Four Quick Steps to Production: Using Model-Based Design for Software-Defined Radio

Part 3—Mode S Signals Decoding Algorithm Validation Using Hardware in the Loop

By Di Pu and Andrei Cozma

Introduction

After implementing any signal processing algorithm in MATLAB or Simulink,® the next natural step is to verify the algorithm's functionality using real data acquired from the actual SDR hardware system that it is going to run on. As a first step, the verification of the algorithm is done using different sets of input data captured from the system. This helps validate the algorithm's functionality, but does not guarantee that the algorithm will perform as expected in environmental conditions other than the ones used to make the data captures, or what the behavior and performance will be for different settings of the analog front end and digital blocks of the SDR system. In order to verify all of these aspects, it is very beneficial if the algorithm can be run online to receive live data as input and to tune the settings of the SDR system for optimal performance. This part of the article series discusses the software tools provided by Analog Devices to allow direct interaction between MATLAB and Simulink models with the FMCOMMSx SDR platforms and shows how these tools can be used to verify the ADS-B models presented in Part 2 of the article series.[2]

MATLAB and Simulink IIO System Object

Analog Devices provides a complete software infrastructure that enables MATLAB and Simulink models to interact in real time with FMCOMMSx SDR platforms that are connected to FPGA/SoC systems running Linux. This is possible due to an IIO System Object™[3] that is designed to exchange data over TCP/IP with the hardware system in order to stream data to and from a target, control the settings of a target and monitor different target parameters such as the RSSI. Figure 1 presents the high level architecture of the software infrastructure and the data flow between the components in the system.

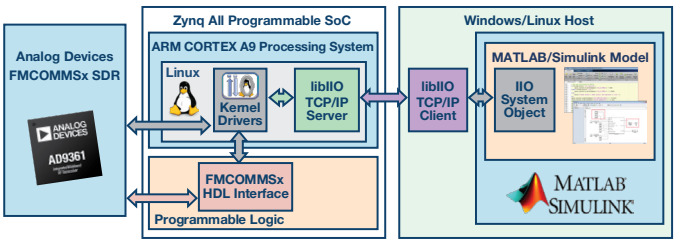


Figure 1. Software infrastructure block diagram.

The IIO System Object is based on the MathWorks System Objects specification[4] and exposes data and control interfaces through which the MATLAB/Simulink models communicate to IIO-based platforms. These interfaces are specified in a configuration file that links the System Object interface to IIO data channels or to IIO attributes. This makes the implementation of the IIO System Object generic, allowing it to work with any IIO platform just by modifying the configuration file. Some of the platforms for which configuration files and examples are available on the ADI GitHub repository[5] include the AD-FMCOMMS2-EBZ/AD-FMCOMMS3-EBZ/AD-FMCOMMS4-EBZ/AD-FMCOMMS5-EBZ SDR boards and the high speed data acquisition AD-FMCDAQ2-EBZ board. The communication between the IIO System Object and the target is accomplished through the libiio server/client infrastructure. The server runs on an embedded target under Linux and manages real-time data exchange between the target and both local and remote clients. The libiio library abstracts the low level details of the hardware and provides a simple yet complete programming interface that can be used for advanced projects with a variety of language bindings (C, C++, C#, Python).

The next sections of the article provide real life examples on how the IIO System Object can be used for validating the ADS-B MATLAB and Simulink models. An AD-FMCOMMS3-EBZ SDR platform[6] connected to a ZedBoard[7] running the Analog Devices Linux distribution were used as the SDR hardware system for verifying the operation of the ADS-B signals detection and decoding algorithm, as shown in Figure 2.

Figure 2. Hardware setup for ADS-B algorithm validation.

