Introduction to System Objects and System Toolboxes

Tabrez Khan
Application Engineer
Agenda

- Introduction
  - What are System Objects
- Working with System Objects
- Introduction to System Toolboxes in MATLAB
- How System Toolboxes Address Design Flow Challenges
- Summary, Q&A
What are System objects?

System objects are data-driven algorithm objects in MATLAB for efficient processing of streaming data.

- System objects provide the following unique functionality (not available in MATLAB functions):
  1. Implicit state management, indexing and buffering
  2. Support for fixed-point arithmetic
  3. Support for C and HDL code generation
  4. Consistent API for system design capabilities

* with MATLAB Coder, HDL Coder
+ with Fixed-Point Toolbox
What do System Object Bring to The Table
Batch Processing

- Load the entire data set and process it all at once
Stream Processing

- Load a frame of the signal and process it before moving on to the next frame
Batch Processing

Requires entire dataset to play audio

```
[audio, Fs] = wavread('dspafxf_8000.wav');
filtCoeff = fir1(40, 0.8, 'low');
audio_filt = filter(filtCoeff, 1, audio);
wavplay(audio_filt, Fs);
```

“audio” data uses more space than needed (double vs. uint16)

Loads entire dataset into workspace
Stream Processing in MATLAB

```matlab
%% Streaming the MATLAB way
% set up initializations
filename = 'dspafxf_8000.wav';
Fs = 8000;
info = mmfileinfo(filename);
um_samples = info.Duration*Fs;
frame_size = 40;
filtCoeff = fir1(40, 0.8,'low');
filtState = zeros(1,numel(filtCoeff)-1);
output = zeros(1,num_samples);

%% Processing in the loop
index= 1;
while index < (num_samples-frame_size+1)
    data = wavread(filename,[index index+frame_size-1]);
    [datafilt,filtState] = filter(filtCoeff,1,data,filtState);
    output(index:index+frame_size-1) = datafilt;
    index = index + frame_size;
end
wavplay(output,Fs);
```

More code than in batch processing example

Explicit state management

Explicit indexing

Need to maintain output buffer
System Objects Make Stream Processing Easier

```
% set up initializations
filename = 'dspafxf_8000.wav';
hAudio = dsp.MultimediaFileReader(filename, ...
    'SamplesPerAudioFrame', 40, 'AudioOutputDataType', 'double');
hFilt = dsp.DigitalFilter(…
    'TransferFunction', 'FIR (all zeros)', ...
    'Numerator', fir1(40, 0.8, 'low');
hPlayer = dsp.AudioPlayer('SampleRate', 8000);

%% Processing in the loop
while ~isDone(hAudio)
    data = step(hAudio);
    datafilt = step(hFilt, data);
    step(hPlayer, datafilt);
end
```

Initialize objects

“In-the-loop” code is much simpler

Implicit states and indexing

Audio player runs in-the-loop
Example: Filtering Audio Data From a File

```plaintext
while ~isDone(hSigsource)
    y = step(hSigsource);   % Read from audio file
    filtout = step(hFilt, y); % Filter the signal
    step(hAudioout, filtout); % Play the output signal
end
```
Working with System Objects

All system objects have a common workflow:

1. Create the System object
2. Specify properties of the System object
3. Run algorithm by calling the `step()` method
4. If your System object opens a file or a device, release the object when you are done by calling its `release()` method.
Example: A Signal Processing Source System Object

%% Step 1: Create System object
hChirp = dsp.Chirp;

%% Step 2: Assign properties to System object
set(hChirp,'SweepDirection', 'Bidirectional', ...  
   'SamplesPerFrame', 200, 'SampleRate', 4000);

%% Step 3: Run algorithm by calling the step() method
for n = 1:300
    y = step(hChirp); % Generate one frame of chirp signal
    plot(y);          % Plot the frame
    drawnow;          % Force updating the figure window
end
# Features of programming with functions or System objects

