Leveraging Mathworks Tools to Accelerate the Prototyping of Custom 5G Applications in Hardware

Joshua Geyster
Open and Embedded Systems
MIT Lincoln Laboratory
Lexington, MA
joshua.geyster@ll.mit.edu

Dr. Karen Gettings
Open and Embedded Systems
MIT Lincoln Laboratory
Lexington, MA
karen.gettings@ll.mit.edu

Dr. Paul Monticciolo
Open and Embedded Systems
MIT Lincoln Laboratory
Lexington, MA
Paul.Monticciolo@ll.mit.edu

Matthew Rebholz
Advanced RF Techniques and Systems
MIT Lincoln Laboratory
Lexington, MA
mjrebholz@ll.mit.edu

***Abstract*—The ubiquity and flexibility of 5G networks make it an attractive technology to customize for particular needs. In this work, we highlight some of the challenges associated with implementing custom 5G applications in hardware and the growing trend of utilizing high-level synthesis tools to relieve these issues. We present an overview of the 5G resource grid and the MATLAB-Simulink-HDL Coder workflow. We then demonstrate the workflow through an example 5G resource grid transmitter design.**

Keywords—5G, Custom 5G, MATLAB, Simulink, HDL Coder, HLS, Hardware Implementation, 5G Transmitter.

I. Introduction

As the rollout and adoption of 5G networks is currently underway around the world [1], many are beginning to look at what is next in the realm of wireless communications. This includes the development of custom applications and research into the next generation of wireless communications.

A resource grid of subcarriers and OFDM symbols is defined for each numerology and carrier. There is one set of resource grids per transmission direction (uplink or downlink) [2]. The 5G architecture introduced a much more flexible numerology than its predecessor, LTE. This flexibility enables 5G to accommodate users with differing quality of service constraints with greater ease. In order to exploit 5G's flexibility to its fullest potential, custom applications/algorithms are developed to make the most of a communication's allotted carrier bandwidth. One such example of a 5G customization, is an optimization of resource allocation for heterogenous services, done by the authors in [3]. In this 5G application, greater total throughput was achieved by cleverly scheduling users in need of low latency and users in need of high data rates within the same bandwidth.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering's FutureG Office under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

When a custom wireless application transitions from ideas and simulations into hardware, the options an engineer has have their own costs and benefits. One option is that they acquire dedicated base stations and user equipment. This additional hardware carries with it the support and tools of the manufacturer, but with a relatively steep financial cost. For prototyping, this extra cost may be out of budget. An alternative to acquiring this dedicated hardware is programming FPGAs to stand in as the transmitters and receivers. This has been an active area of research, for example [4].

FPGAs are able to provide flexibility and reusability at a relatively small price point, but have their own drawbacks. One major issue plaguing FPGAs is the time and expense required to program them. Hardware description languages (HDLs), such as VHDL and Verilog, have a steep learning curve and can be strenuous to debug and optimize. Additionally, synthesis and implementation steps are required before the hardware can be programmed, which can often take several hours to complete [5].

In an effort to alleviate some of these FPGA programming challenges there has been a growing rise in the use of high-level synthesis (HLS) tools over the past decade. These tools aim to accelerate the hardware implementation process by allowing an engineer to build their design on a higher layer of abstraction. This often takes the form of a graphical interface, such as in the case of MathWorks' Simulink and HDL Coder, or a general-purpose programming language, such as C/C++ in the case of Vivado HLS [6]. These tools save time and resources by automating and simplifying HDL development.

In this paper, we start by presenting a brief background of the 5G resource grid. Then, we discuss the workflow required to take a custom 5G design from a MATLAB algorithm to hardware through the use of Simulink and HDL Coder. Lastly, we present an example transmitter design that can be used to quickly prototype different grid configurations.

II. The 5G Resource Grid

5G, like its predecessor LTE, is what is known as a modulated multicarrier waveform [7]. These waveforms create a natural grid of their time and frequency components, often referred to as their resource grid. The frequency component of the waveform's grid is made up of a number of adjacent subcarriers separated in frequency by a number known as the

subcarrier spacing (SCS). Every group of 12 subcarriers is known as a physical resource block (PRB). The time domain component of the grid is broken down into 10 ms frames, 1 ms subframes, and a variable number of slots.

