APPLICATION NOTE

Wireless Network Hardware Implementation with SDR

MathWorks®

Wireless system engineers often want to verify and optimize their system architecture, baseband signal processing algorithm, system control protocols for software-defined radio (SDR), and FPGA prototype long before the hardware design team gets the first silicon prototype. In some cases, the SDR/FPGA prototype is the final product.

This application note describes how to convert Simulink® system models to SDR/FPGA implementations. We use a relatively complex Simulink model [1] to create an SDR wireless network prototype and measure the real network throughputs with over-the-air testing.

The prototype uses Simulink, Communications Toolbox™, Stateflow®, DSP System Toolbox™, Fixed-Point Designer™, and HDL Coder™ (all from Release 2017b or later). The prototype also uses Xilinx® Vivado® Design Suite.

You can follow along by *downloading Simulink files for this example.*

System Modeling and Prototyping

You can use the Simulink model as a basic framework for designing a wireless transceiver and modeling a wireless network. The wireless transceiver consists of a physical (PHY) layer and a medium access control (MAC) layer. The QPSK modulation is used in the PHY and the Carrier Sense Multiple Access and Collision Avoidance (CSMA/CA) function in the MAC. The wireless network model is composed of multiple transceiver nodes communicating over fading channels.

We use Fixed-Point Designer and HDL Coder to conduct a quick implementation of the SDR transceiver prototype from the transceiver model. The prototype will be implemented in the FPGA part of the SDR board. The baseband Tx and Rx complex I/Q signals are directly interconnected with the AD9361 radio front end in the SDR board. We will implement the two-node network as shown in Figure 1.

Starting with the given transceiver model, we need to make the following changes to accommodate the requirement of HDL Coder and the data format of the AD9361 SDR interface:

1. As shown in the model, several Stateflow charts have been used for MAC, LLC, and other bit manipulation blocks. To generate the HDL code using HDL Coder, all the variables in the Stateflow charts must be fixed point. We have converted the variables in all Stateflow charts to integers or Booleans by inspection. This conversion is straightforward because all operations used in the Stateflow charts are either logical or integer operations. All variable data types in the Stateflow charts should be locked against future changes by Fixed-Point Designer.
2. The random generator function **`randi`** used in the MAC and LLC charts of the previous mode [1] is not supported by HDL Coder. So, it is replaced by the PN Sequence Generator block to feed random numbers as the inputs to the MAC and LLC charts.
3. The receiver 12-bit ADC sample output from the AD9361 is a 16-bit fixed-point complex number with the 12 effective bits placed at the last 12 LSBs. In fractional decimal, this presents the input range of (-0.0625, 0.0625) [2]. A scaling block is added in the receiver model to reduce the underflow of the input. Similarly, the 12-bit DAC input of the transmitter is a 16-bit complex input with the 12 effective bits located at the first 12 MSBs, which sets the Tx signal range to (-1, 1). The baseband Tx signal needs to be scaled properly in this range.
4. To save the FPGA resource in the implementation and properly filter the Rx and Tx signals, the digital filters in the AD9361 are used to replace the Tx/Rx match filters in the previous model [1].

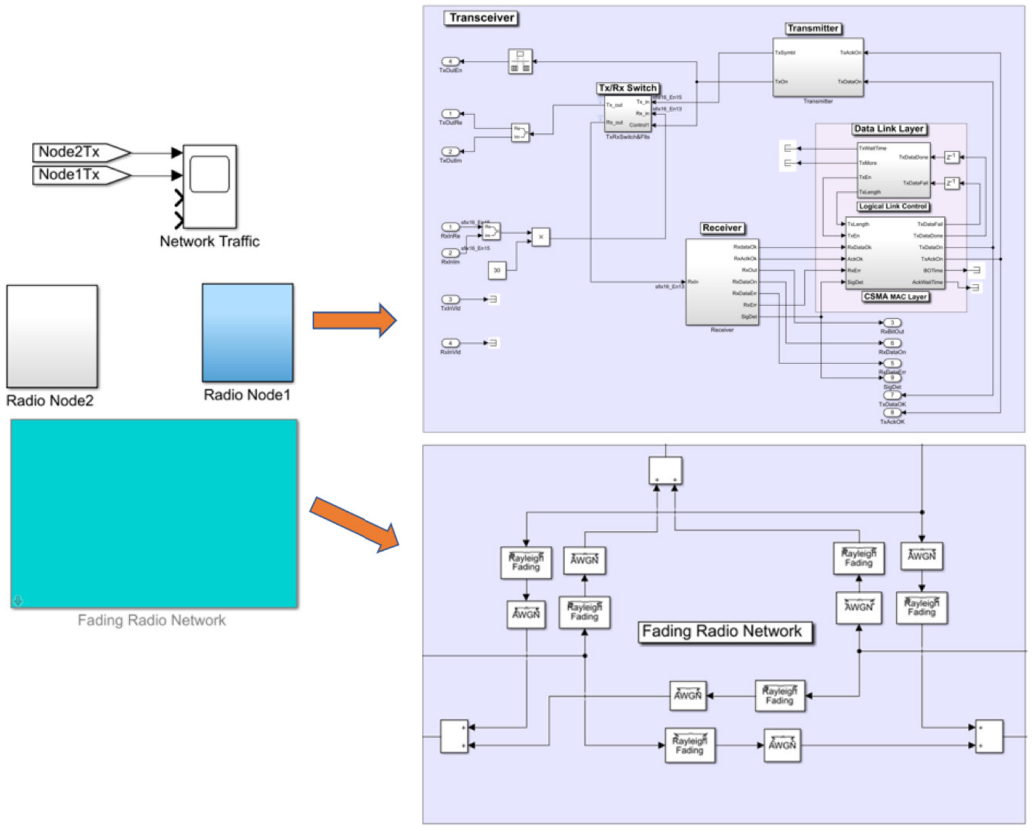


Figure 1. Wireless network and transceiver model for prototyping.

