WHITE PAPER

Deploying 5G NR Wireless Communications on FPGAs: A Complete MATLAB and Simulink Workflow

MathWorks®

Introduction

Algorithmic innovation drives progress in wireless communications, enabling advances in personal connectivity, space and satellite communications, high-reliability automated driving systems, and the Internet of Things (IoT). Designing, implementing, and testing these systems requires close collaboration across multiple disciplines.

Deploying algorithmic models to FPGA hardware makes it possible to do over-the-air testing and verification. Automatically generating HDL code directly from the system-level algorithms and models eliminates the need for engineers to rely on specification documents or manually architect and write code.

This paper describes the process of converting MATLAB® algorithms and Simulink® models directly into HDL for FPGAs: Topics include:

- 5G New Radio (NR) standard-compliant algorithmic modeling with MATLAB and 5G Toolbox™
- Transitioning from a frame-based MATLAB algorithm to a streaming Simulink implementation
- Fixed-point implementation using Fixed-Point Designer™ and target hardware knowledge
- Speeding design by using proven intellectual property (IP) blocks
- Generating HDL and deploying on target hardware, in this case a Xilinx® Zynq® device

A 5G NR cell search design is used to illustrate the process. MathWorks originally created this design to meet critical customer requirements. The design has evolved into a reference application that is available with Wireless HDL Toolbox™. This workflow also uses 5G Toolbox and HDL Coder™.

Implementing Wireless Communications Algorithms in Hardware

Communications engineers can use MATLAB and Simulink, along with add-on toolboxes, to develop, simulate, and refine their applications at the algorithm level. But implementing these algorithms for prototyping or production deployment requires coordination across a range of roles, greatly increasing project complexity.

Different Skills and Working Environments

Figure 1 shows the skills required to build a wireless communications application in hardware. Engineers typically have expertise in only one or two of these areas, so projects often involve coordination across roles and departments.

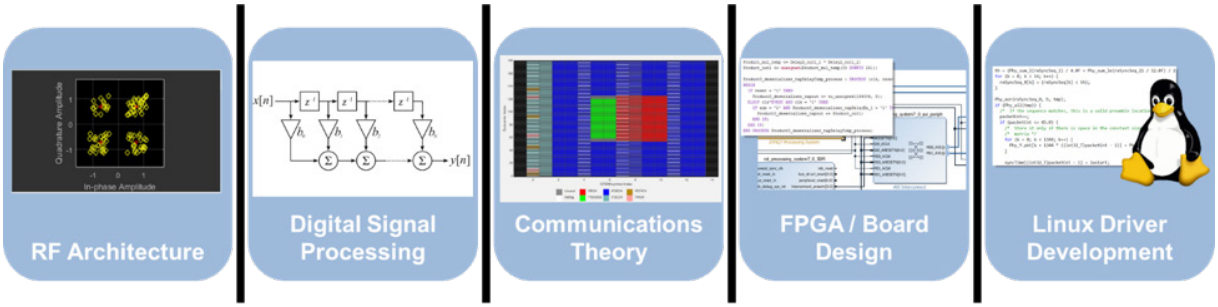

Figure 1. The skills and working environments required to prototype or deploy a wireless communications application on hardware.

Hardware Design Is Different

Programming the hardware on an FPGA or ASIC device is very different from writing software that will be compiled to a set of instructions running on a general-purpose processor. Digital hardware is a prebuilt circuit of fixed resources that operates on a stream of data. While hardware delivers speed, it requires some expertise to structure it. Some of the high-level differences are:

- Streaming data: You can load data of any size or number of dimensions into software. However, hardware is a circuit that streams through 1s and 0s. This means that algorithms must be adapted to work on a stream of data over time, and include some logic to manage the flow of data.
- Parallelism: Hardware circuits can process streams of data in parallel, which speeds up processing, but the timing of parallel paths needs to be coordinated for downstream operations to ensure that the data arrives when expected.
- Fixed-point data types: Floating-point math typically requires more operations and wider data word lengths than fixed point. Because hardware is a fixed set of resources and most teams would prefer to use the smallest chips they can, fixed point is widely used in hardware design. But the process of converting from floating-point algorithms to fixed-point hardware-ready implementations can be challenging even for experienced hardware designers.
- Managing resource usage and clock cycles: Often a first-pass attempt to target an FPGA results in something that either will not fit on the device's resources or runs more slowly than anticipated. Understanding how to optimize for these two orthogonal parameters represents one of the larger learning curves of hardware design.
- Interaction with software and other hardware devices: System-on-chip (SoC) devices combine at least one processor with FPGA hardware fabric on a device. The hardware and software communicate data back and forth through memory locations with prespecified addresses. Accessing these registers, as well as other external memory and chip I/O, requires an understanding of the architecture of the device.

Successful FPGA hardware targeting requires adapting the algorithm to operate efficiently given the parameters listed above. Close collaboration between communications/DSP engineers and their hardware design counterparts delivers the best results. Engineers who are communicating via specification

documents and synthesis reports cannot efficiently implement innovative algorithms on FPGA hardware; it requires close collaboration and a common design exploration environment.

Workflow: Bridging the Algorithm-Hardware Design Divide

The Right Tools for the Job

MATLAB is good for developing mathematical algorithms, manipulating large data sets, exploring mathematics, and visualizing data. But hardware implementation requires timing, parallelism, and custom fixed-point word lengths. These constructs are native to specialized hardware description languages (HDLs) such as VHDL® and Verilog®, but these HDLs have no connection from MATLAB. This requires algorithm designers to describe their intent in a written specification, to be interpreted by designers as they write their detailed HDL and verification test benches.

Simulink is a visual design and simulation environment that is tightly coupled with MATLAB and bridges algorithm design to hardware implementation. With built-in timing, Simulink enables you to design and simulate parallel architectures and to visualize data types as they propagate through the design. Simulink also models embedded soft-ware and analog circuitry, enabling you to simulate your full system at a high level before implementing it in hard-ware. Once you are ready for implementation, you can use HDL Coder and Embedded Coder® to generate synthesizable HDL and embedded C code to deploy on your target device.

For this reason, many teams use Simulink as a collaboration environment between algorithm and hardware design, as shown in Figure 2.

