The MathWorks

MATLAB® & SIMULINK®

MATLAB for signal processing

Houman Zarrinkoub, PhD.
Product Manager
Signal Processing Toolboxes

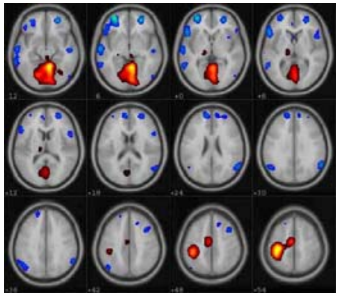
houmanz@mathworks.com

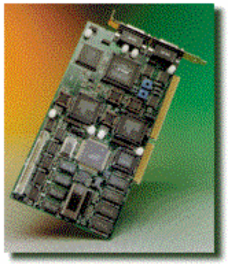
Outline

- Introduction
- Filter Design, simulation and implementation
- Adaptive and Multirate filters
- Spectral analysis of signals
- Fixed-point representation of signals and filters
- Path to C and HDL implementation
- Algorithm verification & validation
- Summary
- Q & A

Ubiquitous signal processing across industries

- Aerospace and Defense
- Automotive
- Communications
- Electronics and Semiconductor
- Computers and Office Equipment
- Education

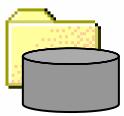




MATLAB as the platform for Signal Processing & Technical Computing

Analysis and Modeling
Visualization
Algorithm Development
Prototyping & Simulation
Application Deployment
Verification & Validation

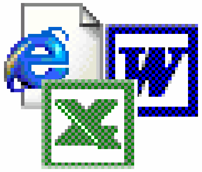
Data



Software

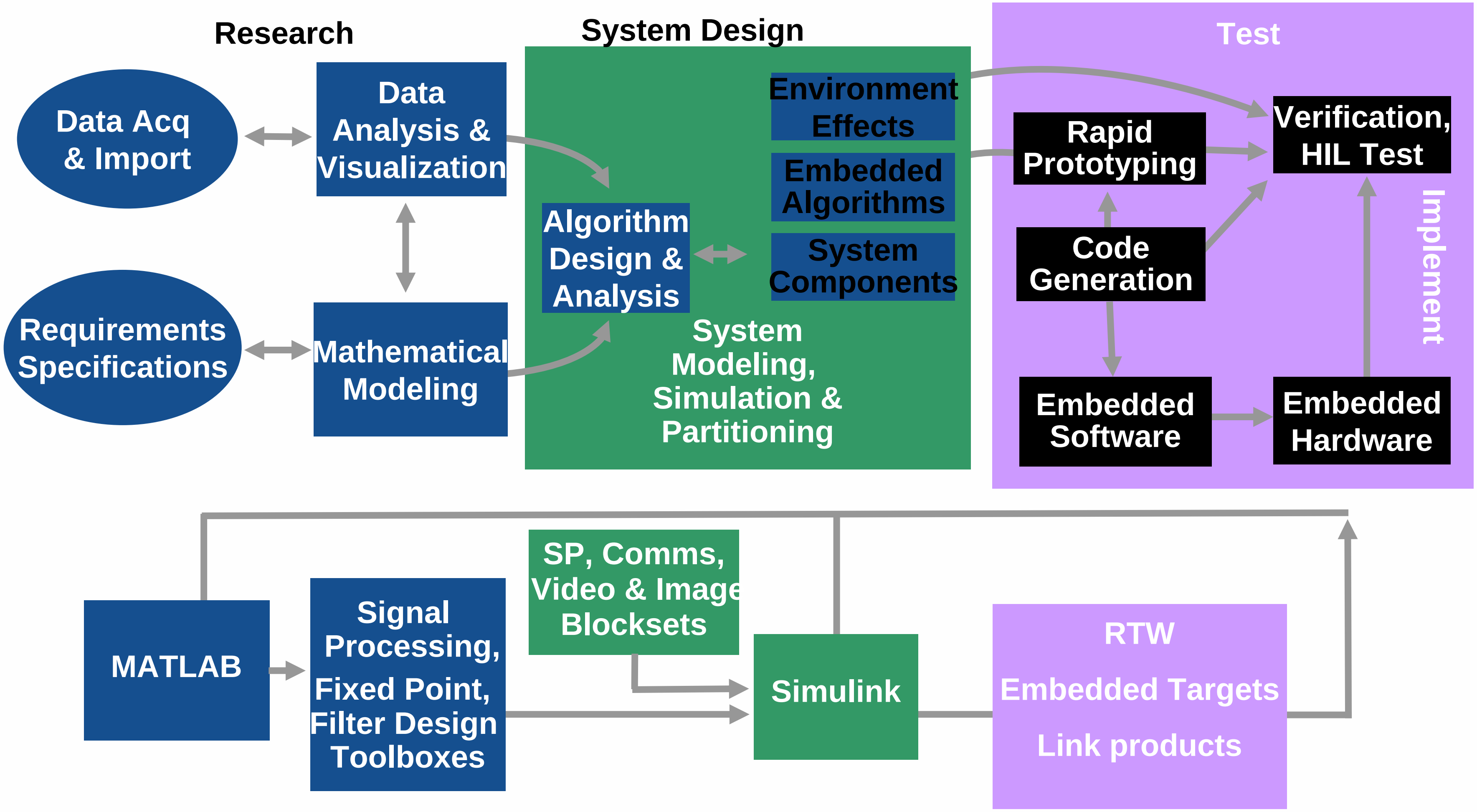
Hardware

Reporting and Documentation



Application Deployment

MATLAB for algorithm development
Simulink for System & Product development



MATLAB Tools for Signal Processing

- Analysis of signals and design of filters
  - Signal Processing toolbox
  - Filter Design toolbox
- Fixed-Point representation of signals
  - Fixed-Point toolbox
- Related products
  - Wavelet, Statistics, Image Processing toolboxes
- System-level design
  - Simulink and Signal Processing Blockset
- Path to HDL implementation
  - Filter Design HDL Coder
- Hardware and software verification
  - Link products (CCS and ModelSim)

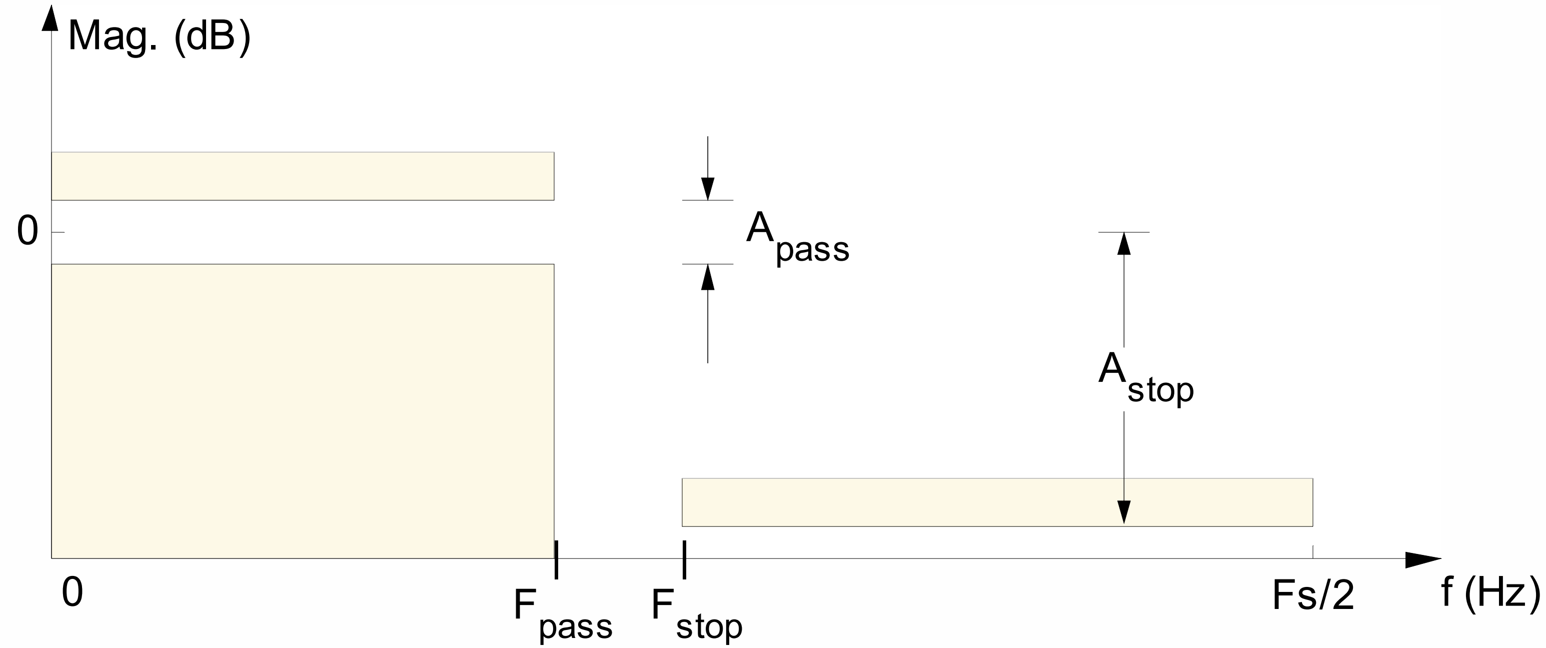
Filter design, simulation & implementation