MATLAB ADS-B Algorithm Validation Using the IIO System Object

To validate the MATLAB ADS-B decoding algorithm operation with real-time data acquired from the AD-FMCOMMS3-EBZ SDR platform, a MATLAB script has been developed to perform the following operations:

- Calculate the earth zone according to user input
- Create and configure the IIO System Object
- Configure the AD-FMCOMMS3-EBZ analog front end and digital blocks through the IIO System Object
- Receive data frames from the SDR platform using the IIO System Object
- Detect and decode the ADS-B data
- Display the decoded ADS-B information

After an IIO System Object is constructed it must be configured with the IP address of the SDR system, the target device name and input/output channels sizes and numbers. Figure 3 presents an example on how to create and configure the MATLAB IIO System Object.

```
% System Object Configuration
s = iio_sys_obj_matlab; % MATLAB libiio Constructor
s.ip_address = ip;
s.dev_name = 'ad9361';
s.in_ch_no = 4;
s.out_ch_no = 4;
s.in_ch_size = n;
s.out_ch_size = n;

s = s.setupImpl();
```

Figure 3. MATLAB IIO System Object creation and configuration.

The IIO System Object is then used to set the attributes of AD9361 and to receive the ADS-B signals. The attributes of AD9361 is set up based upon the following considerations:

```
% Set the attributes of AD9361
if strcmp(source,'pre-captured')
    input_content{s.getInChannel('RX_LO_FREQ')} = 6e9;
elseif strcmp(source,'live')
    input_content{s.getInChannel('RX_LO_FREQ')} = 1.09e9;
else
    error('Please select a data source: pre-captured or live.');
end
input_content{s.getInChannel('RX_SAMPLING_FREQ')} = 12.5e6;
input_content{s.getInChannel('RX_RF_BANDWIDTH')} = 4e6;
input_content{s.getInChannel('RX1_GAIN_MODE')} = 'fast_attack';
input_content{s.getInChannel('RX1_GAIN')} = 0;
input_content{s.getInChannel('RX2_GAIN_MODE')} = 'fast_attack';
input_content{s.getInChannel('RX2_GAIN')} = 0;
input_content{s.getInChannel('TX_LO_FREQ')} = 6e9;
input_content{s.getInChannel('TX_SAMPLING_FREQ')} = 12.5e6;
input_content{s.getInChannel('TX_RF_BANDWIDTH')} = 4e6;
```

Figure 4. MATLAB libiio sets the attributes of AD9361.

The sampling rate is quite straightforward with the AD9361-based platforms. The transmit data rate normally equals the RX data rate, and ultimately depends on the baseband algorithm. In this example, since the decoding algorithm is designed to work with the sampling rate of 12.5 MSPS, the data rate of AD9361 is set accordingly. By doing this, the received samples can be applied directly to the decoding algorithm, without any additional decimation or interpolation operations.

The RF bandwidth control sets the AD9361's RX analog baseband low-pass filter's bandwidth to provide antialiasing and out-of-band signal rejection. In order to successfully demodulate the received signals, the system must maximize the signal-to-noise ratio (SNR). In order to do this, the RF bandwidth needs to be set as narrow as possible while meeting flatness and the out-of-band rejection specification to minimize in-band noise and spurious signal levels. If the RF bandwidth is set wider than it needs to be, the ADC's linear dynamic range will be reduced due to the extra noise. Similarly, ADC's spurious-free dynamic range will be reduced due to the lower out-of-band signal rejection resulting in overall receiver dynamic range reduction. Therefore, setting the RF bandwidth at an optimal value is critical to receive desired in-band signals and reject out-of-band signals. By observing the spectrum of received signals, we find 4 MHz is a proper value for the RF bandwidth.

Besides setting up the analog filters of AD9361 via RF bandwidth attribute, we can also improve the decoding performance by enabling the digital FIR filters on AD9361 via the IIO System Object, as shown in Figure 5. According to the spectrum characteristics of the ADS-B signal, we design an FIR filter with data rate of 12.5 MSPS, pass band frequency of 3.25 MHz and stop band frequency of 4 MHz. In this way, we can further focus on the bandwidth of interest.

```
s.writeFirData('adsb.ftr');
```

Figure 5. Enable the proper FIR filter on AD9361 via libiio.