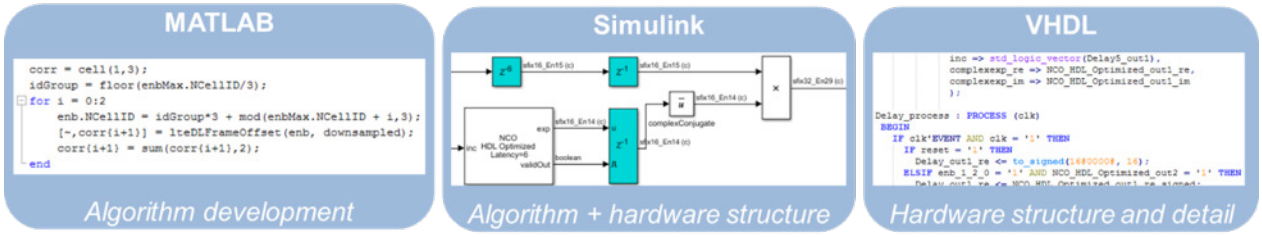

Figure 2. Algorithm and hardware design environments. Algorithm developers use MATLAB; hardware designers use VHDL or Verilog. Simulink bridges this gap, enabling these two roles to collaborate toward a higher-quality implementation.

Design Overview

To illustrate this workflow, we will use a 5G NR wireless design, but the principles discussed apply to any wireless design.

The NR HDL Cell Search design detects and demodulates 5G NR synchronization signal blocks (SSBs). Primary and secondary synchronization signals are used by the user equipment (UE) to obtain the cell identity and frame timing from a gNodeB (gNB) base station.

Figure 3 shows a high-level block diagram of the design's architecture. The Digital Down-Converter (DDC) block corrects frequency offsets before the PSS block returns the strongest primary synchronization signal (PSS). The strongest PSS is used to perform orthogonal frequency division multiplexing (OFDM) demodulation. The final stage searches for the secondary synchronization signal (SSS) within the appropriate resource elements.

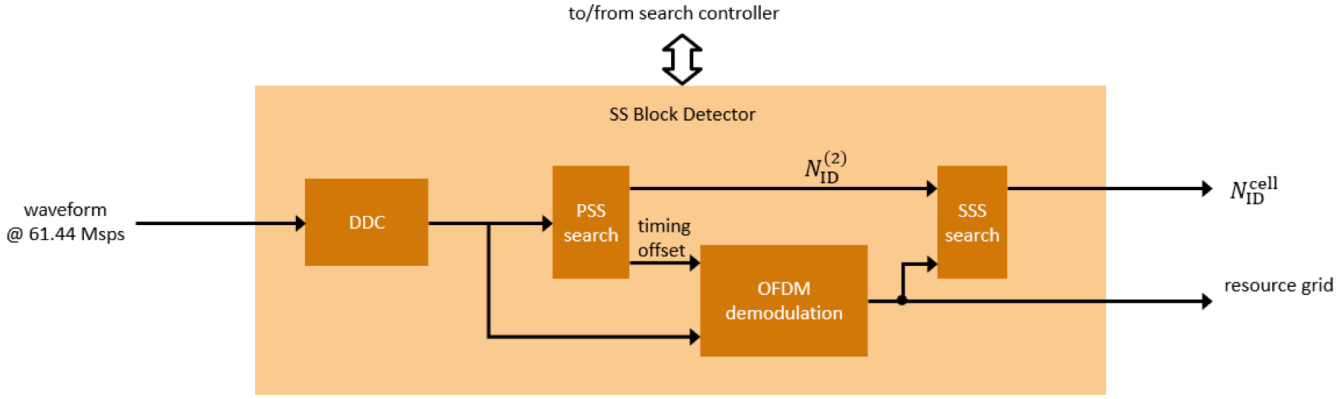

Figure 3. High-level block diagram of the NR HDL Cell Search hardware architecture.

Certain considerations and tradeoffs were made in adapting these algorithms for streaming hardware, some of which are highlighted in the description of the workflow components below.

Workflow Overview

There are many paths from a MATLAB algorithm to an FPGA. The top-down refinement approach shown in Figure 4 helps teams deploy their wireless designs to FPGAs in a deterministic yet agile way.

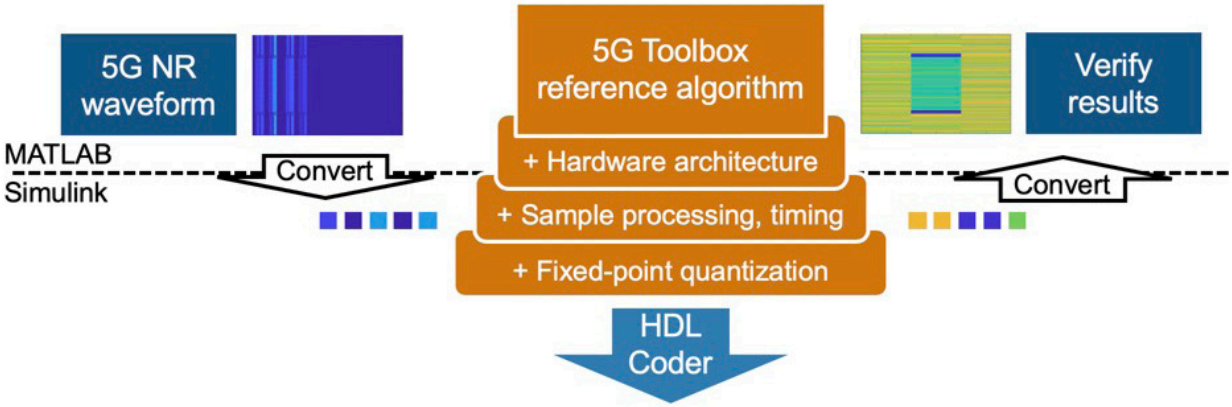

Figure 4. Top-down refinement workflow from wireless algorithm to FPGA deployment.

These refinement steps do not need to follow the order shown in the diagram; the ordering of refinements will depend on the application and your preferences. This approach enables a gradual transition to Simulink while staying connected to MATLAB. Because Simulink has a built-in notion of time, adding sample-based stream processing is a good step to start refining in Simulink. Each of these refinement steps should be verified against the previous stage or against the original algorithm results.

The blue boxes in Figure 4 represent elements of the test bench: generating an input waveform and verifying the results. There are multiple layers to these, depending on the workflow stage. And as you add refinements to the design, your test bench will use the previous stage as a reference to compare against, so the previous stage becomes part of the test bench. It is important to sufficiently verify each stage so you can identify and fix bugs when they are introduced so that you have a known-good version to verify downstream refinements.

Reference Algorithm

Wireless communications design often starts with algorithm development and testing in MATLAB. This MATLAB algorithm becomes the "golden reference" from which the rest of the process proceeds. In traditional workflows, the algorithm would serve as the source of the functional specification document. In this workflow, it represents both the starting point for further refinement and an executable model to verify downstream refinements.