The modified Simulink model file is named **`Transceiver _ N1 _ N2 _ double.slx`** and can be found in the download.

Creating a Fixed-Point Variable Model

To implement the transceiver model, all double variables in the Simulink model need to be converted to fixed-point variables. We use Fixed-Point Designer for the conversion. The difficult part of the conversion is the blocks with feedback loops, such as the ones in the decision feedback equalizer. This conversion process will be iterated for optimal results over the various testing scenarios. We take the outputs of the Tx/Rx switch as the conversion reference points for transmission and the outputs of the equalizer as the points for receiving, as in Figure 2.

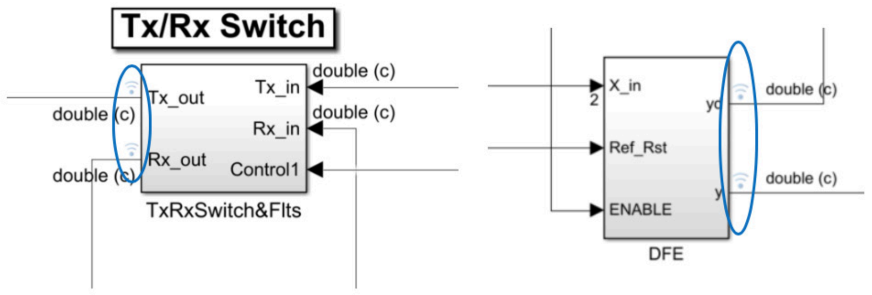


Figure 2. The signals logged for comparison of the fixed-point conversion.

After the conversion, we can use Fixed-Point Designer to visualize the conversion loss. Figure 3 shows that after the conversion, the EVM displayed in the constellation diagram is still satisfactory. As shown in Figure 4, the amplitude conversion error is less than 0.005 with the signal amplitude of 1. Then, the calculated SNR after the conversion is approximately 50 dB, which is ignorable to channel noise and transceiver impairments. This fixed-point model conversion process can be repeated for other selected network conditions for further optimization.

