

CP-OFDM PUSCH Model-Based Design for 5G New Radio Transmitter on ZCU216 RFSoc

James Craig¹, Louise H. Crockett¹, Robert W. Stewart¹, Ian Bowyer², and Garrey Rice²

¹Department of Electronic & Electrical Engineering, University of Strathclyde, Glasgow, Scotland

²MathWorks, Glasgow, Scotland

¹{j.craig, louise.crockett, r.stewart}@strath.ac.uk

²{ibowyer, grice}@mathworks.com

Abstract—5G New Radio (NR) aims to provide a technological solution to the growing demand for faster data rates and lower latency in mobile communications through its improved features and flexibility; however, this comes at the cost of increased design complexity when targeting hardware devices. To address this, high-level design tools such as Simulink can be used with iterative model-based design to shorten development cycles and reduce human error. This paper demonstrates this design flow through the implementation of a hardware-compatible NR Physical Uplink Shared Channel (PUSCH) model, targeting the Zynq UltraScale+ Radio Frequency System-on-Chip (RFSoc) ZCU216 development board. By interfacing with the hardware through MATLAB, the design supports multiple symbol modulation schemes, and parameters such as the number of layers, antenna ports, and selected precoding matrix can be altered. The design was tested by targeting a 10 MHz bandwidth, 60 kHz subcarrier spacing (SCS) waveform with 132 active subcarriers and met timing with 0.276 ns Worst Negative Slack (WNS) for a 245.76 MHz clock frequency, demonstrating its standard compliancy. The Cyclic Prefix Orthogonal Frequency Division Multiplexing (CP-OFDM) modulated signal generated on the board was looped back to MATLAB and verified against an existing software implementation, and the hardware usage was recorded.

Index Terms—5G mobile communication, Field programmable gate arrays, Model-driven development, New Radio, Physical layer

I. INTRODUCTION

The 5G New Radio (NR) standard was developed to address the increasing demands for higher bandwidth, faster data rates, and lower latency, and paves the way for mmWave transmissions. Its performance characteristics cater for three main categories of applications: Ultra Reliable Low Latency Communications (URLLC), Massive Machine Type Communications (mMTC), and Enhanced Mobile Broadband (eMBB).

To help organise the complex processing required in 5G NR, the standard splits the processing steps into multiple layers and channels. This paper focuses on the low physical (PHY) layer and, more specifically, on the Physical Uplink Shared Channel (PUSCH) which handles uplink data transmission.

A. Physical Uplink Shared Channel (PUSCH)

The PUSCH is responsible for taking data from the Uplink Shared Channel (ULSCH) and further processing it so that it

can be assigned to the available time and frequency elements that make up the resource grid. In order, the stages performed by the PUSCH are scrambling, symbol modulation, layer mapping, transform precoding, precoding, and resource block mapping. The resulting resource grid is then turned into a transmittable signal through Cyclic Prefix Orthogonal Frequency Division Multiplexing (CP-OFDM) modulation. Due to the heavy focus on flexibility in 5G NR, there are many customisable parameters and optional processing stages which may be included. An overview of the main processing steps is displayed in Fig. 1.

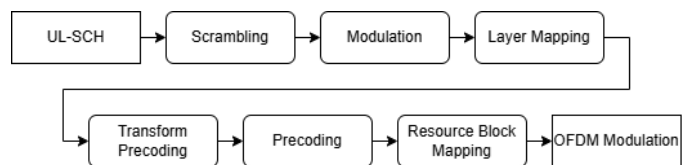


Fig. 1. Main Processing Stages in PUSCH.

Each of these stages is examined in more depth in Section II, where details of their implementation are discussed.

B. Radio Frequency System on Chip (RFSoc)

A common approach when developing 5G NR applications is to validate the initial design in a software environment, as the code can be altered easily and quickly. However, to progress the design towards a real-time implementation, it is often necessary to prototype the desired algorithm as a hardware design on a Field Programmable Gate Array (FPGA) or Radio Frequency System-on-Chip (RFSoc) device. By leveraging the advantages provided by these devices, such as the high level of parallelism provided by the programmable logic, efficient and cost-effective implementations can be achieved.