Adsb.ftr is a file containing the coefficients of an FIR filter designed using the Analog Devices AD9361 Filter Wizard MATLAB application.[8] This tool provides not only a general-purpose low-pass filter design, but it also provides magnitude and phase equalization for other stages in the signal path.

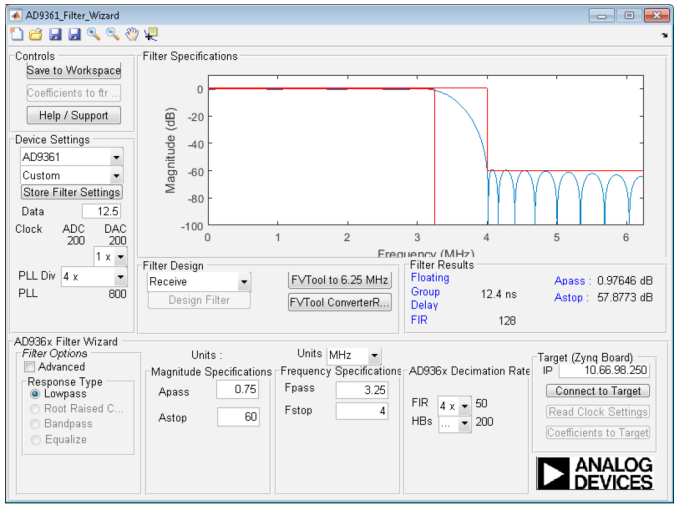


Figure 6. FIR filter designed for ADS-B signals using the MATLAB AD9361 Filter Wizard.

The versatile and highly configurable AD9361 transceiver has several gain control modes that enable its use in a variety of applications. The Gain Mode parameter of the IIO System Object selects one of the available modes: manual, slow_attack, hybrid, and fast_attack. The most frequently used modes are manual, slow_attack, and fast_attack. Manual gain

control mode allows the baseband processor (BBP) to control the gain. Slow_attack mode is intended for slowly changing signals, while fast_attack mode is intended for waveforms that "burst" on and off. Gain mode highly depends on the strength of received signals. If the signal is too strong or too weak, it is suggested to use manual mode or slow_attack. Otherwise, fast_attack is a good option. In the case of ADS-B the fast_attack gain mode provides the best results due to the bursty nature of these signals. Fast_attack mode is a requirement for this waveform since there is preamble, and the AGC needs to react fast enough so that the first bit is captured. There is a difference between attack time—the time it takes to ramp down gain—and decay time—how long it takes to increase gain—in the absence of a signal. The goal is to quickly turn down the gain, so that a valid "1" can be seen on the first bit, but not increase the gain between bit times.

In the end, depending on how you set up the TX_LO_FREQ and RX_LO_FREQ, there are two ways of using this model: using precaptured data (RF loopback) and using live data off the air.

Precaptured Data

In this case, we are transmitting and receiving some precaptured ADS-B signals using AD-FMCOMMS3-EBZ . These signals are saved in a variable called "newModeS."

```
input_content(1) = (2^13).*newModeS./sqrt(2);
input_content(2) = (2^13).*newModeS./sqrt(2);
input_content(3) = (2^13).*newModeS./sqrt(2);
input_content(4) = (2^13).*newModeS./sqrt(2);
```

Figure 7. Define input using precaptured ADS-B signals.

The requirement for this case is to make TX_LO_FREQ = RX_LO_FREQ, and it can be any LO frequency value that AD-FMCOMMS3-EBZ supports. Due to the nature of precaptured data, there is plenty of ADS-B valid data in there, so it is a good way to verify whether the hardware setup is appropriate.

Live Data

In this case, we are receiving the real-time ADS-B signals over the air, instead of the signals transmitted by AD-FMCOMMS3-EBZ. According to ADS-B specification, it is transmitted at the center frequency of 1090 MHz, so the requirements for this case is:

- RX_LO_FREQ=1090 MHz, TX_LO_FREQ far away from 1090 MHz in order to avoid interference.
- Use a proper antenna on the receiver side, which is capable of covering the 1090 MHz band, such as an ADS-B Double 1/2 Wave Mobile Antenna[9]; using a poorly tuned or poorly made antenna will result in a lack of range for your air radar.