<table>
<thead>
<tr>
<th>Features</th>
<th>Functions</th>
<th>System Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory-efficient stream processing (sources &amp; sinks)</td>
<td>Manual Data indexing &amp; buffering</td>
<td>Automatic Data indexing &amp; buffering</td>
</tr>
<tr>
<td>Update of internal states and buffers</td>
<td>Manual</td>
<td>Automatic</td>
</tr>
<tr>
<td>Separation of initialization from in-loop processing</td>
<td>Manual Use persistent variables</td>
<td>Automatic Initialize: object creation In-loop processing: step method</td>
</tr>
<tr>
<td>Separation of constant (ROM) from variable (RAM) properties</td>
<td>Manual Definition of tunability</td>
<td>Automatic Definition of tunability</td>
</tr>
<tr>
<td>Simulation speed</td>
<td>Baseline</td>
<td>Faster (*)</td>
</tr>
<tr>
<td>C code generation support</td>
<td>Limited</td>
<td>Majority</td>
</tr>
<tr>
<td>Fixed-point support</td>
<td>Limited</td>
<td>Majority</td>
</tr>
<tr>
<td>Code maintainability, API</td>
<td>Nominal</td>
<td>Better</td>
</tr>
</tbody>
</table>

(*) Except functions that use optimized libraries
Benefits of Using System Objects

• Easier processing of streaming data
  • Implicit management of states, indices, and buffers
• Simple and efficient code
  • Algorithm parameters setup only once
  • Subsequent calls to `step()` method without any parameters
    - Avoid repeated input argument verification and validation in a loop
• Access to new algorithms previously not available in MATLAB
• Support for C and HDL code generation†
• Support for fixed-point arithmetic‡

† with MATLAB Coder, HDL Coder
‡ with Fixed-Point Toolbox
Agenda

- Introduction
  - What are System Objects
- Using System Objects
- Introduction to System Toolboxes in MATLAB
- How System Toolboxes Address Design Flow Challenges
- Summary, Q&A
System Designer’s Challenges

Rapid Innovation

“I need to find innovative algorithms and create a working system prototype very quickly.”

Faster Simulations

“I have to process large amounts of data and test my simulations for several parameter value sets. Simulation speed is the biggest bottleneck.”

Standardized System Design API

“I often have to work with other engineers’ code and it takes me a long time to understand their code. If we could all use standardized APIs, code sharing and reuse would be much better.”

Design Flow Gaps

“I need to rewrite my high-level MATLAB algorithm into low-level code like C/C++ for implementation purposes. I then need to verify that the rewritten code works the same way as the original MATLAB code.”
System Toolboxes Address System Design Challenges

**Rapid Innovation**

Pre-defined algorithms enable faster design iterations very quickly.

**Standardized System Design API**

“I often have to work with other engineers’ code and it takes me a long time to understand their code. If we could all use standardized APIs, code sharing and reuse would be much better.”

**Faster Simulations**

Streaming techniques speed up simulations parameter value sets. Simulation speed is the biggest bottleneck.

**Design Flow Gap**

Algorithm support for fixed-point and C code generation enable design continuity in MATLAB codee works the same way as the original MATLAB code.”
What are System Toolboxes?

- A new set of products that spans between MATLAB and Simulink providing a more unified environment for design, analysis, and system simulation

- System Toolboxes:
  - Contain a rich application specific set of algorithms for simulation and design in both MATLAB and Simulink
  - Provide a standardized API and object oriented design methodology for reusable component design
  - Support both stream and batch processing in MATLAB and Simulink
  - Contain both fixed-point and floating point algorithm implementations
  - Support both MATLAB & Simulink based code generation
System Toolboxes Enable System Design for DSP, Comms, and Video Applications

DSP System Toolbox
Platform for Signal Processing System Design

Phased Array System Toolbox
Phase Array and Radar System Design

Communications System Toolbox
Communications System Design

Computer Vision System Toolbox
Image Processing and Computer Vision
DSP System Toolbox

Over 300 algorithms for
- Advanced filter design
- FFTs
- Multirate DSP
- Linear algebra routines

Algorithm libraries in MATLAB

Algorithm libraries in Simulink
Demo: Adaptive Noise Cancellation of Streaming Audio