The 5G NR wireless standard is deep and complex, but you can speed the algorithm development process by using prebuilt standard-compliant functions from 5G Toolbox. These algorithms, combined with the wireless functions from Communications Toolbox™, let you quickly build a known-good 5G communications module so you can focus on developing your unique algorithm.

The NR HDL Cell Search design uses the PSS search, OFDM demodulation, and SSS search steps shown in the ***NR Synchronization Procedures*** example that ships with 5G Toolbox.

Hardware Architecture

Architecting an algorithm for implementation begins with partitioning. In the 5G Toolbox example ***NR Synchronization Procedures***, the MATLAB code for waveform generation, channel propagation, the receiver, and examining results is combined in one script. Implementing the receiver as shown in Figure 4 requires partitioning the receiver functionality and defining inputs and outputs.

The Wireless HDL Toolbox reference application includes a MATLAB version of the partitioned receiver, ***NR HDL Cell Search MATLAB Reference***. In this version, the receiver functionality that performs the high-speed signal processing tasks is partitioned into its own MATLAB function, which will be deployed to hardware with inputs and outputs as shown in the orange box in Figure 5. The blue boxes in Figure 5 will become the test bench for the hardware. The search control functionality can be implemented as software, targeting the processor on a system-on-chip (SoC) device. In this reference application, it is implemented in MATLAB as part of the test bench. This type of hardware-software partitioning, with the software controlling the hardware acceleration, is a common approach for wireless applications.

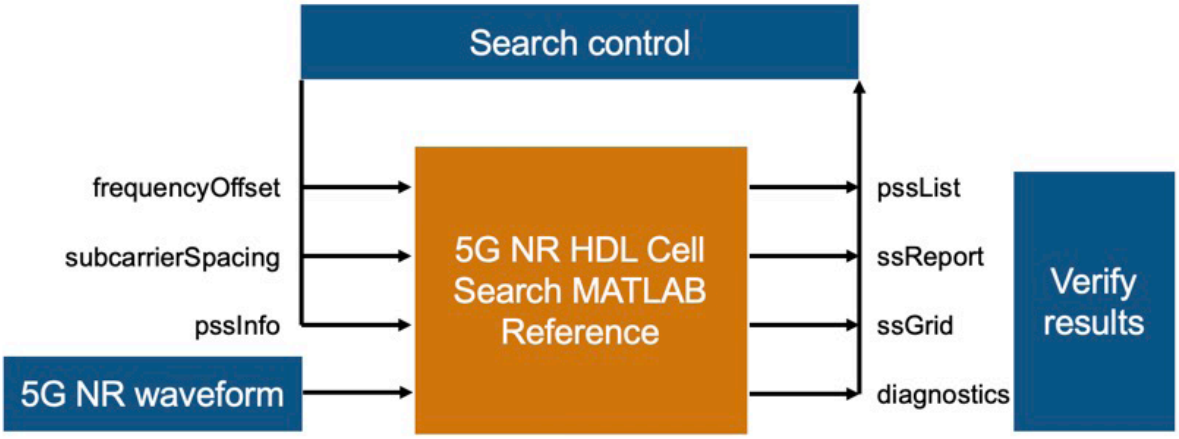

Figure 5. Partitioning of hardware functionality from software and test bench.

The hardware is architected to first correct the frequency offset passed from the search control and reduce the sampling rate. This allows the software to search across a range of coarse frequency offsets. The hardware performs PSS correlation and measures the residual fine frequency offset, passing the results back to the search control, which determines the strongest peak. This information is then passed back to the hardware along with the fine frequency offset value, where OFDM demodulation and SSS detection is performed.

All the processing to this point has operated on the entire waveform, or "frame-based" processing. The hardware is architected in anticipation of it being implemented in a real-world mode, where signals stream in continuously over time. The next step adapts the algorithms to work on a stream of samples.

Sample-Based Processing and Timing

Adapting your algorithm to work with streaming sample data requires some key changes in thinking. Not only do you have to change the way the algorithm works, but you also need to manage the stream of data. This is the largest change you will need to make, but a necessary one for targeting hardware. This mode of operation is also typically easier to design using Simulink because of its built-in notion of sample time.

In the NR HDL Cell Search design, detecting the PSS involves correlating a known sequence with the incoming signal. Correlation is straightforward in a pure mathematical algorithm—slide one signal across the other, sum the products of the samples, and identify a sufficient peak amongst the sums.

In the case of PSS detection, each of the three cell identity PSSs needs to be cross-correlated with the incoming signal. The PSSs are static values likely stored in local on-chip RAM, and the incoming signal is streaming in, so the sliding of one signal over another happens naturally. But we still need to buffer a window of the incoming stream to perform the cross-correlation calculations. This is all handled in the Discrete FIR Filter HDL Optimized blocks in the PSS correlators, shown in Figure 6. These blocks offer a choice of serial and parallel architectures, and automatically update their latency for simulation based on the parameter settings.

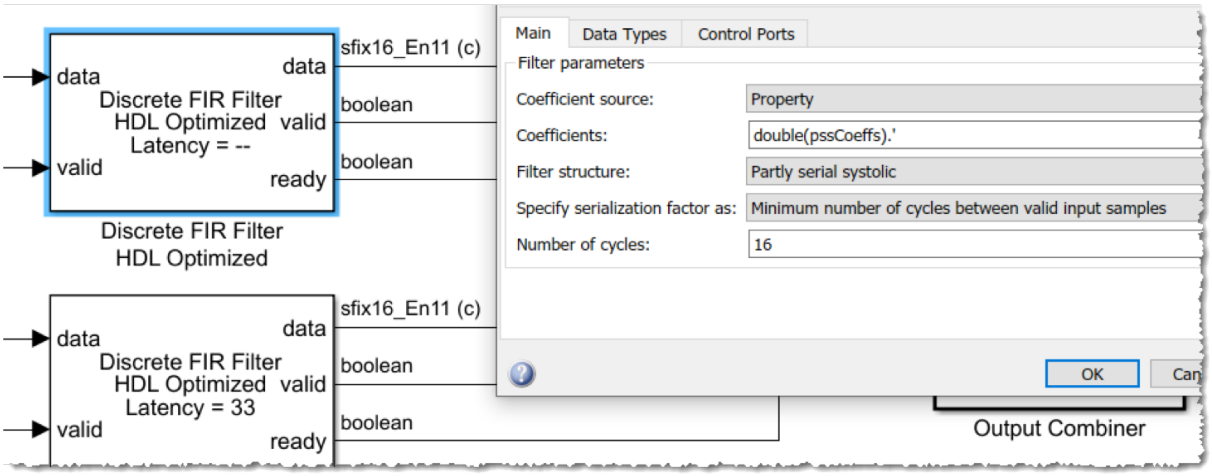

Figure 6. Configuring a Discrete FIR Filter HDL Optimized block.