The Zynq UltraScale+ RFSoc ZCU216 development board is the ideal platform for prototyping such designs in hardware, due to its high-speed reconfigurable FPGA fabric [1]. The ZCU216 device also features multiple Radio Frequency (RF) Digital-to-Analogue Converters (DACs) and Analogue-to-Digital Converters (ADCs) which make it well suited for

This work was supported by MathWorks.

high frequency Multiple-Input Multiple-Output (MIMO) applications, such as those required for 5G NR wireless transmitters and receivers. Furthermore, a wide range of industry-standard development tools support the board, with MathWorks providing several reference designs which accelerate prototyping, leading to the selection of this device for development purposes.

C. Literature Review

Since before its standardisation, 5G has been a focus of academic research, being a logical continuation of the previous generations of wireless mobile communications. Many papers have been published proposing technologies to support it or discussing implementation details as various aspects of the NR standard become finalised. In [2], the authors consider a software prototype of a 5G NR transceiver with a focus on the Low Density Parity Check encoder and decoder, limited to 2 by 2 MIMO Spatial Multiplexing (SM).

In [3], the authors implement a 5G NR Radio Access Network (RAN) PHY-layer Distributed Unit (DU) and Radio Unit (RU) on an FPGA device through MATLAB and Simulink, part of which consists of a PUSCH receiver design. They test this design with signals captured in their laboratory and from signals provided by 5G Toolbox, however, if they also had a PUSCH transmitter design then this could also be deployed to the FPGA which would allow a greater degree of control over the signal transmission for testing various uplink scenarios.

Both [4] and [5] consider FPGA implementations of 5G NR data channels, the Physical Downlink Shared Channel (PDSCH) and the Physical Uplink Control Channel (PUCCH) respectively, and demonstrate the benefits of creating hardware designs for these aspects of 5G NR. The PUSCH can similarly be implemented on hardware, but examples of this are missing from the literature.

The process of converting a software algorithm into a hardware-compatible design is often a complex process due to the technical characteristics of the target device, and this problem is compounded by the complex and flexible computations that are included throughout the 5G NR standard. Manually programming these algorithms in a low-level Hardware Description Language (HDL) is a lengthy process prone to human error, and so it is desirable to develop and model them in a high-level environment before deployment.

Many papers such as [5] and [6] utilise HDL Coder to automatically generate HDL from high-level block-based designs in Simulink, but without thorough testing in simulation this can lead to errors in the generated bitstream which can be time consuming and costly to fix. The effectiveness of the MATLAB-Simulink-HDL Coder workflow for 5G designs is highlighted in [6]. There are also examples that use the PYNQ software framework, such as in [4], where the authors develop a 5G NR PDSCH transceiver.

D. Contributions

This paper demonstrates the novel implementation of a CP-OFDM PUSCH design, targeted at the Zynq UltraScale+

RFSoc ZCU216 development board, which leverages the benefits of Software Defined Radio (SDR) to maintain flexibility, parameterised through MATLAB, while also achieving an efficient, single-chip solution that can be used to generate waveforms for testing various uplink scenarios.

This design is used to showcase the reduction in development time and increase in productivity generated by utilising a model-based design workflow featuring MATLAB, Simulink, and HDL Coder to refine a system design in simulation early in the development process. By employing an iterative, concurrent design methodology, bitstream generation and redesign time can be minimised which allows applications to be rapidly developed and deployed to hardware. This is particularly beneficial in the creation of 5G NR designs due to the high level of complexity and flexibility present in such systems.

The Simulink model and accompanying MATLAB scripts have been made available for reference, see [7] and [8], and can be used to generate CP-OFDM PUSCH signals to test 5G NR uplink scenarios or can be incorporated into a larger 5G NR transceiver application. Through this research, the timing and resource utilisation of the design is evaluated, and key design takeaways are highlighted.

The remainder of this paper is structured as follows: Section II explains the overall design flow used throughout the development work, and how this approach was applied to creating a hardware-compatible CP-OFDM PUSCH design. The steps taken to verify and test the design are covered in Section III, and further discussion and evaluation of the design is provided in Section IV, alongside a summary of the generated results. The paper is concluded and the key takeaways are highlighted in Section V.