- Use System objects to maintain filter states automatically
- Visualize changing filter coefficients as they adapt to input signal
- Listen to output audio in real-time

```matlab
%% Stream Processing Loop
numplays = 0;
while numplays < 3
    [y, eof] = step(hAudioIn);       % Read from audio file
    noise = randn(s,40,1);          % Produce random data
    noisefilt = step(hFilt, noise); % Filter the random data
    desired = noisefilt + y;        % Construct LMS 'desired' input
    [out, err, w] = step(hLms, noise, desired); % Run the LMS filter
    step(hAudioOut, double(err));   % Play the output signal
    step(hAudioFile, double(err)); % Play the output signal
    hplot(w);                       % Plot the weights
    numplays = numplays + eof;      % Update number of plays
end
```

>> dspAcousticNoiseCanceller
System Toolboxes Enable System Design for DSP, Comms, and Video Applications

DSP System Toolbox
Platform for Signal Processing System Design

Phased Array System Toolbox
Phase Array and Radar System Design

Communications System Toolbox
Communications System Design

Computer Vision System Toolbox
Image Processing and Computer Vision
# Phased Array System Toolbox

## Phased array design and analysis
- Linear, rectangular, conformal geometries
- Shading, tapering
- Element position and orientation
- Gain, delay, steering vector

## Temporal processing
- Time varying gain, pulse compression
- Coherent, non-coherent integration
- Signal detection and ROC curves
- CFAR processing, range/Doppler estimation

## Waveform design and analysis
- Pulsed CW
- LFM and stepped FM
- Staggered PRF
- Ambiguity function
- Matched filter

## Spatial processing
- Digital beamforming: narrowband & broadband, Conventional, MVDR, LCMV, Frost, time delay, time delay LCMV, subband phase shift
- DOA processing: Monopulse, MVDR, beamscan, ESPRIT, Root-MUSIC

## Signal modeling framework
- Monostatic and multistatic scenarios
- Point target and Swerling target models
- Narrowband and broadband modeling
- Platform motion

## Space-time adaptive processing
- Displaced phase center array (DPCA)
- Adaptive DPCA
- Sample matrix inversion (SMI)
- Angle-Doppler response
Phased Array Systems: Block Diagram View

Waveform Generator → Transmitter → Transmit Array

Environment, Targets, and Interference

Signal Processing ← Receiver ← Receive Array
Demo: Uniform Linear Array (ULA)

½ wavelength spacing
End-to-End Monostatic Radar System

- Model a complete end-to-end radar system in MATLAB
- Model spatial effects (multi-target dynamics)
- Visualize temporal performance of the system
System Toolboxes Enable System Design for DSP, Comms, and Video Applications

**DSP System Toolbox**
Platform for Signal Processing System Design

**Phased Array System Toolbox**
Phase Array and Radar System Design

**Communications System Toolbox**
Communications System Design

**Computer Vision System Toolbox**
Image Processing and Computer Vision
Communications System Toolbox

Over 100 algorithms for
- Modulation, Interleaving, Channels, Source Coding
- Error Coding and Correction
- MIMO, Equalizers, Synchronization
- Sources and Sinks, SDR hardware

Algorithm libraries in MATLAB

Algorithm libraries in Simulink
Communications System Toolbox
System level communications modeling and simulation for MATLAB and Simulink

Key capabilities:

- Streaming sources and visualizations (eye diagrams, constellations, EVM, BER)
- Library of algorithms for encoding/decoding, interleaving, filtering, mod/demod, equalization and synchronization
- Channel models
- Simulation tools for BER and channel visualization
- Hardware interfaces for SDR (USRP2)
- Support for code generation and fixed-point modeling
System Toolboxes Enable System Design for DSP, Comms, and Video Applications

**DSP System Toolbox**
Platform for Signal Processing System Design

**Phased Array System Toolbox**
Phase Array and Radar System Design

**Communications System Toolbox**
Communications System Design

**Computer Vision System Toolbox**
Image Processing and Computer Vision
Computer Vision

- Using images and video to detect, classify, and track objects, activities, or events in order to “understand” a real-world scene
Typical Computer Vision Applications

