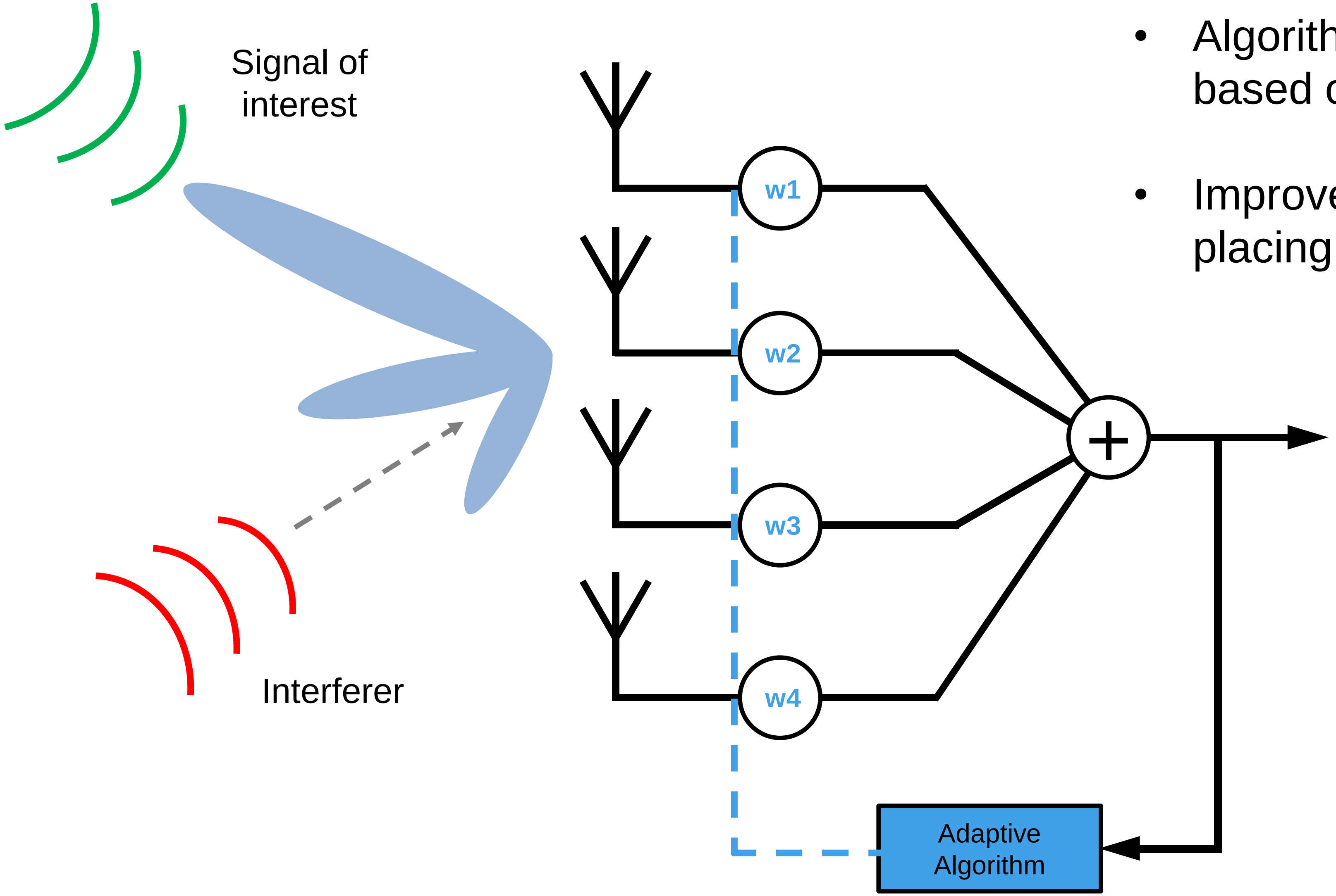
FPGA Adaptive Beamforming with HDL Coder and Zynq RFSoC

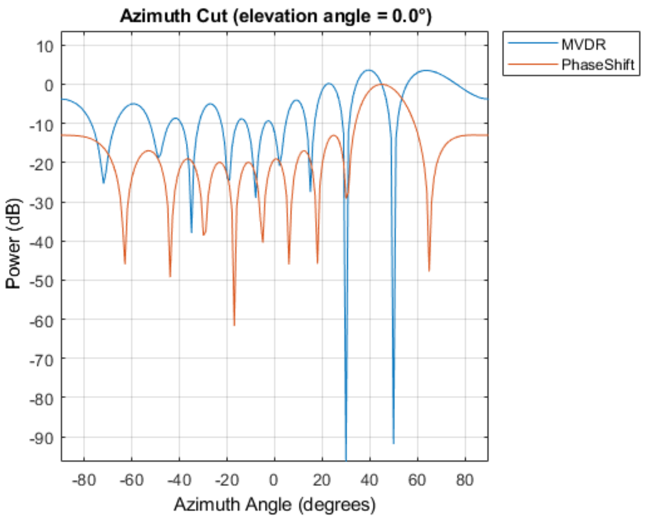
Agenda

- Introduction: Motivation and Challenges
  - Applications: Radar, Comms and Wireless
  - Hardware FPGA challenges
- Theory and Implementation
  - Linear algebra
  - QR Decomposition
  - Matrix Divide
- Zynq RFSoC and HDL Coder Implementation
  - MATLAB MVDR reference code
  - HDL Coder implementation
  - Hardware Prototyping – live demo