- Signal Processing & Filter Design toolboxes
- Single-rate filters
  - Lowpass, highpass, bandpass, etc.
  - Designed based on spectral specifications
  - Employed across many applications (i.e., modeling linear time-invariant systems)
- Adaptive filters
  - Modeling linear time-varying systems
  - Learn and adapt to changes of the desired signal
  - Important applications in noise and echo cancellation
- Multirate filters
  - Different sampling frequency for input and output
  - Used extensively in wireless receivers & digital audio systems

Example workflow: lowpass filter design

- Classical function-based approach
  - Command-line or GUI-based (fdatool)
- New object-based approach
  - Design: advantages of fdesign objects
  - Implementation: advantages of filter objects
    - Dfilt (single-rate digital filter)
    - Mfilt (multirate filter)
    - Adaptfilt (adaptive filter)

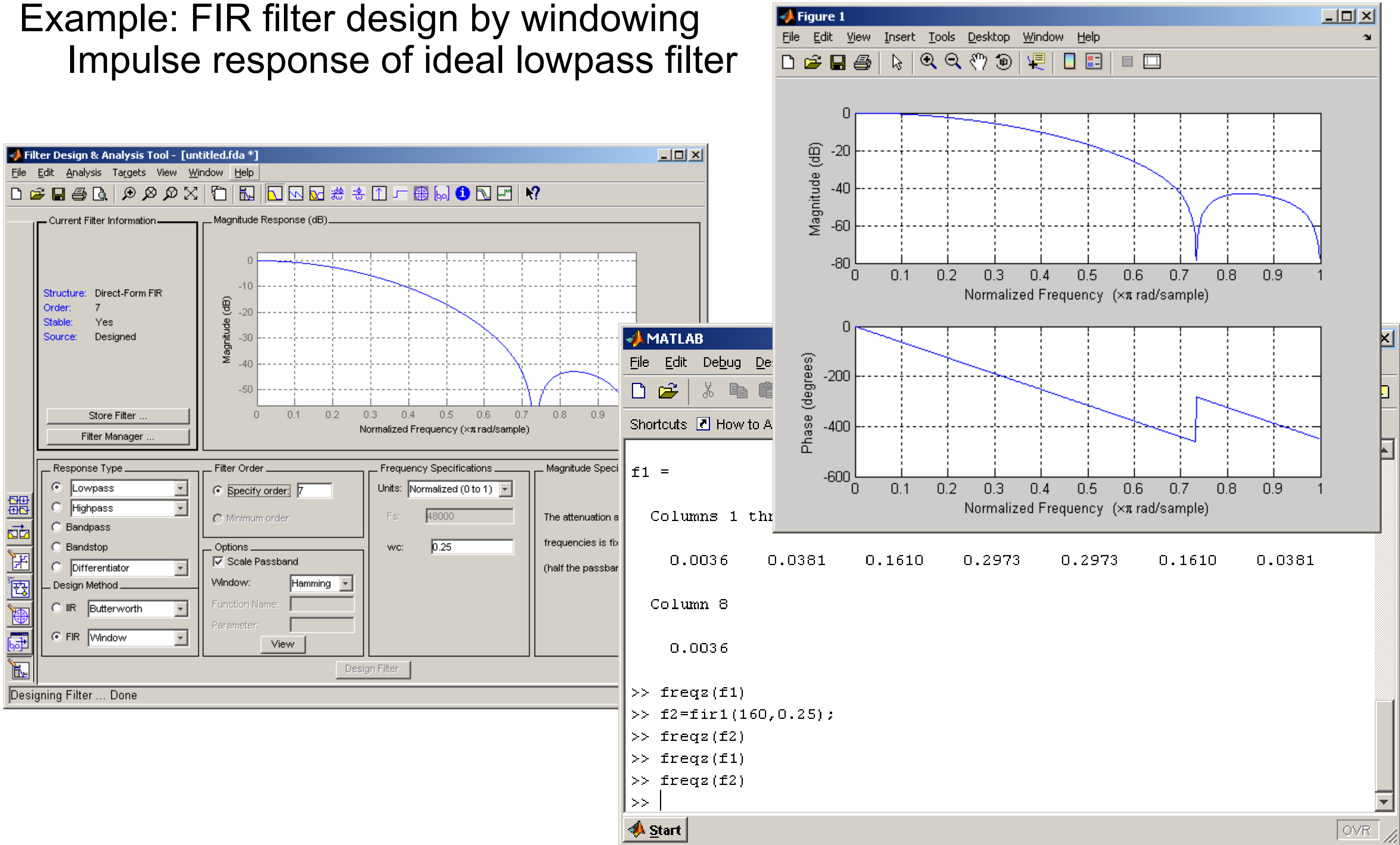
Typical Lowpass Design Specifications



Classical function-based filter design

Example: FIR filter design by windowing

Impulse response of ideal lowpass filter



An alternative to function-based design

- Process of function-based design is sub-optimal
  1. Choose a design method first
  2. Guess its parameters and then design
  3. Look at filter response to see if meets requirements
  4. Iterate by trial-and-error until requirements satisfied

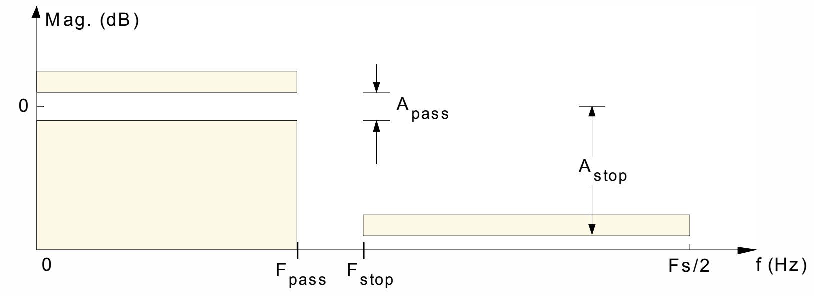
- Not efficient for assessing design trade-offs

- Fdesign: A more optimal design methodology
  1. First, set the design requirement
  2. Find out what design methods can meet them
  3. Then iterate through design methods and find the best

Filter design based on fdesign object

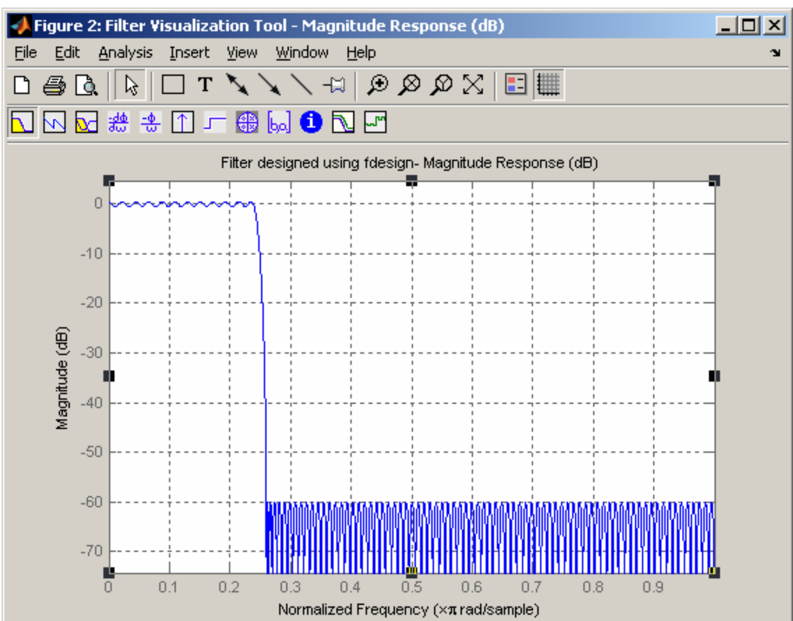
```
>> fd=fdesign.lowpass;
>> fd.Fpass=0.24; fd.Fstop=0.26

fd =

            Response: 'Lowpass'
       Specification: 'Fp,Fst,Ap,Ast'
         Description: {4x1 cell}
 NormalizedFrequency: true
               Fpass: 0.24
               Fstop: 0.26
               Apass: 1
               Astop: 60

>> designmethods(fd)


Design Methods for class fdesign.lowpass (Fp,Fst,Ap,Ast):


butter
cheby1
cheby2
ellip
equiripple
ifir
kaiserwin
multistage

>> df3=design(fd,'equiripple')
```