One key difference between LTE and 5G is the introduction of a flexible SCS. In LTE, the SCS was fixed at 15 kHz; however, in 5G this number can be either 15 kHz, 30 kHz, 60 kHz, 120 kHz, or 240 kHz, with a few exceptions depending on the transmit signal. Based on the SCS of the waveform, the duration of each slot of the waveform changes. While each subframe remains 1 ms, the number of slots per subframe increase by powers of 2 as the SCS is increased. Each slot contains 14 symbols, except when using extended cyclic prefix with a SCS of 60 kHz, where the number of symbols is 12.

For a waveform with a 15 kHz SCS there is only 1 slot per subframe, so the slot duration is also 1 ms. For a waveform with a 30 kHz SCS, there is then 2 slots per subframe, so the slot duration of each slot becomes 0.5 ms. The full breakdown of PRB bandwidths and slot durations as a function of the SCS can be found in Table I.

Subcarrier Spacing (kHz)	PRB Bandwidth (kHz)	Slots / Subframe	Symbols / Slot	Slot Duration (ms)
15	180	1	14	1
30	360	2	14	0.5
60	720	4	14 (12)	0.25
120	1440	8	14	0.125
240	N/A	16	14	0.0625

Table I. PRB/Slot Grid Bandwidth and Duration as a Function of SCS

By having a flexible numerology, 5G seeks to meet the requirements of the three main development use cases: extreme mobile broadband (eMBB), ultra-reliable and low latency communications (URLLC), and massive machine type communications (mMTC). eMBB users prioritize high data rates, URLLC users prioritize low latency, and mMTC users prioritize having many connections [8].

Among the possible numerologies available for 5G waveforms, there exist limitations based on what frequency range the transmitted signal lies within. For 5G, there are two frequency ranges: ffrequency range 1 (FR1) lies between 410 MHz and 7.125 GHz; and frequency range 2 (FR2) lies between 24.25 GHz and 52.6 GHz [7].

For signals within FR1, available SCS include 15KHz, 30 kHz and 60 kHz. For signals within FR2, available SCS include 60 kHz, 120 kHz and 240 kHz. However, a SCS of 240 kHz is only allowed for the downlink synchronization signal block (SSB) and 60 kHz SCS signals cannot be used for SSBs [7].

5G communications are split into downlink and uplink messages and the contents of the resource grid vary based on the signal being sent. Depending on the current step of the communication, the grid may contain control/synchronization information, transmit data, or both. Once an engineer is aware of the grid required for their custom application, they can move to simulation and hardware implementation.

III. The MATLAB-Simulink-HDL Coder Workflow

In order to quickly get a design working in hardware, it may be beneficial to develop the algorithms and structure in a high-level language. MathWorks technology coupling Matlab, Simulink and HDL coder has proven successful at aiding the FPGA implementation of algorithms developed in Matlab for a large number of applications, for example [9][10][11][12]. In our research, MATLAB and Simulink are helpful because of the various 5G applications and toolboxes available. Additionally, these programs allow all of the required working environments, such as RF architecture, digital signal processing, and FPGA design, to all be bundled up in one location [9], and have been successful at several hardware demonstrations for multiple applications.

To begin this design process, an algorithm for the custom design should first be developed in MATLAB. The process can be expedited using an example design from the 5G Toolbox and the pre-built functions from either the 5G Toolbox or the Communications Toolbox [9]. This floating-point MATLAB algorithm will serve as the "golden reference" to which one can compare the Simulink and Hardware results [9].

In addition to the pre-built functions, the 5G toolbox offers a 5G waveform generator application which can be used to quickly visualize and generate both uplink and downlink resource grids of various configurations. A snapshot of the waveform generator application for a custom downlink signal is shown in Fig. 1.