- **Defense**
  - Video surveillance
  - Automatic target recognition

- **Security**
  - Iris or fingerprint recognition
  - People tracking

- **Consumer electronics**
  - Smile detection in new digital cameras
  - Gaming systems (Microsoft Kinect)

- **Navigation**
  - Robotic navigation (DARPA Grand Challenge)
  - Automotive safety systems
Computer Vision System Toolbox

Design and simulate computer vision and video processing systems

- Feature detection and extraction
- Registration and stereo vision
- Object detection and tracking
- Motion estimation
- Video processing, file I/O, display, and graphic overlays
Demo: Face Tracking with CAMShift

- **Workflow**
  - Detect face and nose
  - Use Hue channel from HSV space
  - Initialize histogram tracker
Summary: Benefits of System Toolboxes

- System Toolboxes provides system design capabilities in MATLAB and Simulink

  ✓ Pre-defined algorithms enable faster design iterations

  ✓ Streaming simulation techniques enable modeling of real-time signal processing systems

  ✓ Object-oriented implementation of algorithms speed up system simulations

  ✓ Algorithm support for fixed-point and C code generation enable design flow continuity
Training Services

*Exploit the full potential of MathWorks products*

Flexible delivery options:
- Public training available in several cities
- Onsite training with standard or customized courses
- Web-based training with live, interactive instructor-led courses

More than 30 course offerings:
- Introductory and intermediate training on MATLAB, Simulink, Stateflow, code generation, and Polyspace products
- Specialized courses in control design, signal processing, parallel computing, code generation, communications, financial analysis, and other areas

www.mathworks.in/training
# Public Trainings in the next Few Months

<table>
<thead>
<tr>
<th>Course</th>
<th>Dates</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink for System and Algorithm Modeling</td>
<td>20 Aug 2012 – 21 Aug 2012</td>
<td>Bangalore</td>
</tr>
<tr>
<td>Embedded Coder for Production Code Generation</td>
<td>22 Aug 2012 – 24 Aug 2012</td>
<td>Bangalore</td>
</tr>
<tr>
<td>MATLAB Fundamentals</td>
<td>03 Sep 2012 – 05 Sep 2012</td>
<td>Bangalore</td>
</tr>
<tr>
<td>MATLAB Programming Techniques</td>
<td>06 Sep 2012 – 07 Sep 2012</td>
<td>Bangalore</td>
</tr>
<tr>
<td>MATLAB Fundamentals</td>
<td>24 Sep 2012 – 26 Sep 2012</td>
<td>Pune</td>
</tr>
<tr>
<td>Simulink for System and Algorithm Modeling</td>
<td>27 Sep 2012 – 28 Sep 2012</td>
<td>Pune</td>
</tr>
<tr>
<td>MATLAB Based Optimization Techniques</td>
<td>17 Oct 2012</td>
<td>Bangalore</td>
</tr>
</tbody>
</table>

Email: [training@mathworks.in](mailto:training@mathworks.in)  
URL: [http://www.mathworks.in/services/training](http://www.mathworks.in/services/training)  
Phone: 080-6632-6000
# MathWorks India Contact Details

<table>
<thead>
<tr>
<th>URL:</th>
<th><a href="http://www.mathworks.in">http://www.mathworks.in</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail:</td>
<td><a href="mailto:info@mathworks.in">info@mathworks.in</a></td>
</tr>
<tr>
<td>Technical Support:</td>
<td><a href="http://www.mathworks.in/myservicerequests">www.mathworks.in/myservicerequests</a></td>
</tr>
<tr>
<td>Tel:</td>
<td>+91-80-6632 6000</td>
</tr>
<tr>
<td>Fax:</td>
<td>+91-80-6632 6010</td>
</tr>
</tbody>
</table>

- **MathWorks India Private Limited**
  Salarpuria Windsor Building
  Third Floor, 'A' Wing
  No.3 Ulsoor Road
  Bangalore - 560042, Karnataka
  India

**Thank You for Attending**
**Talk to Us – We are Happy to Support You**