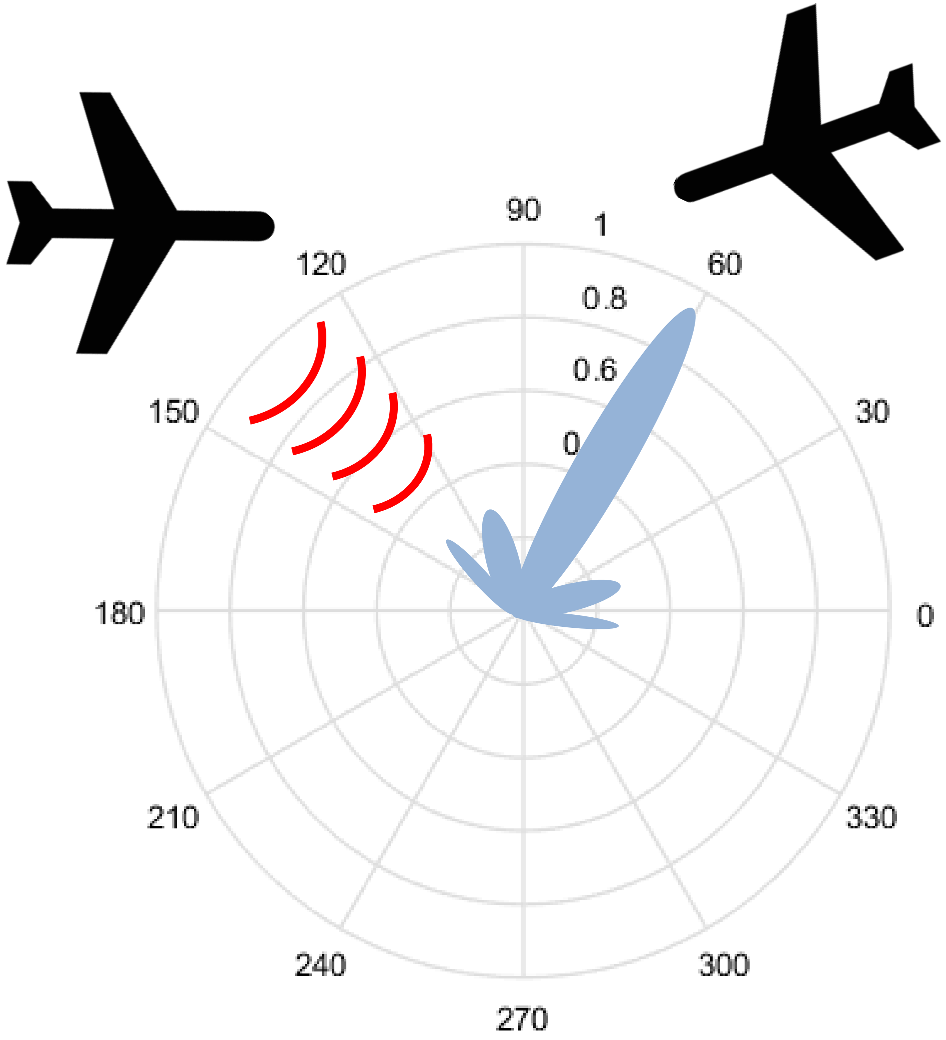
Adaptive Beamforming



- Algorithm chooses optimal weights based on receive data statistics
- Improve SNR by automatically placing nulls at interference angles



Applications: Radar



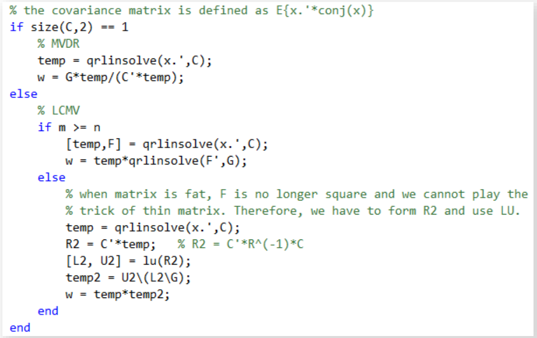
- Increase angular resolution
- Suppress interference

Applications: 5G

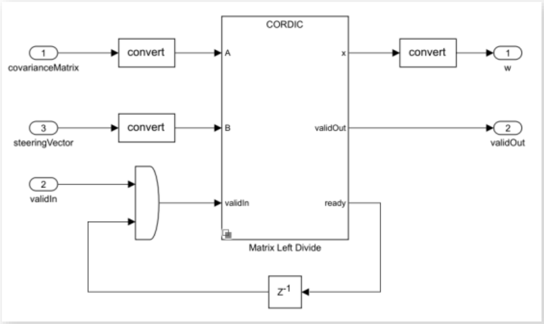
- Increase number of simultaneous users
- Improve throughput and coverage

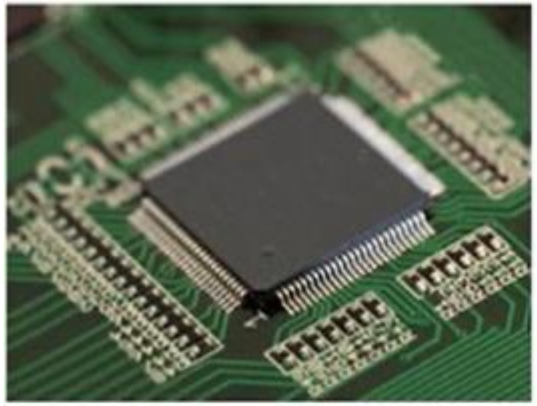
FPGA Implementation Challenges

- Fixed-Point Math
- Performance vs Area tradeoffs
- Data Rate vs Clock Rate
- Project Timeline



```
% the covariance matrix is defined as E{x.'*conj(x)}
if size(C,2) == 1
    % MVDR
    temp = qrlinsolve(x.',C);
    w = G*temp/(C'*temp);
else
    % LCMV
    if m >= n
        [temp,F] = qrlinsolve(x.',C);
        w = temp*qrlinsolve(F',G);
    else
        % when matrix is fat, F is no longer square and we cannot play the
        % trick of thin matrix. Therefore, we have to form R2 and use LU.
        temp = qrlinsolve(x.',C);
        R2 = C'*temp;    % R2 = C'*R^(-1)*C
        [L2, U2] = lu(R2);
        temp2 = U2\(L2\G);
        w = temp*temp2;
    end
end
```

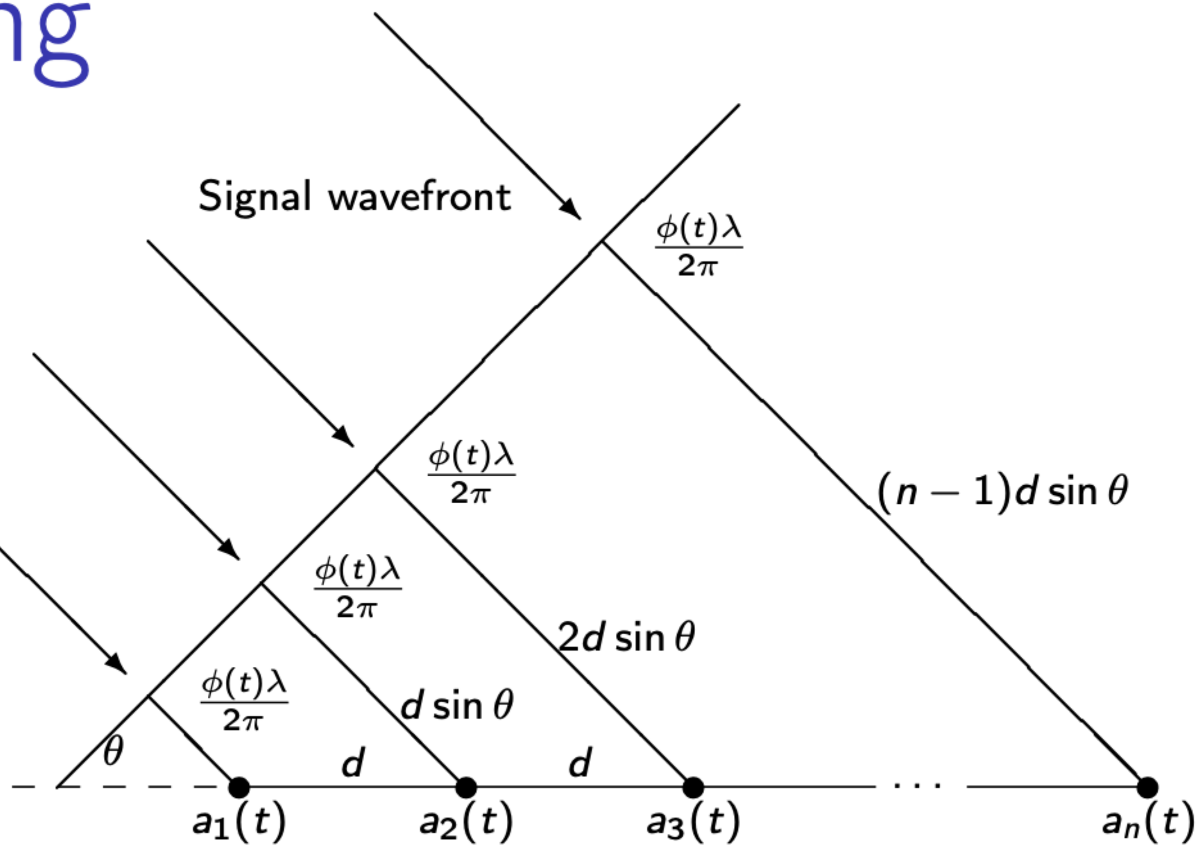




Agenda

- Introduction: Motivation and Challenges
  - Applications: Radar, Comms and Wireless
  - Hardware FPGA challenges
- **Theory and Implementation**
  - Linear algebra
  - QR Decomposition
  - Matrix Divide
- Zynq RFSoC and HDL Coder Implementation
  - MATLAB MVDR reference code
  - HDL Coder implementation
  - Hardware Prototyping – live demo

Beamforming



m samples, n antenna elements, $m \gg n$.

m-by-n data matrix A.