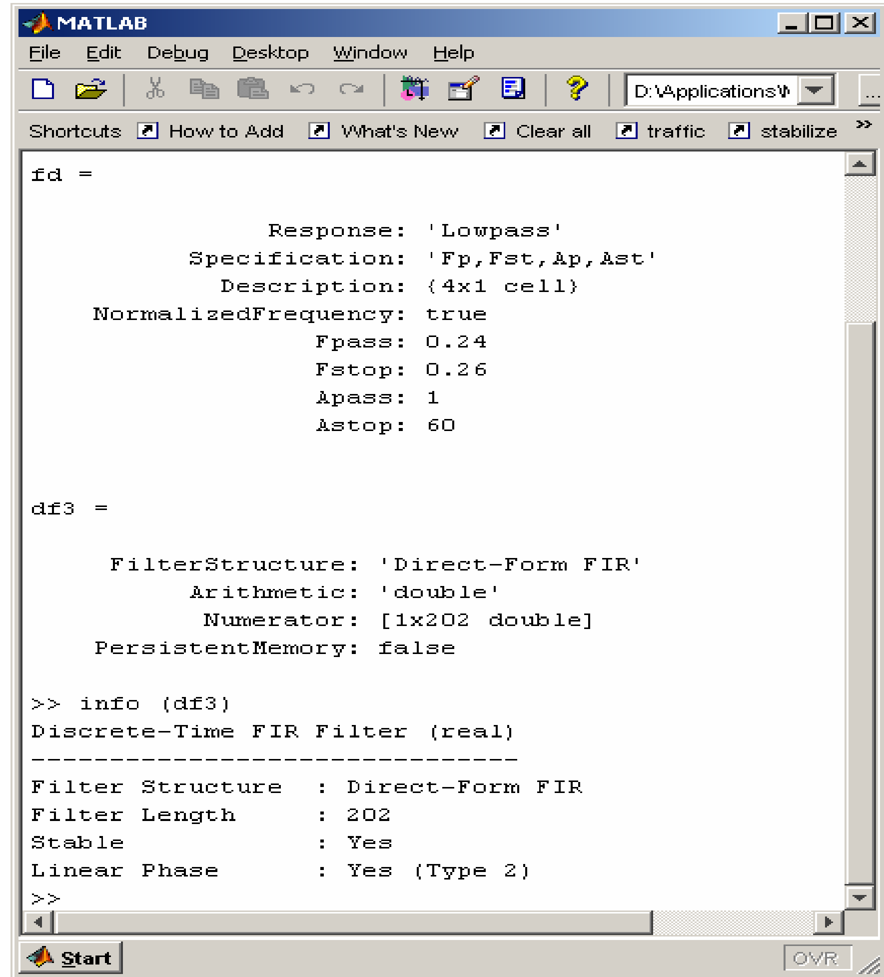
- Tradeoff analysis between Stopband attenuation and Filter order
- Filter order relates to algorithmic delay and computational complexity of filter



Capturing design as a filter object

- Designed filter represented as
  - Coefficients as MATLAB vectors
  - Captured as dfilt object
- Filter objects facilitate task of analyzing the design

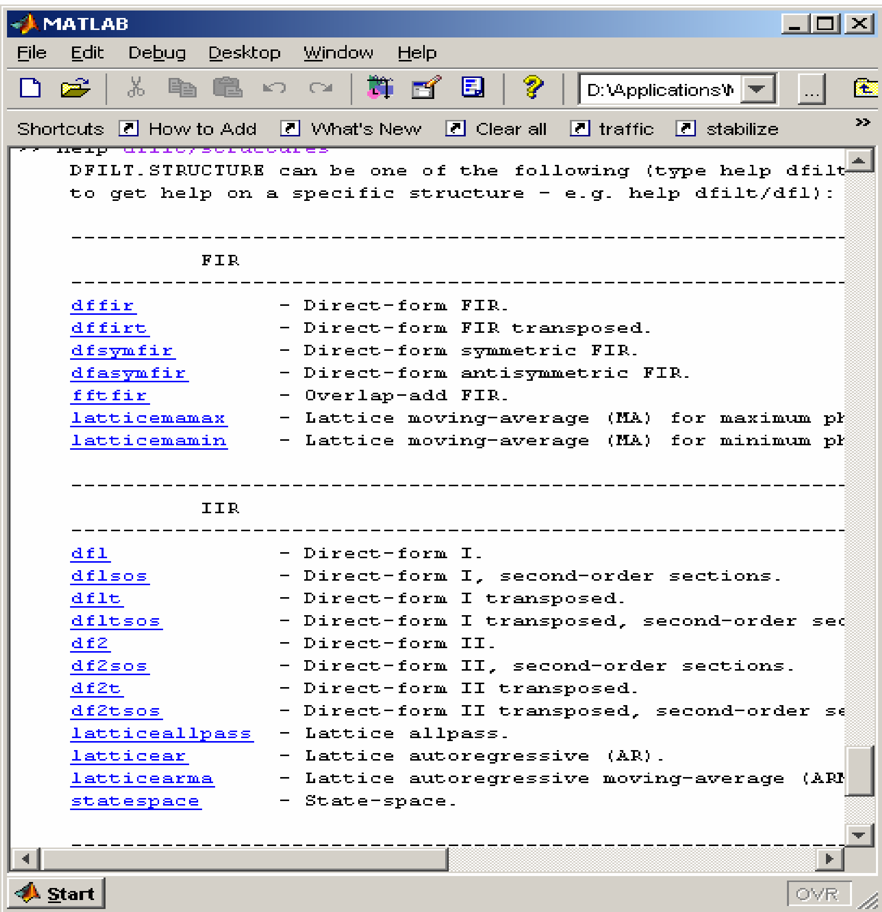




Advantages of using filter objects

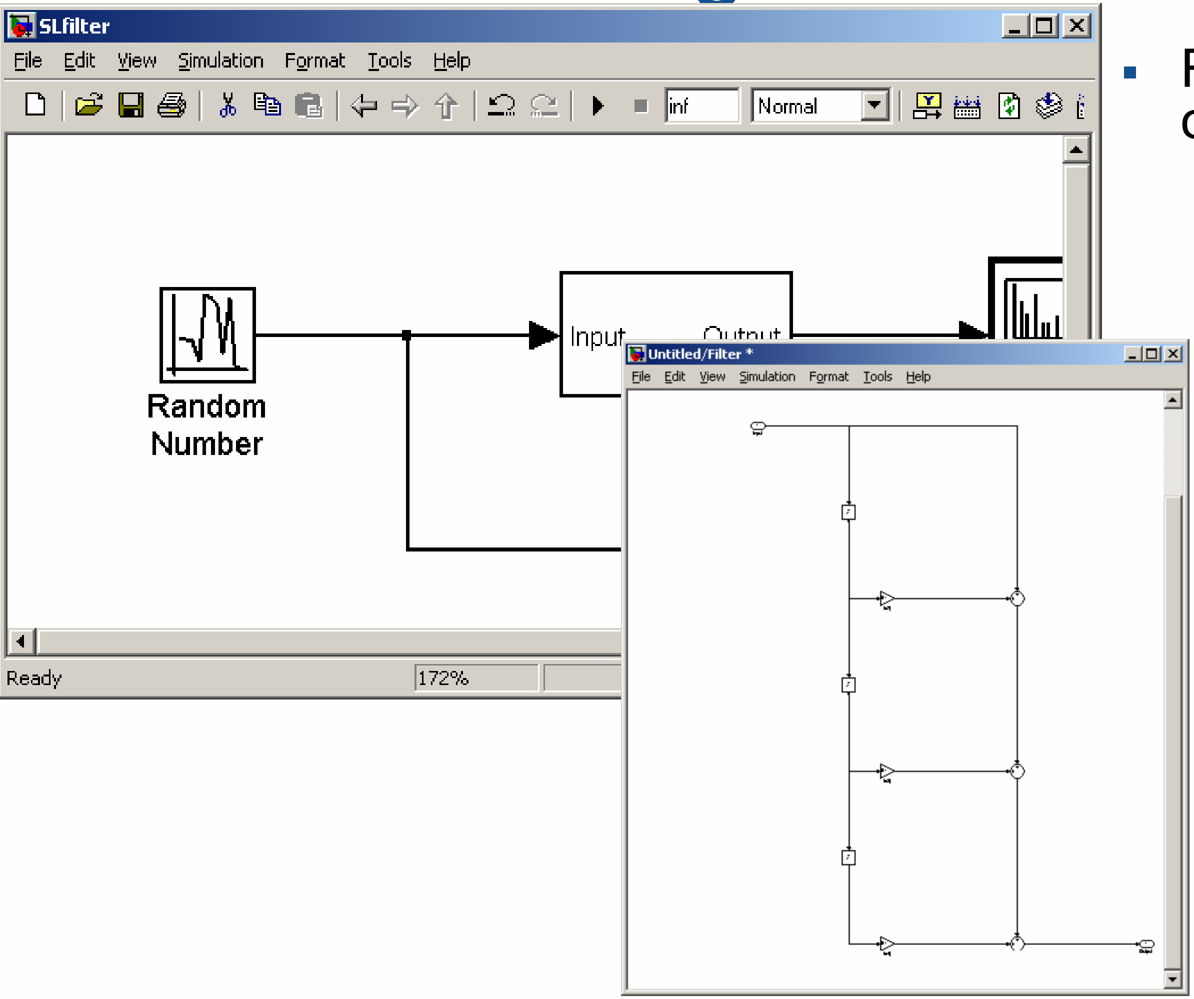
- Consolidated visualization and analysis (fvtool)
- Trade-off analysis for filtering via various structures
  - Overloaded filter function
  - List of supported filter structures
- Path to simulation and automatic code generation
  - Simulink model
  - Generate HDL code
- Automatic estimation of computational complexity
  - Examining the Simulink model
  - Use of cost function

