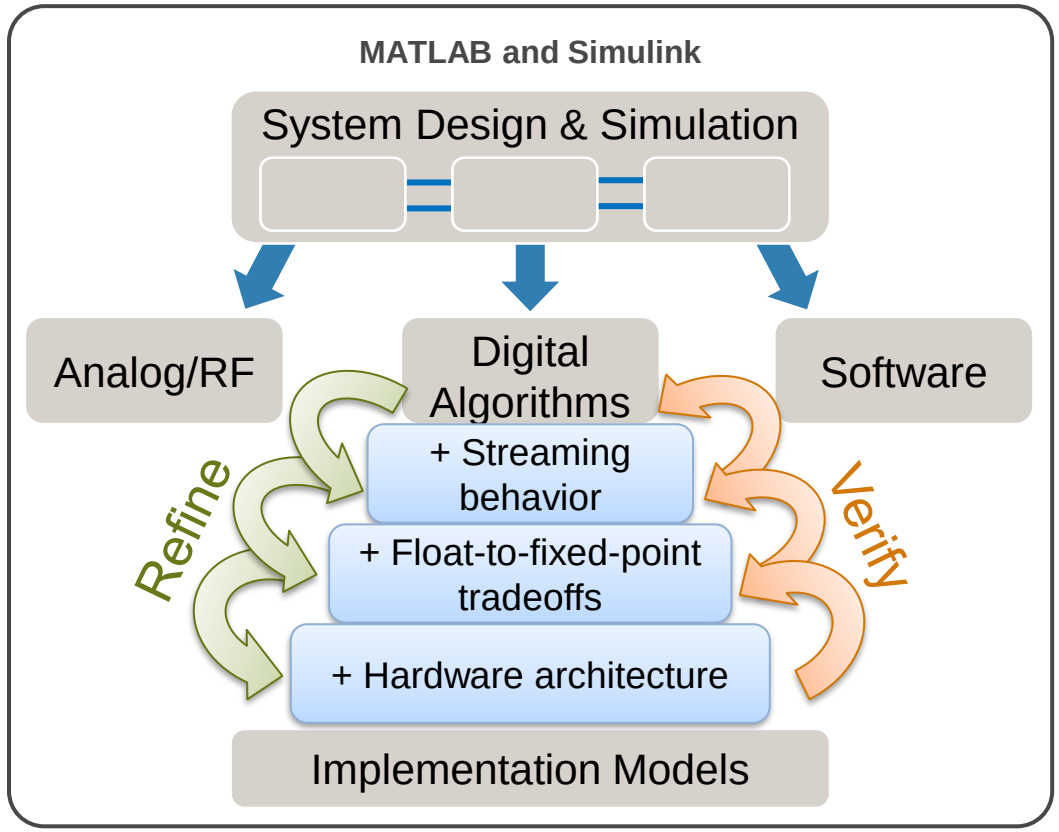


Generate SystemVerilog DPI/UVM verification test bench components

- ✓ Executable specification – no miscommunication
- ✓ Change the model, re-simulate and re-generate
- ✓ Saves RTL verification time/effort

Collaborate on Top-Down Refinement and Verification

- All roles contribute to high-level design and exploration
- Eliminate costly system-level bugs early
- Communicate via executable models
- Generate stimuli and scoreboards for verification



> "Simulink helps system architects and hardware designers communicate. It is like a shared language that enables us to exchange knowledge, ideas, and designs."
>
> **Marcel van Bakel**
> **Philips Healthcare**

Learn More

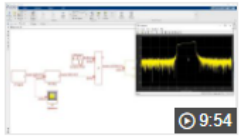
MATLAB EXPO 2021

Qualcomm	Prototype Platform for Imaging Applications
Cadence	Accelerating Design, Data Visualization and Analysis of Analog and Mixed-Signal Systems
NI	SDR Solutions with NI Hardware and MathWorks Software

Video Series

Developing Radio Applications for RFSoC with MATLAB & Simulink

Part 1: Hardware/Software Co-Design Workflow
Target SoC architectures like Xilinx UltraScale+ RFSoC devices using Model-Based Design. Build Simulink models of hardware/software platforms to make design decisions.



Part 2: System Specification and Design
System specifications for a range-Doppler radar are the driver for hardware/software implementation decisions when targeting SoC architectures like Xilinx RFSoC devices.

Part 3: Hardware/Software Partitioning
Perform simulation and analysis of the SoC architecture of the Xilinx RFSoC to investigate hardware/software partitioning of the range-Doppler radar algorithm.

Part 4: Code Generation and Deployment
Use SoC Blockset to automate the process of C and HDL code generation from Simulink models, and to automatically deploy the range-Doppler radar algorithm to a Xilinx ZCU111 development kit.

Products and Solutions

- HDL Code Generation and Verification
- MATLAB for FPGA, ASIC, and SoC Development
- HDL Coder
- HDL Verifier
- Fixed-Point Designer
- SoC Blockset

MATLAB EXPO 2021

Thank you