With everything set up properly, in order to run the MATLAB model, simply use the following command:

```
[rssi1,rssi2]=ad9361_ModeS('ip','data source',channel);
```

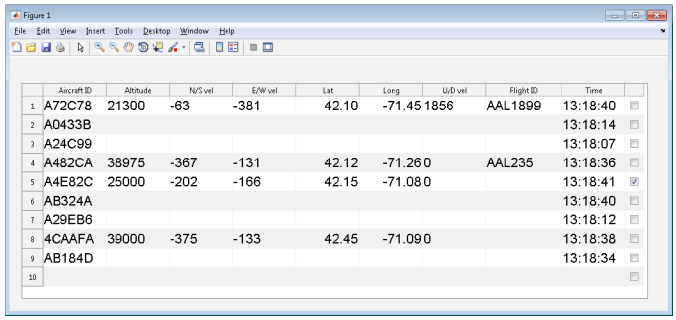
where ***ip*** is the IP address of the FPGA board, and ***data source*** specifies the data source of the received signal. Currently, this model supports data sources of 'precaptured' and 'live'.

Channel specifies whether signals are received using Channel 1 or Channel 2 of the AD-FMCOMMS3-EBZ.

For example, the following command receives the precaptured data on Channel 2:

```
[rssi1,rssi2]=ad9361_ModeS('192.168.10.2','pre-captured',2);
```

At the end of the simulation, you will get the RSSI values on both channels, as well as the result tables shown below:



	Aircraft ID	Altitude	N/S vel	E/W vel	Lat	Long	U/D vel	Flight ID	Time
1	A72C78	21300	-63	-381	42.10	-71.45	1856	AAL1899	13:18:40
2	A0433B								13:18:14
3	A24C99								13:18:07
4	A482CA	38975	-367	-131	42.12	-71.26	0	AAL235	13:18:36
5	A4E82C	25000	-202	-166	42.15	-71.08	0		13:18:41
6	AB324A								13:18:40
7	A29EB6								13:18:12
8	4CAAFA	39000	-375	-133	42.45	-71.09	0		13:18:38
9	AB184D								13:18:34
10									

Figure 8. Result table shown at the end of the simulation.

This result table shows the information of aircrafts appearing during the simulation. With a proper antenna, this model is able to capture and decode the aircraft signals in an 80 mile range with AD-FMCOMMS3-EBZ. Since there are two types of Mode S messages (56 μs or 112 μs), some messages contain more information than the other.

When trying out this model with the real-world ADS-B signals, the signal strength is very important for successful decoding, so make sure to put the antenna in a good line of sight location with the aircraft. The received signal strength can be seen by looking at the RSSI values on both channels. For example, if receiving the signals on Channel 2, the RSSI of Channel 2 should be significantly higher than that of Channel 1. You can tell whether there is any useful data by looking at the spectrum analyzer.

RF Signal Quality

For any RF signal, there needs to be a quality metric. For example, for signals like QPSK, we have error vector magnitude (EVM). For ADS-B signals, it isn't enough to look at the output of a slicer for correct messages, as shown in Figure 8. We need a metric to define the quality of ADS-B/pulse position modulation, so that we can tell whether one setting is better than the other.

In ModeS_BitDecode4.m function, there is a variable *diffVals*, which can be used as such a metric. This variable is a 112 × 1 vector. It shows for each decoded bit in one Mode S message, how far is it away from the threshold. In other words, how much margin each decoded bit has with respect to a correct decision. It is obvious the more margin a bit has, the more confident the decoded result is. On the other hand, if the margin is low, it means the decision is in the border area, so it is very likely that the decoded bit is wrong.

The following two figures compare the *diffVals* values obtained from the ADS-B receivers with and without the FIR filter. By looking at the y-axis, we find with the FIR filter, *diffVals* is larger regardless of whether it is at the highest point, lowest point, or average. However, when there is no FIR filter, the *diffVals* of several bits are very close to 0, which means the decoded results could be wrong. Therefore, we are able to verify that using a proper FIR filter improves the signal quality for decoding.