II. METHODOLOGY

A. Design Flow

To ensure that the model was operating correctly throughout the development phase, an iterative design approach was adopted based on the MATLAB, Simulink, and HDL Coder workflow recommended by MathWorks [9]. An overview of the design flow is shown in Fig. 2.

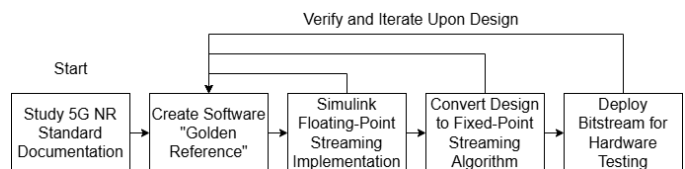


Fig. 2. Overview of Design Flow.

The design is split into subsystems based on domain knowledge of the 5G NR standard documentation relating to the PHY layer and, more specifically, the PUSCH. This enables modular testing of the design as the inputs and outputs of each subsystem can be monitored, and this is extremely useful for narrowing the search for errors when debugging.

Using model-based system engineering, a software implementation of the PUSCH is created in MATLAB using

standard-compliant functions from 5G Toolbox. This is used as a “Golden Reference” throughout the design process, and the outputs from each subsystem are compared against it to ensure that they are within error tolerances.

To introduce the concepts of parallelism and sample time to the design, a floating-point streaming algorithm is developed in Simulink. This prepares the design for hardware implementation, but also has the benefits of quick simulation which allow errors to be identified and fixed quickly. Since generating bitstreams from hardware designs is a complex and lengthy process, any issues in the design at the deployment stage can be costly to fix. To help identify and resolve these issues early in development, the intermediate outputs are exported to MATLAB using “To Workspace” Simulink blocks within valid enabled subsystems, and are verified against the software reference.

Fixed-Point Designer is an exceptionally useful tool when converting the design from a floating-point streaming algorithm to a fixed-point design for implementation on an FPGA [10]. After running a simulation of the full-precision design, it can provide an overview of quantisation effects such as overflows and precision loss, and recommends data types which can be automatically applied. It also generates tables and histograms showing the data ranges for each point in the design, which is vital for gaining an understanding of the data flow through the model. An example of the histograms generated by Fixed-Point Designer is shown in Fig. 3.

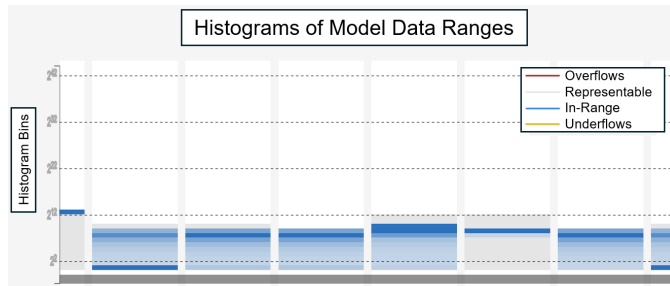


Fig. 3. Fixed-Point Designer Simulation Data Range Visualisation.

With the design converted to fixed-point, the outputs are compared with both the floating-point and software versions to ensure that they are still in alignment. By developing these designs concurrently then any issues in the model are identified quickly, and the modular approach allows the exact location of errors to be isolated efficiently. Since the design uses many of the HDL compatible blocks provided by MathWorks, then HDL code is easily generated from the design using HDL Coder.

The bitstream and host interface scripts are also generated from the design using HDL Coder. This bitstream is then deployed to the board for hardware testing, with the host interface script allowing the design to be parameterised from MATLAB over the AXI4-Lite and AXI4-Stream interfaces to set the various PUSCH parameters, and to load it with the desired codeword at run time.

B. CP-OFDM PUSCH Design

The PUSCH was selected from the 5G NR standard as a suitable section to implement as it is under-represented in the literature while featuring many computations that could benefit from hardware implementation. The details of the PUSCH are found in the Technical Specification (TS) 38.211 [11] which is developed by the Third Generation Partnership Project (3GPP). The model is split into various subsystems based on the required processing stages and an overview of the design in Simulink can be seen summarised in Fig. 4.