Simulation and implementation in MATLAB



- Advantage of using dfilt objects
  - Filtering with overloaded filter function
  - Choose among various filter structures
  - Direct control over states of filter

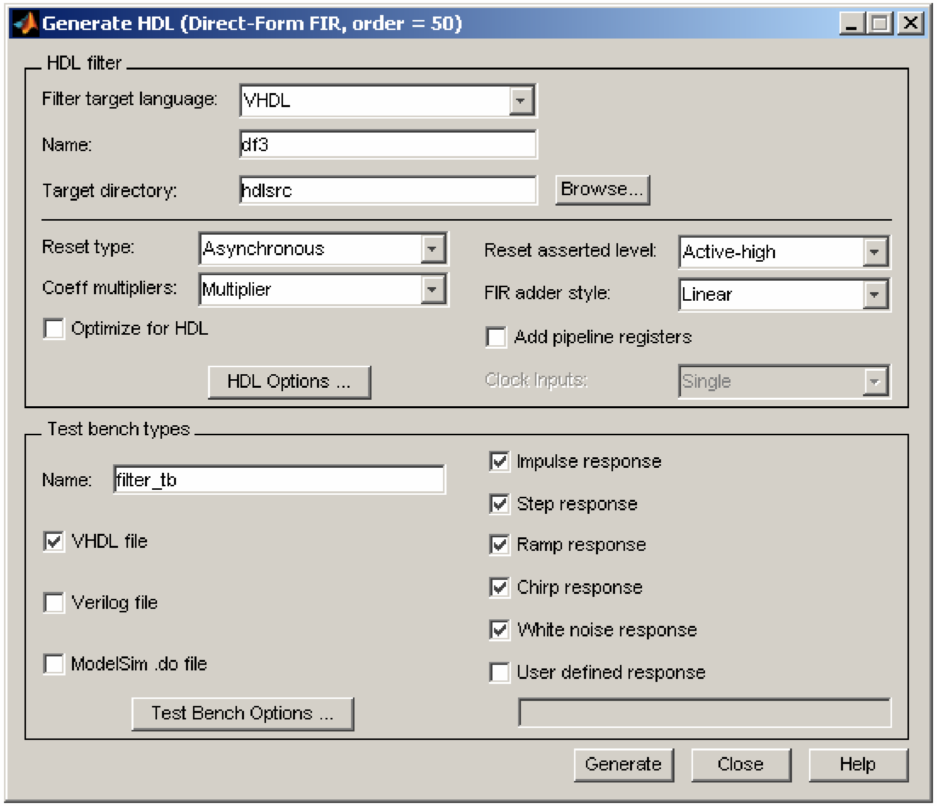
Path to system-level simulation with Simulink & Signal Processing Blockset



- Realizemdl method of filter objects
  - Generates a Simulink model representing the designed filter
  - Implemented with delay, sum and gain blocks
  - Reflects the structure of the filter
  - Helps visualize the computational complexity

Automatic HDL code generation from filter objects

- Functionality of Filter Design HDL Coder
- Supports both VHDL and Verilog code
- Command-line with generatehdl method
- GUI-based as a target in fdatool



Estimation of filter computational complexity

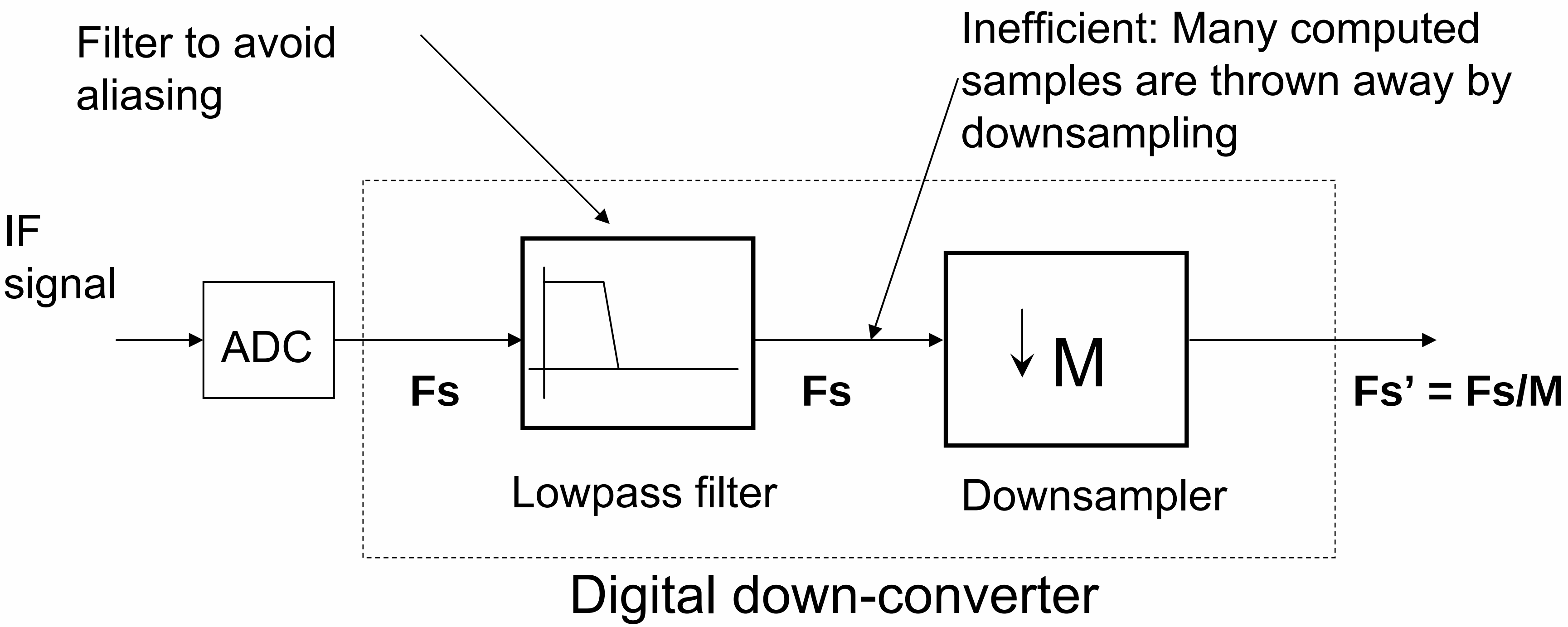
- Examine realized Simulink model to estimate number of additions & multiplications per sample
- Together with sampling frequency estimate Number of Operations per second
- Use the Cost method of filter objects
- Important tool in studying design tradeoffs in terms of quality and complexity

Direct-Form FIR filter	
Sampling Frequency (MHz)	100
Filter order	202
Number of Multipliers	642
Number of Adders	641
Number of States	630
Multiplications per input sample	42.8
Additions per input sample	42.7
Operations per second (MOPS)	**8550**

Multirate filters

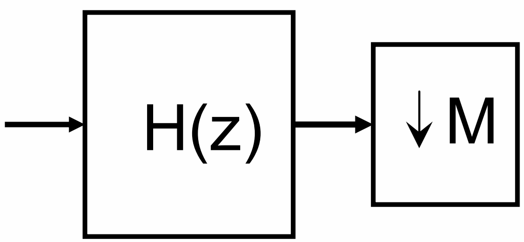
- An important class of filters
- Widespread use in high data-rate signal processing
- Major applications:
  - Wireless receivers
  - Digital audio systems
- Design challenge
  - Meet spectral specification
    - Minimize aliasing effect
  - Minimize the computational cost
    - Use efficient filter structures to avoid wasting processing power