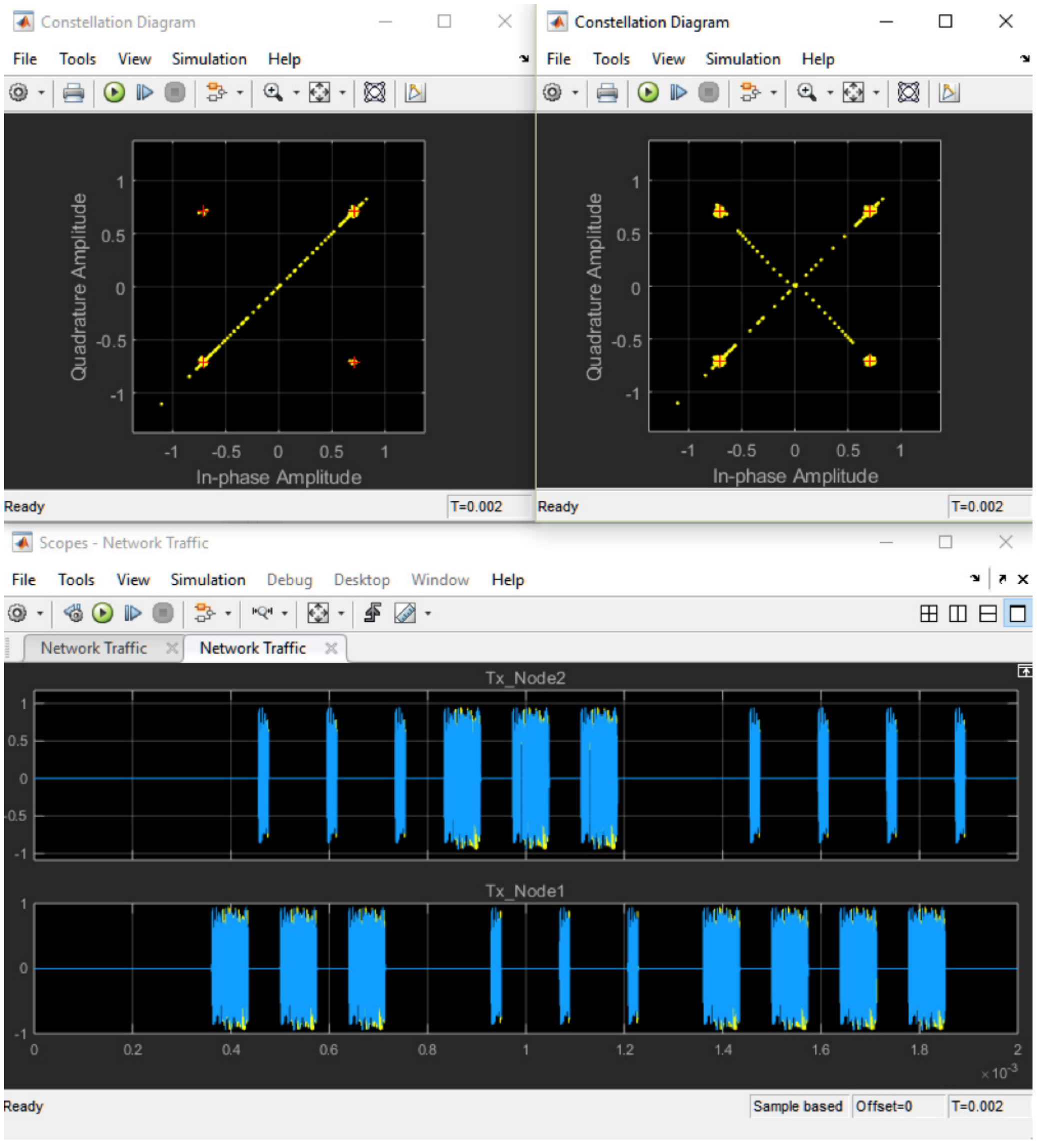
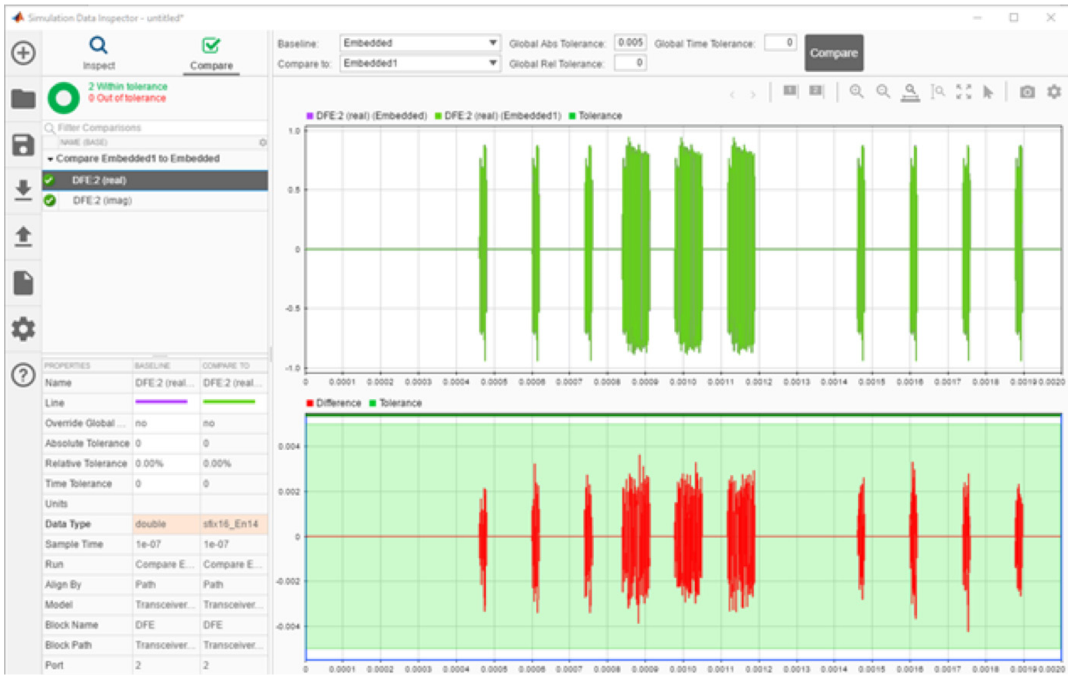


Figure 3. Simulation results with the fixed-point variables.



Baseline: Embedded
Compare to: Embedded1
Global Abs Tolerance: 0.005
Global Rel Tolerance: 0
Global Time Tolerance: 0

PROPERTIES	BASELINE	COMPARE TO
Name	DFE:2 (real...	DFE:2 (real...
Line	🟪	🟩
Override Global ...	no	no
Absolute Tolerance	0	0
Relative Tolerance	0.00%	0.00%
Time Tolerance	0	0
Units		
Data Type	double	sfix16_En14
Sample Time	1e-07	1e-07
Run	Compare E...	Compare E...
Align By	Path	Path
Model	Transceiver...	Transceiver...
Block Name	DFE	DFE
Block Path	Transceiver...	Transceiver...
Port	2	2

Figure 4. Comparison of the fixed-point and double variable models.

The converted and optimized fixed-point Simulink transceiver model is stored in the file **`Transceiver _ N1 _ N2 _ fpt.slx`**.

Generating HDL Code and the FPGA Bitstream

HDL Coder converts the Simulink model to Verilog® or VHDL® RTL code, if the blocks in the Simulink model are supported in HDL and all the variables are fixed point. The HDL Workflow Advisor eases the conversion process. The transceiver is considered to be an HDL IP core. At one end, this IP core interfaces with the AD9361 radio front end through a parallel port. This parallel port is bidirectional. The 12-bit ADC and DAC data of Channel 1 and Channel 2 of the AD9361 radio are received and transmitted through this port. At the other end, the transceiver IP core is connected to the ARM core on the FPGA board and then through the Ethernet port to the host computer. Figure 5 shows the structure of these interfaces.