The input to the PUSCH is a stream of bits from the UL-SCH after they have been processed using Forward Error Correction (FEC) and rate matching. This is simulated by the input handling subsystem which paces the data required for all the following subsystems by toggling a valid signal. Using the same clock rate across the design, the correct data rate can be ensured at the output by varying the valid duty cycle. This handles the increased data requirements of higher symbol modulation schemes and increased number of layers configured by the user at runtime. This is discussed in further detail in Section III.

The first processing step of the PUSCH is scrambling which introduces a pseudo-random element to the codeword to ensure uniform power distribution across time and frequency resources. This also provides data privacy, and assists in interference management by providing a more accurate channel estimation. The pseudo-random sequence is defined by a length-31 Gold sequence which is initialised with a value based on the Radio Network Temporary Identifier (RNTI), Network Identification (NID), and Random Access Preamble Index (nrapi), each of which can be altered at runtime from the host interface scripts in MATLAB.

Following scrambling, codeword modulation converts the scrambled codeword into a stream of complex-valued symbols suitable for wireless transmission using either QPSK, 16QAM, 64QAM, 256QAM, or $\pi/2$ -BPSK, depending on the runtime parameters selected in MATLAB. The choice of modulation scheme affects the data rate and spectral efficiency of the transmission as a trade-off against the signal’s robustness to noise and interference. The “NR Symbol Modulator” block from Wireless HDL Toolbox is used to accelerate implementation [12].

Layer mapping then directs the symbols into one to four parallel streams for transmission from multiple antennas based on the number of selected layers, which enables MIMO techniques such as precoding and beamforming. This adds spatial diversity to the signal by utilising multipath to reduce the effects of fading.

Once the data is mapped to a number of layers, there is an optional processing stage of transform precoding, which adapts the data to a form suitable for Discrete Fourier Transform-Spread-Orthogonal Frequency Division Multiplexing (DFT-s-OFDM). This stage was not included in the design to reduce complexity and so it will not be discussed further in this paper.

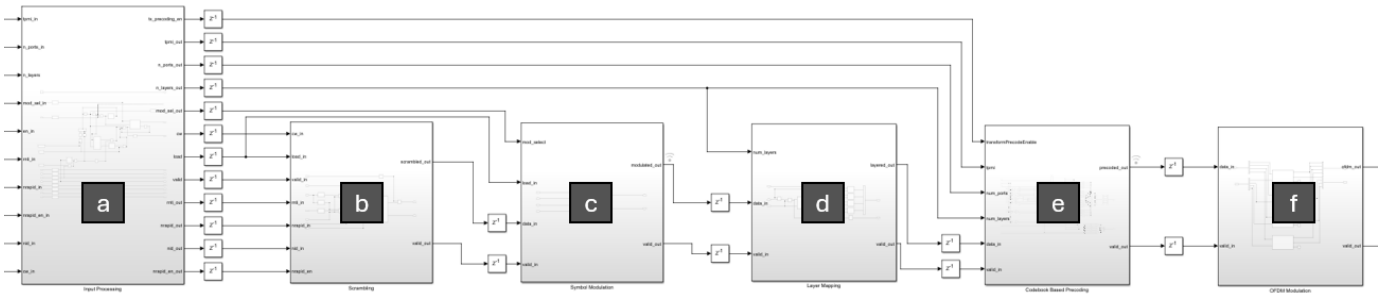


Fig. 4. PUSCH Design in Simulink showing Subsystem Splits between: (a) Input Processing, (b) Scrambling, (c) Symbol Modulation, (d) Layer Mapping, (e) Codebook-Based Precoding, (f) CP-OFDM Modulation.

Next, the precoding stage multiplies the data stream by the selected precoding matrix to transform the data from parallel streams based on layers to streams related to the number of antenna ports. The approach in this paper focuses on codebook-based precoding, where the precoding matrix is selected from a set of predefined matrices based on a transmitted Precoding Matrix Indicator (PMI), but there also exists the non-codebook based precoding approach where the precoding matrix is generated based on an estimation of the channel from the channel state information reference signals.

For example, 1 layer can be mapped to 4 antenna ports to improve signal robustness to channel effects, or 4 layers can be mapped to 4 antenna ports to prioritise data throughput. The number of layers and antenna ports enabled, as well as the selected PMI, can be altered in MATLAB at runtime through the use of the host interface scripts.

