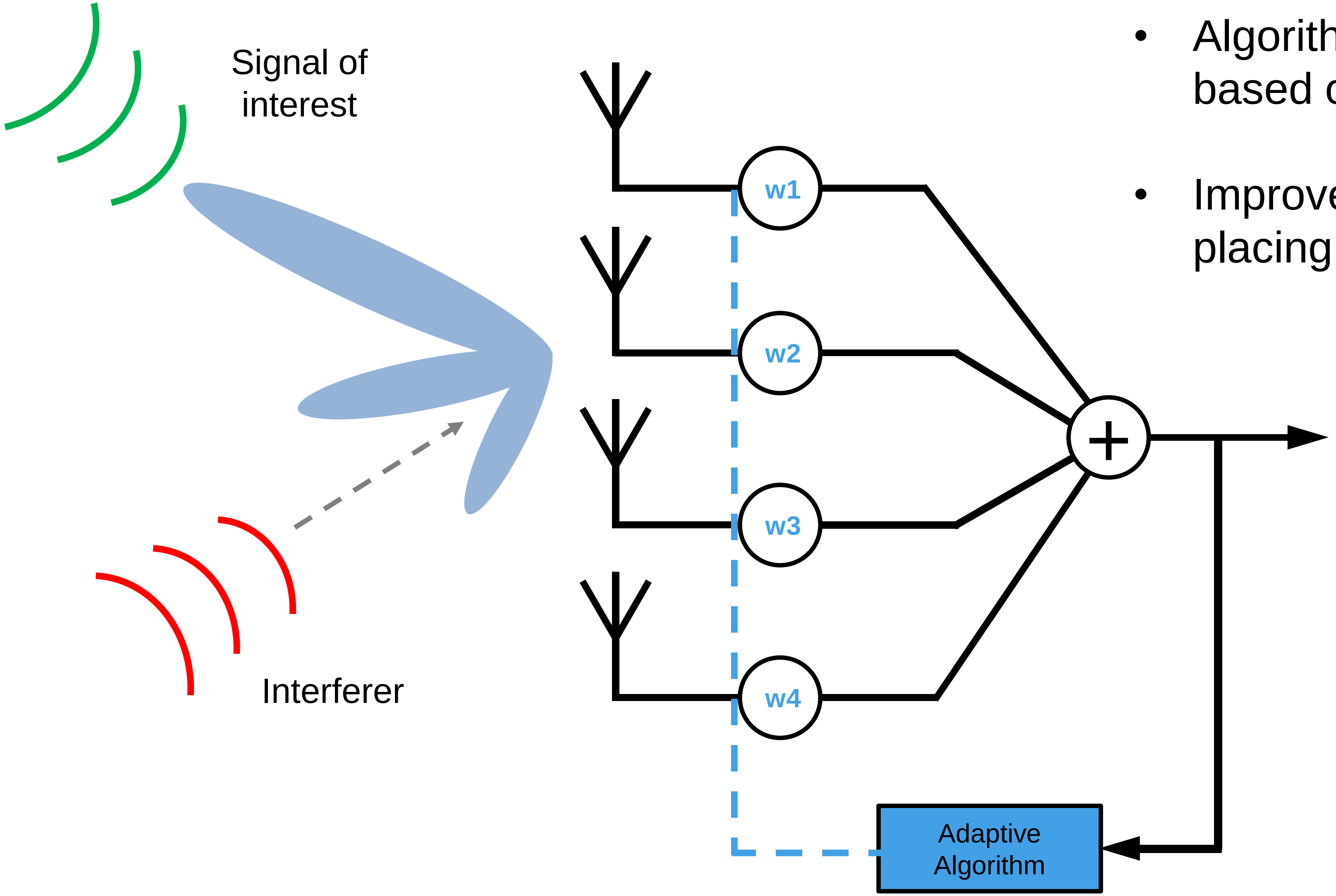
Hardware-Efficient Linear Algebra for Radar and 5G

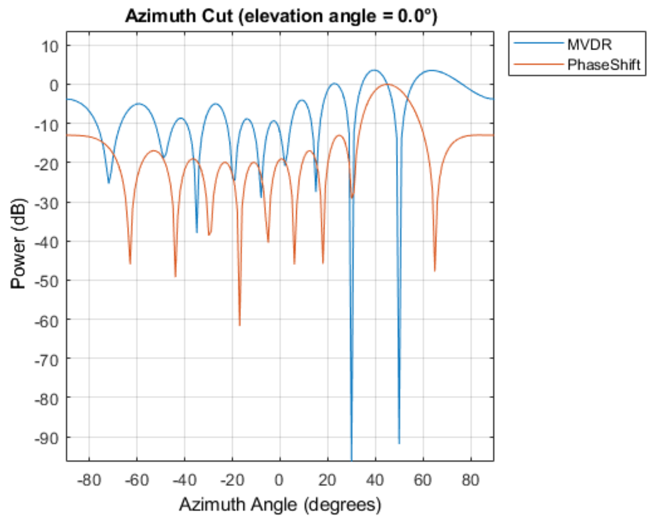
Agenda

- Introduction
  - Applications: Radar, Comms and Wireless
  - Hardware Prototyping – live demo
- Theory and Implementation
  - Linear algebra
  - Matrix decomposition: QR vs Cholesky
  - Latency vs. area tradeoffs
- HDL Coder Implementation
  - HDL Coder implementation
  - Resource mapping and utilization