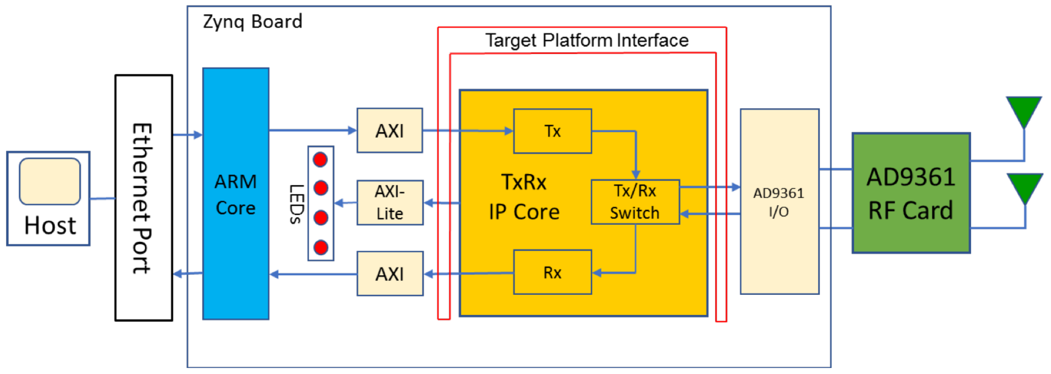


Figure 5. Wireless transceiver hardware structure.

As shown in Table 1, the target platform interface connects the transceiver IP core to the ADI 9361 radio, the ARM AXI and AXI-Lite ports, and the LEDs on the Zynq® board. Table 1 also gives the descriptions of the interface mappings of the transceiver IP core. This mapping table is implemented in Step 1.3 of the HDL Workflow Advisor.

The port names in the table are in the model file **`Transceiver _ N1 _ N2 _ fpt _ HDL1.slx`**.

Port Name	Target Platform Interfaces	From	To	Note
RxInRe	Rx Data I1 In [0:15]	Radio	FPGA	Receiving in-phase ADC data (16 bits) from Radio Channel 1
RxInIm	Rx Data Q1 In [0:15]	Radio	FPGA	Receiving quadrature ADC data (16 bits) from Radio Channel 1
TxOutRe	Tx Data I1 Out [0:15]	FPGA	Radio	Transmitting in-phase DAC data (16 bits) to Radio Channel 1
TxOutIm	Tx Data Q1 Out [0:15]	FPGA	Radio	Transmitting quadrature DAC data (16 bits) to Radio Channel 1
TxInVld	Tx Data Valid In [0]	ARM	FPGA	Data from ARM ready to transmit
RxInVld	Rx Data Valid In [0]	Radio	FPGA	Received ADC data ready
TxOutVld	Tx Data Valid Out [0]	FPGA	Radio	DAC data ready to transmit
RxOutVld	Rx Data Valid Out [0]	FPGA	ARM	Received data ready for ARM
RxOut	AXI-Lite x"108"	FPGA	ARM	Received bitstream output to ARM
RxDataErr	AXI-Lite x"104"	FPGA	ARM	Received data CRC error indicator to ARM
TxDataOK	LED General Purpose [0]	FPGA	LED 0	Data frame transmitting indicator
TxAckOK	LED General Purpose [1]	FPGA	LED 1	ACK frame transmitting indicator
SigDet	LED General Purpose [2]	FPGA	LED 2	Rx signal detection indicator

Table 1. Target platform interface of FPGA to radio and ARM.

By following the steps of the HDL Workflow Advisor, we generate the FPGA bitstream for the transceiver block TxRx_N1. The HDL Workflow Advisor also creates a Zynq radio software interface model, shown in Figure 6. This Simulink interface model represents the transmission block and the receiving block of the AD9361 radio card and the ARM interfaces to the IP core.