Example: decimator of a receiver Lowpass filter + downsampler

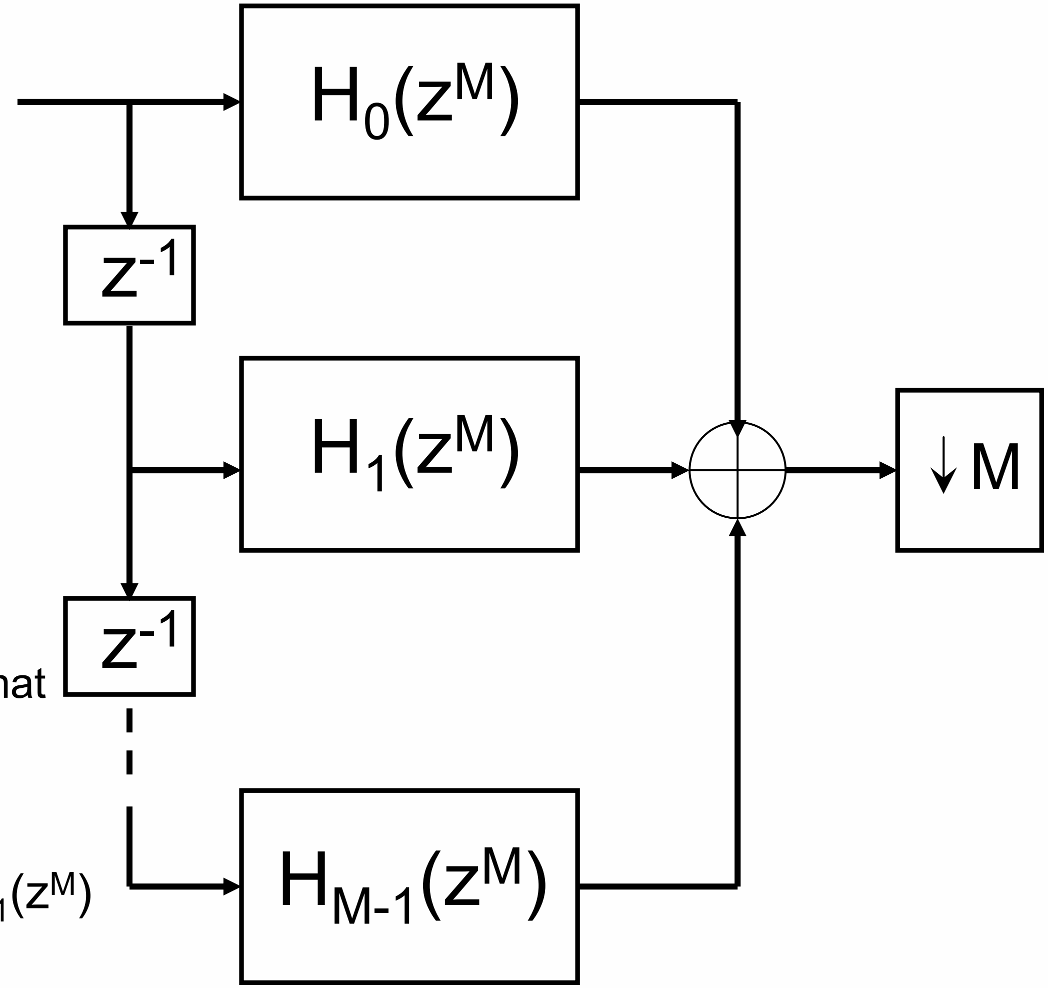


Reestablish efficiency: Polyphase filter structure

Any lowpass filter



≡



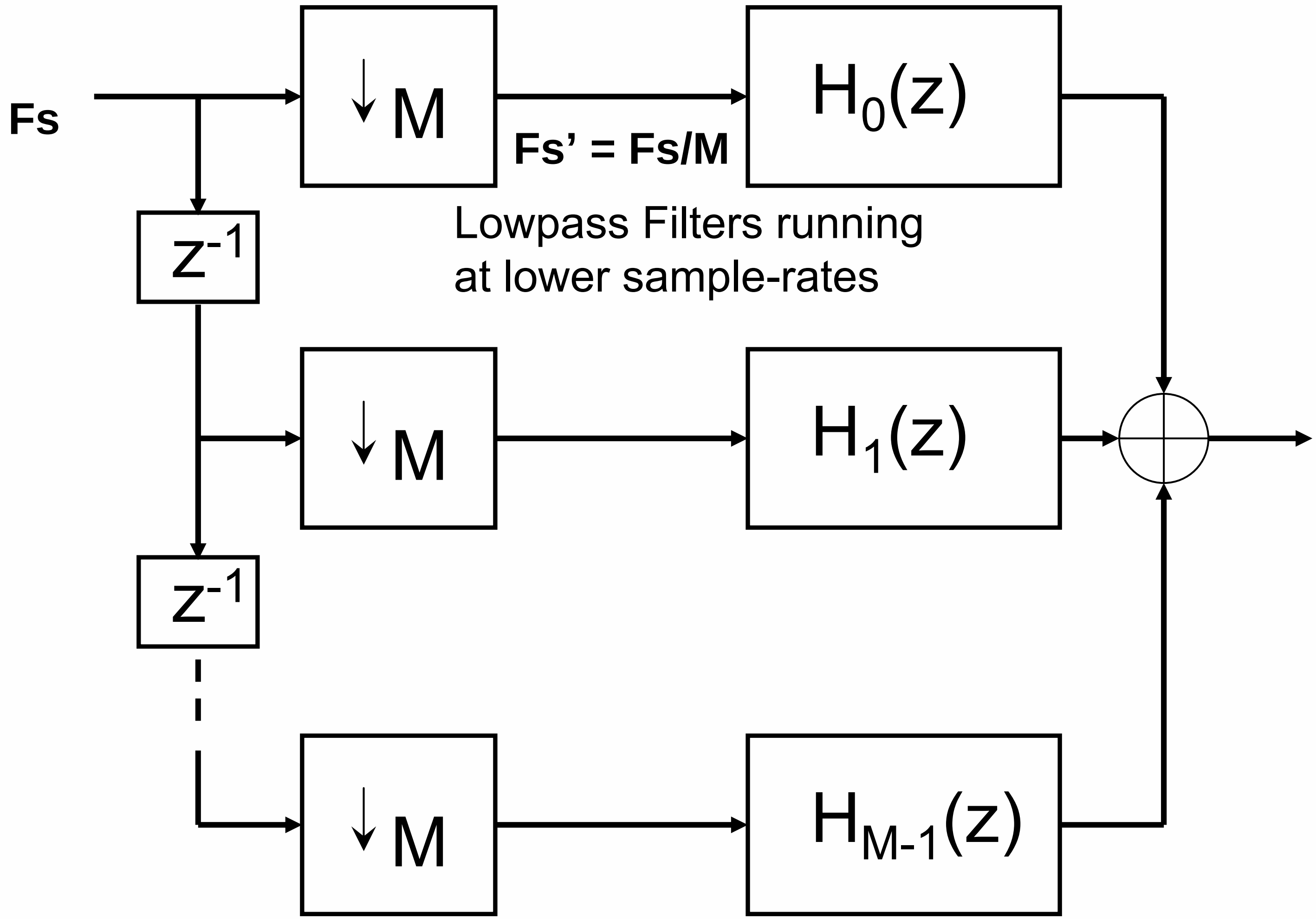
Can be represented in polyphase format

$H(z) =$

$b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N} =$

$H_0(z^M) + z^{-1} H_1(z^M) + \ldots + z^{-M+1} H_{M-1}(z^M)$

Efficient Polyphase Decimators

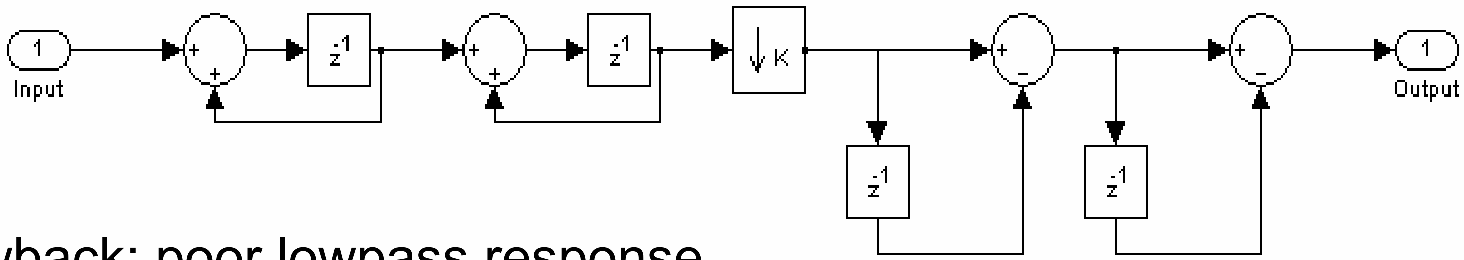


Efficient Multirate Filters

- **Interpolators**
  - Polyphase FIR interpolator
  - Hold interpolator
  - Linear interpolator
  - Frequency Domain interpolator
  - Cascaded Integrator-Comb (CIC) interpolator
- **Decimators**
  - Polyphase FIR decimator
  - Transposed polyphase FIR decimator
  - CIC decimator
- **Sample-rate converters**
  - Polyphase FIR SRC
  - Polyphase fractional decimator
  - Polyphase fractional interpolator