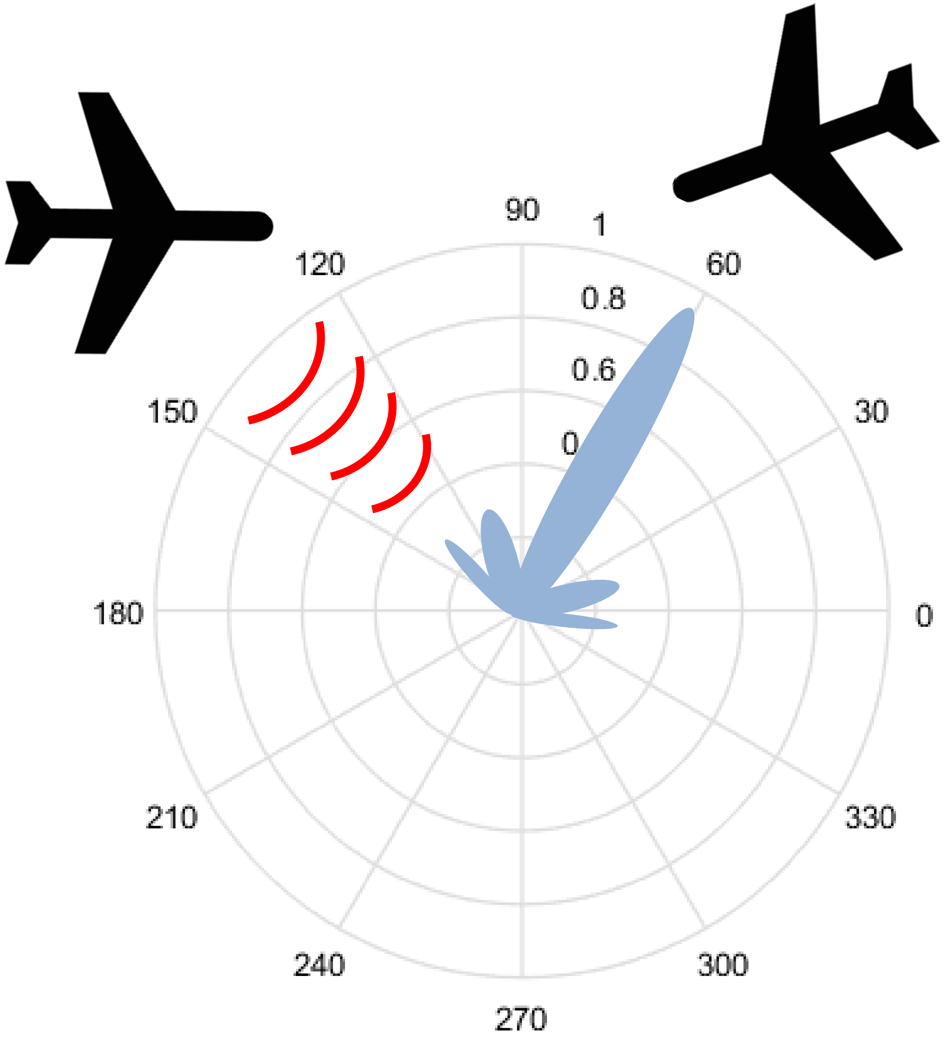
Adaptive Beamforming



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



Applications: Radar



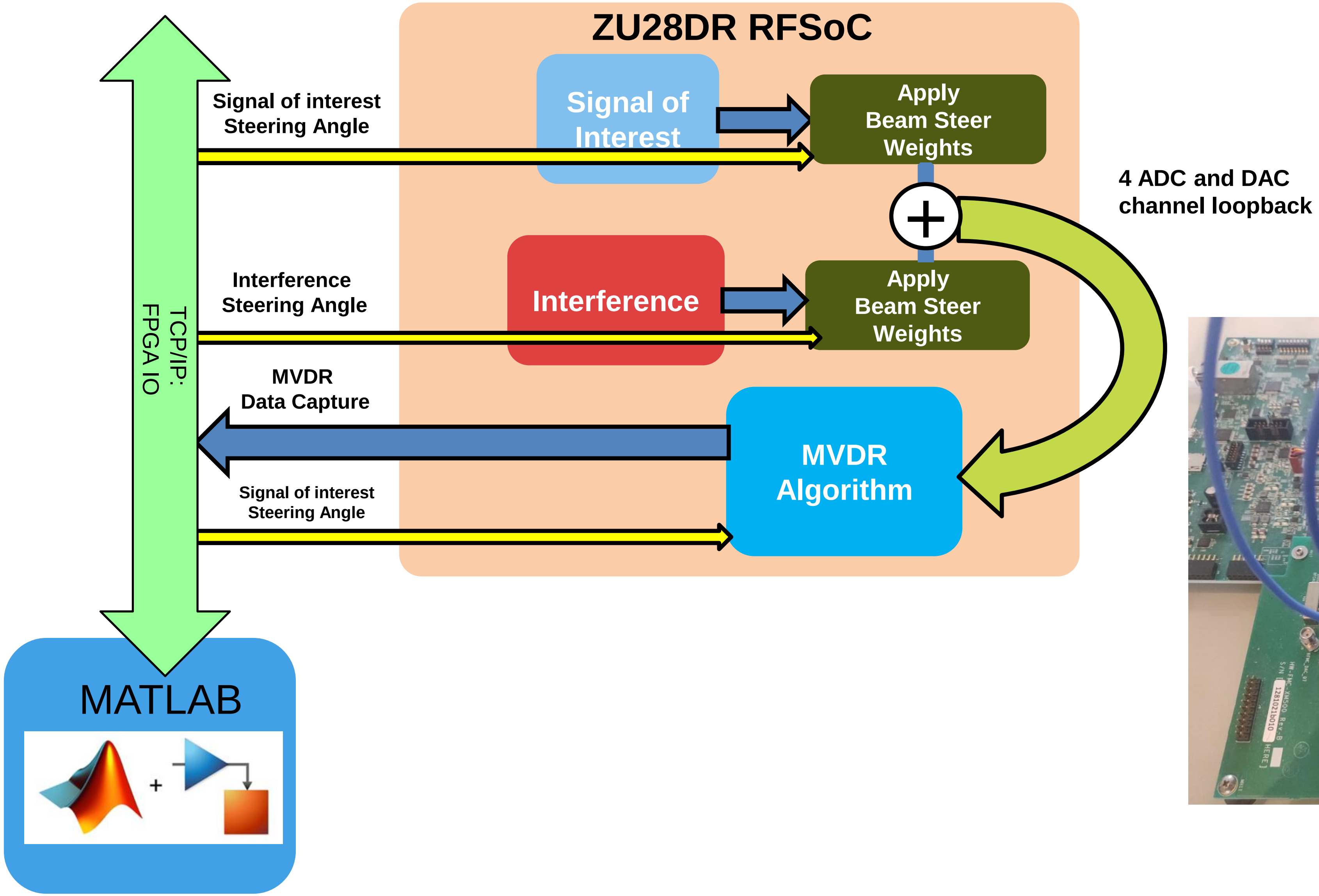
- Increase angular resolution
- Suppress interference

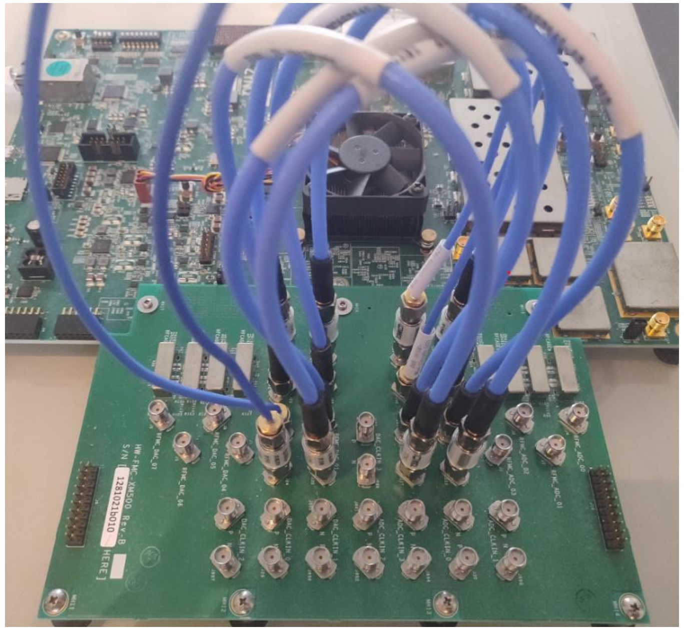
Applications: 5G

- Increase number of simultaneous users
- Improve throughput and coverage

Beamforming Demonstration

Test Setup

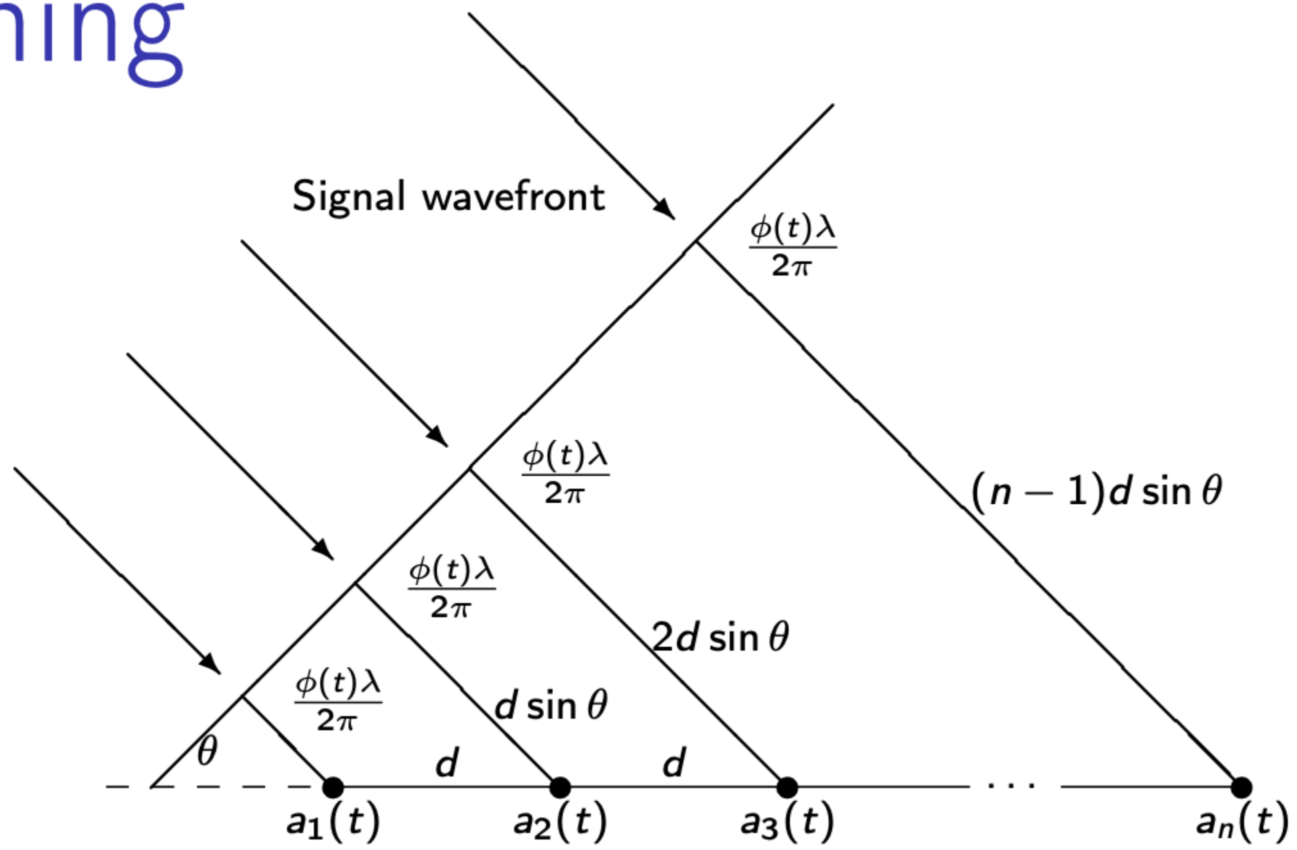




Agenda

- Introduction
  - Applications: Radar, Comms and Wireless
  - Hardware Prototyping – live demo
- **Theory and Implementation**
  - Linear algebra
  - Matrix decomposition: QR vs Cholesky
  - Latency vs. area tradeoffs
- HDL Coder Implementation
  - HDL Coder implementation
  - Resource mapping and utilization

