embedded world 2014
Exhibition & Conference
... it´s a smarter world

# *System objects: Design, optimization, and C code generation for signal processing in MATLAB®*

Marco Roggero
Application Engineer
The MathWorks GmbH
Aachen, Germany
marco.roggero@mathworks.de

Youssef Abdelilah
Senior Product Manager
The MathWorks
Natick, MA, United States
youssef.abdelilah@mathworks.com

***Abstract*—Applications that run on embedded electronic devices are continuously increasing in complexity and are processing ever larger amounts of data. To develop these applications, engineers need a design flow that supports rapid development of proof-of-concept algorithms and simulation with complete and realistic data. Further, engineers are looking for ways to directly implement these algorithms on embedded platforms.**

**System Toolboxes introduce new libraries of sophisticated algorithms implemented with System objects, as well as MATLAB® functions and Simulink® blocks. System objects are available for signal processing applications including computer vision, image processing, communications, and phased array systems.**

**System objects make it possible to perform stream-based or frame-based signal processing with a reduced memory footprint, which makes real-time simulations possible and memory-efficient for applications that handle signals and large amounts of data.**

**The separation of initialization steps from in-the-loop processing when using System objects provides a significant speed up for most algorithms. Additional speed can be achieved by automatic conversion of MATLAB functions into C code to be used as a MEX (MATLAB-executable) function.**

**System objects support automatic generation of C code for standalone PC applications, dynamic libraries, and embedded targets (with code optimization targeting specific embedded processor families using floating-point or fixed-point arithmetic).**

**In this article, using examples from the signal processing, image processing, and communications fields, we illustrate how System objects can simplify and speed algorithm development, accelerate simulations in MATLAB, reduce the memory footprint when working with signals and large amounts of data, and automatically generate embeddable C code.**

***Keywords— System objects; MATLAB; signal processing; stream processing; frame-based processing;***

## I. INTRODUCTION

System objects are MATLAB® objects that support a simplified object-oriented workflow to facilitate the creation, configuration, and execution of algorithms, providing easy access to real-time data and support for automatic code generation.
Each System object contains configurable properties and methods for specifying parameters, options, and operations that can be used and invoked for a specific object.
For example, to create an object named "Microphone" using the class dsp.AudioRecorder you can type this command in MATLAB:

```
>> Microphone = dsp.AudioRecorder
```

This will create a dsp.AudioRecorder object with default properties (Figure 1).

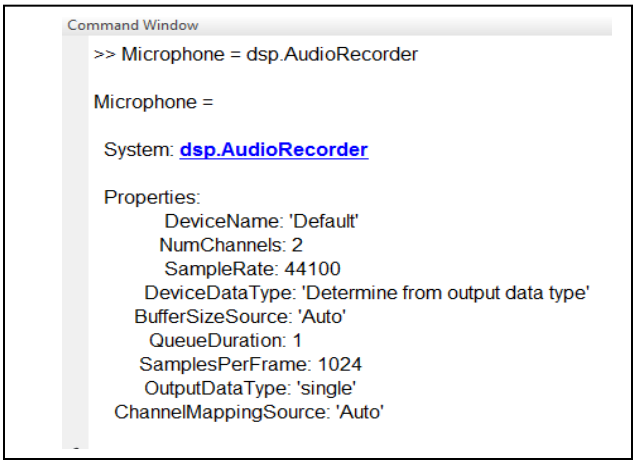


Fig. 1. Default properties of a dsp.AudioRecorder object.

To run the System object, you use the "step" method; for example:
>> audioIn = step(Microphone);

The command above creates a variable named audioIn containing a frame for the channels managed by the selected audio recorder device.

At the end of your script or function, you can use the "release" method to release any resources allocated by the System object; for example:

>> release(Microphone)

System objects support real-time simulations with a reduced memory footprint, faster execution, automatic generation of embeddable C/C++ code for fixed-point and floating-point data types, and the use of code replacement libraries for code optimization when targeting specific embedded processor families such as ARM Cortex processors.