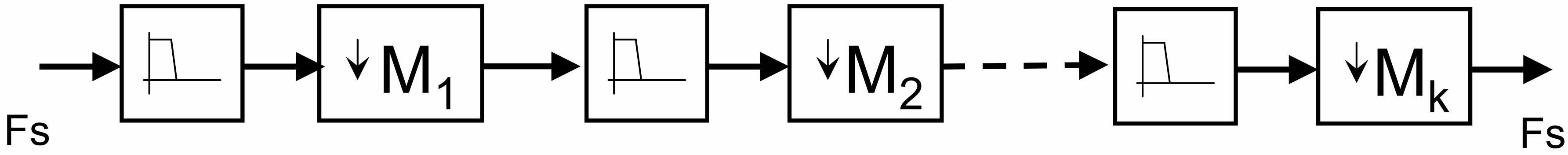
Featuring multistage CIC Filters

- Very computationally efficient: No multipliers



- Drawback: poor lowpass response
- Need cascading with a compensation filter
- Multistage cascades reduce computational cost

$$\mathbf{M = M_1 * M_2 * \ldots * M_k}$$

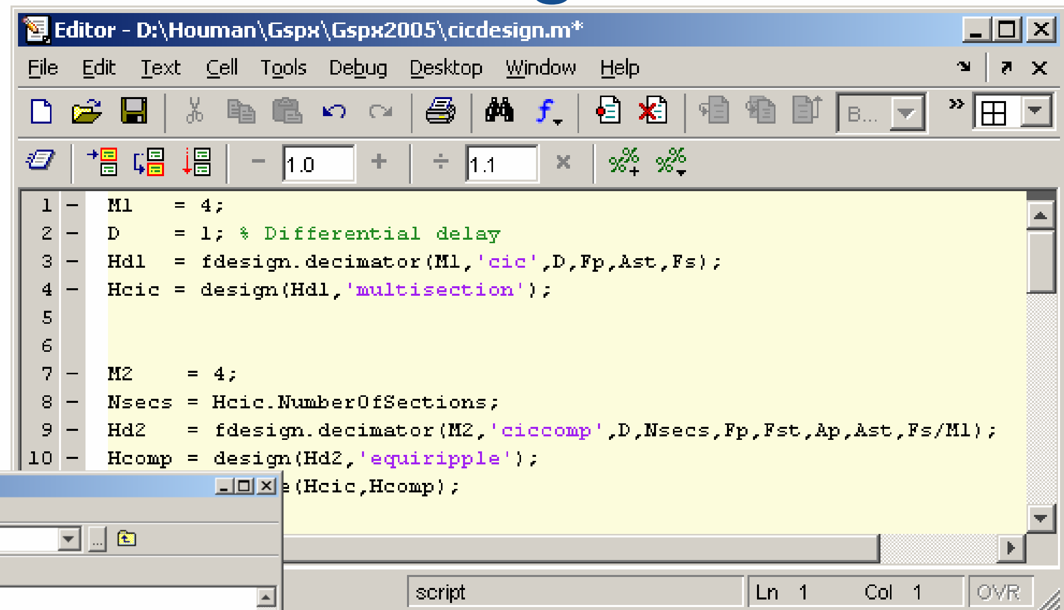


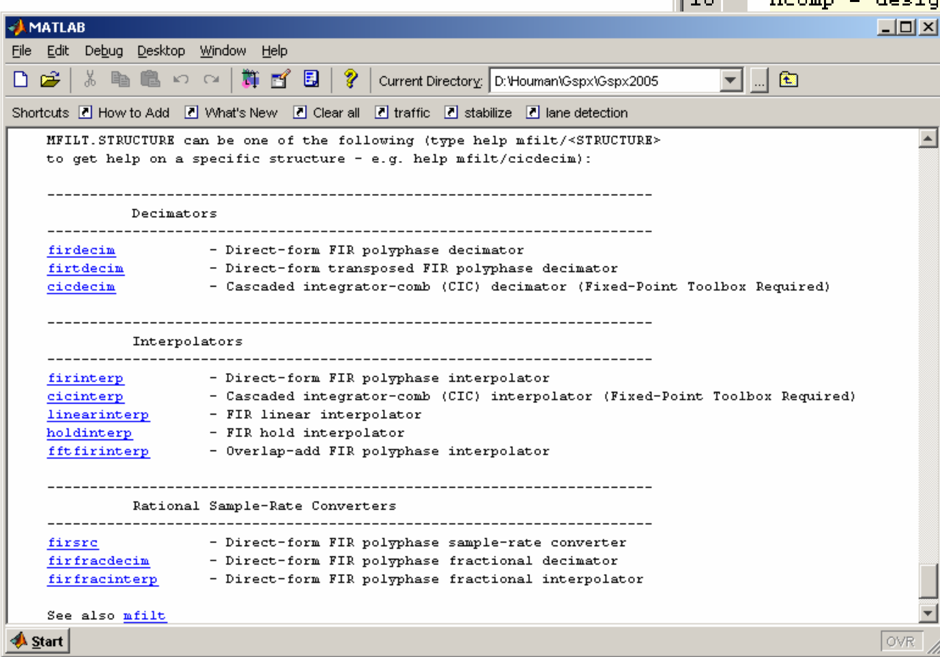
The MathWorks

MATLAB® & SIMULINK®

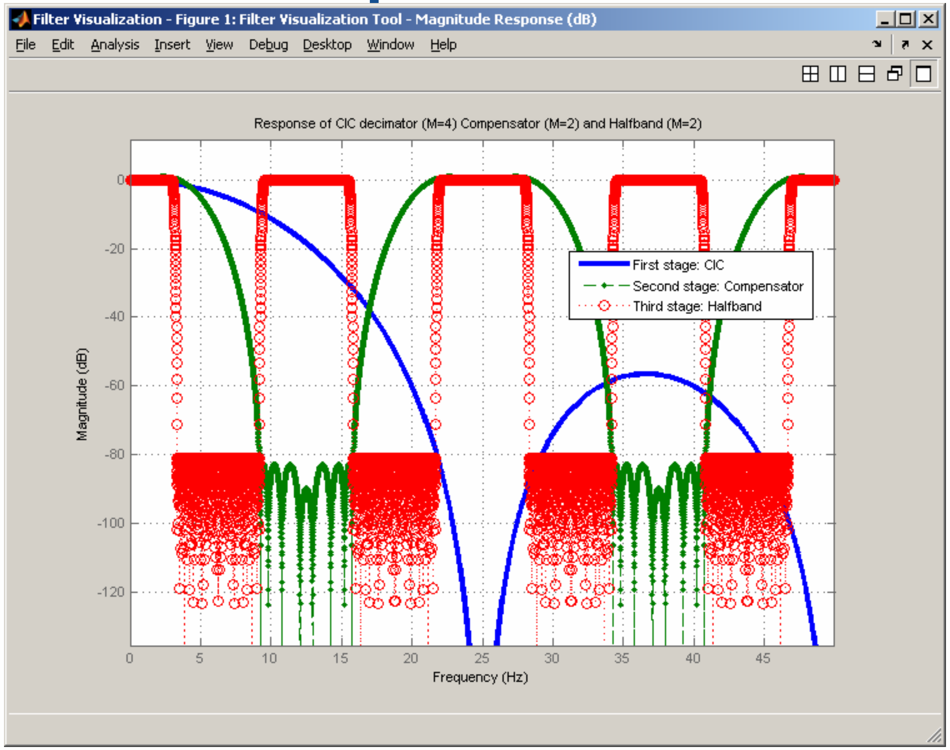
Design of cascaded multistage decimators