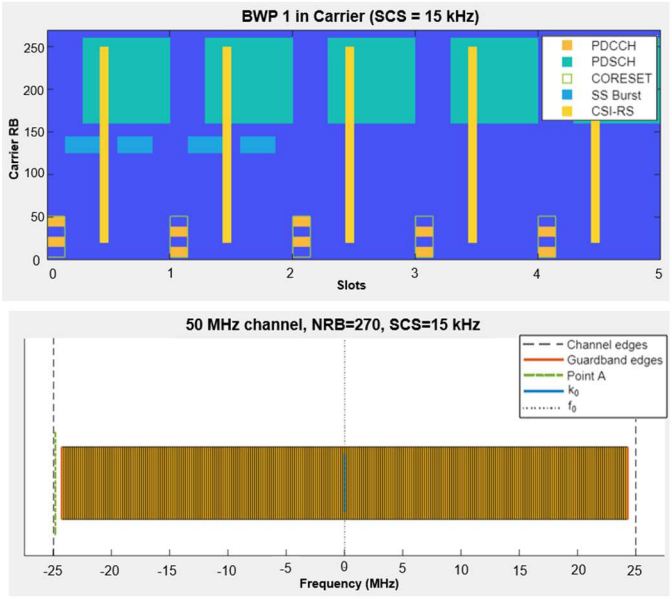

Fig. 1. 5G Waveform Generator Interface from the 5G Toolbox App for a Custom Downlink Waveform.

After the algorithm is designed, the custom MATLAB application is then implemented in Simulink to include information on data flow, resource usage, and timing to enable porting the algorithm into hardware [9]. Simulink is able to better model the behavior of hardware due to its built-in concept of sample time.

To implement an algorithm in hardware using the MathWorks methodology, one must first partition the algorithm into its individual components [9]. These components may include the memory, processing, control, or interface portions of the design. Another key decision to make is what operations can be done in parallel, since both Simulink and the hardware allow for parallelization.

Once the Simulink design is planned out, one can begin building the individual blocks. For applications targeted at hardware, note that only a subset of the Simulink blocks are compatible for HDL code generation. A full list of compatible blocks can be found by entering "*hdllib"* into the MATLAB command window. Delay blocks, which map to flip-flops in hardware, may need to be added between combinational components in order to meet timing constraints. The locations of required delay blocks can usually be found by analyzing path of worst negative slack (WNS) following the implementation of the HDL Code in a synthesis tool such as Vivado.

After laying out the components of the design in Simulink, the Simulink blocks can be organized into groups, referred to as subsystems. Masked subsystems and variant subsystems can be used to have alternate configurations of the same subsystems throughout a design. Once the desired subsystems are formed, the individual blocks and their subsystems must be connected together and the data types for the signals should be assigned. In hardware, operating on floating point numbers can be very resource intensive so it is good practice to convert any fractional numbers into fixed-point values. The required bit widths and fraction lengths for each signal depend on the application, the hardware resource utilization, and how much quantization error is acceptable.

Once one's blocks and subsystems are laid out and they have been assigned hardware-friendly data types, they should be packaged into a single subsystem that will serve as the design-under-test (DUT). The DUT subsystem will serve as the top-level module, and HDL code will be generated for all the blocks within it. The exact shape of the DUT subsystem depends on the application and the desired interfaces. The in-ports and out-ports of the DUT will become the interfaces to which external signals will be connected during code generation.

To interface with the DUT once it's in hardware there are a few options that can be used. An AXI-Lite interface can be used to read and write single words. This can be useful for setting control registers, reset signals, and/or enable signals. For reading and writing streams of data AXI-Stream interfaces can be added. For both AXI-Lite and AXI-Stream interfaces, additional hardware and ports will need to be added to the DUT. For the stream interfaces this includes data, valid, and ready signals. In addition, a FIFO may need to be added to buffer up sequential valid data read by the DUT or prior to performing a write out of the DUT.

As shown in Fig. 2, additional blocks and subsystems may be added outside of the DUT to provide stimulus to and aid in Simulink simulation. The Simulink data inspector or To Workspace blocks can be used to verify that the Simulink model behaves as intended. If constructed with HDL compatible blocks, the DUT should behave similarly in hardware as it does in the Simulink simulator.

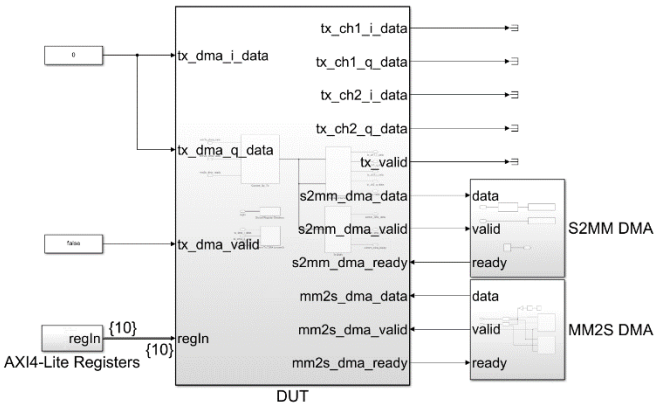

Fig. 2. Example DUT and External Blocks used for Simulink Simulation.