Following precoding, the generated symbols are mapped to virtual resource blocks, and then to physical resource blocks. To conform to transmit power requirements, the symbols are multiplied by an amplitude scaling factor, and are then mapped to the resources allocated for PUSCH transmission. For this design, the data pacing in the input processing subsystem handles the allocation of PUSCH data to resource elements. This means that the resource grid can be generated seamlessly, with subframes being generated per enabled antenna port.

Once the PUSCH has been generated, CP-OFDM is used to process the resource grid into the uplink signal, with one CP-OFDM PUSCH symbol being generated per column of each resource grid. The “Introduction to Custom OFDM” example provides a reference design for the process, and the “OFDM Modulator” block from Wireless HDL Toolbox is used to reduce development time [12]. The CP-OFDM PUSCH symbols generated on the hardware are then looped back to MATLAB using the AXI4-Stream protocol for decoding and verification through comparison against the software reference, although in a deployment scenario they would instead be interpolated and sent to the DACs to be transmitted through a loopback cable or over a wireless RF channel. The output signal consists of a 16-bit signed complex sample, with 14-bit fractional precision, per antenna port, and so to transmit this

data over the AXI-Stream interface it is concatenated into a single 128-bit stream.

III. VERIFICATION AND TESTING

Throughout the design process, a concurrent, modular design approach was taken, with the output from each subsystem compared against the respective stage of the PUSCH software reference. For the fixed-point model, the absolute error is calculated to ensure it falls within tolerances for quantisation and the additional difference caused by propagating those errors throughout the rest of the design. As an additional check, the generated CP-OFDM signal is provided as an input to a software decoder to verify that the signal can be successfully decoded and matched to the initial input codeword.

For testing, the design targets a 10 MHz bandwidth 60 kHz Subcarrier Spacing (SCS) waveform with 132 active subcarriers (11 resource blocks) in 256-point Fast Fourier Transform (FFT) as a proof of concept, but the FFT size and number of active subcarriers could be increased by modifying the design in future iterations. The use of a 60 kHz SCS with extended cyclic prefix mode was selected to streamline the design process to establish the initial system, but future work could include the option for regular cyclic prefix mode which would allow alternate SCS to be used (this would require changes to the CP-OFDM system so that variable cyclic prefix lengths could be defined). Due to the more relaxed constraints of lower SCS, the use of 60 kHz SCS confirms that their requirements could be effortlessly met, and the board has sufficient capability to target higher SCS if required.

After thorough testing to ensure that the design operates as expected in simulation for both floating-point and fixed-point arithmetic, HDL Coder is used to generate the hardware design which is then loaded onto the Zynq UltraScale+ RFSoc ZCU216 development board for hardware testing. By utilising HDL Coder Support Package for AMD RFSoc Devices and the generated host interface scripts, input parameters such as the modulation scheme and number of layers are directly written to registers using the AXI4-Lite protocol at runtime, and a desired codeword and the resulting CP-OFDM PUSCH signal are streamed to and from the board to MATLAB using

the AXI4-Stream interfaces. The hardware setup is shown in Fig 5.



Fig. 5. Experimental Setup showing Zynq UltraScale+ RFSoc ZCU216 and Host PC.

For the hardware implementation, additional care is taken to ensure that the correct data rates are achieved at each point in the system so that the data throughput desired at the transmitter can be achieved regardless of previous bottlenecks in the system. This is achieved by pacing the data through the input processing subsystem with a valid signal so that a consistent rate is achieved at the output.

First, the total required input data for a single subframe is calculated based on the bits per symbol (B) for the selected symbol modulation scheme and the number of layers (N_l). Then, the input rate (f_i) is calculated based on the difference between the device clock rate (f_c) and the required data rate at the output. For example, a target OFDM output sample rate of 15.36 MHz means that with a clock rate of 245.76 MHz, the oversample factor (R) is 16. The input rate is then calculated using (1).

$$f_i = \frac{B \times N_l}{R} \times f_c \quad (1)$$

To simplify the design, the entire model operates at the device clock rate but uses a valid signal to pace the data based on the configured parameters. When the design is operating with pi/2-BPSK modulation and only one active layer, valid data is only input once every 16 clock cycles, whereas with 64QAM modulation and 4 active layers then the input valid data rate is equal to the device clock rate.