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 the m-by-n data matrix
- $m \gg n$
- $A^H A$ is the n-by-n estimate of the covariance matrix
- $b = \begin{bmatrix} 1 \\ e^{(2\pi d/\lambda)\sin(\theta) i} \\ e^{2(2\pi d/\lambda)\sin(\theta) i} \\ \vdots \\ e^{(n-1)(2\pi d/\lambda)\sin(\theta) i} \end{bmatrix}$ is the steering vector

Minimum Variance Distortionless Response (MVDR) Beamformer

Covariance matrix solve

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

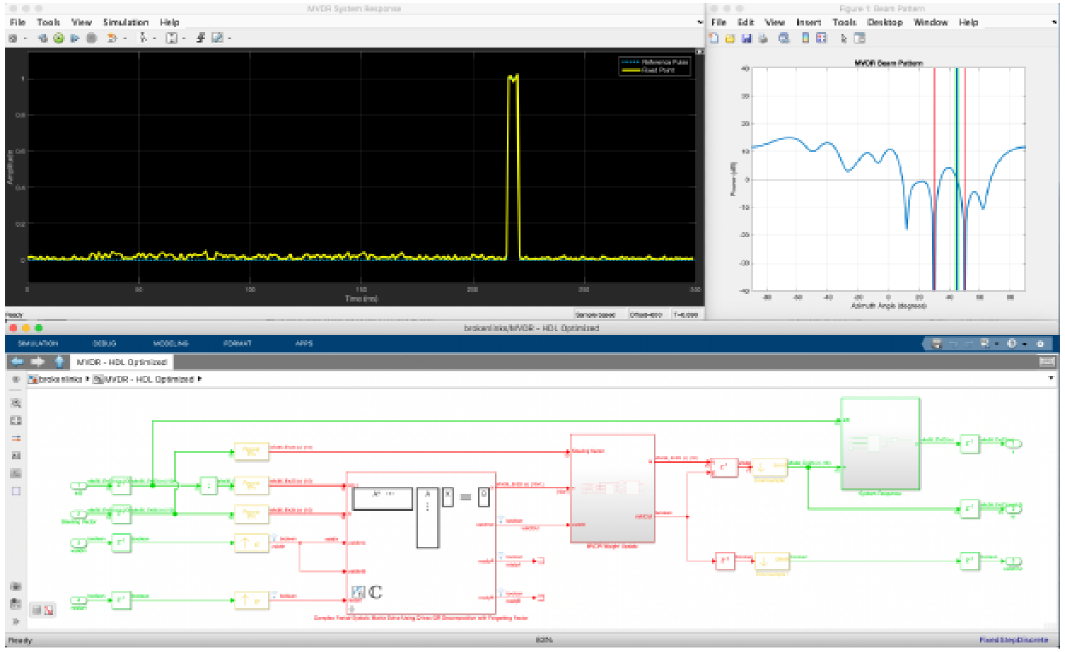
MVDR weight vector

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

MVDR response

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

Example



MVDR Beamformer in MATLAB

Covariance matrix solve	MATLAB
$(A^H A)x = b$	`x = (A'*A)\b;`

MVDR weight vector

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

```
w = x/(b'*x);
```

MVDR response

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

```
y = w'*a
```

Resist the urge to use inverse

Covariance matrix solve

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

MATLAB

```
x = (A'*A)\b;
```

~~`x = inv(A'*A)*b;`~~

Inverses have problems with roundoff errors and high dynamic range.

- Roundoff errors. $3x = 21 \rightarrow x = 0.3333 \cdot 21 = 6.9993$
- Big numbers get small and may underflow. $10000x = 20000 \rightarrow x = 0.0001 \cdot 2000$
- Small numbers get big and may overflow. $0.0001x = 0.0002 \rightarrow x = 10000 \cdot 0.0002$

Cholesky factorization

Covariance matrix solve

$(A^H A)x = b$

MATLAB

```
R = chol(A'*A)
```

Forward and backward substitute

```
x = R\(R'\b)
```

Cholesky requires symmetric positive definite input, which $A^H A$ is.

`R = chol(A'*A)` $\Rightarrow$ `R` is upper triangular and `R'* R = A'*A`

Using QR decomposition without computing Q

Covariance matrix solve

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

MATLAB

```
[~,R] = QR(A,0)
```

Forward and backward substitute

```
x = R\(R'\b)
```

QR vs. Cholesky

Computing $A^H A$ squares the condition number of A,
but may be better for faster updates in some hardware applications.

`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

$$R = \mathrm{chol}(A'A) \to A'A = R'R$$
$$[Q, R] = \mathrm{qr}(A, 0) \to A'A = R'Q'QR = R'R$$

MATLAB Fixed-Point using Q-less QR

MATLAB

```
[~,R] = QR(A,0)
x = R\(R'\b)
```

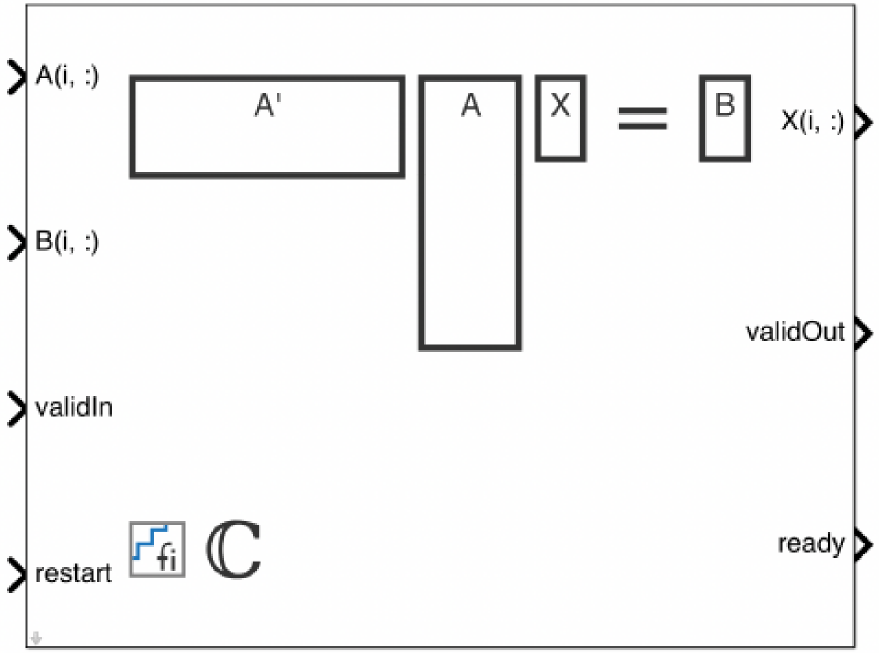
Fixed point

```
x = fixed.qlessQRMatrixSolve(A,b)
```