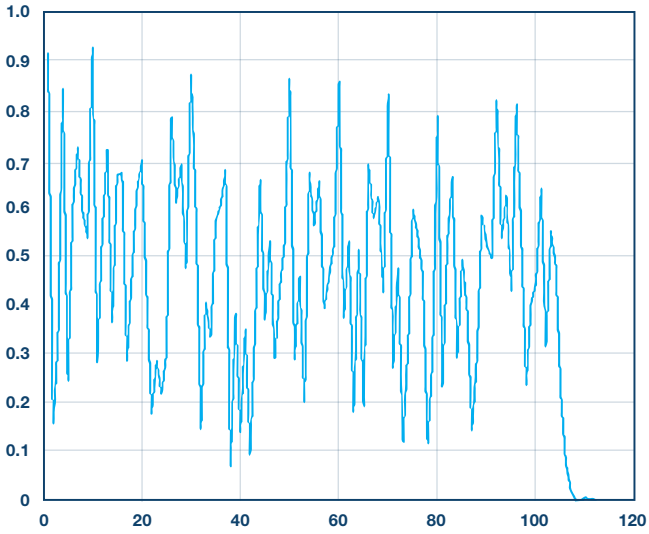


Figure 9. diffVals values obtained from the ADS-B receiver with FIR filter.

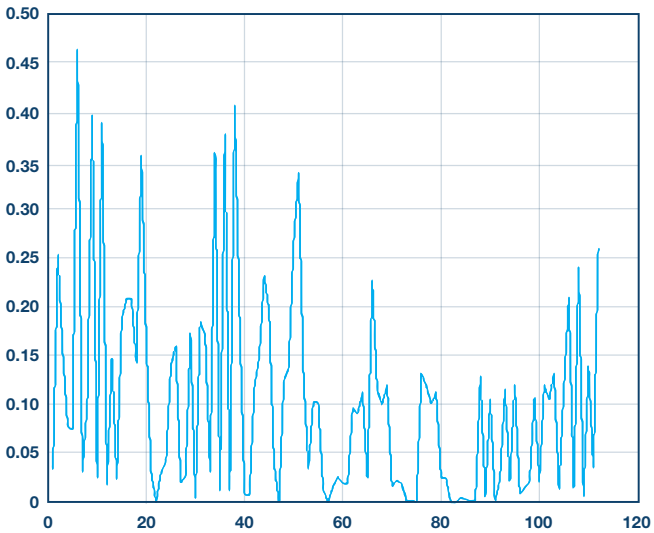


Figure 10. diffVals values obtained from the ADS-B receiver without FIR filter.

The MATLAB ADS-B algorithm using the IIO System Object can be downloaded from the ADI GitHub repository.[10]

Simulink ADS-B Algorithm Validation Using the IIO System Object

The Simulink model is based upon the model introduced in Part 2 of the article series.[2] The detector and decoding piece comes directly from that model, and we add the Simulink IIO System Object to conduct the signal reception and hardware in the loop simulation.

The original model works with sample time = 1 and frame size = 1. However, the Simulink IIO System Object works in a buffer mode—it accumulates a number of samples and then processes them. In order to make the original model work with the System Object, we added two blocks between them: unbuffer to make frame size = 1 and rate transition to make sample time = 1. By doing this, we can keep the original model intact.

The Simulink IIO System Object is set up as following. Similar to the MATLAB one, it creates a System Object, and then defines the IP address, device name, and input/output channels number and sizes related to this System Object.