- Design
  - fdesign objects
- Implementation
  - mfilt objects





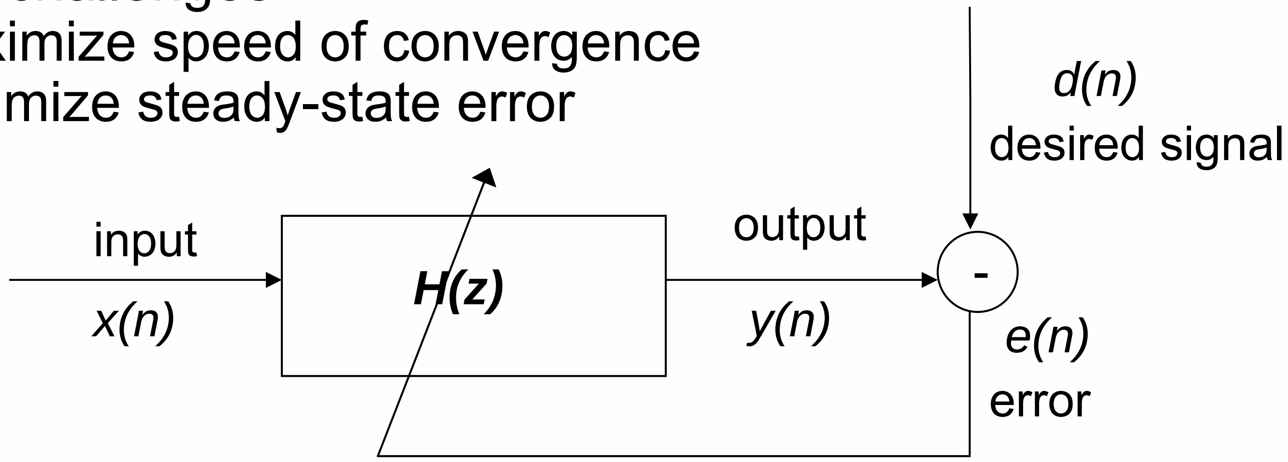
CIC + multi-stage polyphase and half-band compensators: Filter response



CIC with 2-stage Compensator	
Sampling Frequency (MHz)	100
Decimation Factor	4 x 2 X 2
Number of Multipliers	86
Number of Adders	94
Number of States	166
Multiplications per input sample	6.0625
Additions per input sample	12.125
Operations per second (MOPS)	**1818**

Adaptive filters

- Tracking a desired signal by adapting a filter based on error between desired signal and filter output
- Applications include:
  - Acoustic echo cancellation
  - Adaptive Noise Canceling (ANC)
  - Equalization in Digital Communications
  - Active Noise Control
- Design challenges
  - Maximize speed of convergence
  - Minimize steady-state error



Adaptive Filtering Algorithms in Filter Design Toolbox

- **Gradient-based**
  - LMS
  - Normalized LMS
  - Block LMS
  - Delayed LMS
  - Adjoint LMS
- **Sign Algorithms**
  - Signed-error
  - Signed-data
  - Signed-sign
- **Affine projection**
  - Direct matrix inversion
  - Recursive updates
  - Block AP
- **Active noise control**
  - Filtered X LMS
- **Recursive least-squares**
  - RLS, RW-Kalman
  - Sliding-window RLS
  - Householder
  - Householder sliding-window
  - QR decomposition
- **Frequency-domain**
  - FDAF
  - Unconstrained FDAF
  - Partitioned-block FDAF
  - Unconstrained PBFDAF
- **Fast algorithms**
  - FTF, SWFTF
  - GAL, Least-squares lattice

Using Adaptfilt filter object

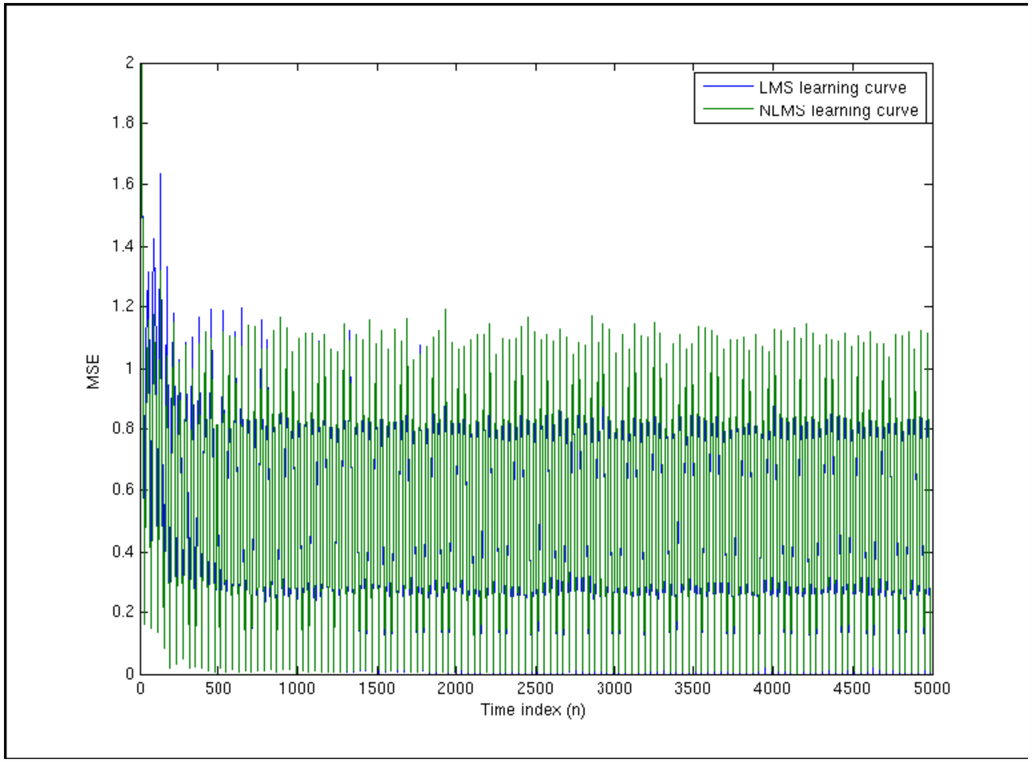
- Construction

hlms = adaptfilt.lms(7);

- Filtering with overloaded filter function
- Compute mean squared error

msetlms = msesim(hlms,v2,x,M);
msenlms = msesim(hnlms,v2,x,M);

- Trade-off between convergence & steady state MSE



Spectral analysis

- Time-frequency duality
- Gain insight from analyzing spectral content
- Power spectral density as Fourier transform of signal auto-correlation
- Spectrum objects to study power spectrum

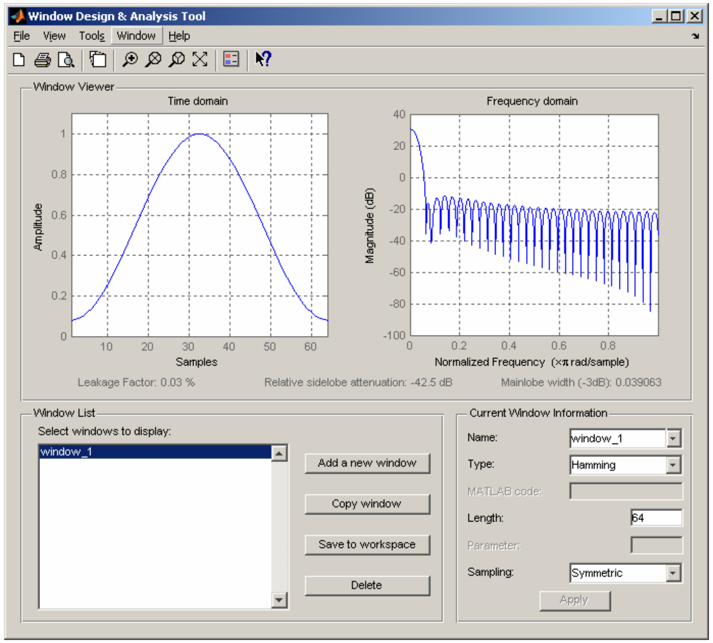
```
h= spectrum.periodogram;
h =
    EstimationMethod: 'Periodogram'
           FFTLength: 'NextPow2'
          WindowName: 'Rectangular'
```

Signal Processing Toolbox spectral analysis techniques

- Periodogram
- Welch
- MTM (Thomson multitaper method)
- Burg
- Covariance
- Modified Covariance
- Yule-Walker
- MUSIC (Multiple Signal Classification)
- Eigenvector

Benefits of spectral objects

- Estimating the spectral characteristics of systems operating on received signals
- Effect of windowing and overlaps on power spectral estimate
- Wintool



Fixed-Point Signal Processing

- Link between algorithm development and hardware implementation
- Lower cost: driver for using fixed-point processors
- Design challenges:
  - Conversion of design to fixed-point
  - Model the effect of finite word lengths
  - Ensure adherence to specifications before hardware prototyping

What is Fixed-Point?

- Finite word length arithmetic with a fixed number of fractional digits

```
>> a=fi(pi, true, 8, 5);
>> bin(a)
0   1   1 .  0   0   1   0    1
s   2   1 . 1/2 1/4 1/8 1/16 1/32
>> double(a)
3.15625
```

Fixed-Point in MATLAB®

- Fixed-point numeric object `fi`
  - Bit-faithful fixed-point math in MATLAB
  - Fixed-point algorithm development in M
  - Natural MATLAB syntax

```
>> a=fi(0.1);
>> bin(a)
ans =

0110011001100110
```

Benefits of `fi`?