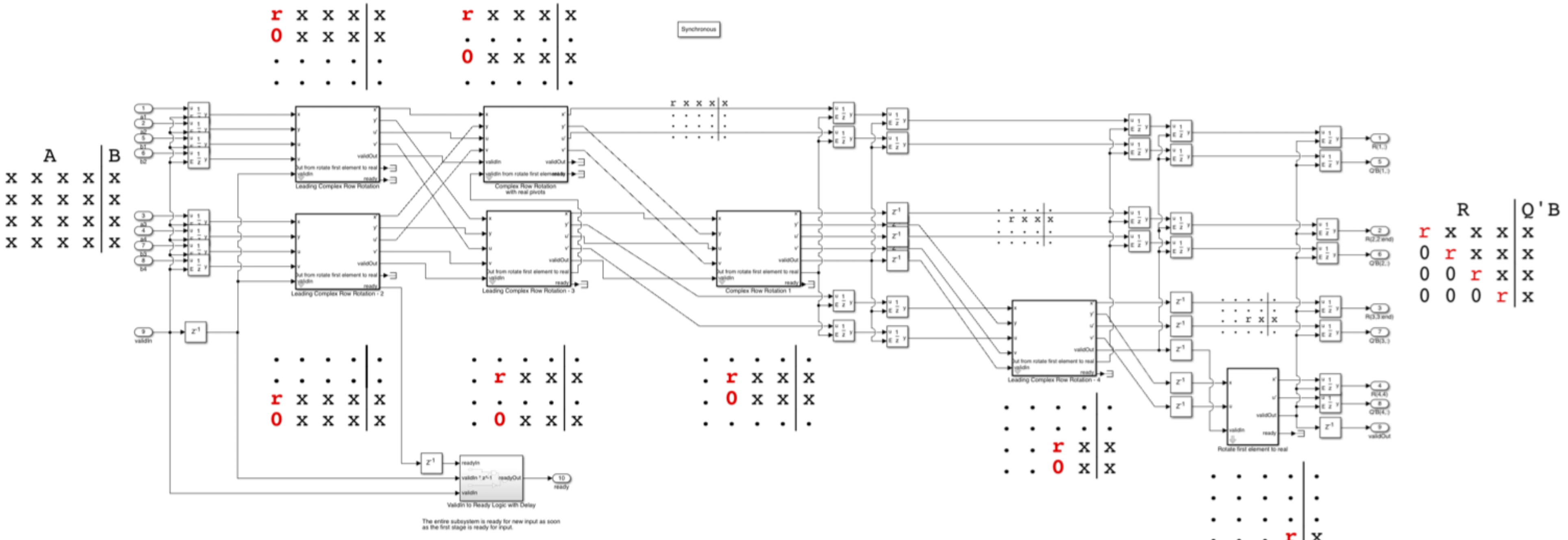
HDL-Optimized Simulink

Matrix Solve Using Q-less QR Decomposition

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



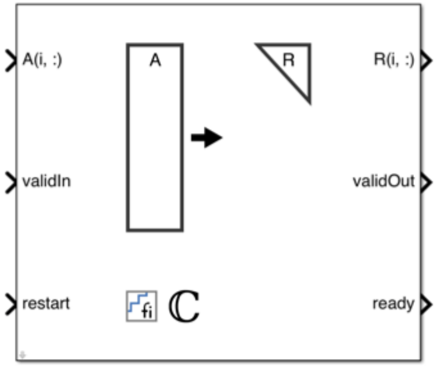
Systolic: One cell for each zero ($\mathcal{O}(mn)$ cells). High area, Low latency.

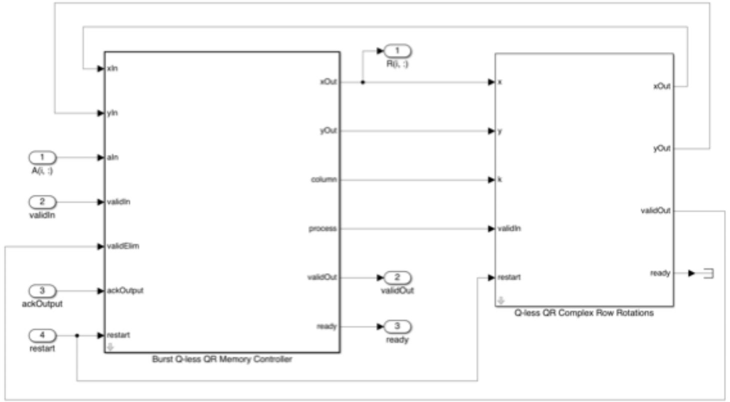


Complex 4x4 CORDIC Q'B, R

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.

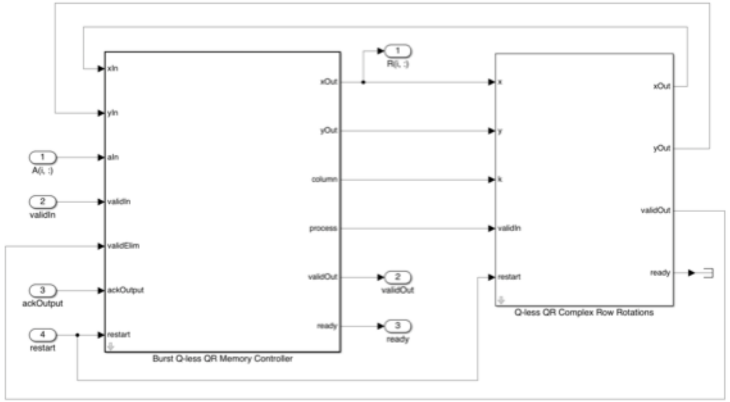




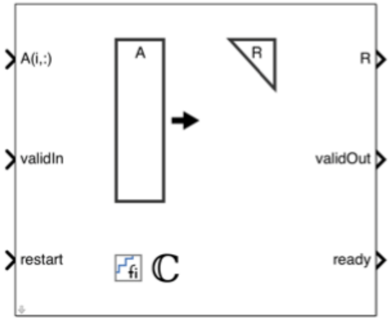
Method	*Input*	*Ready*	*Latency*	*Area*	*Release*
Systolic	Matrix	C	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	R2019a Example
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a Library blocks
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b Library blocks
Partial-Systolic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

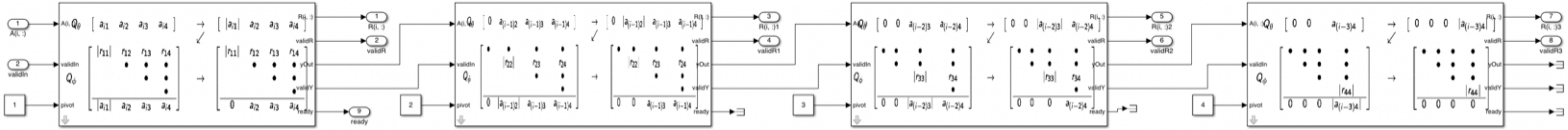
Partial-Systolic: (n cells). Medium area, Medium latency.

Cell internals



Block



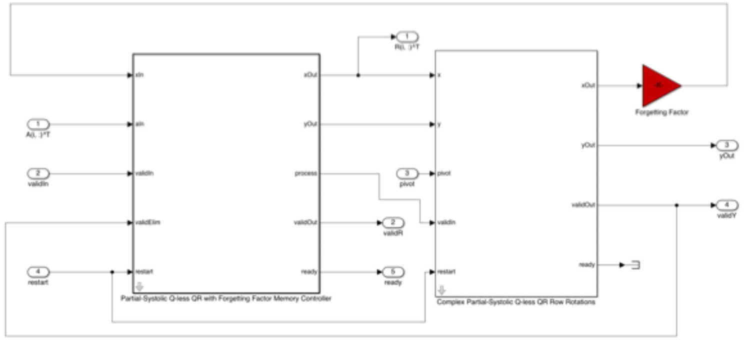


Cell 1. Update R(1,:) Cell 2. Update R(2,:) Cell 3. Update R(3,:) Cell 4. Update R(4,:)

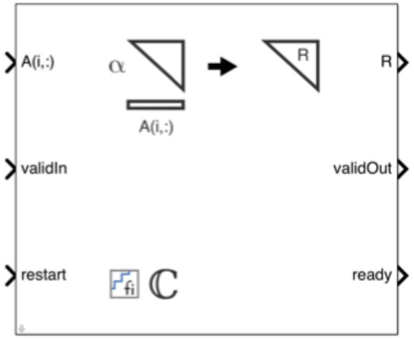
Method	*Input*	*Ready*	*Latency*	*Area*	*Release*
Systolic	Matrix	C	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	R2019a Example
Burst	Row	$\mathcal{O}(n)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(n)$	R2020a Library blocks
Partial-Systolic	Row	C	$\mathcal{O}(m)$	$\mathcal{O}(n^2)$	R2020b Library blocks
Partial-Systolic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

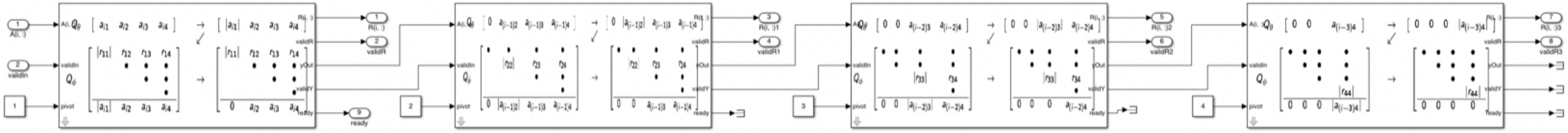
Partial-Systolic with Forgetting Factor (n cells):Continuously update

Cell internals with forgetting factor



Block

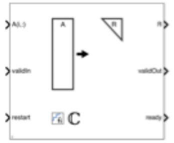




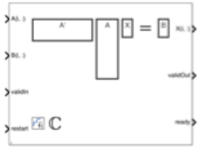
Cell 1. Update R(1,:)   Cell 2. Update R(2,:)   Cell 3. Update R(3,:)   Cell 4. Update R(4,:)