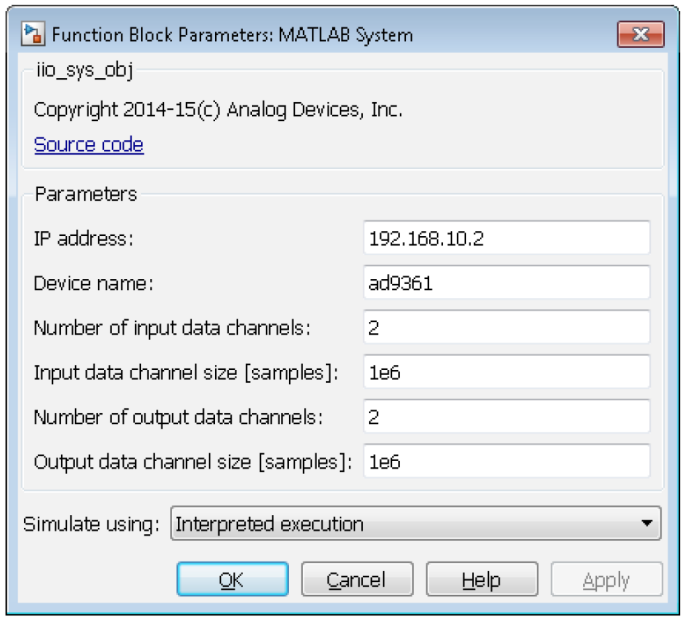


Figure 12. Simulink IIO System Object.

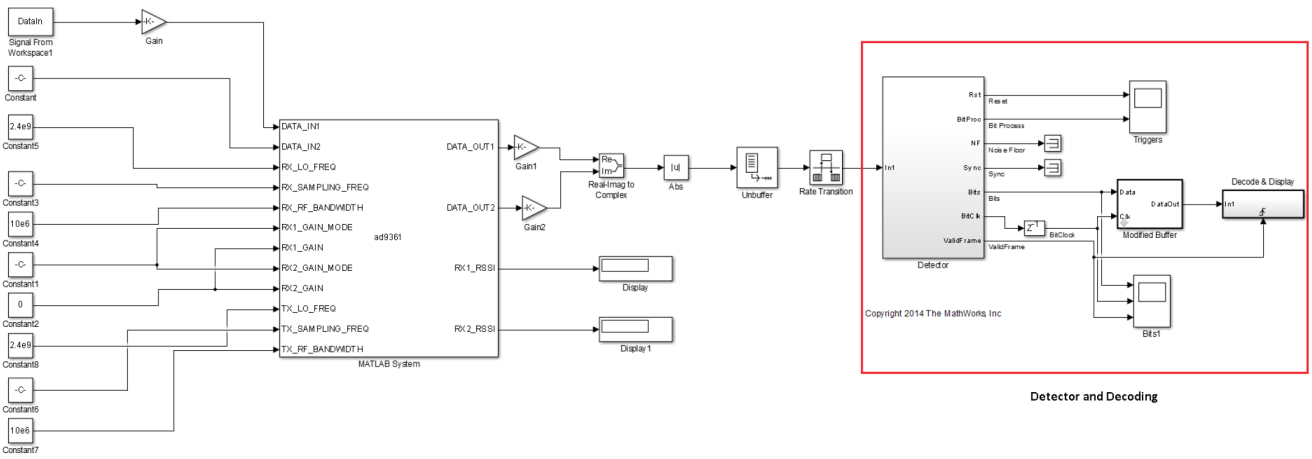


Figure 11. Simulink model to capture and decode ADS-B signals.

The input and output ports of this Simulink block corresponding to an IIO System Object are defined through the properties dialog of the object's block as well as through a configuration file that is specific to the targeted ADI SDR platform. The input and output ports are categorized as data and control ports. The data ports are used to receive/transmit buffers of continuous data from/to the target system in a frame-based processing mode, while the control ports are used to configure and monitor different target system parameters. The number and size of the data ports are configured from the block's configuration dialog while the control ports are defined in the configuration file. The attributes of AD9361 are set up according to the same factors as introduced in MATLAB model. All the theories and methods employed in the MATLAB model can be applied here.