In hardware, we can perform the three PSS correlations in parallel, detect the peaks, then pass the results to the control software. To determine the peaks, we check whether each correlation result is higher than a threshold that is proportional to the average signal power. If the threshold is crossed, we can search this local window of time for the true peak value. Figure 7 illustrates this architecture, with the PSS correlation results and average signal power plotted.

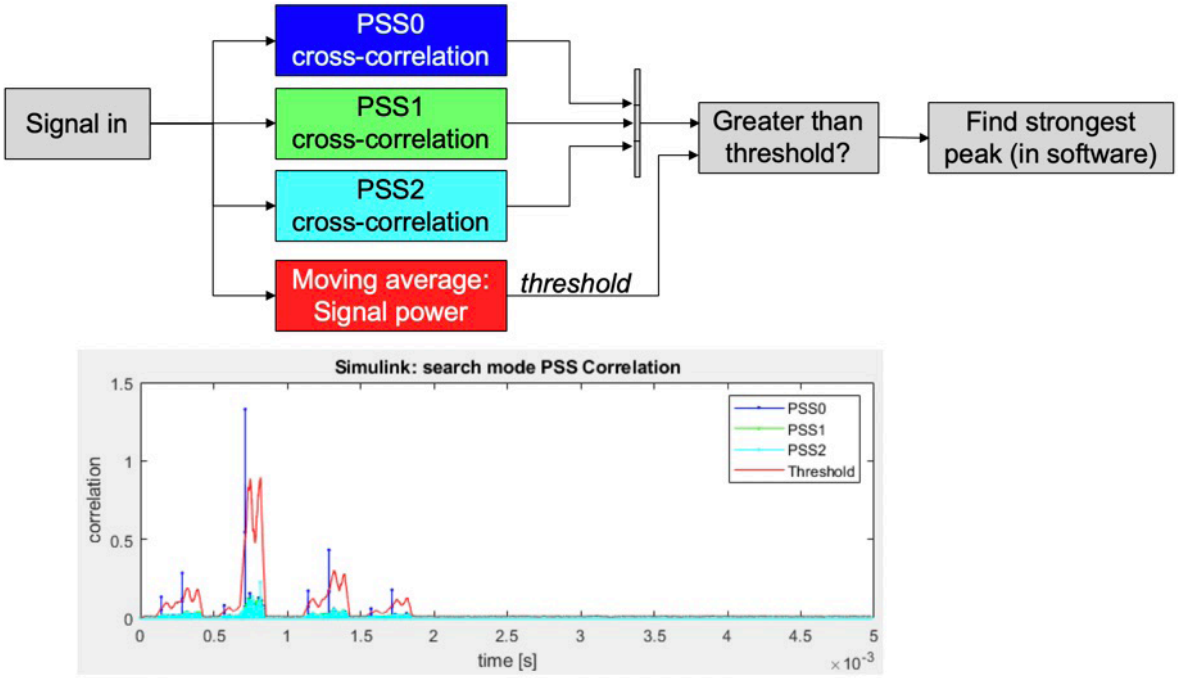

Figure 7. Flowchart of PSS detection (top), and plot of the results of the PSS cross-correlation and the moving average threshold (bottom).

Calculating the average signal power for a continuous stream of data requires a different approach than calculating a global average in MATLAB. This also requires a sliding window approach, which is why the threshold plot in Figure 7 varies over time. The average signal power is measured across the same time window as the correlators. This approach delivers the required detection performance with a small amount of storage required. This design tradeoff requires close collaboration between algorithm developers and hardware engineers.

A moving average algorithm is shown in Figure 8. Delay blocks such as "Z^{-1}" will map to registers in hardware to store values. The "Z^{-128}" block inserts a FIFO buffer in parallel to the incoming sample, allowing the calculation to subtract out the sample that was 129 clock cycles previous—so it constantly stores the most recent 128 samples for the moving average. The blue registers in Figure 8 serve as pipeline stages. Because mathematical operations consume time in hardware, breaking them up to reduce the amount of computation between stages reduces the clock period, increasing the maximum clock frequency. You can let HDL Coder automatically insert pipeline stages; however, it's good practice to model the right number of delays in your design so that during simulation you can make sure the parallel paths are correctly synchronized before you generate HDL. Note how in Figure 8, the parallel validIn signal has matching delays, so when it's used for the enabled delay it arrives on the right clock cycle. If you let HDL Coder insert pipeline delays as part of optimizations such as adaptive pipelining, ensure that "Balance delays" is turned on in "Optimization" settings (it is by default).

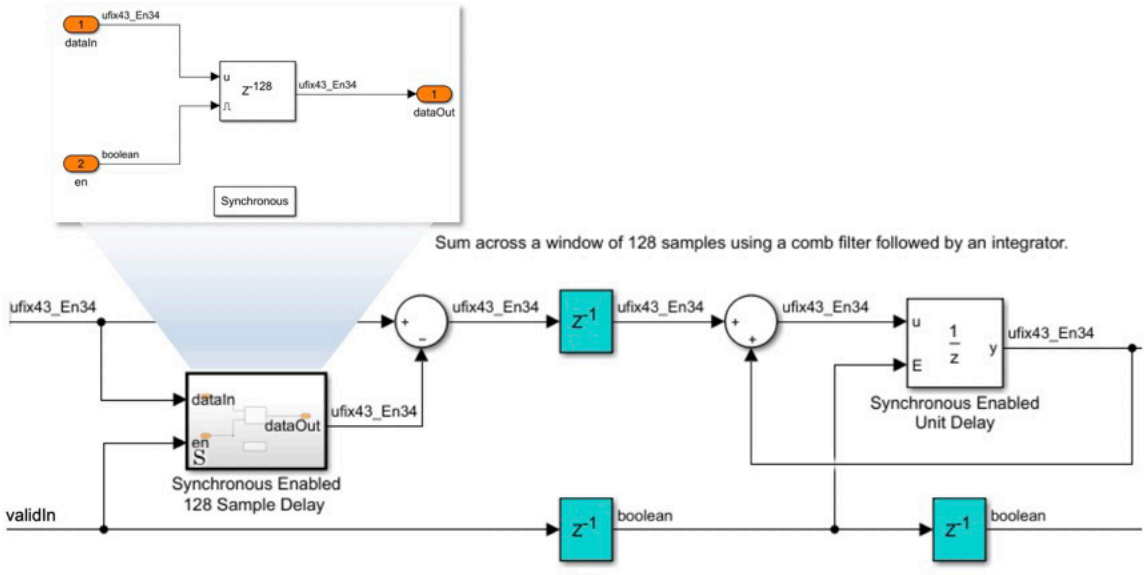

Figure 8. Moving average algorithm in Simulink with hardware implementation detail.