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

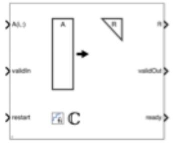
MATLAB functions



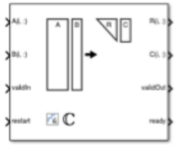
`fixed.qlessQR`



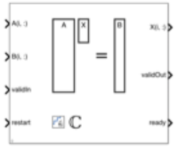
`fixed.qlessQRMatrixSolve`



`fixed.qlessQRUpdate`

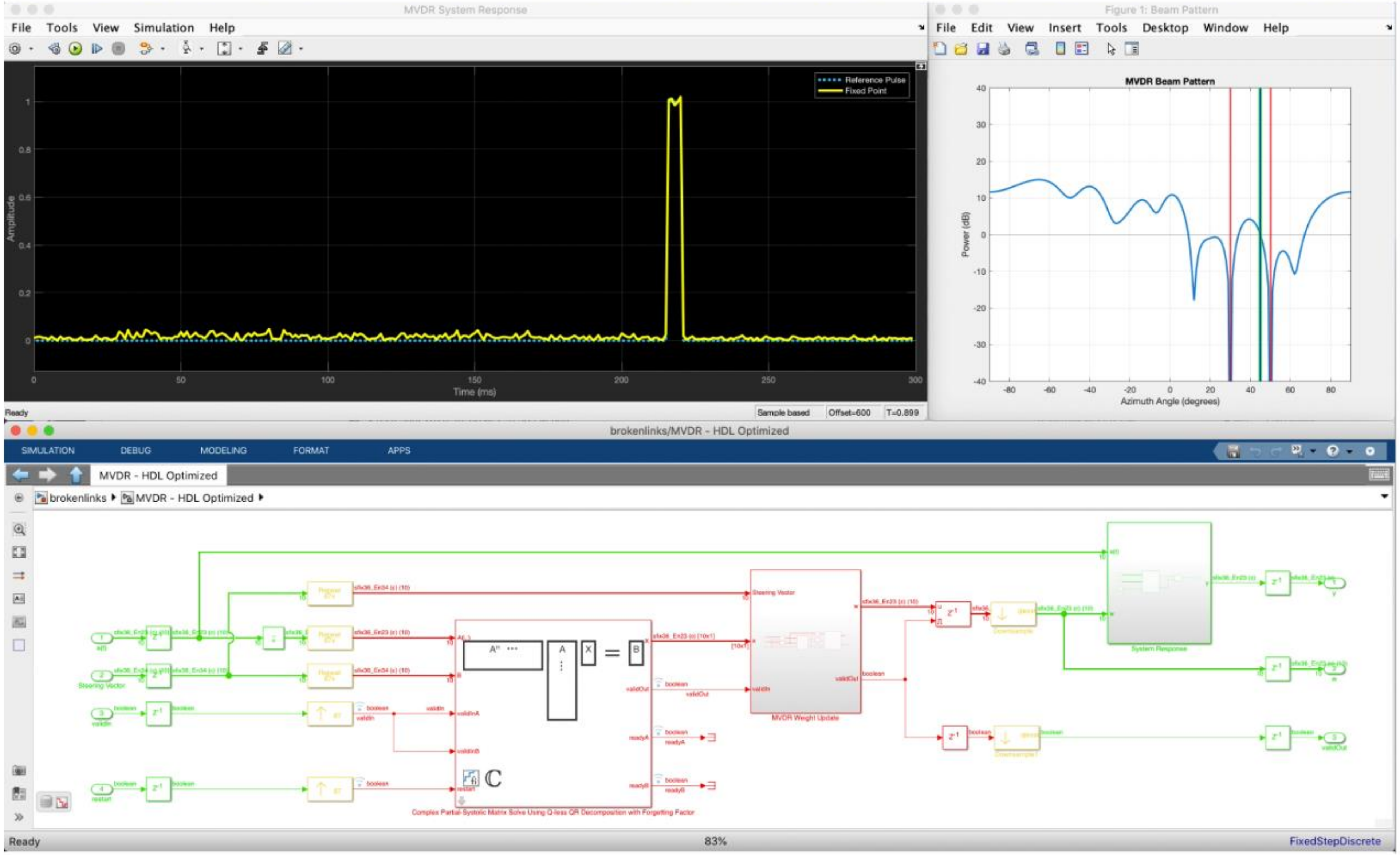


`fixed.qrAB`



`fixed.qrMatrixSolve`

White-box MVDR reference model in doc



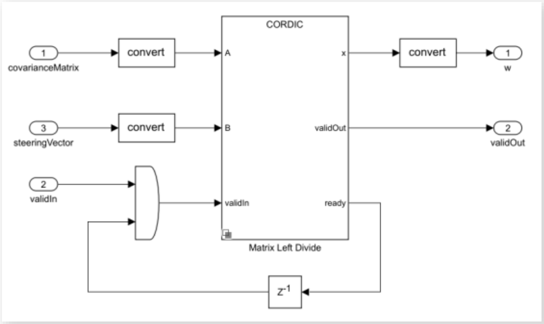
Agenda

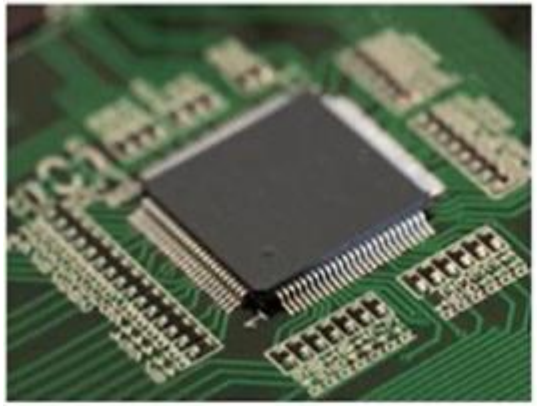
- Introduction
  - Applications: Radar, Comms and Wireless
  - Hardware Prototyping – live demo
- Theory and Implementation
  - Linear algebra
  - Matrix decomposition: QR vs Cholesky
  - Latency vs. area tradeoffs
- **HDL Coder Implementation**
  - HDL Coder implementation
  - Resource mapping and utilization