The following sections cover characteristics and features of System objects, differences between System objects and standard MATLAB functions, and conditions to keep in mind to make the most of the advantages offered by System objects.

## II. REAL-TIME SIMULATIONS

Two main conditions have to be satisfied for real-time simulations. First, the simulation must be fast enough to process the incoming data. Second the memory needed for data processing must not exceed the simulation system's available memory.
For each incoming frame, time is needed both for data acquisition and data processing. Real-time signal processing is only possible if the frame time is longer than the total time needed to acquire the data and process it (Figure 2).

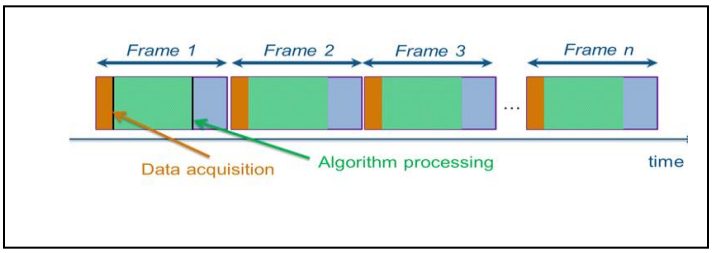


Fig. 2. Data acquisition and algorithm processing times.

### A. Algorithm acceleration

System objects help accelerate MATLAB algorithms in three main ways. First, most System objects use precompiled functions. Second, System object initialization is decoupled from in-the-loop processing. Third, most System objects support automatic C code generation so they can be converted into MEX functions.

Note that not every System object is guaranteed to be faster than comparable MATLAB code that provides similar results. Nevertheless, System objects do make most algorithms faster. To get the best results, we recommend dividing algorithms into smaller steps and replacing each step's MATLAB functions with System objects as appropriate. After comparing the performance of the algorithm with and without System objects, select the version that provides results faster.

To see how System objects decouple object creation and the object's use in the processing loop, examine the code shown in Figure 3.

The code is an example of creating a test bench for an audio signal processing system. Two System objects are initialized, used for 20 seconds in the processing loop, and then released.

```
%% Create and Initialize
SamplesPerFrame = 1024;
Fs = 44100;

Microphone=dsp.AudioRecorder('SamplesPerFrame',Sam
plesPerFrame);
Spectra=dsp.SpectrumAnalyzer ('SampleRate',Fs);
```

```
%% Stream processing loop
tic;

while toc < 20
     % Read frame from microphone
       audioIn = step(Microphone);

     % View audio spectrum
       step(Spectra,audioIn);
End
```

```
%% Terminate
release(Microphone)
release(Spectra)
```

Fig. 3. Script that creates a test bench for audio signal processing systems.

Fig. 4 shows the spectrum of an audio signal visualized with the Spectrum Analyzer System object.

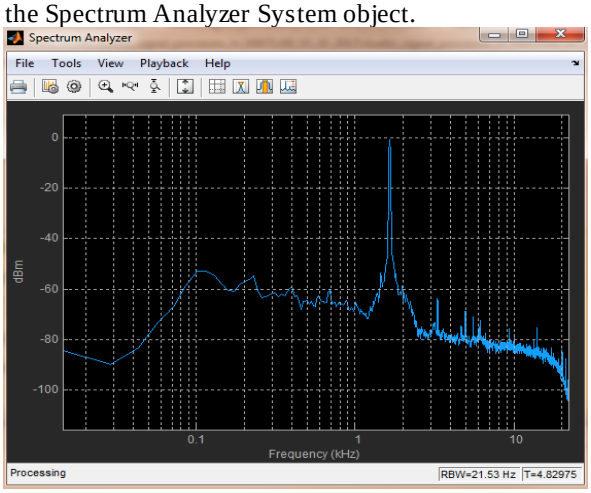


Fig. 4. Signal spectrum visualized with the Spectrum Analyzer System object.