$a(t)$ is an n-by-1 column vector. $a(t)^H$ form the rows of A.

Unified notation

- A is an m-by-n data matrix
- $m \gg n$
- Beamformer problem: Solve $(A^H A)x = b$ where b is the steering vector.

Beamformer problem

1. Estimate correlation matrix: $m\mathcal{R}_{aa} = \sum_{t=1}^{m} a(t)a(t)^H = A^H A$
2. Solve $(A^H A)x = b$ where b is the steering vector.
3. Compute upper-triangular Cholesky factor of $(A^H A)$ and do forward and backward substitution.
4. QR vs. Cholesky
    - $R = \mathrm{chol}(A^H A) \rightarrow A^H A = R^H R$
    - $[Q, R] = \mathrm{qr}(A) \rightarrow QR = A \rightarrow A^H A = R^H Q^H QR = R^H R$

QR vs. Cholesky

Avoid computing $A^H A$ if you can. Never compute inverse.

Direct least-squares solution	Normal equations least-squares
$Ax = b$	$x = (A^H A)^{-1} A^H b$
$(A^H A)x = b$	$x = (A^H A)^{-1} b$
`R = fixed.qlessQR(A)`	`R = chol(A'*A)`
`5.6648   2.3256  -0.8496`	`5.6648   2.3256  -0.8496`
`     0   3.5967  -0.9131`	`     0   3.5967  -0.9131`
`     0        0   2.4822`	`     0        0   2.4822`

```
fixed.qlessQR(A) == [~,R] = qr(A,0)
```

Minimum Variance Distortionless Response (MVDR) Beamformer

Matrix Solve Using Q-less QR Decomposition

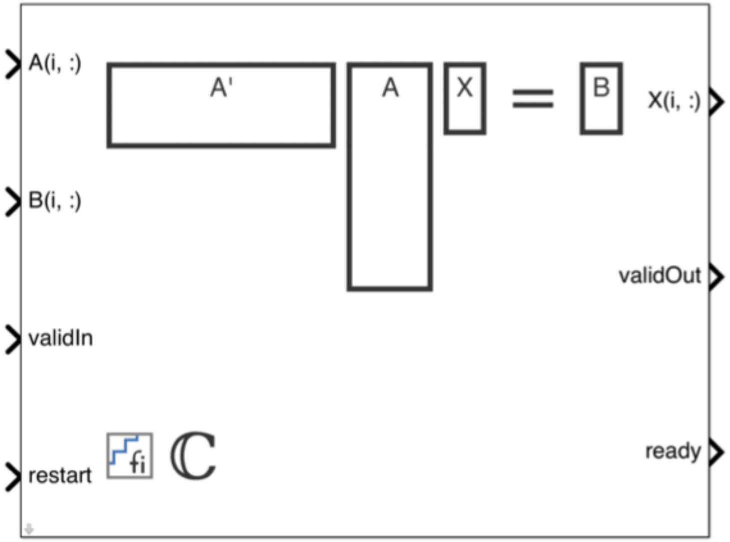
$$(A^H A)x = b$$

MVDR weight vector

$$w = \frac{x}{b^H x}$$

MVDR response

$$y = w^H a(t)$$



Method	*Input*	*Ready*	*Latency*	*Area*	*Release*
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b

MVDR with continuously streaming data

Matrix Solve Using Q-less QR Decomposition with Forgetting Factor

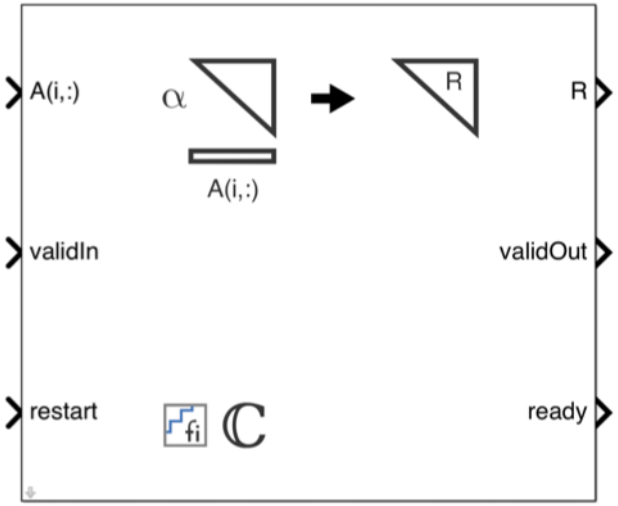
$$(A^H A)x = b$$

MVDR weight vector

$$w = \frac{x}{b^H x}$$

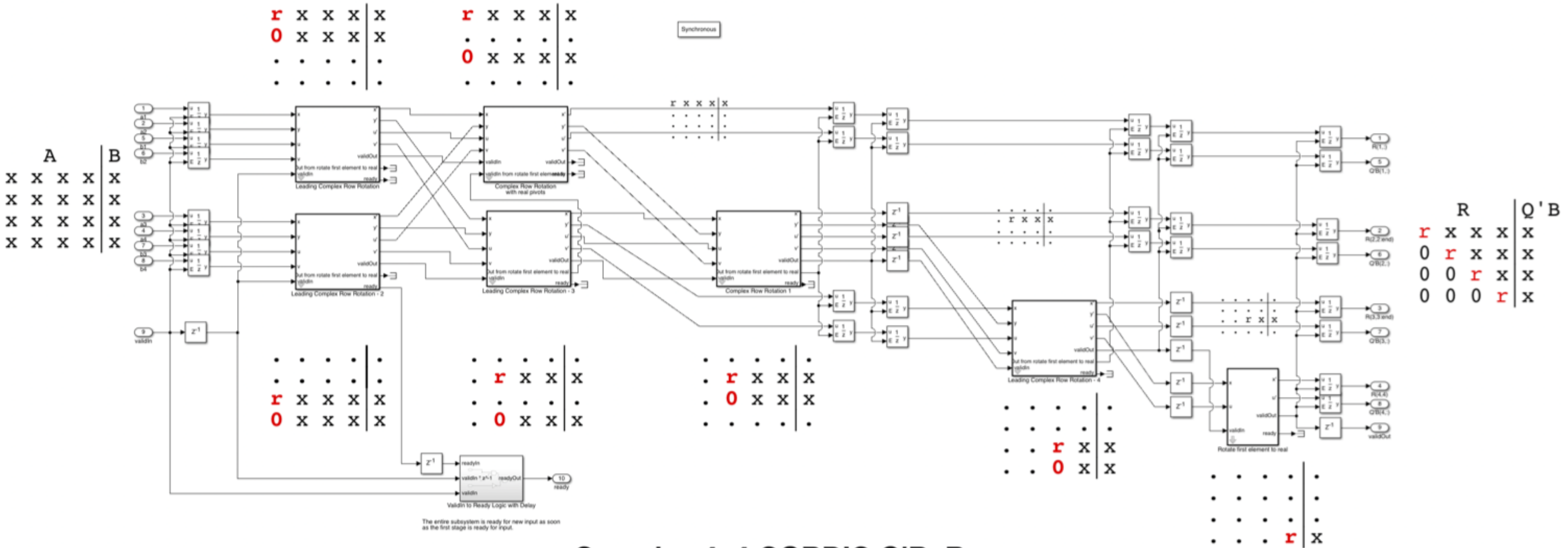
MVDR response

$$y = w^H a(t)$$



Method	Input	Ready	Latency	Area	Release
Partial-Systolic	Row	C	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	R2020b