After verifying that the Simulink model behaves as intended, HDL code and an IP Core for the DUT can be generated using the HDL Workflow Advisor in Simulink. It is within this workflow advisor tool that one selects their target FPGA, the interfaces for the DUT ports, and the target frequency. Additionally, HDL Coder offers a number of HDL options that can be used to optimize the generated code, such as adaptive pipelining and various clock/reset settings. Through the workflow advisor, HDL code can be generated and a project for the specified synthesis tool can be created. From there synthesis, implementation, and bitstream generation can either be done through the advisor or by opening up the generated project for the synthesis tool and running the tasks there. After generating the bitstream file, the target FPGA can be programed a number of ways including using a MATLAB script, the workflow advisor, or the synthesis tool.

With the FPGA programed, there are a limited number of ways to debug issues with the design. Debugging the design while in Simulink is recommended. If needed, collecting a stream of data from the FPGA via the AXI interface is one way to debug the design while in hardware. Another hardware debugging method available is the use of FPGA Data Capture via HDL Verifier. This allows the designer to add test points to the Simulink design which will automatically create capture logic within the DUT at the time of code generation. Given a trigger condition, these test points can be read into MATLAB for debugging. One note on test points however is that can cause significant block RAM utilization because the captured data needs to be collected and stored prior to the connected host computer reading the data.

IV. Custom 5G Resource Grid Transmitter

To demonstrate the MATLAB-Simulink-HDL Coder Workflow and to aid the prototyping of custom grid configurations, a hardware compatible resource grid transmitter was developed. In the following sections we outline the design workflow, operation, and test results of the transmitter.

A. Implementing the HDL Coder Workflow

To host our custom 5G resource grid transmitter we chose a ZCU102 eval board with a FMCOMMS 3 RF transceiver. This was chosen because it is one of several target layouts supported by the HDL Coder Support Package for Xilinx Zynq Platform. With this in mind, we determined that our transmit waveforms would need to be within FR1 and at most 50MHz in bandwidth. This is because the FMCOMMS 3 transceiver can only handle a sampling rate up to 61.44 MHz.

After laying out some constraints, we began to determine what main components our transmitter design would need. First, we would need memory to hold both the resource grid and the list of cyclic prefix lengths. For the memory, we would also need control logic to handle addressing. In addition to memory, the other main piece of the transmitter design would be the OFDM modulator to take the grid from the frequency domain to the time domain. Lastly, we would need AXI interface logic to set control registers, read data from MATLAB, and to write data to MATLAB.

To serve as the "golden reference" for the design, resource grids from the 5G waveform generator application were created. The OFDM demodulation function from the 5G toolbox was used to translate the application's time-domain waveform into its associated grid.

For the Simulink model, in order to facilitate prototyping of different grid configurations more quickly, we decided that using random-access memory (RAM) would be a better choice than look-up tables (LUT). This is because LUTs are static and can only contain the values specified at the time of code generation. Therefore, in order to swap resource grids, the code would have to be regenerated and the synthesis, implementation, and bitstream generations steps would need to be repeated. By instead using RAM, the stored resource grid can be swapped by reading in a new configuration from MATLAB via the AXI-Stream interface.

There are a few limitations as to what grids can be loaded into the implemented design. At the time of code generation, a clock speed must be specified, therefore while SCS can be swapped on the fly, the sampling rate of the 5G waveform must remain the same. The sampling rate for the various supported 5G waveform configurations range from 1.92 MHz up to 61.44 MHz by powers of 2. These sample rate values are equivalent to the SCS times the OFDM modulator's FFT size. The supported FFT sizes rage from 128 to 4096, also by powers of 2.

In order to enable swapping resource grids, the array of cyclic prefix lengths fed in to the OFDM modulator had to also be configurable. This is because the number of cyclic prefix array elements is equivalent to the number of symbols per subframe and therefore changes with SCS.