With data streaming in continuously, the algorithms need to know when a sequence such as a packet starts and ends, or is interrupted. This requires control signals in hardware, which at a minimum consist of "valid" signals, but often include "enable," "start," and "end." In the NR HDL Cell Search design, the OFDM demodulator uses a MATLAB Function block to generate a valid signal based on the startIn and validIn signals, as shown in Figure 9.

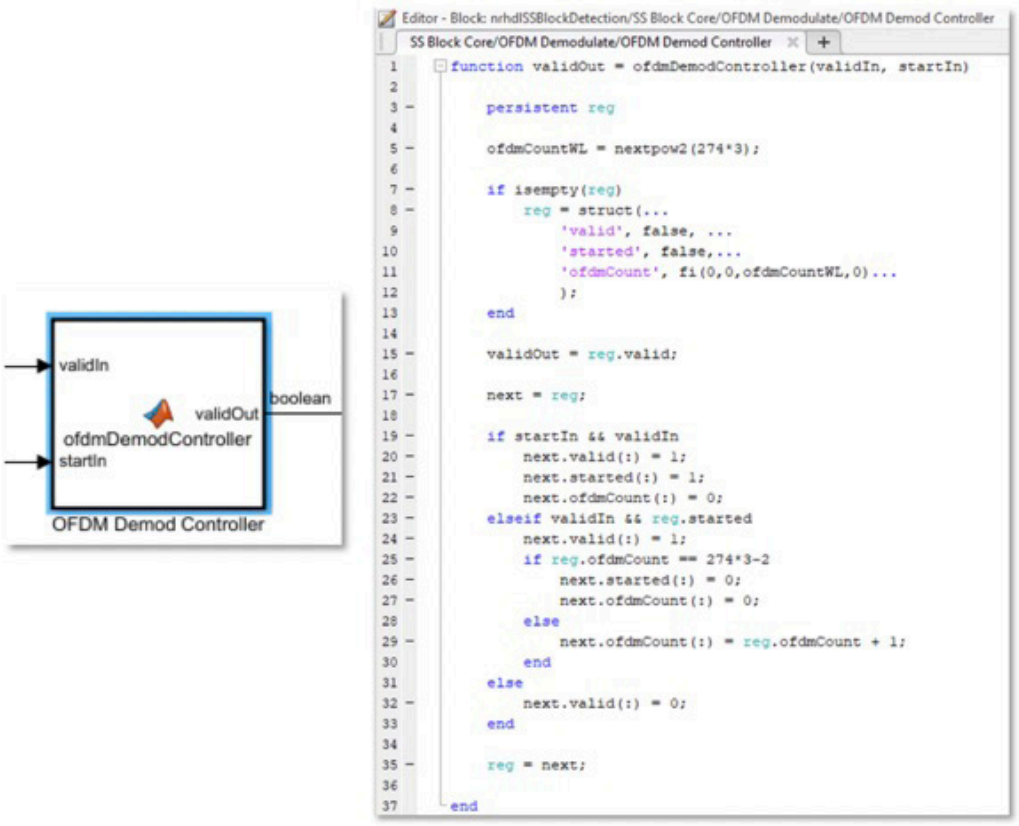

Figure 9. OFDM Demodulation Controller MATLAB Function block and corresponding MATLAB code..

The streaming algorithms described earlier use FIFO buffers, a simple form of memory, to temporarily store a window from the data stream. Data stored or used non-sequentially must use addressable RAM or read-only LUT resources. Memory architecture can greatly affect the performance and resource usage of hardware implementations. Figure 10 shows how memory is used in the SSS detection subsystem. The MATLAB Function block controls the state, which is either "idle," writing the OFDM-demodulated symbols to the RAM, or reading them out for correlation versus the SSS sequences. The SSS sequences are fixed values so they are stored in a LUT, which is a read-only memory resource.

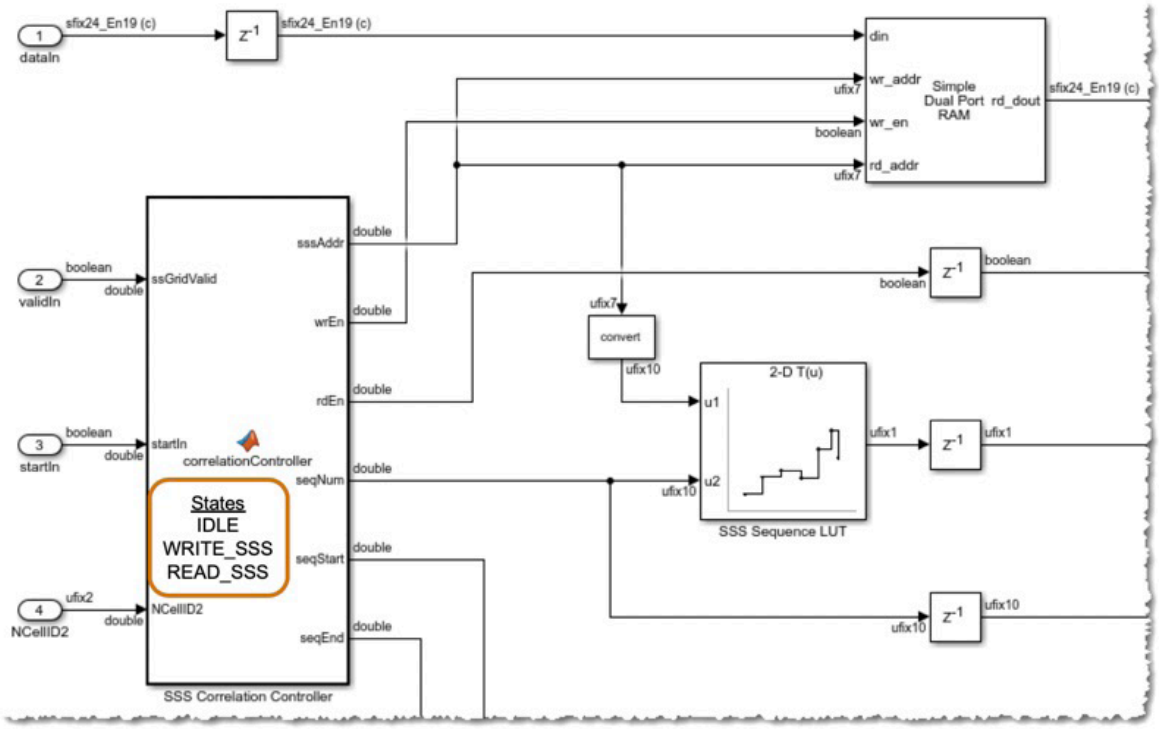

Figure 10. Using RAM and LUT hardware resources for the input to the SSS detection algorithm.