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
```





HDL Implementation Workflow

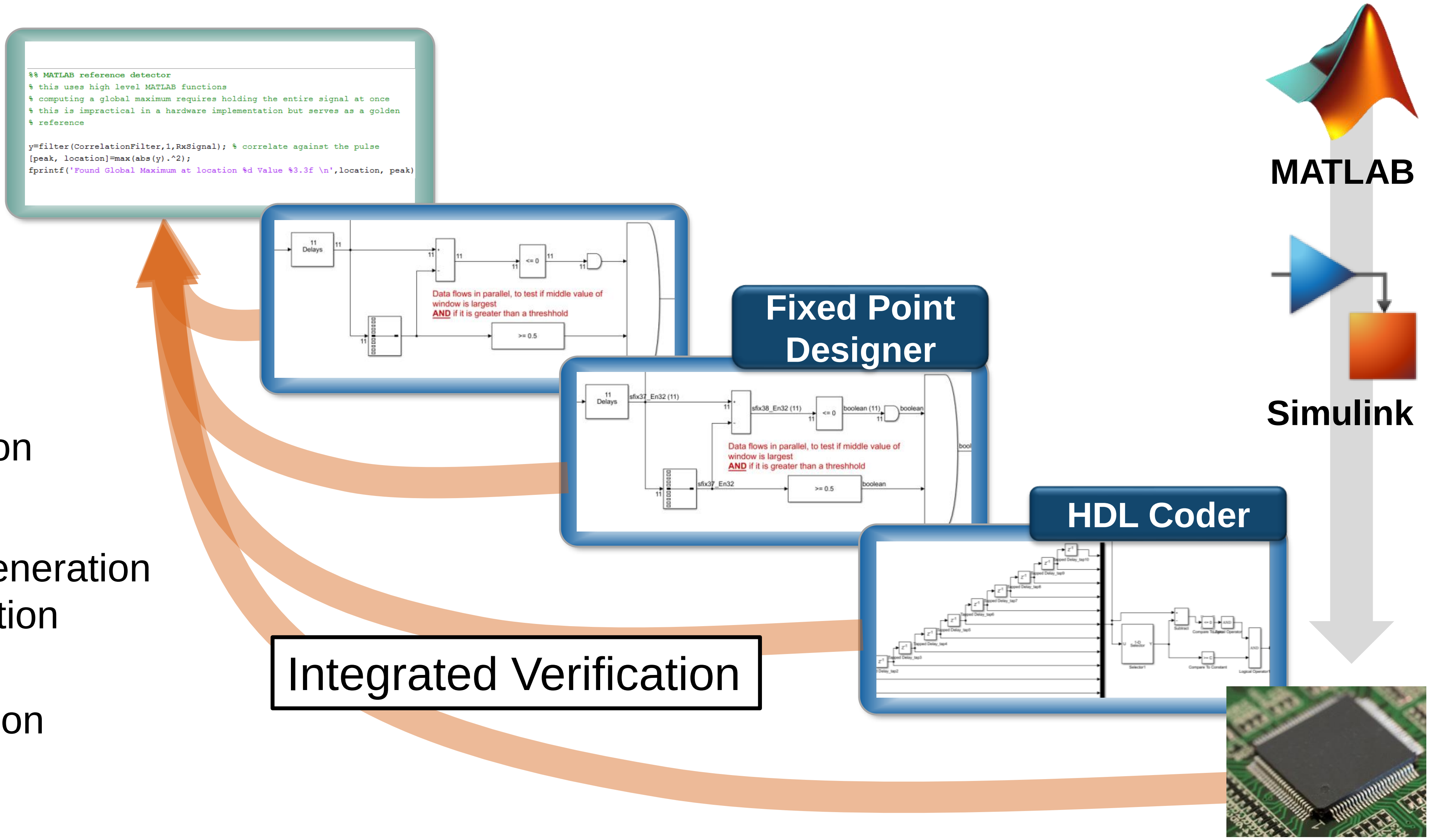
MATLAB Reference

Hardware Architecture

Fixed-point Implementation

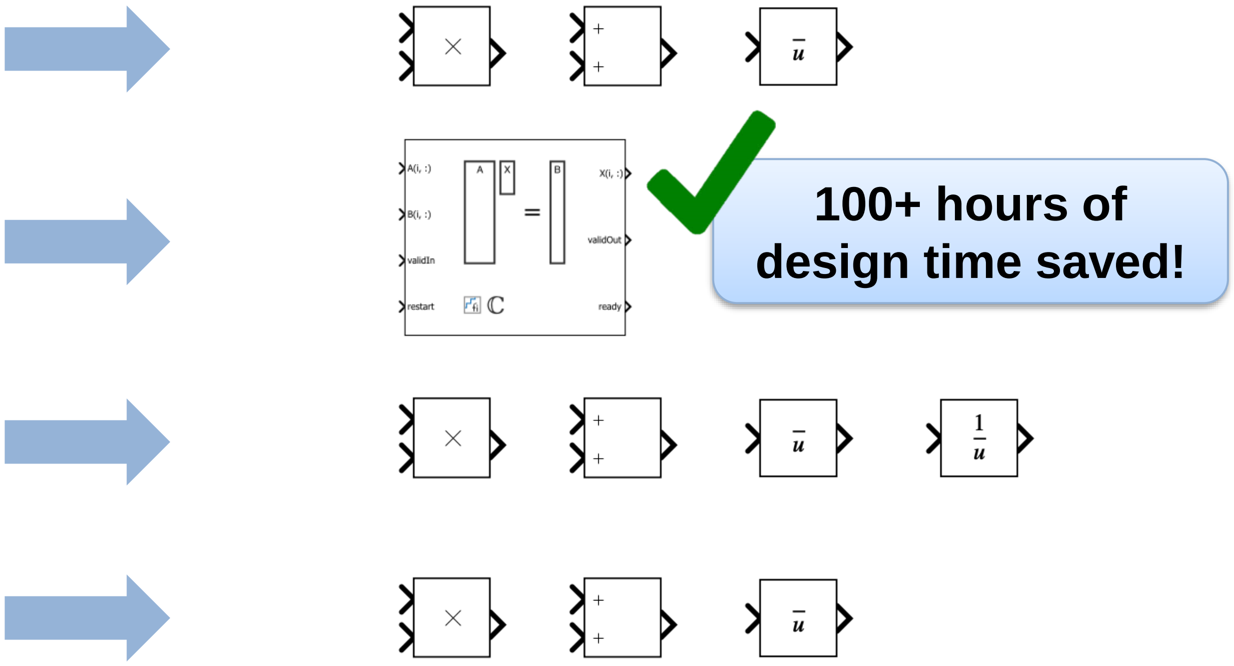
HDL Code Generation and Optimization

HDL Verification and Targeting

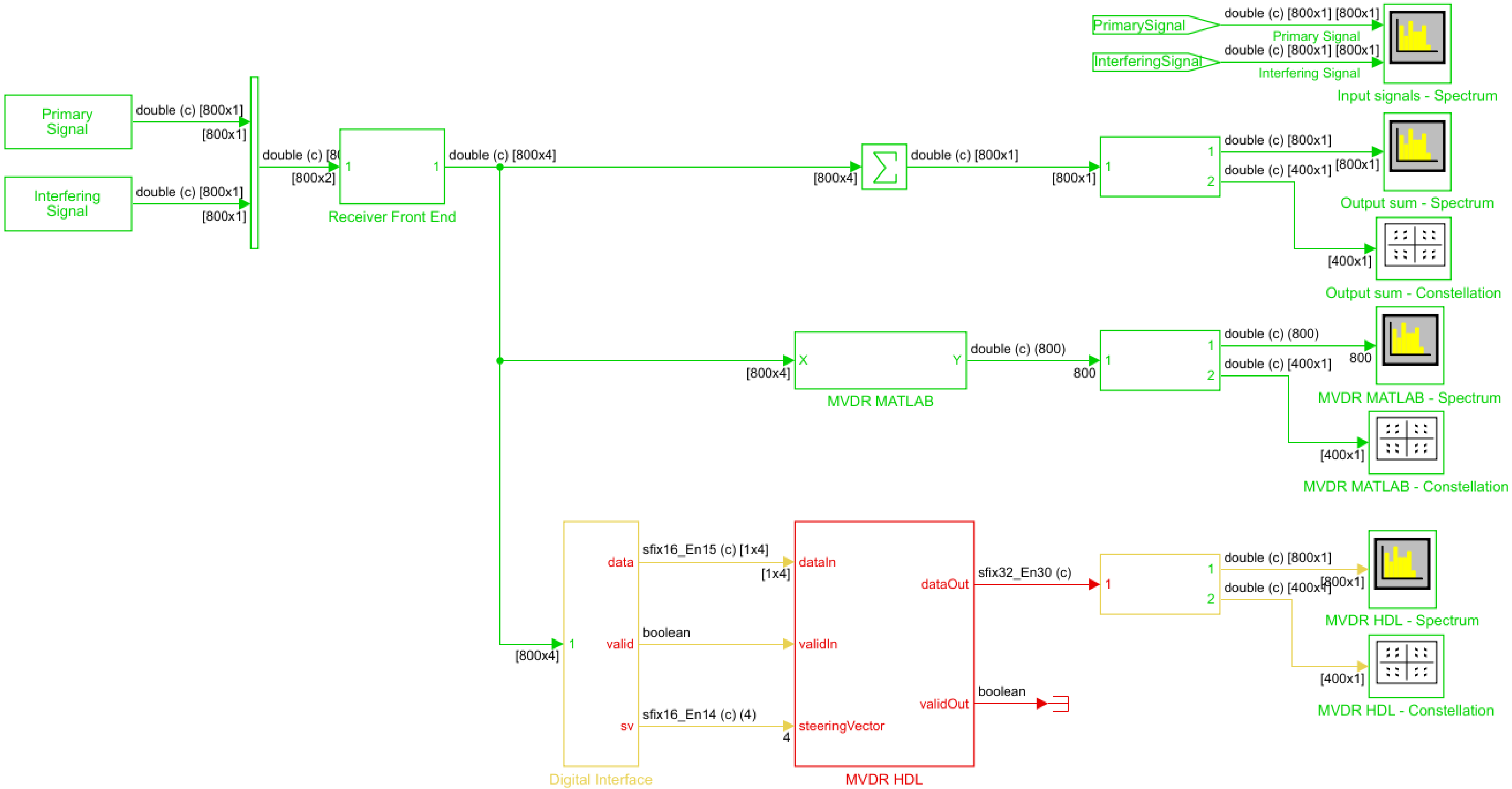


MATLAB MVDR reference code

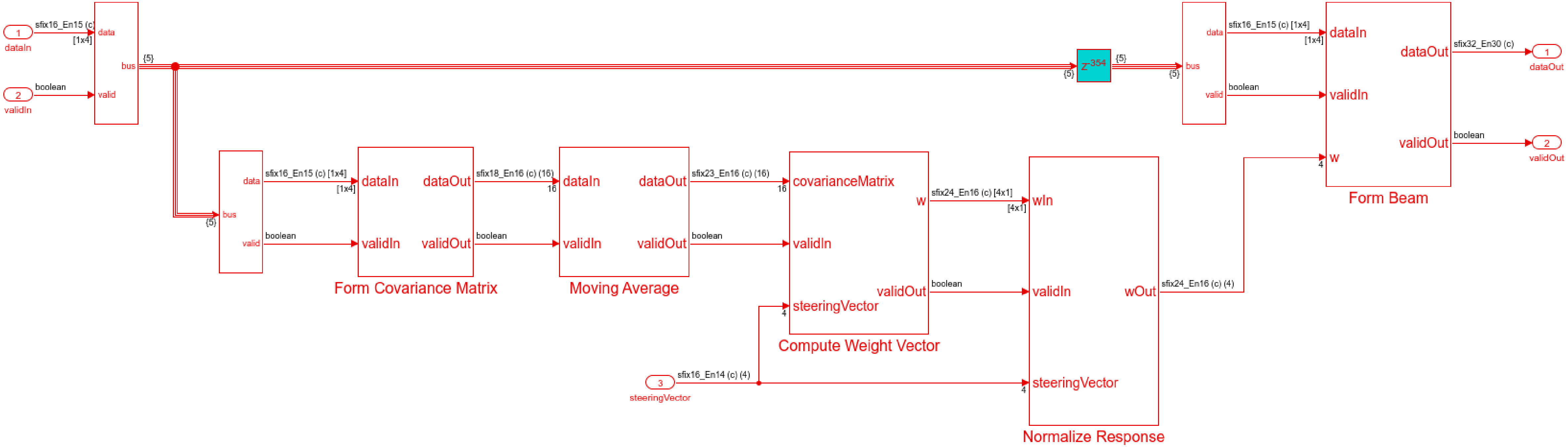
```
function Y = mvdr_beamform(X, sv)