To configure this array of lengths, a small additional RAM was used which is also loaded by the AXI-Stream interface. To arbitrate whether the grid RAM or the cyclic prefix RAM receives the input write data, a control flag was added which can be set with the AXI-Lite interface.

A number of other AXI-Lite control signals were added including the number of total elements and number of subcarriers in the loaded resource grid, the size of the of the OFDM modulator's Fast Fourier Transform (FFT), the number of cyclic prefix lengths in the array, a reset signal for the input FIFO, a write cyclic prefix flag, and a start transmitter signal.

To aid in debugging, an AXI-Stream interface was also used to write data from the FPGA to MATLAB. Additional control signals for this interface include a capture start flag, a capture length, and a select signal to choose what data to be written to MATLAB.

B. 5G Resource Grid Transmitter Operation

The operation of the transmitter begins by setting the control registers for the grid that needs to be loaded. Written grid and cyclic prefix data arrive from the AXI-Stream interface and passes through an input FIFO. Based on the status of the writeCyclicPrefix flag, that data either makes its way to the grid RAM or the cyclic prefix RAM. Counters and control logic set the address to which the data is written in the respective RAM.

Once both RAMs have their values populated, transmission can be enabled. With transmission enabled, another counter addresses the grid RAM and the IQ data for each grid element is read out sequentially. The valid flag for the grid data is monitored and a counter for the cyclic prefix watches for the transition from grid symbol to grid symbol. Based on the grid symbol corresponding to the current data from the grid RAM a cyclic prefix length is generated. This first half of the design is shown in Fig. 3.

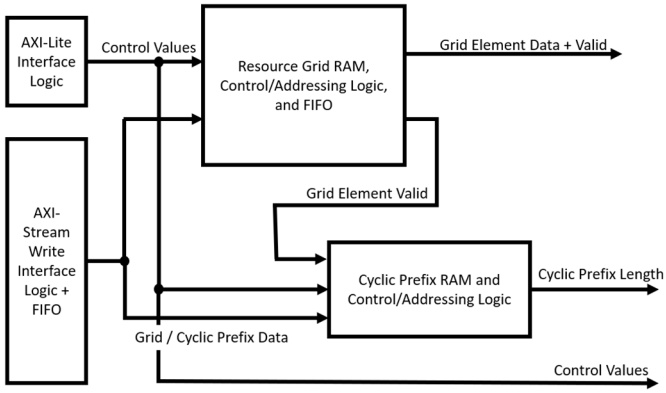

Fig. 3. First half of 5G Resource Grid Transmitter Design.

The grid data is then passed through another FIFO before entering the OFDM modulator Simulink block. This Simulink block is made up of three independent OFDM modulators, one for each possible SCS at the specified sample rate, each with a different value for their max FFT size parameter. Of the three OFDM modulators, only one is used at a time. This was done because it was found that if the max FFT size parameter of the OFDM modulator block differed from the actual FFT size, the OFDM modulator would produce cycles of invalid data to make up the difference in size. To ensure valid data is constantly streamed without interruption, the design swaps the active OFDM modulator based on the SCS of the loaded grid.

Following OFDM modulation, the waveform gets scaled by the square-root of the FFT size to undo the butterfly divisions during the OFDM computations. After scaling, the waveform's bit widths are adjusted and the signal is sent to the FMCOMMS 3 transceiver for transmission. This second half of the design is shown in Fig. 4.

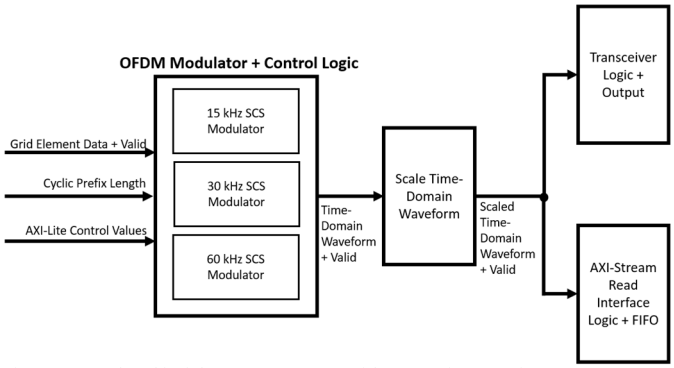

Fig. 4. Second Half of the 5G Resource Grid Transmitter Design.