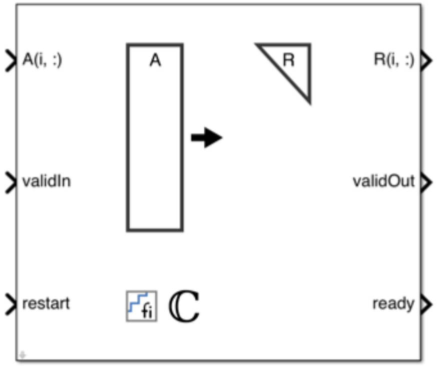
Systolic: One cell for each zero ($\mathcal{O}(mn)$ cells). High area, Low latency.

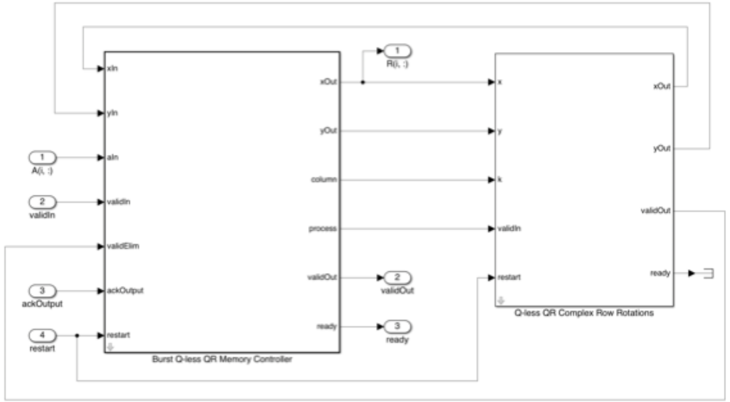


Complex 4x4 CORDIC Q'B, R

Method	Input	Ready	Latency	Area	Release
Systolic	Matrix	C	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	R2019a Example
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a Library blocks
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b Library blocks
Partial-Systolic with Forgetting Factor	Row	C	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	R2020b Library blocks

Burst: One cell for all zeros (1 cell). Low area, High latency.

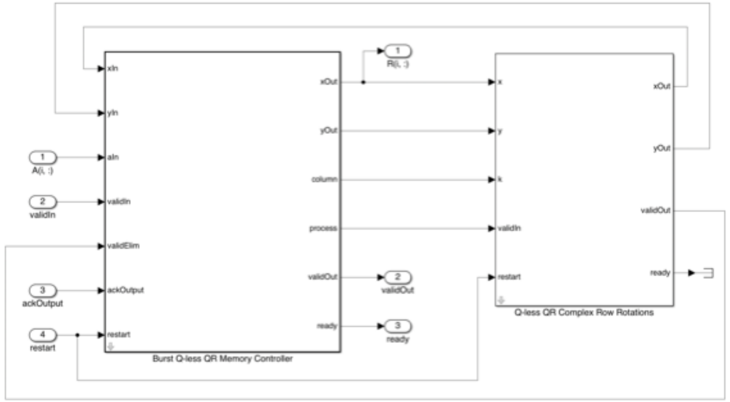




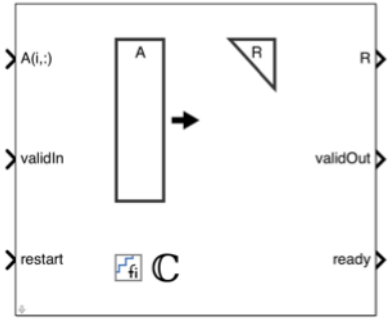
Method	*Input*	*Ready*	*Latency*	*Area*	*Release*
Systolic	Matrix	C	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	R2019a Example
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a Library blocks
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b Library blocks
Partial-Systolic with Forgetting Factor	Row	C	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	R2020b Library blocks

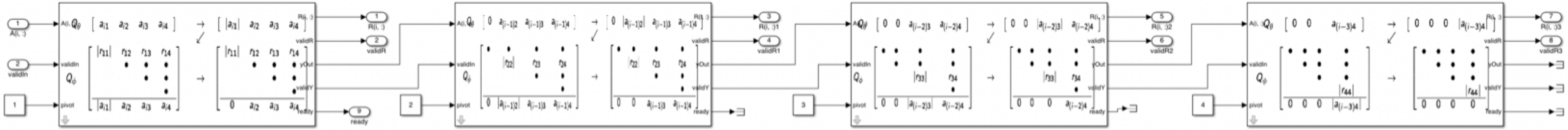
Partial-Systolic: (n cells). Medium area, Medium latency.

Cell internals



Block



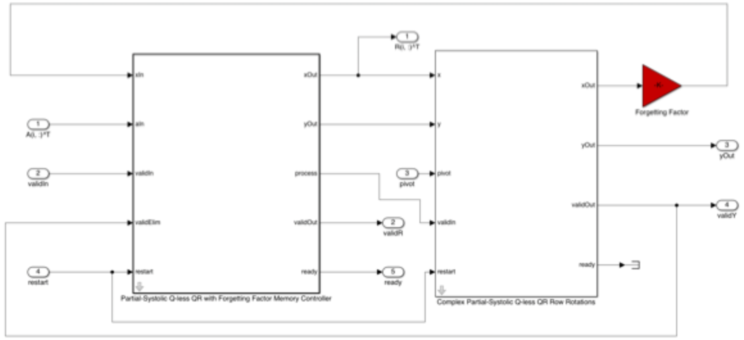


Cell 1. Update R(1,:) Cell 2. Update R(2,:) Cell 3. Update R(3,:) Cell 4. Update R(4,:)

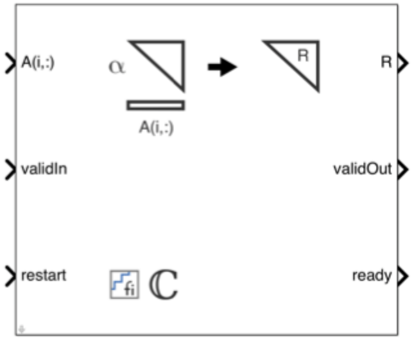
Method	*Input*	*Ready*	*Latency*	*Area*	*Release*
Systolic	Matrix	C	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	R2019a Example
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a Library blocks
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b Library blocks
Partial-Systolic with Forgetting Factor	Row	C	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	R2020b Library blocks

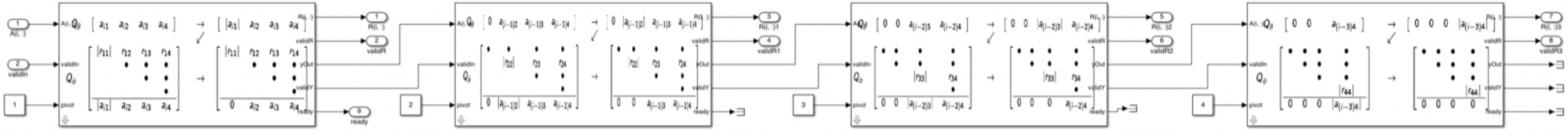
Partial-Systolic with Forgetting Factor (n cells):Continuously update