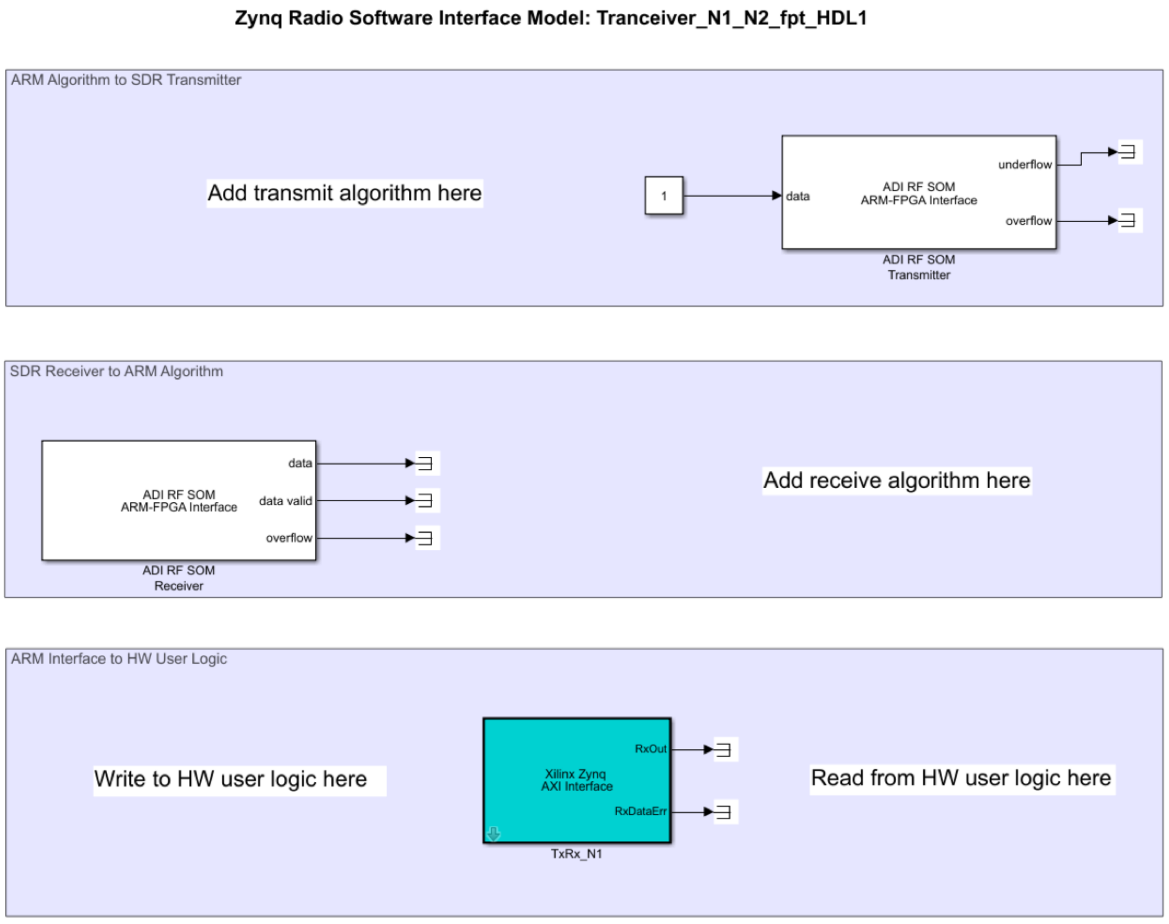

Zynq Radio Software Interface Model: Tranceiver_N1_N2_fpt_HDL1
ARM Algorithm to SDR Transmitter
Add transmit algorithm here
1
data
ADI RF SOM ARM-FPGA Interface
underflow
overflow
ADI RF SOM Transmitter
SDR Receiver to ARM Algorithm
ADI RF SOM ARM-FPGA Interface
data
data valid
overflow
ADI RF SOM Receiver
Add receive algorithm here
ARM Interface to HW User Logic
Write to HW user logic here
Xilinx Zynq AXI Interface
RxOut
RxDataErr
TxRx_N1
Read from HW user logic here

Figure 6. Zynq radio software interface model.

To generate the FPGA bitstream for the TxRx_N2 transceiver block, right click the TxRx_N2 block and select **Treat as Atomic Unit** in the Block Parameters setting window and deselect the same option on TxRx_N1. Then the same operations can be used to generate the FPGA bitstream file for the second SDR transceiver node.

Building and Measuring the Radio Network

The network prototype needs two ADI RF SOM Zynq boards for the transceivers, named radio node 1 and 2, as shown in Figure 7. The observation radio receiver, named radio node 3, can be an ADI RF SOM or ZedBoard™. The setup order of the wireless network prototype is shown in Figure 7.