C. *Testing of the 5G Resource Grid Transmitter*

After constructing all of the blocks and control logic, testing was done in Simulink. As was shown in Fig. 2, additional blocks were added outside the DUT to play the role of the AXI interfaces. To test the accuracy of the design, a 30 kHz SCS downlink waveform was produced using the 5G Toolbox's Waveform Generator application. This waveform was then configured to act as stimulus for the DUT Simulink model, fed in through the AXI-Stream write interface. The produced Simulink time domain waveform was then compared to the waveform produced by the application. A comparison plot of the results is shown in Fig. 5. Due to quantization caused by the fixed-point data types, there was a maximum error of 8.12e-3 between the Simulink result and the "golden reference."

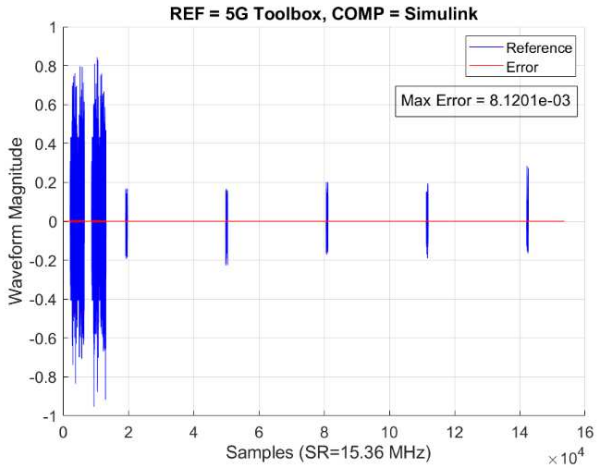

Fig. 5. Comparison between Simulink Waveform and 5G Toolbox Waveform.

Once the Simulink version of the design showed the desired behavior, the DUT was ran through the HDL Coder's workflow advisor and code was generated, synthesized, implemented, and a bitstream for the FPGA was generated. Following bitstream generation, the FPGA was programmed and the hardware was tested with the design. Using the AXI-Stream read interface for debugging, result waveforms were pulled directly from the FPGA and again compared to the 5G waveform generator "golden reference," shown in Fig. 6. Similar to the Simulink results, a small error between the hardware results and the 5G toolbox results was present.

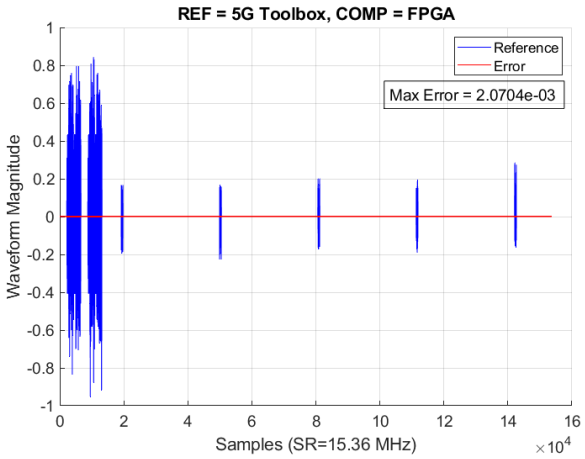

Fig. 6. Comparison between FPGA Waveform and 5G Toolbox Waveform.

Finally, through the use of the FMCOMMS 3 board, the waveform was modulated to a carrier frequency of 2.4 GHz and then transmitted. A secondary FMCOMMS 3 board was used to receive the transmit waveform. A comparison of the received waveform's resource grid to the original 5G toolbox waveform's resource grid can be seen in Fig. 7. Distortion due to channel effects can be seen, however the general timing and shape of the waveform remained the same.