Depending on how you set up the TX_LO_FREQ and RX_LO_FREQ, this Simulink model can be run in two modes: using precaptured data "DataIn" and using live data. Taking the precaptured data, for example, at the end of the simulation, we can see the following results in command window.

```
Aircraft ID 400927      Long Message CRC: 8D40092760C38037389C0EF0029C
Aircraft ID 400927 is at altitude 39000
Aircraft ID 400927 is at latitude 42 19 24.8, longitude -71 8 33.3
Aircraft ID 400927      Long Message CRC: 8D4009279944E7B320048CDB40FA
Aircraft ID 400927 is traveling at 468.363107 knots
Direction West at 230.000000 knots, direction South at 408.000000 knots
Aircraft ID 400927 is going Up at 0.000000 feet/min
```

Figure 13. Results in command window at the end of simulation using precaptured data.

Instead of the result table shown in the MATLAB model, the results here are displayed in the text format.

The Simulink ADS-B model using the IIO System Object can be downloaded from the ADI GitHub repository.[11]

Conclusion

This article talked about hardware in the loop simulation using the libiio infrastructure provided by Analog Devices. Using this infrastructure, the MATLAB and Simulink algorithms for ADS-B signals detection and decoding can be validated with the real-world signals and real hardware. Since the attribute setting is very application and waveform dependent, what works for one waveform will not work for a different one. This is a critical step to ensure that the analog front end and the digital blocks of the SDR system are properly tuned for the algorithm and waveform of interest and that the algorithm is robust enough and works as expected with real life data acquired in varying environmental conditions. Having a verified algorithm, it is now time to move to the next step, which consists of translating the algorithm to HDL and C code using the automatic code generation tools from MathWorks and integrating this code into the programmable logic and software of the actual SDR system. The next part of the article series will show how to generate code and deploy it in the production hardware and will talk about the results obtained by operating the platform with real-world ADS-B signals at an airport. This will complete the steps required to take an SDR system from prototyping all the way to production.

References

[1] Cozma, Andrei, Di Pu, and Tom Hill. "Four Quick Steps to Production: Using Model-Based Design for Software-Defined Radio—Part 1." *Analog Dialogue*, Volume 49, Number 3, 2015.

[2] Donovan, Mike, Andrei Cozma, and Di Pu. "Four Quick Steps to Production: Using Model-Based Design for Software-Defined Radio—Part 2." *Analog Dialogue*, Volume 49, Number 4, 2015.

[3] Analog Devices. "IIO System Object."

[4] MathWorks. "What Are System Objects?"

[5] Analog Devices, "Mathworks_tools." GitHub repository.

[6] Analog Devices. AD-FMCOMMS3-EBZ User Guide.

[7] ZedBoard.

[8] Analog Devices. MATLAB AD9361 Filter Design Wizard.

[9] ADS-B Double 1/2 Wave Mobile Antenna.

[10] MATLAB ADS-B Algorithm Using The IIO System Object Source Code.

[11] Simulink ADS-B Model Using The IIO System Object Source Code.

Acknowledgements

The authors would like to thank Mike Donovan from MathWorks, who contributed to the development of the MATLAB and Simulink ADS-B signal detection and decoding algorithms used in this article.

Di Pu [di.pu@analog.com] is a system modeling applications engineer for ADI, supporting the design and development of software-defined radio platforms and systems. She has been working closely with MathWorks to solve mutual end customer challenges. Prior to joining ADI, she received her B.S. degree from Najing University of Science and Technology (NJUST), Nanjing, China, in 2007 and her M.S. and Ph.D. degrees from Worcester Polytechnic Institute (WPI), Worcester, MA, U.S.A., in 2009 and 2013—all in electrical engineering. She is a winner of the 2013 Sigma Xi Research Award for Doctoral Dissertation at WPI.

Di Pu

Andrei Cozma [andrei.cozma@analog.com] is an engineering manager for ADI, supporting the design and development of system level reference designs. He holds a B.S. degree in industrial automation and informatics and a Ph.D. in electronics and telecommunications. He has been involved in the design and development of projects from different industry fields such as motor control, industrial automation, software-defined radio, and telecommunications.

Andrei Cozma

Also by this Author:

FPGA-Based Systems Increase Motor-Control Performance

Volume 49, Number 1