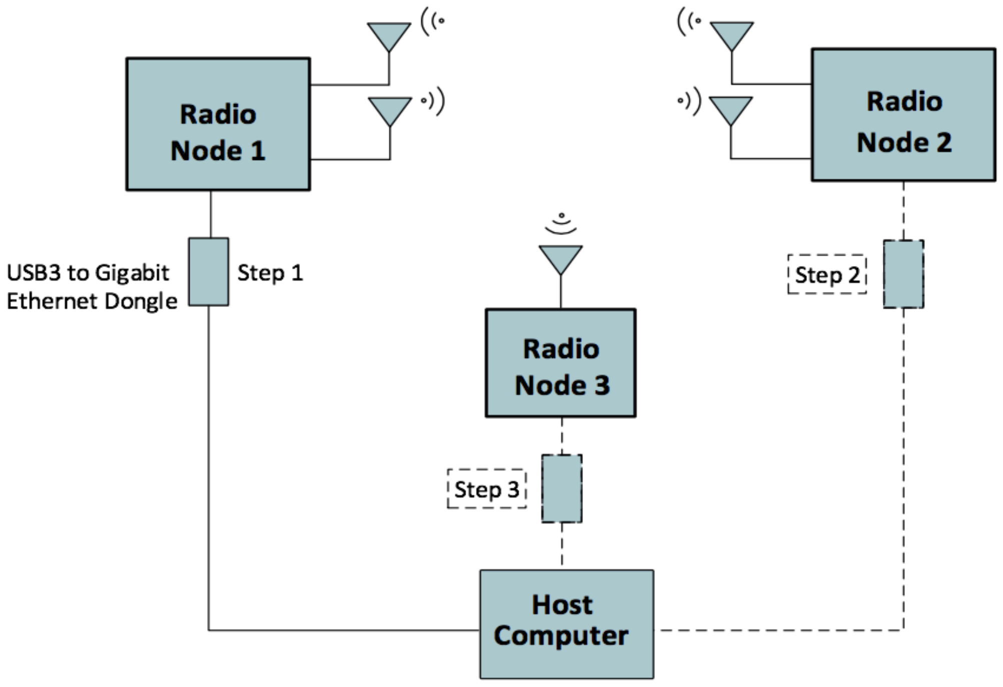


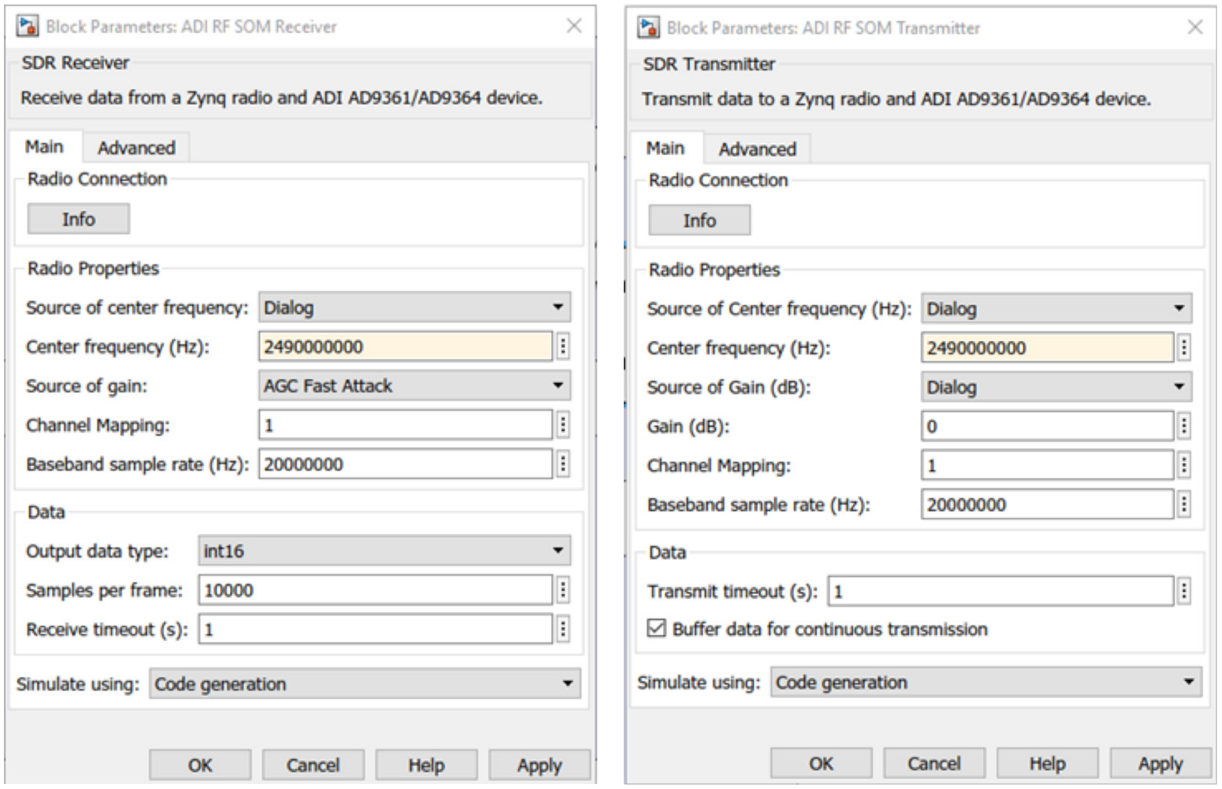
Figure 7. SDR network demo setup.

The hardware support package to the three SDR boards [3] needs to be installed on all three SDR boards beforehand. In this prototype, we use one host computer to operate on the three SDR boards sequentially. The FPGA bitstreams can be loaded into the Zynq transceiver boards respectively by using the following MATLAB commands:

```
>> dev=sdrdev('ADI RF SOM')

>> dev.downloadImage('FPGAImage', 'Bitstreams/TxRx _ Node1.bit')
```

The hardware parameters of the transmitter and the receiver of AD9361 and the Tx and Rx filters are set at the corresponding blocks in the software interface model, as shown in Figure 8. The SDR board can be activated by clicking the "Deployed to Hardware" icon in the toolbar of the interface model.


Block Parameters: ADI RF SOM Receiver

SDR Receiver

Receive data from a Zynq radio and ADI AD9361/AD9364 device.

Radio Properties

Source of center frequency: Dialog

Center frequency (Hz): 2490000000

Source of gain: AGC Fast Attack

Channel Mapping: 1

Baseband sample rate (Hz): 20000000

Data

Output data type: int16

Samples per frame: 10000

Receive timeout (s): 1

Simulate using: Code generation

Block Parameters: ADI RF SOM Transmitter

SDR Transmitter

Transmit data to a Zynq radio and ADI AD9361/AD9364 device.

Radio Properties

Source of Center frequency (Hz): Dialog

Center frequency (Hz): 2490000000

Source of Gain (dB): Dialog

Gain (dB): 0

Channel Mapping: 1

Baseband sample rate (Hz): 20000000

Data

Transmit timeout (s): 1

☑ Buffer data for continuous transmission

Simulate using: Code generation

Figure 8. Parameters for radio receiver and transmitter.

After the two ADI RF SOM boards are activated, the network will start communicating and all four LEDs near the antennas of both boards should light up. We use the third SDR to observe the signal traffic over the air. The setup of the wireless network prototype is shown in Figure 9.