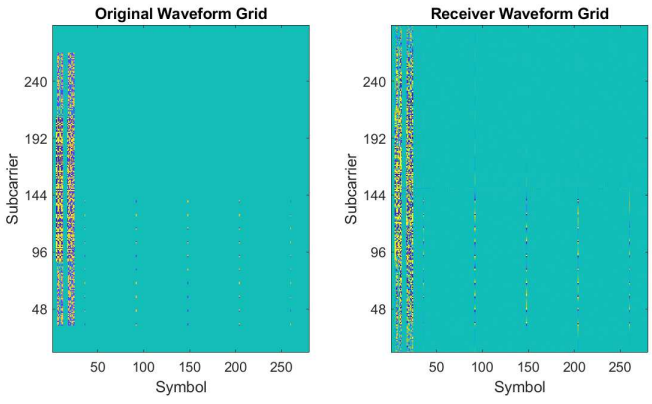

Fig. 7. Comparison between Received Waveform's Grid and the Original Waveform's Grid.

Although error existed in the Simulink, FPGA, and received waveform we determined that this was within acceptable levels for the continued use of this design. In the future, additional receive processing such as filtering, channel estimation, and

carrier offset estimation may be added to further refine the received waveform.

V. Conclusion

In this work, we explored MATLAB, Simulink, and HDL Coder as solutions to some of the challenges faced while trying to implement custom 5G applications in hardware. The capabilities of these tools were demonstrated using a custom 5G resource grid transmitter design. By exploiting the pre-built MATLAB functions and Simulink blocks, in conjunction with the HDL Coder, we experienced significant improvement in productivity and a speedup of the design timeline. In the future, we plan on experimenting with these tools further both in 5G and in other applications.

Acknowledgment

The authors would like to thank Jeff Miller from MathWorks for his support throughout this effort.

References

[1] Richter, Felix. "Global 5G Adoption to Hit One Billion in 2022." Digital image. June 22, 2022. Accessed July 13, 2023. https://www.statista.com/chart/9604/5g-subscription-forecast/

[2] 5G NR Physical channels and modulation (3GPP TS 38.211 version 15.2.0 Release 15), 2018. Accessed July 13, 2023. https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/15.02.00_60/ts_138211v150200p.pdf

[3] N. Ferdosian, S. Berri and A. Chorti, "5G New Radio Resource Allocation Optimization for Heterogeneous Services," in *2022 International Symposium ELMAR*, Zadar, Croatia, 2022, pp. 1-6, doi: 10.1109/ELMAR55880.2022.9899817.

[4] H. Nguyen and S. Nguyen, "FPGA-based Implementation and Evaluation of Realtime OFDM Phase Compensation in 5G," 2021 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 2021, pp. 131-134, doi: 10.1109/ATC52653.2021.9598312.

[5] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in Heterogeneous Computing," in *Scientific Programming*, vol. 18, Oct. 2010. doi: 10.3233/SPR-2009-0296.

[6] R. Nane et al., "A Survey and Evaluation of FPGA High-Level Synthesis Tools," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591-1604, Oct. 2016, doi: 10.1109/TCAD.2015.2513673.

[7] M. Enescu et al., "PHY Layer," in 5G New Radio: A Beam-based Air Interface, *Wiley*, 2020, pp.95-260, doi: 10.1002/9781119582335.ch3.

[8] W. Saad, M. Bennis and M. Chen, "A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems," in *IEEE Network*, vol. 34, no. 3, pp. 134-142, May/June 2020, doi: 10.1109/MNET.001.1900287.

[9] MathWorks, "Deploying 5G NR Wireless Communications on FPGAs: A Complete MATLAB and Simulink Workflow", *MathWorks, https://www.mathworks.com/campaigns/offers/deploying-5g-nr-on-fpgas-white-paper.html*

[10] Cousins, D. and Rohloff, K., "Accelerating Computations on Encrypted Data with an FPGA," Accessed July 13, 2023. https://www.mathworks.com/company/newsletters/articles/accelerating-computations-on-encrypted-data-with-an-fpga.html

[11] N. Othman, F. Mahmud, A. K. Mahamad, M. Hairol Jabbar and N. A. Adon, "Cardiac excitation modeling: HDL coder optimization towards FPGA stand-alone implementation," 2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014), Penang, Malaysia, 2014, pp. 507-511, doi: 10.1109/ICCSCE.2014.7072771.

[12] S. Spanò, L. Canese and G. C. Cardarilli, "Profiling of CNNs using the MATLAB FPGA-based Deep Learning Processor," 2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Villasimius, SU, Italy, 2022, pp. 121-124, doi: 10.1109/PRIME55000.2022.9816841.