Understanding memory sizes and your hardware device capabilities is important for meeting your performance and resource usage goals. In this case, the RAM is dual-port, meaning it can be written to while it is being read. Its address line is 7 bits, so it can store data to 128 locations, and the data is 48 bits wide, so this requires 6144 bits of RAM, which in most cases will map to block RAM on the device.

When you are ready to simulate the Simulink hardware implementation, Simulink interacts with the MATLAB workspace. The top level of the NR HDL Cell Search design uses From Workspace blocks to take the frame-based signal data, in the MATLAB workspace structure "in," and stream it into the hardware implementation sample by sample. The outputs are collected in the To Workspace blocks and written to the structure "out" for use by MATLAB. Figure 11 shows the Simulink diagram with some of the MATLAB code that sets up the input parameters, runs the Simulink model in search mode, reads the results, then runs the Simulink model in demodulation mode. What is not shown in Figure 11 is that the MATLAB reference algorithm is also called to verify the results of the streaming Simulink implementation.

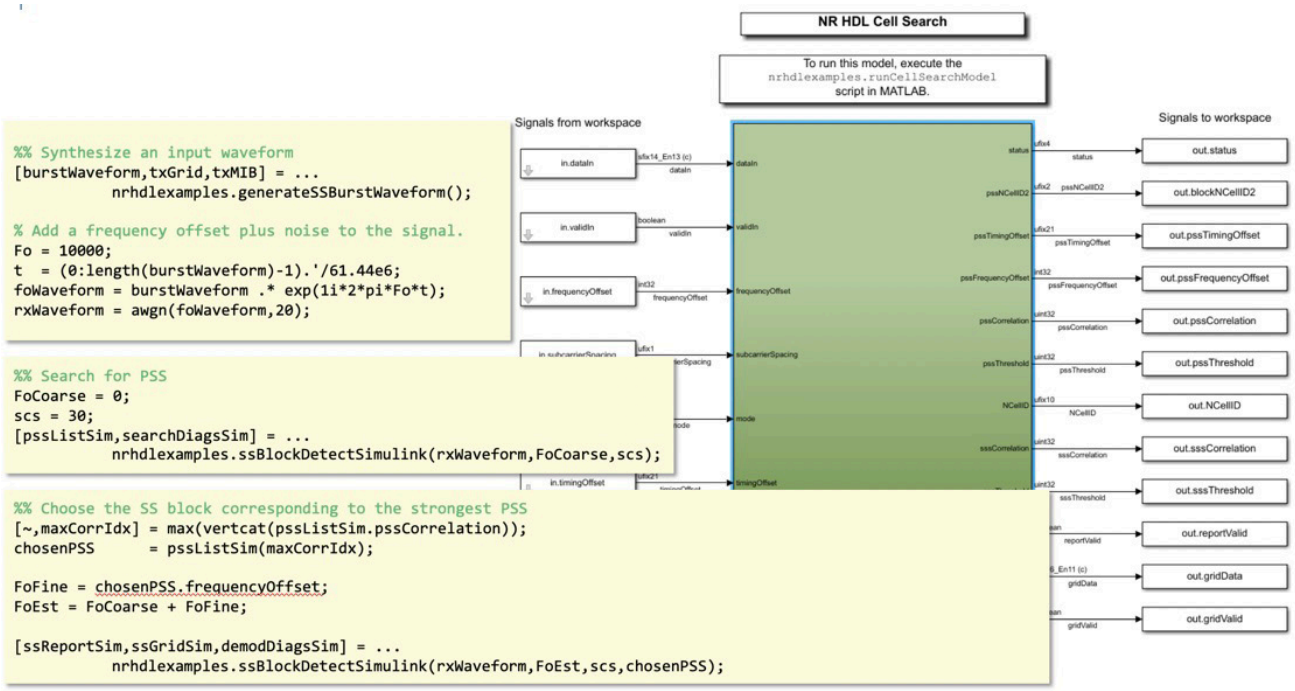

Figure 11. Top-level Simulink diagram for the NR HDL Cell Search reference application, with MATLAB code to synthesize a 5G NR-compliant waveform for the input, call Simulink in PSS search mode from MATLAB, and pass the strongest PSS back to the Simulink model for SSS detection.

Leveraging Proven IP Blocks

When implementing hardware architectures of streaming algorithms, it is usually more practical to reuse prebuilt blocks than to take on the cost and risk of designing them from scratch. These IP blocks can come from external sources or other projects within your company. No matter the source, there are three key principles to using them:

1. **Use built-in customizability instead of changing the design itself.** This customizability typically takes the form of parameters. For instance, with the Discrete FIR Filter HDL Optimized block used in the PSS correlators as shown in Figure 6, you can reuse coefficients from the MATLAB workspace, customize the HDL filter architecture, serialize the operations, and configure other parameters. Once the parameters are set and the design is built, the block will simulate this hardware implementation, including its latency given its settings. This implementation will be optimized and generated as HDL downstream.
2. **Use IP blocks that easily plug into your design.** The OFDM Demodulator block in the SS Block Core subsystem of the NR HDL Cell Search design, shown in Figure 12, illustrates this principle. This block supports demodulation for a variety of OFDM-based standards: 5G, LTE, Wireless Local Area Network (WLAN 802.11a/b/g/n/ac), WiMAX (802.16 m and e), digital video broadcast (DVB), and digital audio broadcast (DAB). Depending on your design requirements, you can quickly set parameters for the cyclic prefix, OFDM symbol guardbanding, and the FFT implementation. Note that the control functionality on the input uses the startIn signal to determine when to reset the demodulator and pass samples into the OFDM demodulator by driving the valid signal.

3. **Select IP blocks that have been proven to deliver the functionality and performance you need.** This may seem obvious, but if you do not adhere to this principle, any issues you run into will be difficult to address because the design is unfamiliar to you. This is why the *NR HDL Cell Search design* has been targeted to Xilinx Zynq hardware and tested in the field using over-the-air signals.