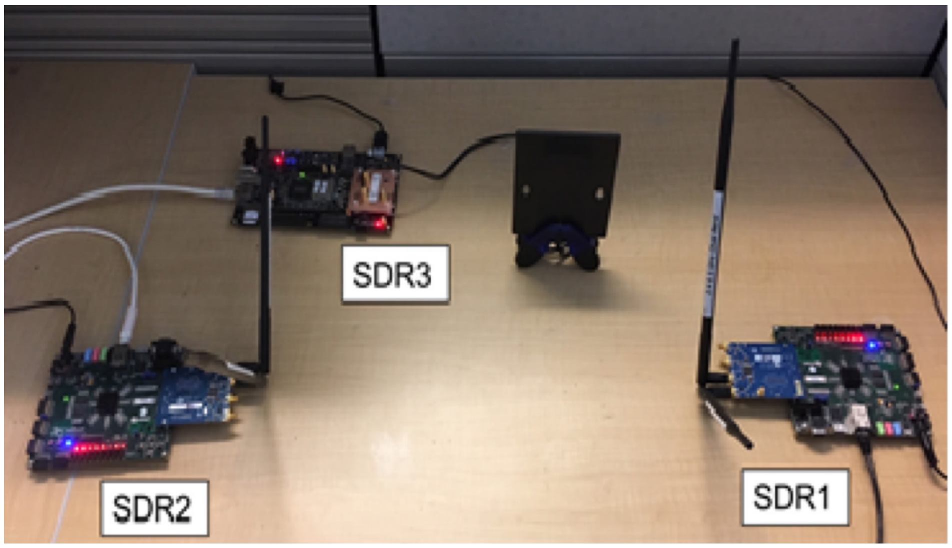


Figure 9. The setup of the wireless network prototype.

To calculate the throughput, we rely on the fact that a successfully transmitted data frame is always followed by an acknowledge frame [1]. In the network setup, we place the observer radio deliberately close to radio node 1 such that the received signal from node 1 has higher amplitude than that from node 2. In this way, the observer node knows which node is transmitting a signal frame. The frame type, data frame, or ACK frame is identified by the frame width. The throughput is calculated for each node by taking off the interframe gaps and the overheads of frame preambles. Here we use a Simulink model with the Stateflow chart, shown in Figure 10, acting on the receiver signal power envelope to calculate the throughputs for each node.

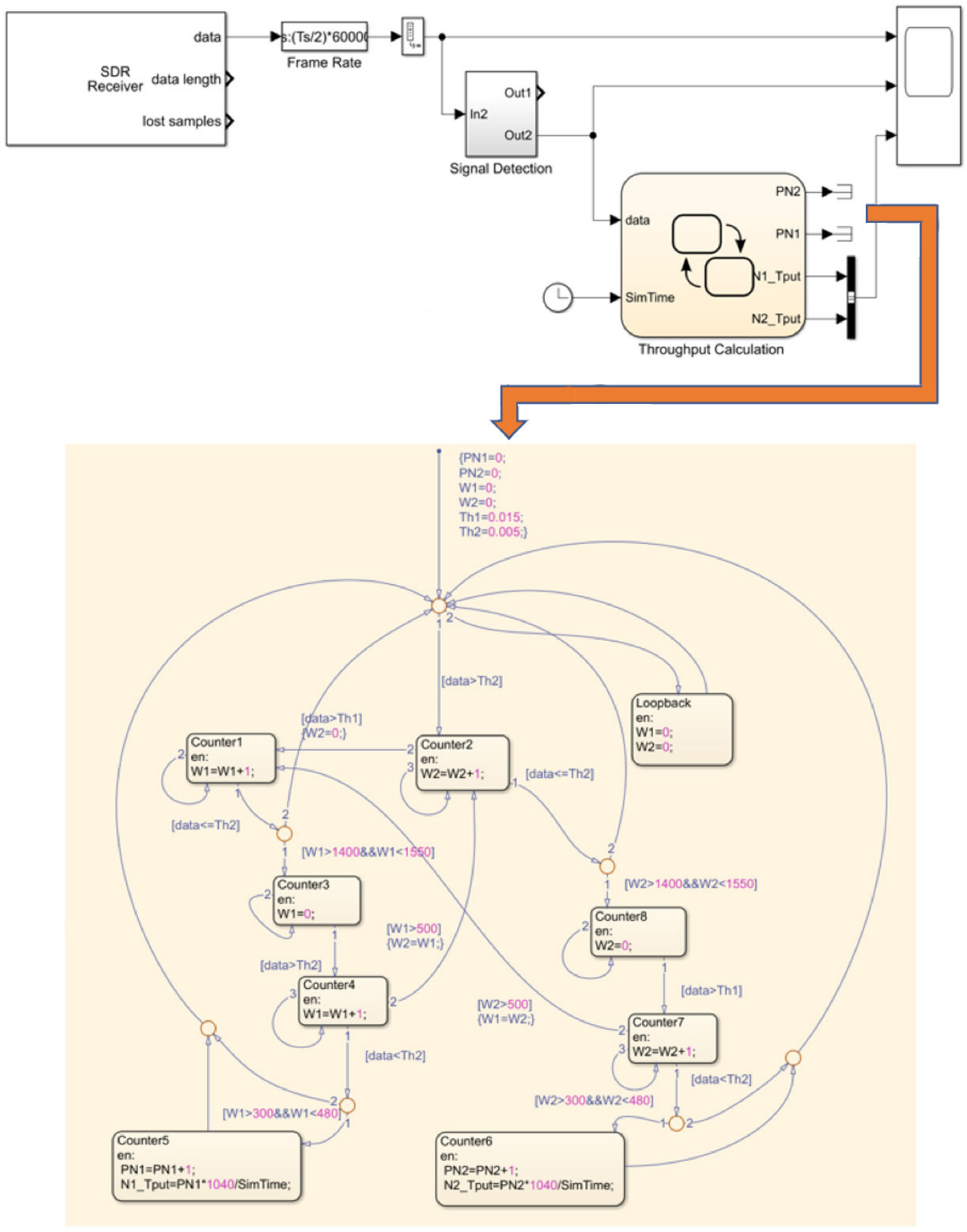


Figure 10. Stateflow chart of calculating node throughput.