Cell internals with forgetting factor



Block

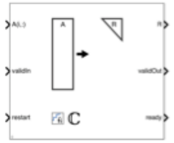




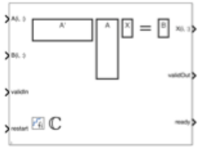
Cell 1. Update R(1,:) Cell 2. Update R(2,:) Cell 3. Update R(3,:) Cell 4. Update R(4,:)

Method	*Input*	*Ready*	*Latency*	*Area*	*Release*
Systolic	Matrix	C	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	R2019a Example
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a Library blocks
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b Library blocks
Partial-Systolic with Forgetting Factor	Row	C	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	R2020b Library blocks

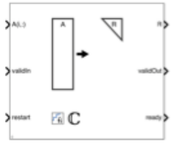
MATLAB functions



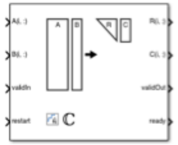
`fixed.qlessQR`



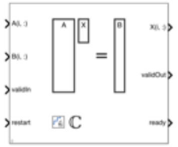
`fixed.qlessQRMatrixSolve`



`fixed.qlessQRUpdate`



`fixed.qrAB`



`fixed.qrMatrixSolve`

Agenda

- Introduction: Motivation and Challenges
  - Applications: Radar, Comms and Wireless
  - Hardware FPGA challenges
- Theory and Implementation
  - Linear algebra
  - QR Decomposition
  - Matrix Divide
- **Zynq RFSoC and HDL Coder Implementation**
  - MATLAB MVDR reference code
  - HDL Coder implementation
  - Hardware Prototyping – live demo