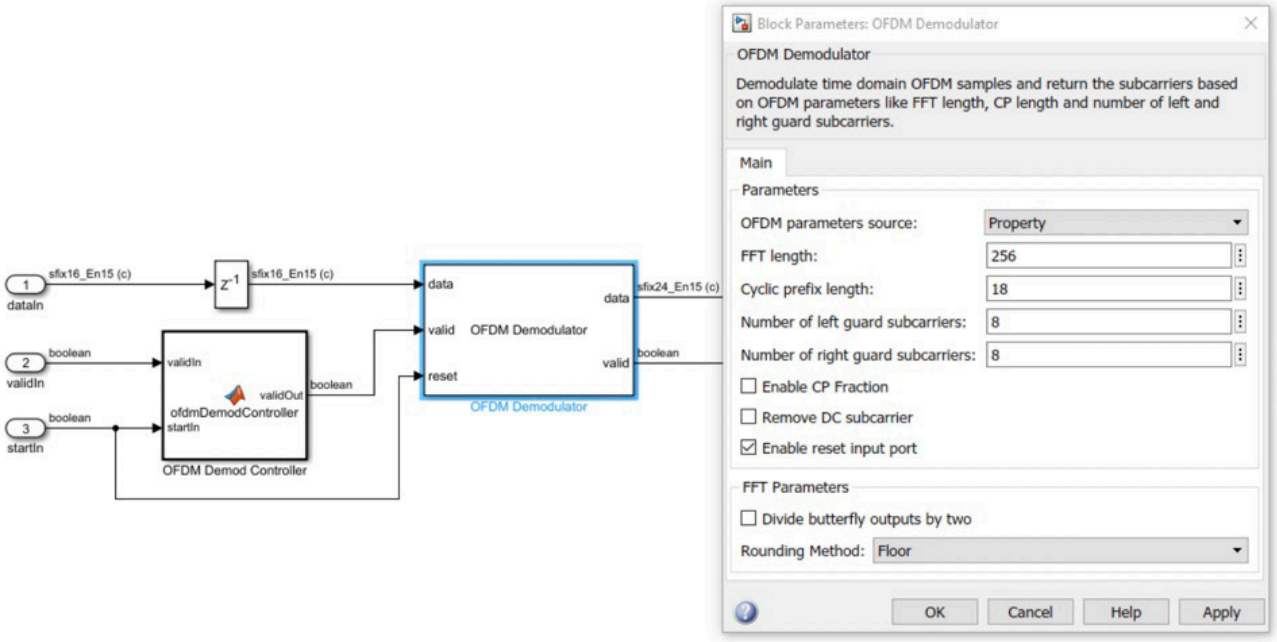

Figure 12. IP blocks that plug into your design's context.

Fixed-Point Implementation

Most digital hardware designs use fixed-point math for more efficient implementation. Trading off accuracy versus efficiency can be challenging, especially when there are feedback loops or high-dynamic-range equations. HDL Coder offers *native floating-point* implementation, which can be useful in the divide operations required for OFDM equalization in LTE. You can isolate these operations with data type conversion blocks and generate single-precision floating-point operations for scalable accuracy.

For most designs, including the 5G NR Cell Search design, a sufficiently accurate fixed-point implementation is real-izable, and you could use Fixed-Point Designer to manage the process. But it helps to have a basic understanding of *fixed-point theory*, especially how word lengths grow in different arithmetic operations. For instance, the word length for the result of a multiplication will be the sum of the word lengths of the inputs, which you can observe in Simulink, as shown in Figure 13. In the notation shown, sfix means signed fixed-point, the first number is the word length, and the second number is the bits of precision. So sfix16_En11 is a signed fixed-point number whose word length is 16, with 11 of those bits being precision bits. When these two 16-bit numbers are multiplied, they produce a 32-bit result.

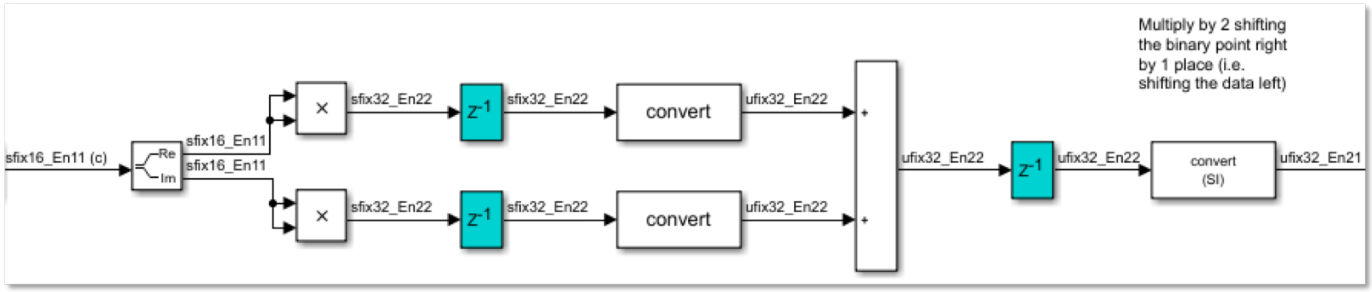

Figure 13. Fixed-point word length propagation through arithmetic operations.

Multipliers are a key area of focus when performing fixed-point conversion because multiplication operations are expensive in terms of hardware resource usage. Most FPGAs have efficient multiplication resources called DSP slices, which are typically 18x18 or 25x18. If your inputs reach the multiplier and are larger than that, your multiplication operation will be split across DSP slices and you can use them up quickly. Again, it is a balancing act of efficiency versus accuracy.

Targeting FPGA Hardware

Once you are satisfied with the accuracy of your fixed-point implementation, you can begin the targeting process with HDL Coder. Start by right-clicking on the SS Block Detection subsystem and selecting **HDL Code > HDL Workflow Advisor**. This wizard-like interface will walk you through the steps to generate VHDL or Verilog HDL to target your device. HDL Coder produces device-independent code with hierarchy and signal names that correspond to your Simulink design, along with all the bit-level port mappings, as shown for a snippet of code in Figure 14.

```
-- -------------------------------------------------------------
--
-- Module: nrhdlSSBlockDetection_Magnitude_Squared_2
-- Source Path: nrhdlSSBlockDetection/SS Block Core/PSS Detection/Correlators/Magnitude Squared 2
-- Hierarchy Level: 5
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY nrhdlSSBlockDetection_Magnitude_Squared_2 IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        enb                               :   IN    std_logic;
        dataIn_re                         :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En11
        dataIn_im                         :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En11
        validIn                           :   IN    std_logic;
        dataOut                           :   OUT   std_logic_vector(31 DOWNTO 0);  -- ufix32_En24
        validOut                          :   OUT   std_logic
        );
END nrhdlSSBlockDetection_Magnitude_Squared_2;


ARCHITECTURE rtl OF nrhdlSSBlockDetection_Magnitude_Squared_2 IS

  -- Signals
  SIGNAL dataIn_re_signed                 : signed(15 DOWNTO 0);  -- sfix16_En11
  SIGNAL dataIn_im_signed                 : signed(15 DOWNTO 0);  -- sfix16_En11
  SIGNAL Delay3_out1_re                   : signed(15 DOWNTO 0);  -- sfix16_En11
  SIGNAL Delay3_out1_im                   : signed(15 DOWNTO 0);  -- sfix16_En11
  SIGNAL Product4_out1                    : signed(31 DOWNTO 0);  -- sfix32_En22
  SIGNAL Product5_out1                    : signed(31 DOWNTO 0);  -- sfix32_En22
```

Figure 14. Snippet of generated VHDL for a magnitude-squared subsystem.