Figure 11 shows the real RF signal observed by radio node 3. The upper plot shows the signals transmitted by node 1 and node 2. We see that the data frames (the signals with wider duration) are always followed by the ACK frames (the signal with narrow duration). This indicates that the two-node network works the same as the Simulink model does. The raw symbol rate is 10 Mbps, and the raw data rate is 20 Mbps.

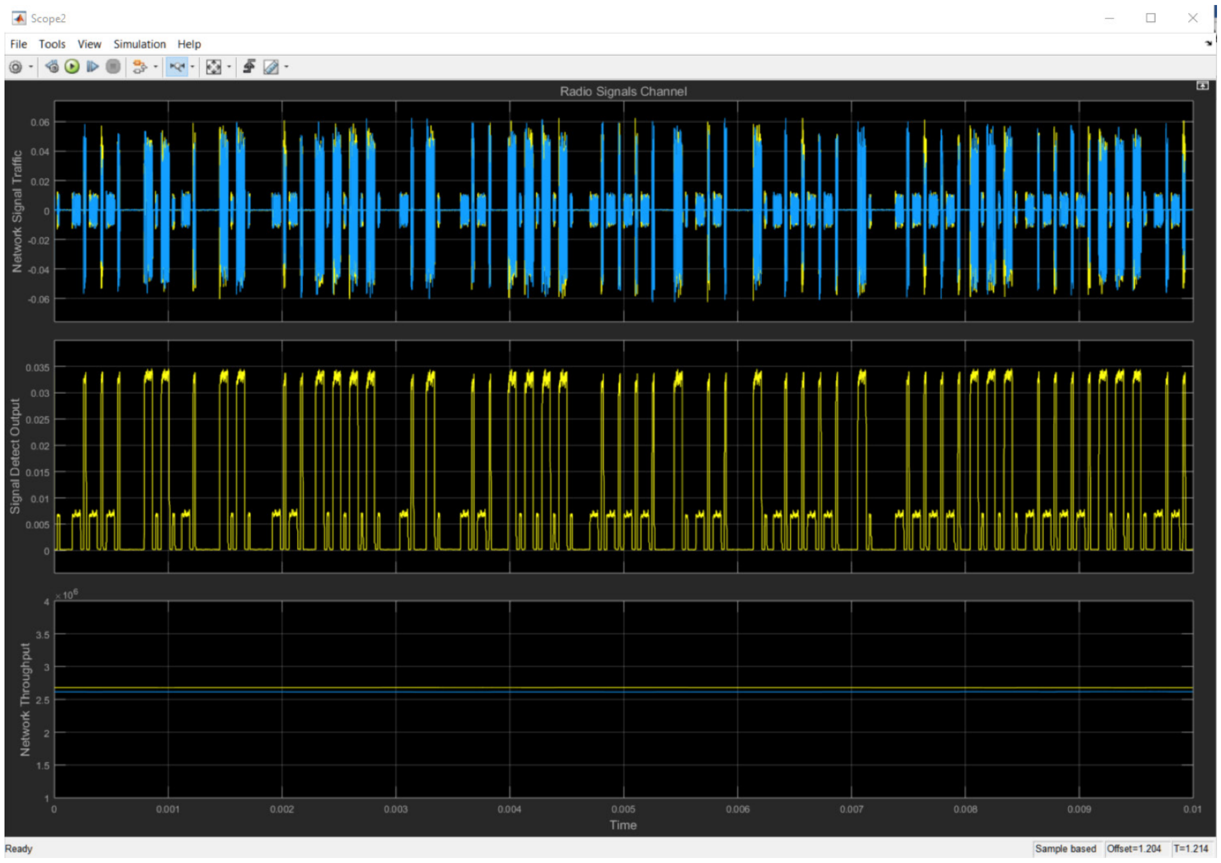


Figure 11. Real RF signals and throughputs monitored by node 3.

The middle plot is the received signal strength. We adjust the antenna location of node 3 such that one signal strength is more than twice the other. As shown in the example, in the Stateflow chart we set the thresholds Th1 and Th2 at 0.005 and 0.015, respectively, to sort out which node the signal comes from.

The bottom plot of Figure 11 shows that the throughputs of both nodes are about 2.6 Mbps, which is close to the throughput 2.7 Mbps calculated in the Simulink model. The small difference of the throughputs between the simulated and the measured data might be caused by the hardware delay ignored in the Simulink model.

Summary

In this application note, we showed the key operations in converting the wireless transceiver Simulink model to the SDR hardware implementation. The hardware implementation (prototype) proves various aspects in system modeling: the system architecture, the modulation and demodulation, the MAC layer, the system timing, the ADC/DAC accuracy, the use of Stateflow chart, the SDR radio settings, and so on. The approach for this SDR wireless network prototype provides a platform and a workflow for rapid prototyping of other wireless networks.

Reference

[1] Chung Wu, "*Wireless Transceiver Design and Network Modeling in Simulink*," Technical Article, MathWorks, December 2017

[2] *AD9361 Interface Description*, Analog Devices Wiki page

[3] *Install Support Package for Xilinx Zynq-Based Radio*, MathWorks Documentation

[4] *Frequency Offset Calibration Using Analog Devices AD9361/AD9364*, MathWorks Documentation