% 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
```



HDL Implementation of MVDR Beamforming



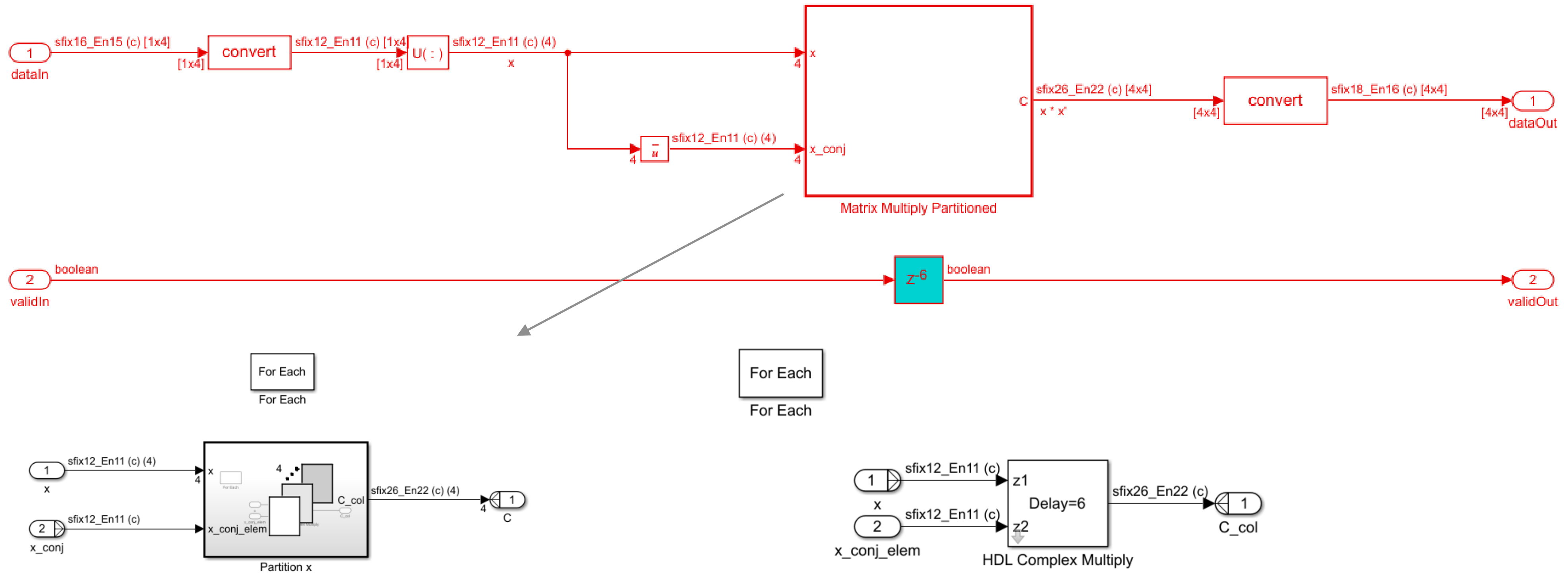
HDL Implementation of MVDR Beamforming



Form Covariance Matrix

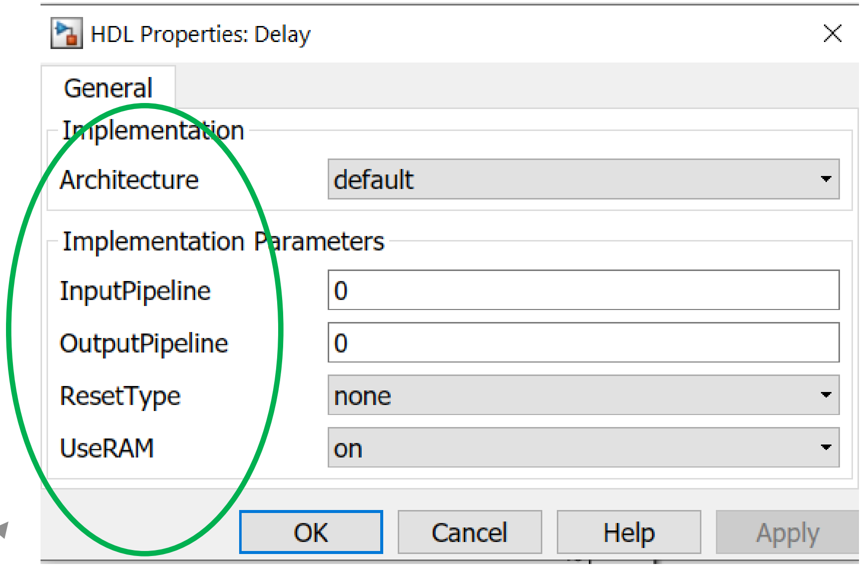
- For Each subsystem
  - Process elements independently
  - Concatenate results into outputs

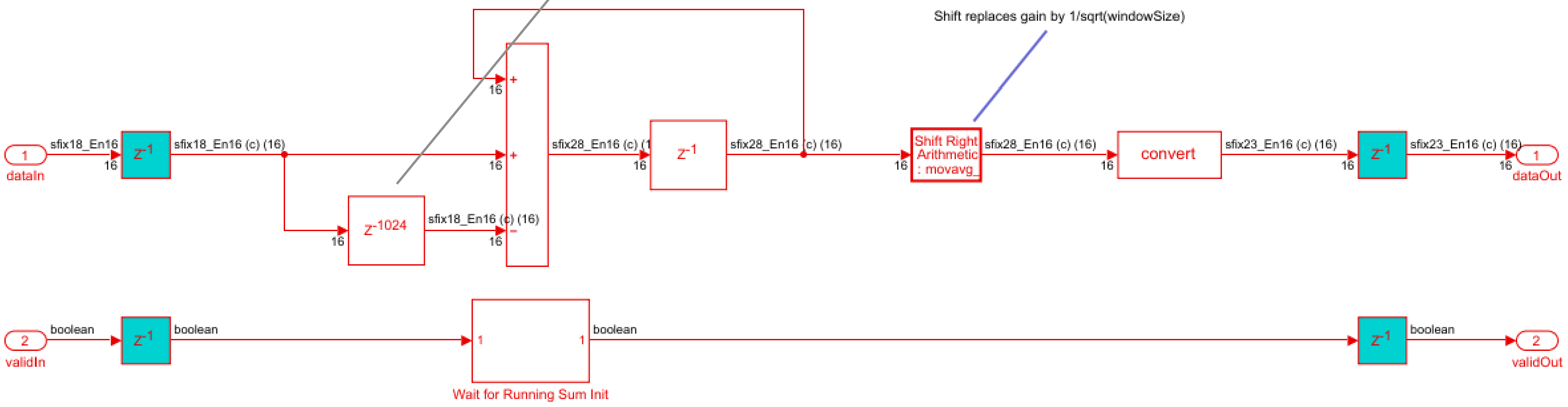
```
% form covariance matrix
Ecx = X.'*conj(X);
```



Moving Average

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

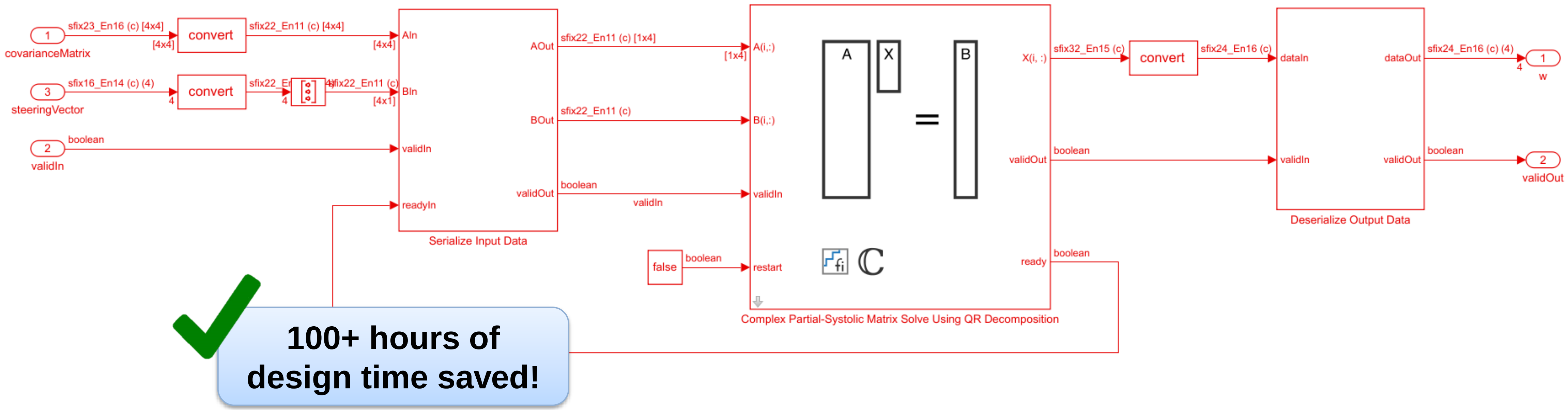




Compute Weight Vector

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

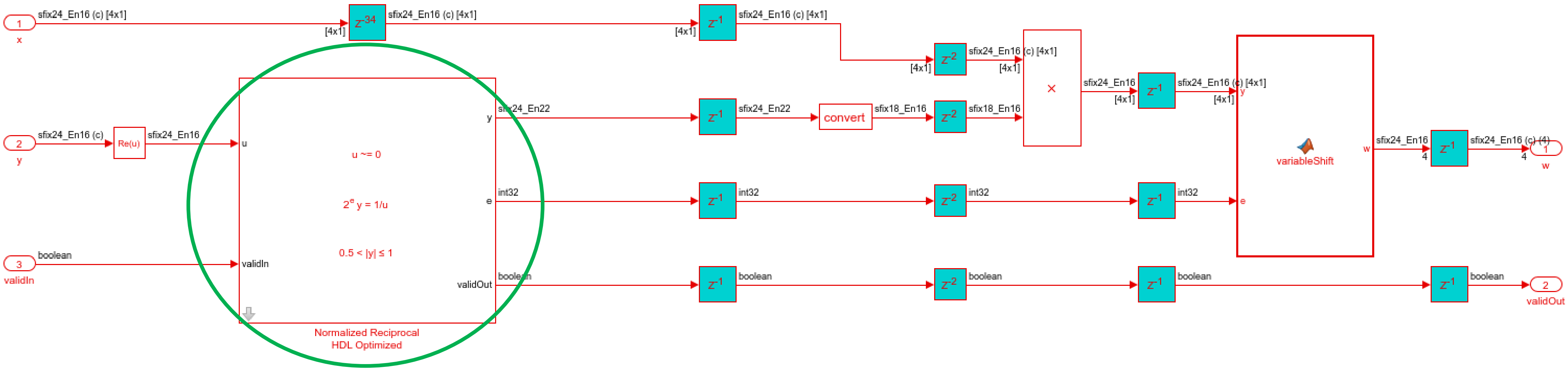
```
% compute weight vector
wp = Ecx\sv;
```



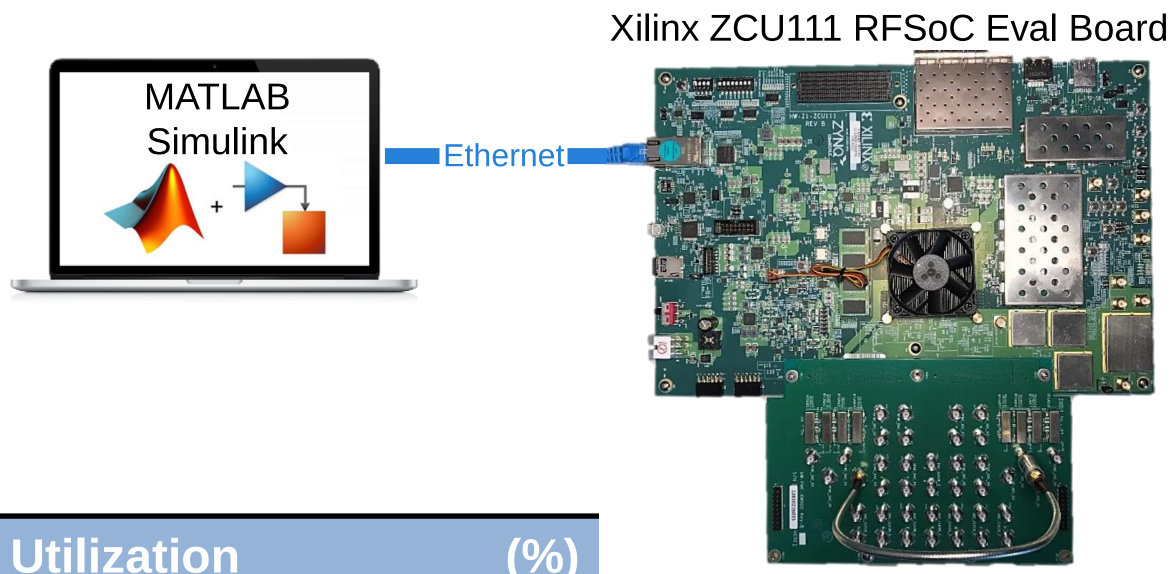
Normalize Response

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

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



Implementation Results