Additionally, to ensure that consistent data is produced at the output of the CP-OFDM subsystem, the data is paced so that the subsystem receives enough data for one CP-OFDM symbol and then is allowed sufficient samples to add the cyclic prefix and guard band before subsequent data is sent. This pacing is also handled by the input processing subsystem, and is combined with the previously mentioned rate calculations through the use of several counters which keep track of the oversample factor and CP-OFDM requirements.

IV. DISCUSSION AND RESULTS

Throughout testing, it was ensured that the maximum absolute difference in the generated CP-OFDM PUSCH complex symbols was kept below 2^{-11} , and that the average absolute difference was kept below 2^{-13} . A binary input codeword is generated with enough data to generate a subframe of PUSCH symbols for the selected modulation scheme and number of antenna ports. This is sent to the development board, through the design, and returned to MATLAB over the AXI4-Stream interface where it is then compared against the 5G Toolbox software reference. The resulting maximum absolute difference was found to be 3.59×10^{-4} and the resulting average absolute difference was found to be 1.09×10^{-4} , equivalent to $2^{-11.44}$ and $2^{-13.17}$ respectively. As a visual representation, the first complex CP-OFDM PUSCH symbol across four antenna ports for the two signals were compared and are shown in Fig 6.

The design has a 0.276 ns Worst Negative Slack (WNS) when targeting a clock frequency of 245.76 MHz, and the resource utilisation of the final design can be seen in Table I.

TABLE I
RESOURCE UTILISATION OF HARDWARE DESIGN (XCZU49DR).

Resource	Utilisation	Available	Utilisation %
LUT	38622	425280	9.08
LUTRAM	2665	213600	1.25
FF	48798	850560	5.74
BRAM	46.50	1080	4.31
DSP	24	4272	0.56
IO	76	408	18.6

As can be seen from Table I, the design only takes up a reasonable 9.08% of the device's Lookup Tables (LUTs), 5.74% of the Flip-Flops (FFs), and 0.56% of the available Digital Signal Processing (DSP) slices, demonstrating that there is still ample space for further development.

The design features many avenues for further expansion and could act as a reference design for any future development work in the field with the possibility to add a variable or increased FFT size, variable SCS and active subcarriers, more flexible MIMO precoding and transform precoding options, and regular cyclic prefix. In addition, due to the modest resource utilisation of the design it could be further expanded to include previous processing steps in the 5G NR chain such as FEC or the ULSCH, or paired with an uplink receiver system to test the design further through loopback or over-the-air scenarios. Finally, a more complete uplink transmitter design could be achieved by including additional 5G NR data channels and reference signals such as the PUCCH or Demodulation Reference Symbols (DM-RS) to the resource grid.

V. CONCLUSION

In conclusion, it has been shown that a CP-OFDM PUSCH signal can be successfully generated with a low resource utilisation implementation, using only 9.08% of the LUTs,