HDL Coder can also drive the FPGA synthesis and targeting process, which of course is specific to your vendor and device. Because there are so many devices and boards available for targeting, this paper cannot cover all of the details beyond the general targeting categories:

Targeting a prototype board. If you are looking to use an off-the-shelf board such as a Xilinx Zynq-based software-defined radio (SDR) kit, you can download a ***hardware support package*** to ease the setup and targeting of both the hardware and the software running on the ARM® processor. ***This example*** uses this NR HDL Cell Search design to illustrate the steps for targeting a Xilinx Zynq-based board.

Targeting a production device. Typically you would work with your hardware team to define a custom board definition, which specifies what device is on the board, what peripherals are on the board, and how the device's pins map to the board-level I/O. When you run the HDL Workflow Advisor, you can select this board as the target in step 1.1. Then you specify the mappings of your design's I/O to either registers to communicate with the software or to the device's pins.

Verification and Debugging

As mentioned in the Workflow Overview, your test bench will evolve along with your design. Here is a brief overview of the main elements of the test bench and how they might evolve.

Stimulus

Early in algorithm development, you can use MATLAB based functionality to synthesize a waveform. In the NR HDL Cell Search design, the test bench uses functionality from 5G Toolbox to synthesize a standard-compliant waveform and pass it through a noisy channel model. No matter how much you model into the stimulus waveform, there will always be unanticipated elements to the signals you encounter when testing in the field. Thus it is good practice to capture live over-the-air waveforms whenever possible, as shown in ***this example*** for an LTE signal, for testing within MATLAB and Simulink before you get onto hardware, where it's difficult to debug.

Reference Model

As you refine your design with hardware detail, implement sample-based stream processing, and quantize to fixed point, you can compare its results against previously verified versions of the algorithm. You can always compare results with those from the original reference algorithm, but often it's more straightforward to compare results from more similar versions of the design. A good example is that once you convert to stream processing, it's easier to compare streaming outputs against each other rather than converting back to frames. In the NR HDL Cell Search example, the Simulink implementation model is compared with the MATLAB hardware architecture model, which was the previous refinement.

Assessment and Debug

When developing initial algorithms, assessment of correctness typically involves examining waveforms, scopes, and spectrum plots. But as you add refinements to the algorithm, verification involves measuring whether added implementation refinements functionally match the original algorithm.

Results from different refinements typically will not exactly match each other, a good example being the comparison of floating-point and fixed-point versions. Thus you will need to determine how to assess whether an algorithm is considered to pass or fail each test. In the NR HDL Cell Search example, the results from the MATLAB and Simulink models are compared by measuring their relative mean-squared error. Another common technique for wireless communications is the bit-error rate. Agreeing on the measurement of pass/fail is a project-level decision.

Debugging test failures is always more easily done at the step where the bug is introduced. Waiting to find bugs later will increase the challenge exponentially, because you will have more detail to wade through, and you will be steps removed from the design decisions that likely led to the introduction of the bug. Once the design is running on the FPGA, limited visibility to internal signals adds another factor to the complexity equation.

Measuring ***coverage*** is a way to assess how much of the functionality of your design has been tested before you move on and generate HDL or program it onto an FPGA. It will also help you assess whether your tests are exercising the same functionality over and over and hence wasting simulation time. You can adjust what tests are run based on the feedback from coverage analysis.

Reducing the Pain of Verification Downstream

Verification and debugging can be extremely time-consuming and painful at lower levels of detail, which is why it's so critical to identify and fix bugs as early as possible. From the register-transfer level (RTL) until implementation on the device, more detail and functionality is added and hence more opportunity for bugs to be introduced. You can leverage the work you have done in MATLAB and Simulink to improve the efficiency of resolving these issues. Techniques such as ***HDL cosimulation*** and ***FPGA data capture*** let you debug from within MATLAB and Simulink, and you can ***generate verification components*** directly from your test bench models to help production verification teams build their test benches.

Results

We deployed the NR HDL Cell Search design to a Xilinx Zynq-7000 All Programmable SoC ZC706 Evaluation Kit with a connected Analog Devices® AD9361 RF card. It scanned a frequency range, and from a live over-the-air signal it detected PSS and SSS for the strongest cell signal. It passed the OFDM-demodulated synchronization block resource grid from the SoC to MATLAB, where it was successfully decoded using 5G Toolbox.

Perhaps more importantly, we started with a MATLAB algorithm, refined it with hardware implementation details, and automatically generated and targeted this device from the same environment. This approach enabled us to easily verify each successive refinement toward hardware to ensure that it delivered the same results as the MATLAB algorithm. The simulations used 5G NR waveforms and we also detected PSS and SSS using live over-the-air signals.

The skills to accomplish all of this rarely exist within one engineer; the collaboration environment shown here brings these skills together to enable engineers from across roles and departments to design, simulate, and iterate more quickly. Simulating early in the system-level context with real

stimuli helps engineers find and fix functional and performance issues before detailed hardware design begins. Changes can be implemented quickly and simulated, and then updated code can be automatically generated to target hardware. This agility enables engineers to focus their efforts on higher-order solutions that have a larger effect on the resulting product than if they were constrained by having already written thousands of lines of VHDL.

Whether your end goal is a prototype or production deployment, this workflow speeds the targeting of high-quality wireless communications algorithms to FPGA hardware.

Learn More

Watch a video on the NR HDL Cell Search design (reference application available with Wireless HDL Toolbox): *5G NR HDL Cell Search Reference Application*

Download the Zynq SDR hardware support package: *Zynq SDR Support from Communications Toolbox*

Visit the resource center: *Wireless Communications Design with MATLAB*

Explore Services for Getting Started

MathWorks offers many ways to help you learn to use this workflow effectively, from getting started with an applications engineer, to in-depth live training courses and customized consulting services.

- *MATLAB and Simulink Training*
  - *Designing LTE and LTE Advanced Physical Layer Systems with MATLAB*
  - *DSP for FPGAs*
  - *Software-Defined Radio with Zynq Using Simulink*
- *MathWorks Consulting Services*