In this example, the System objects' initialization is separated from their use. Each System object is declared and its parameters are set once. The processing loop invokes the System objects by calling the "step" method. Each loop iteration uses the existing System objects. After processing is complete, the System objects are released together with their allocated resources. Note that MATLAB functions providing the same results would be reinitialized and released each time they are called. System objects not only contribute to performance improvements, but also simplify code maintenance and reduce coding mistakes.

The next example compares two algorithms for detecting red objects that are not round in a video in which each frame is an RGB image. The first function (Figure 5) was written using only MATLAB functions; the second function (Figure 6) was written using a mix of System objects and MATLAB functions.

Examination of the code from these two examples reveals a few noteworthy observations.

In the System objects example, the System objects are initialized only once within the "if..end" statements. As a result, the System objects and their parameters are declared only the first time this function is called. In subsequent calls of this function, the System object declaration will be skipped and only the processing loop will be executed.

```
Defect_Identification_processing_TBF.m
 5   %% color segmentation and
 6 - INPUT= im2single(INPUT);
 7 - set(imag_input, 'CData', INPUT);
 8 - image_red=(2*INPUT(:,:,1)-INPUT(:,:,2)-INPUT(:,
 9 - thresh=0.3;
10 - bin_image = im2bw(image_red,thresh);
11 - BW = bwareaopen(bin_image, 20);
12 - fill = imfill(BW,'holes');
13   %% Extract features
14 - [labeled,numObjects] = bwlabel(fill,4);
15 - stats = regionprops(labeled,'Eccentricity','Bou
16 - eccentricities = [stats.Eccentricity];
17
18   %% Use feature analysis to identify broken obje
19 - idxOfDefects = eccentricities > .6;
20 - statsDefects = stats(idxOfDefects);
21
22   %% Label broken objects
23 - h = nan(1,length(statsDefects));
24 - for idx = 1 : length(statsDefects)
25 -         h(idx) = rectangle('Position',statsDefe
```

Fig. 5. Example of code for object detection written using only MATLAB functions.

```
Defect_Identification_processing_SO.m*
 4
 5     %% Initializing
 6 -   persistent CSC_RGB_YCbCr blob marker
 7 -   if isempty(CSC_RGB_YCbCr)
 8 -   CSC_RGB_YCbCr = vision.ColorSpaceConverter('Con
 9 -   blob = vision.BlobAnalysis('AreaOutputPort',fal
10 -   marker = vision.ShapeInserter('BorderColor','Cu
11 -   end;
12
13    %% Processing Loop
14
15 -   imageYCbCr = step(CSC_RGB_YCbCr, INPUT);
16 -   thresh=0.6;
17 -   bin_image = im2bw(imageYCbCr(:,:,3),thresh);
18 -   [BBOX Eccentricity] = step(blob,bin_image);
19 -   OUTPUT = step(marker, INPUT, BBOX(Eccentricity
20
21 -   end
```

Fig. 6. Example of code for object detection written using a mix of System objects and MATLAB functions.

Compare the lines of code needed for this image processing task (without the System object initialization, which takes place only when the function is called to process the first frame). Note that the first function used 15 lines of code, while the System objects version needs only 5 lines to achieve the same result.

During simulation, the first function supported a speed of 12 frames per second, while the function with System objects version could process 80 frames per second – a rate five times faster than the first function.

For this image processing example we automatically generated C code for the System objects algorithm and compiled it into a MEX function. This new version of the algorithm was able to process more than 130 frames per second – more than 10 times as many as the initial version of the algorithm.

Note: Simulation speed depends not only on the MATLAB algorithm in use but also on the hardware on which it is running and other processes that may be executing on the machine. For all use cases discussed here, comparisons were made using a consistent hardware and software setup.

We performed similar measurements on an acoustic tracker application that estimates the direction from which an audio signal is coming by measuring the time delay of arrival of the sound at an array of four microphones. We found similar results. The algorithm that used System objects was 4.9 times faster than the initial algorithm, which did not make use of System objects. The use of automatic code generation and MEX functions produced an algorithm that was 11.9 times faster than the initial algorithm.