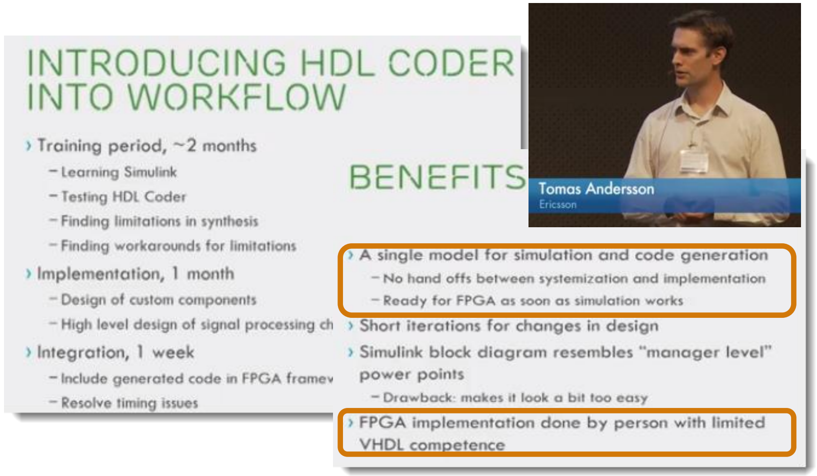


- Device: xczu28dr (ZCU111)
- Maximum frequency: 452 MHz
- Resource utilization:

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

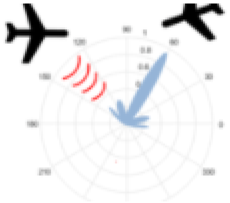
- Getting started:
  - MATLAB Onramp
  - Simulink Onramp
  - HDL pulse detector self-guided tutorial and videos



- 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

Beamforming Demonstration



FPGA-Adaptive-Beamforming-and-Radar-Examples

version 1.0.0.0 (6.87 MB) by Daren Lee STAFF

FPGA/HDL demonstrations for beamforming and radar designs.

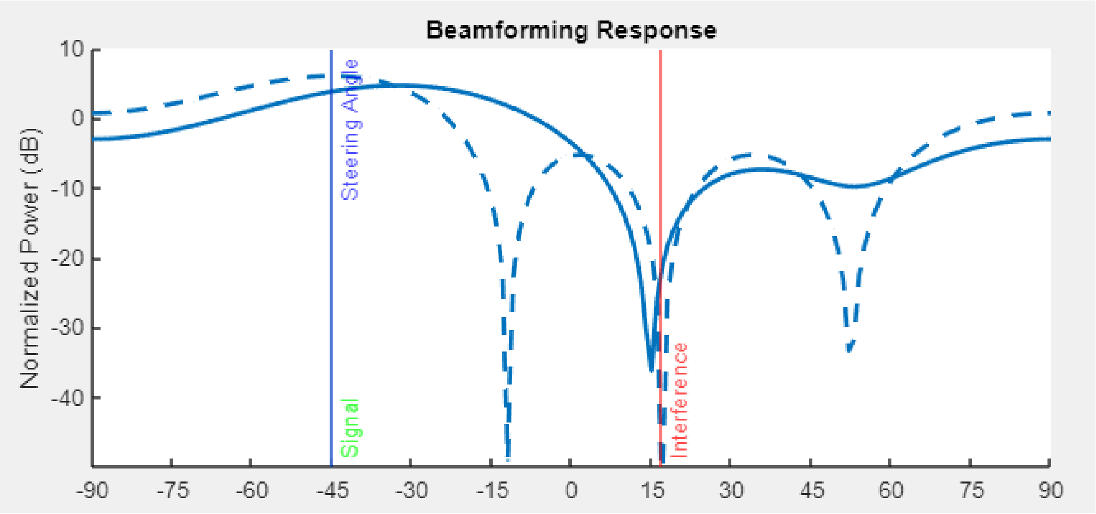
★★★★★ 2 Ratings

30 Downloads

Updated 31 Mar 2021

From GitHub

- ZCU111 RFSoC Adaptive Beamformer demo for 4x4 matrix solve for 4 channel ADC/DAC
- Places nulls in interference locations and maximizes beam pattern for steering direction
- Interactively steer angles for interference and beam pattern at run-time



Download from File Exchange