HDL Implementation Workflow

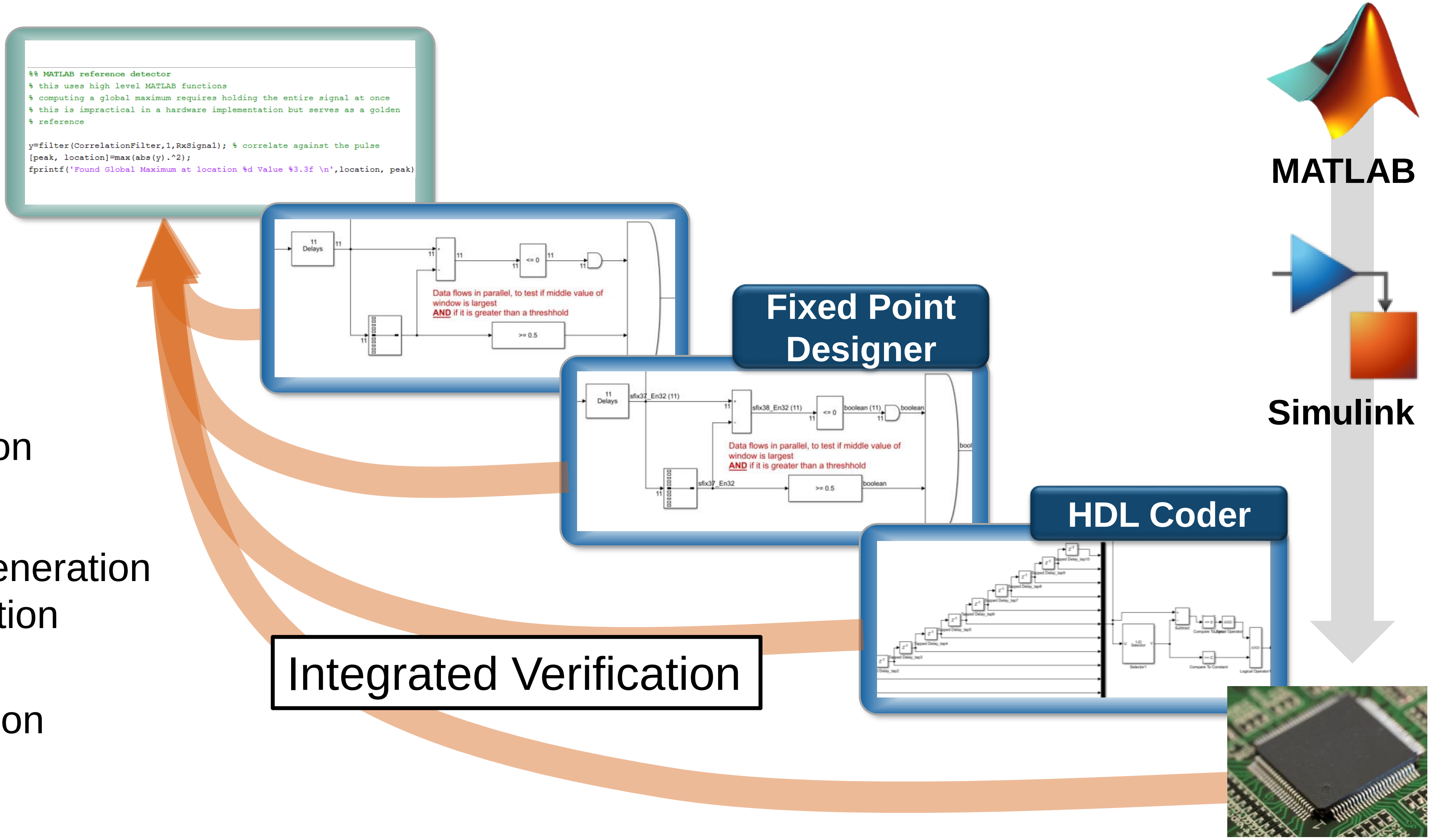
MATLAB Reference

Hardware Architecture

Fixed-point Implementation

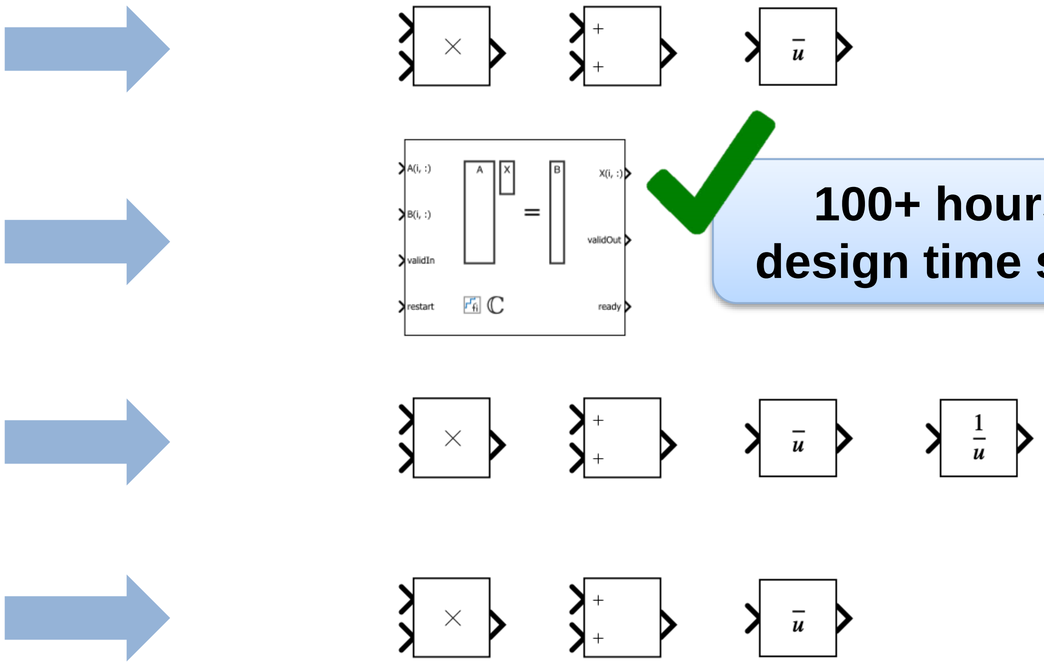
HDL Code Generation and Optimization

HDL Verification and Targeting



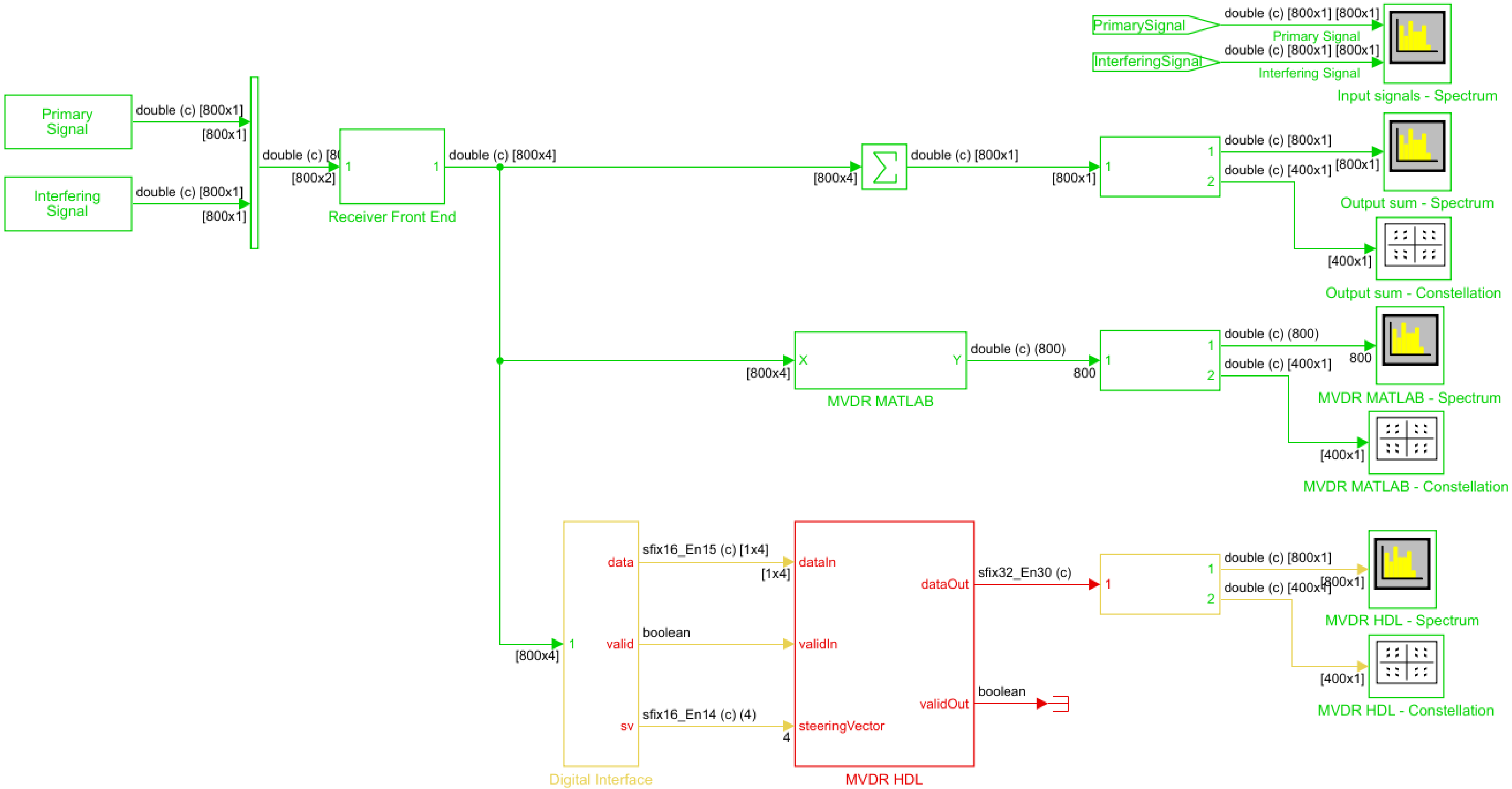
MATLAB MVDR reference code

```matlab
function Y = mvdr_beamform(X, sv)

% form covariance matrix
Ecx = X.'*conj(X);

% compute weight vector
wp = Ecx\sv;

% normalize response
w = wp/(sv'*wp);

% form output beam
Y = X*conj(w);

end
```

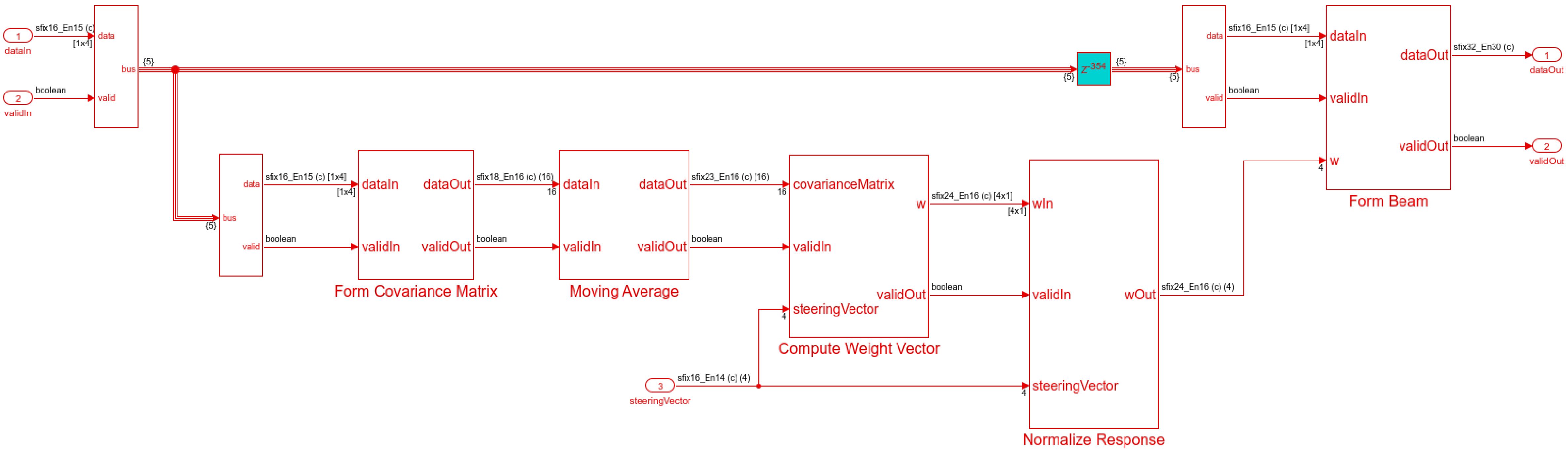


100+ hours of design time saved!