- Quick fixed-point algorithm design and prototyping
- Test vectors for verification and validation
- Arbitrary word lengths (up to 65535 bits)
- Easier algorithm debug and visualization
- Enables fixed-point in Filter Design Toolbox
- Supports Simulink To/From Workspace
- Supported in Embedded MATLAB Function block

Workflow of embedded fixed-point algorithm designer

1. Set-up simulation flow (initialization, loop, termination)
2. Express your floating-point M-code algorithm
   - Focus on algorithmic integrity, proof of concept
3. Simulate
   - iterate on algorithm trade-offs, validate against requirements
4. Convert design to fixed-point
   - Focus of design viability based on implementation constraints
5. Simulate
   - iterate on implementation trade-offs, validate against original requirements
6. Generate code for hardware implementation
7. Validate and verify design after hardware deployment

Conversion of design from floating to fixed-point

Steps involved with translating dynamic range of floating-point signal to convert design into fixed-point

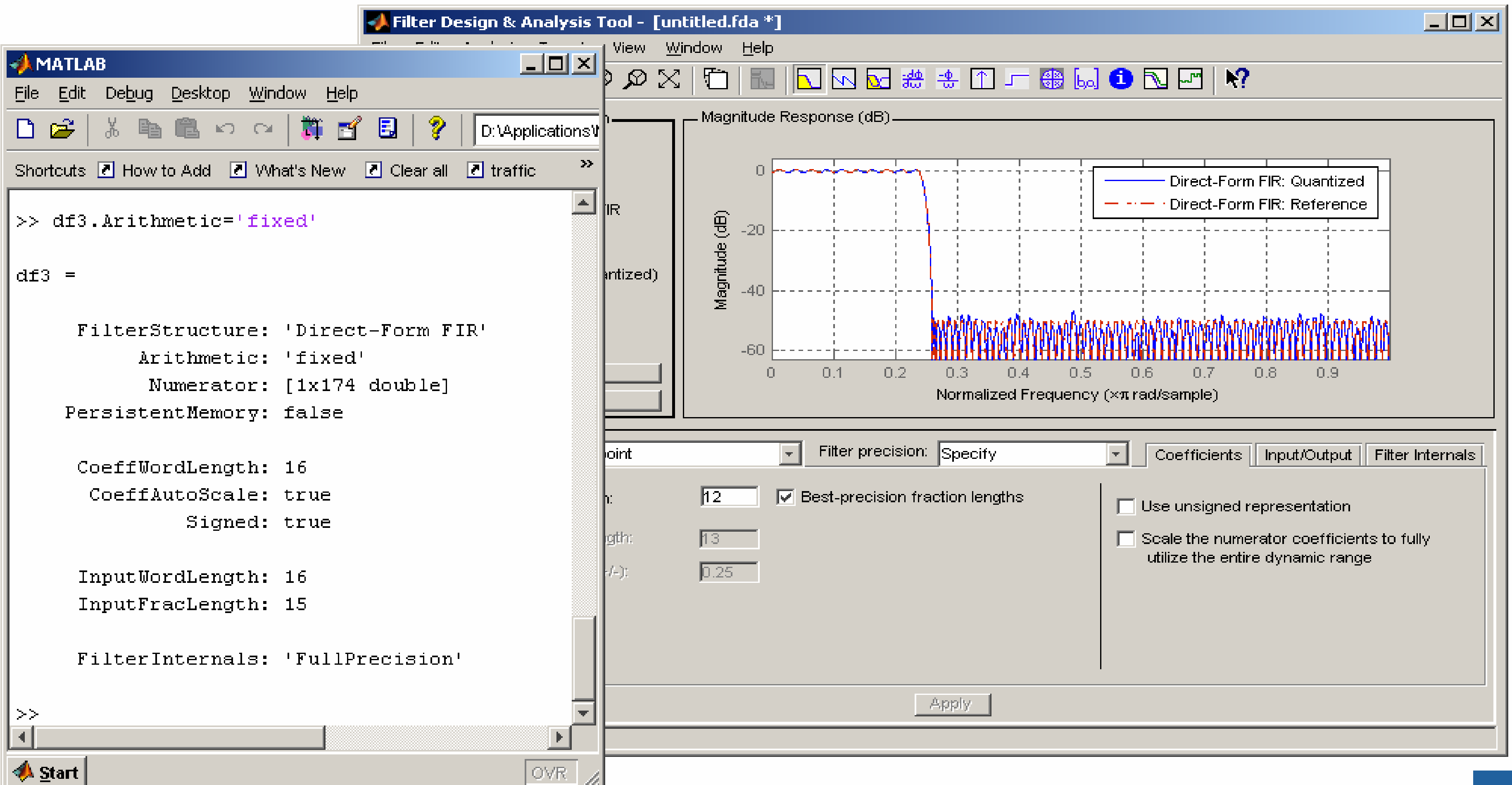
1. Compute the range of the min/max logs
2. Compute the integer part such that the range will not overflow
3. Compute the fraction length
4. Construct the fixed-point numeric type object

```
1. A = max(abs(double(minlog(x))),abs(double(maxlog(x))));
2. integer_part = ceil(log2(A));
3. fraction_length = word_length - integer_part - double(logical(is_signed));
4. T = numerictype(is_signed, word_length, fraction_length);
```

Conversion of filter to fixed-point

Set the fixed-point property of the dfilt object

At command-line or in fdatool GUI

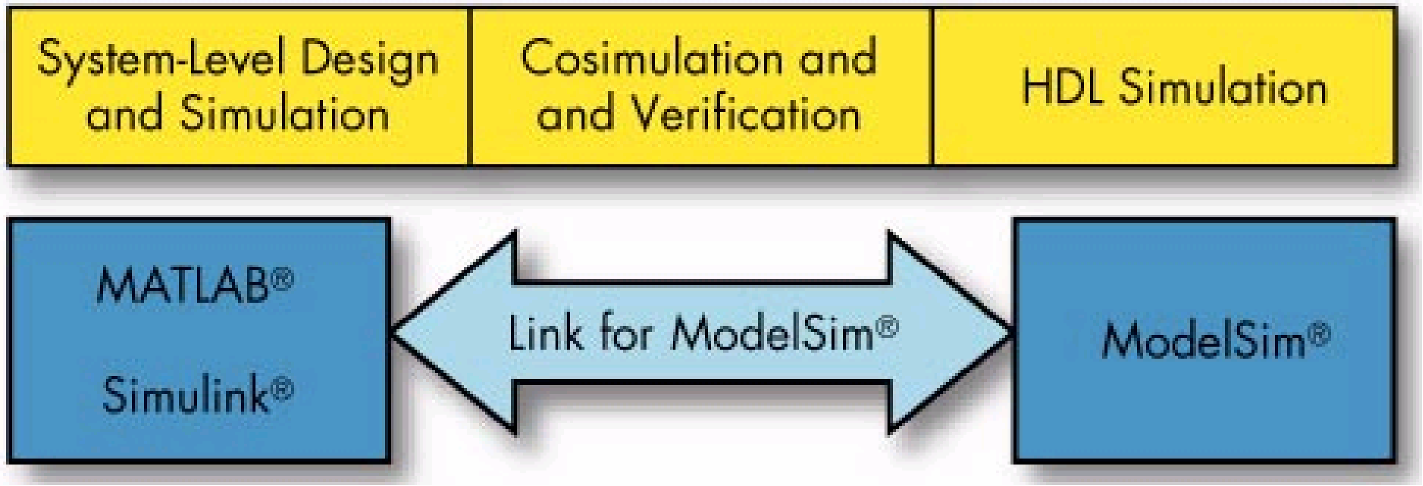


Path to C and HDL Implementation

- System-level simulation and integration
  - Simulink, Signal Processing Blockset
  - Support for single-rate, multirate adaptive filters
    - Realizemdl and block methods
- Automatic C code generation from Simulink
  - Real-Time Workshop
  - Real-Time Workshop Embedded Coder
- Automatic HDL code generation for filters
  - Filter Design HDL Coder
  - Support for single-rate, multirate adaptive filters

Hardware Verification & Validation

- Link for Code Composer Studio
  - TI hardware
- Link for ModelSim
  - Simulate HDL generated using ModelSim



Summary

- MATLAB Signal Processing capabilities are productivity tools designed to respond to everyday challenges of researchers, scientists and engineers in all stages of development process
- These include filter design, implementation, for single-rate, multirate and adaptive filters, spectral analysis, conversion of algorithms and filters to fixed-point and path to automatic hardware code generation and verification

For more information

- About MATLAB and Simulink signal processing products
  - http://www.mathworks.com/products/product_listing/index.html
- About relevant product demos
  - http://www.mathworks.com/products/demos/index.html
- User-contributed examples in MATLAB Central
  - http://www.mathworks.com/matlabcentral