The acceleration achievable with System objects or automatic C code generation depends heavily on several parameters, including the quality of the original MATLAB code, the kind of algorithm and application, and the data types used in the simulation. Figure 7 shows a comparison of results from three different applications using algorithms such as Discrete Cosine Transform (DCT)with standard toolbox functions only, with System objects, and with code generation and Parallel Computing Toolbox (PCT).

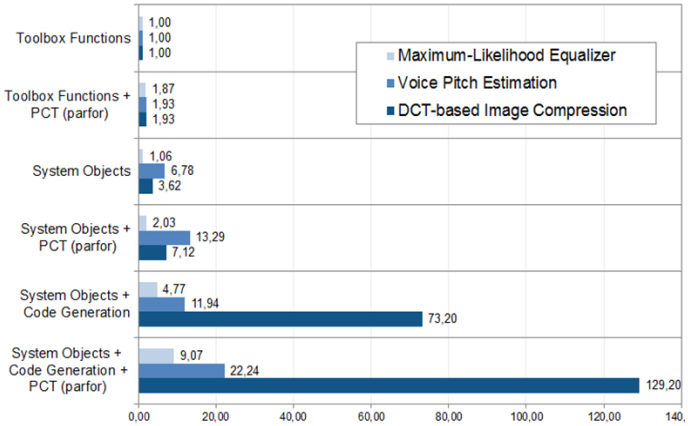


Fig. 7. Comparison of execution speeds for three different applications using different strategies for algorithm acceleration.

### B. *Memory footprint reduction with stream and frame-based processing*

A typical MATLAB function used to process a stream signal will work on the entire variable or file that holds the signal information. If necessary, data will be copied one or more times depending on the algorithm. When processing large amounts of data, this can require a substantial amount of memory to be allocated.

System objects are designed to support automatic and easy stream and frame-based processing [1]. If a particular algorithm requires it, a System object will create copies of processed data and allocate memory. However, since they work on a single frame, the same algorithm will require significantly less memory to be allocated than the same algorithm without System objects (Figure 8).

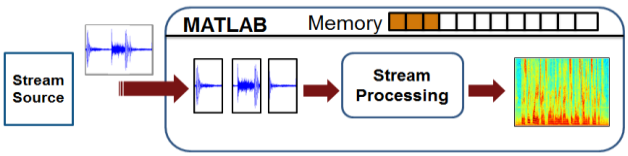


Fig. 8. Frame-based processing requires less memory.

## III. ALGORITHM IMPLEMENTATION

Once the mathematical idea behind an algorithm has been verified via simulation, several tasks remain before it can be tested on a embedded processor. These tasks include selecting the data types supported by the microcontroller, generating C code, and optionally optimizing the code for the target microcontroller.

System objects support most floating-point and fixed-point data types, enabling engineers to simulate and test the algorithm in MATLAB using the actual data type of the final implementation. These simulations can be invaluable in detecting overflow conditions and identifying and resolving discrepancies between the reference algorithm and the final algorithm.

After the algorithm has been verified with embeddable data types, the engineers can automatically generate C code for the embedded processor (as well as for standalone applications or libraries).

Engineers can then use code replacement libraries to optimize the automatically generated C code. Embedded processors and associated compilers have special instructions or *intrinsics* to support certain operations that are used frequently in typical embedded applications. These processor-specific instructions execute much faster than their ANSI/ISO C equivalents. Use of these instructions can improve code performance significantly. Engineers can select the target processor's code replacement library to generate processor-specific code that takes advantage of the processor's special instructions or intrinsics [2].

## CONCLUSIONS

This article has covered the use of System objects to model complex dynamic systems. System objects enable engineers to model real-time signal processing systems using streaming signal processing techniques, run faster simulations with larger data sets, and automatically generate C code from MATLAB signal processing algorithms. As a result, the process of taking a design concept to the final embedded implementation is faster, more efficient, and less costly.

## REFERENCES


[1] http://www.mathworks.com/discovery/stream-processing.html

[2] http://www.mathworks.com/help/rtw/examples/optimizing-embedded-code-via-code-replacement-library-1.html