HDL Implementation of MVDR Beamforming



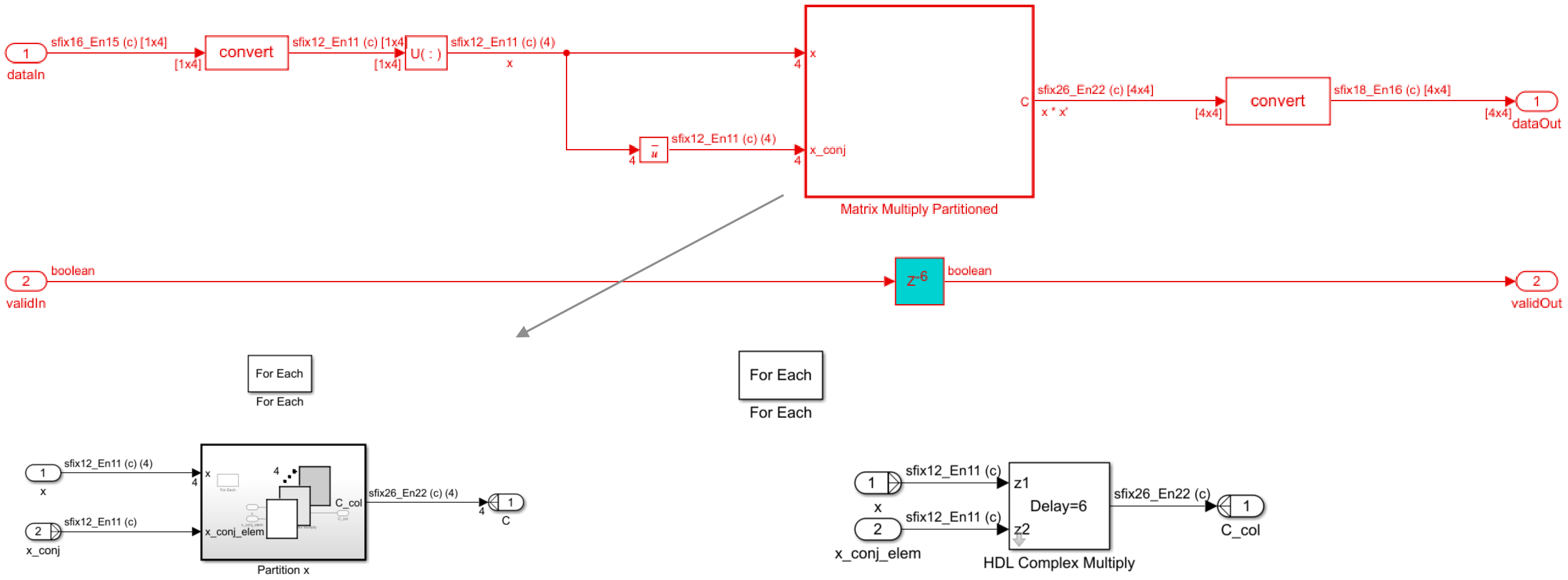
HDL Implementation of MVDR Beamforming



Form Covariance Matrix

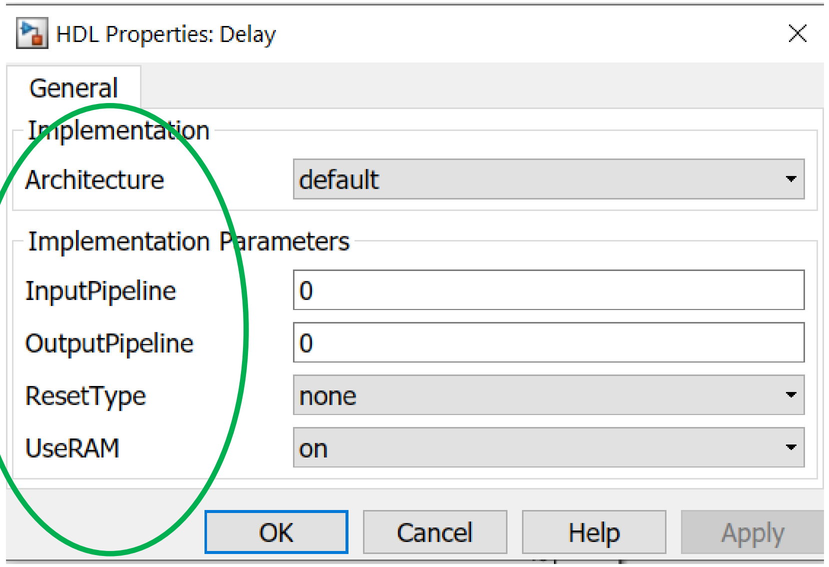
- For Each subsystem
  - Process elements independently
  - Concatenate results into outputs

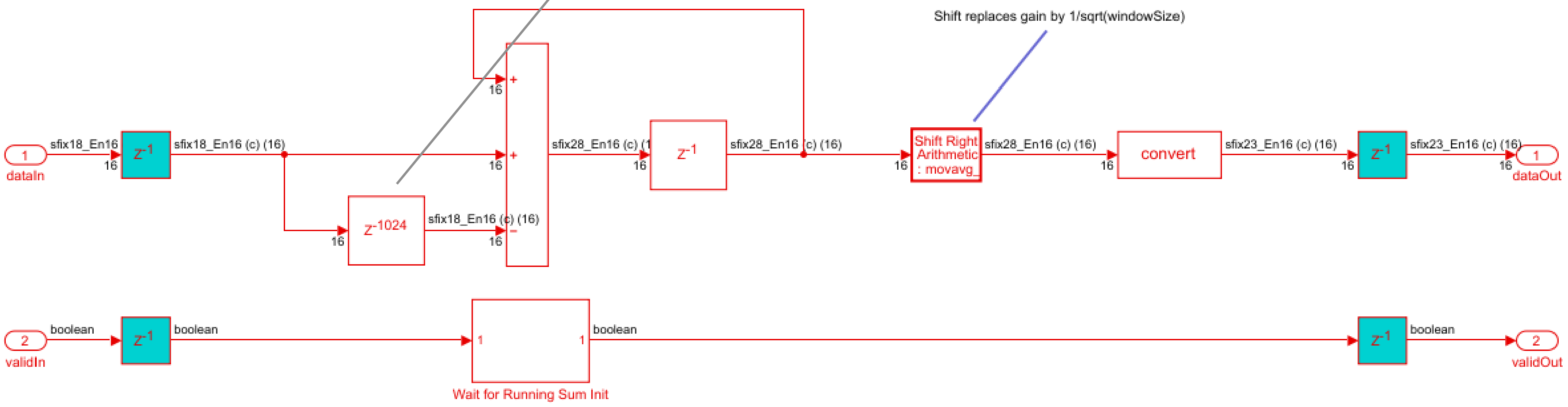
```
% form covariance matrix
Ecx = X.'*conj(X);
```



Moving Average

- Use HDL Implementation properties to map large delays to Block RAM

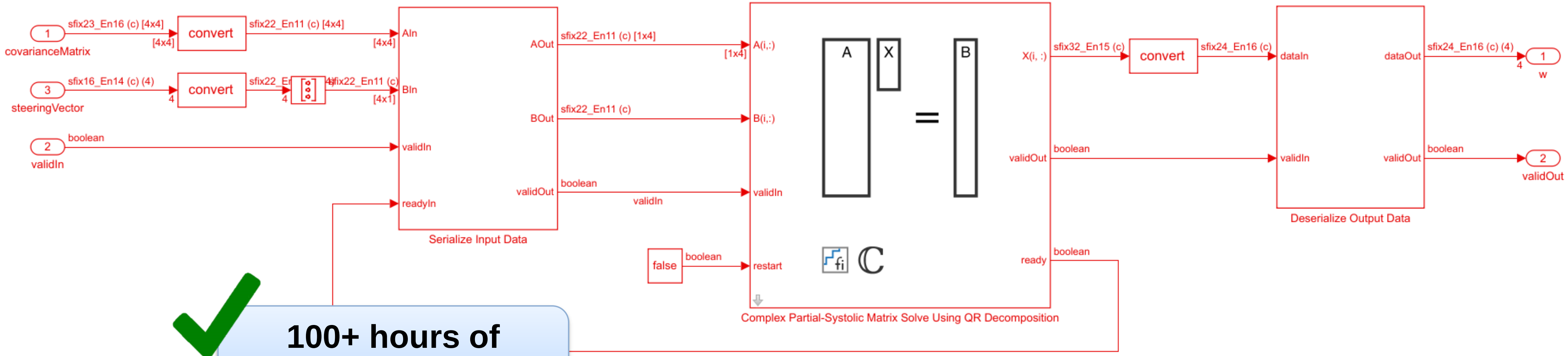




Compute Weight Vector

- Use Complex Matrix Solve block from Fixed-Point Matrix Linear Algebra Library

```
% compute weight vector
wp = Ecx\sv;
```

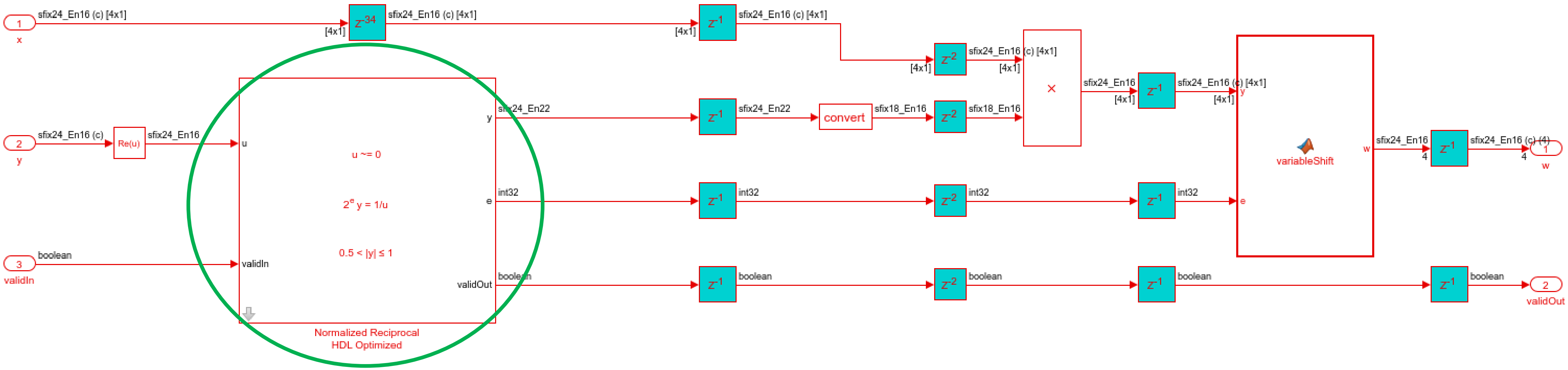


100+ hours of design time saved!

Normalize Response

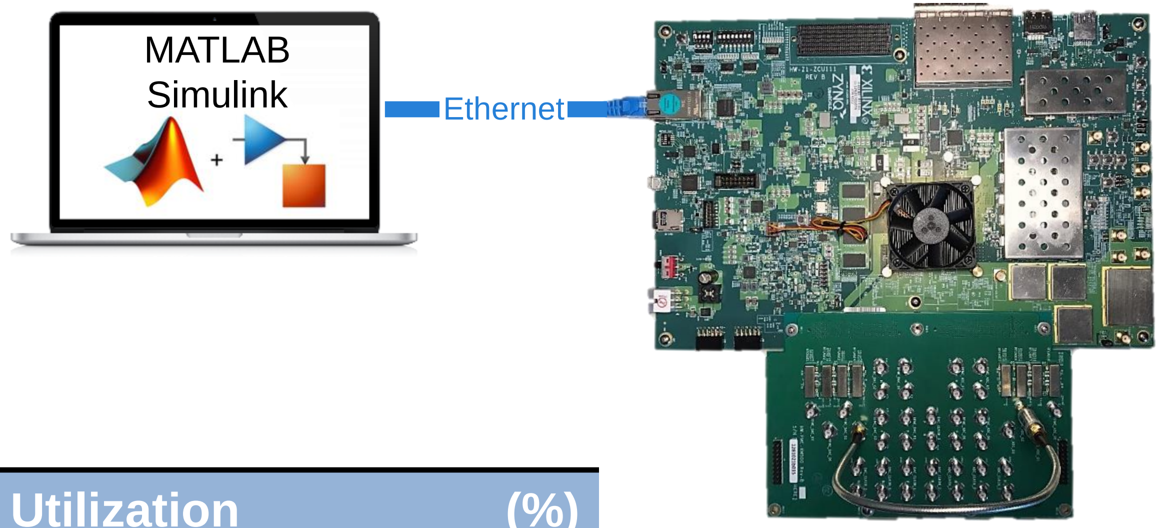
- Perform divide using reciprocal and multiply
- Fixed-point CORDIC reciprocal “just works”

```
% normalize response
w = wp/(sv'*wp);
```



Implementation Results

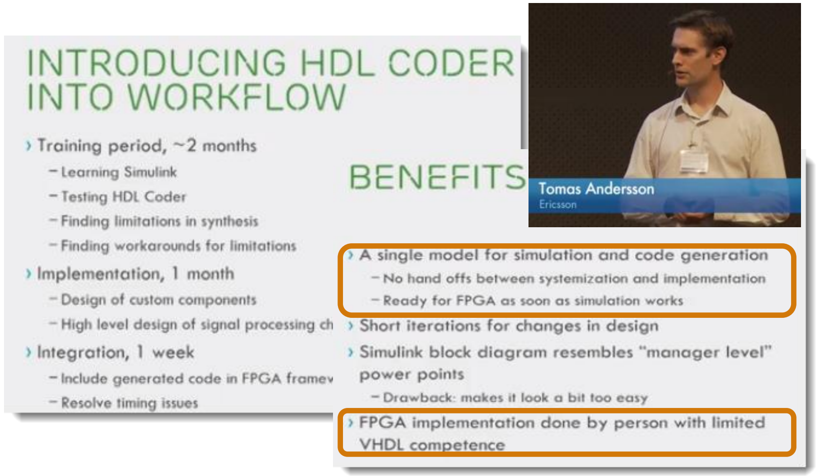
Xilinx ZCU111 RFSoC Eval Board



- Device: xczu28dr (ZCU111)
- Maximum frequency: 452 MHz
- Resource utilization:

Resource	Utilization	(%)
LUT	47K	11.13
LUTRAM	989	0.5
FF	40K	4.7
BRAM	2	0.2
URAM	10	12.5
DSP	92	3.5

Resources to Get Started and Speed Adoption



- Getting started:
  - MATLAB Onramp
  - Simulink Onramp
  - HDL pulse detector self-guided tutorial and videos
- Proof-of-concept guided evaluations
  - **FREE** support via weekly WebEx meetings using custom sample designs
  - MathWorks coaches customers on “how to fish” through weekly WebEx sessions
- Training & consulting services
  - HDL code generation, FPGA signal processing & Zynq programming training courses
  - Consulting service on deep technical coaching, custom design / hardware and more

Q&A