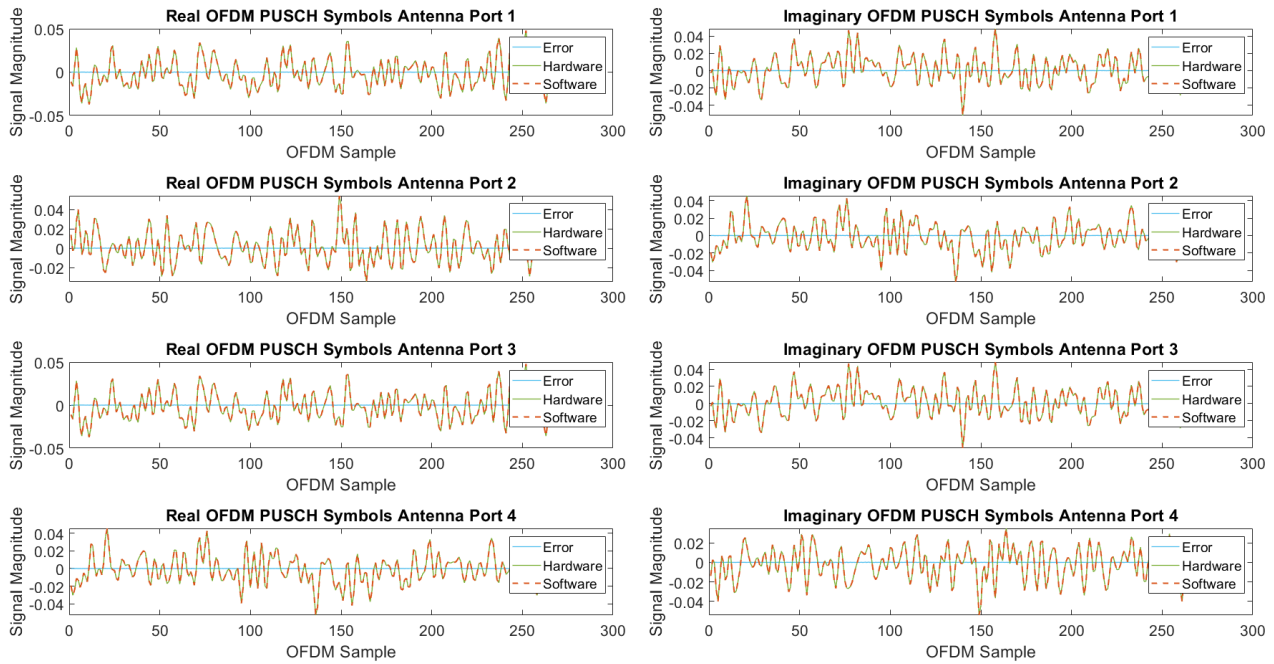


Fig. 6. Comparison of Complex Signal Generated on Hardware against Software Reference.

5.74% of the FFs, and 0.56% of the DSPs available on the XCZU49DR device. The design has been shown to achieve a maximum operation frequency of 245.76 MHz with 0.276 ns WNS when targeting the ZCU216 development board, while maintaining the flexibility to alter parameters such as the modulation scheme and number of antenna ports from MATLAB. This makes it an invaluable tool for generating test waveforms for a variety of uplink MIMO scenarios, and the design has been made available for reference, see [7] and [8].

This paper also demonstrates the effectiveness of the MATLAB, Simulink, and HDL Coder workflow by progressing the design from an algorithm based on the standards documentation, to a software reference, to a deployable hardware design. By adopting an iterative, modular, model-based workflow then any issues were easily identified and rectified without requiring lengthy debugging which cut down on overall development time.

ACKNOWLEDGMENT

The authors would like to extend their gratitude to the StrathSDR team at the University of Strathclyde, and to MathWorks for their support and guidance during the work detailed in this paper.

REFERENCES

- [1] "Zynq UltraScale+ RFSoc ZCU216 Evaluation Kit," AMD, 2025. <https://www.xilinx.com/products/boards-and-kits/zcu216.html> (accessed Feb. 09, 2025).
- [2] G. Cisek and T. P. Zieliński, "Prototyping Software Transceiver for the 5G New Radio Physical Uplink Shared Channel," 2019 Signal Processing Symposium (SPSypm), Krakow, Poland, 2019, pp. 150-155, doi: 10.1109/SPS.2019.8882079.
- [3] F. D. L. Coutinho, J. D. Domingues, P. M. C. Marques, S. S. Pereira, H. S. Silva and A. S. R. Oliveira, "Towards the Flexible and Efficient Implementation of the 5G-NR RAN Physical Layer," 2021 IEEE Radio and Wireless Symposium (RWS), San Diego, CA, USA, 2021, pp. 130-132, doi: 10.1109/RWS50353.2021.9360353.
- [4] C. -Y. Chang and H. -T. Chou, "FPGA Implementation of 5G NR PDSCH Transceiver for FR2 Millimeter-wave Frequency Bands," 2022 IEEE 4th Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan, 2022, pp. 60-63, doi: 10.1109/ECICE55674.2022.10042864.
- [5] Y. -H. Kim, H. Ju, I. -J. Chun, C. B. Jeong and M. -S. Lee, "Design of Low-latency Synthesizable PUCCH Demodulation Unit Using Simulink HDL Coder," 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, Republic of, 2021, pp. 1387-1389, doi: 10.1109/ICTC52510.2021.9620844.
- [6] J. Geyster, K. Gettings, P. Monticciolo and M. Rebholz, "Leveraging Mathworks Tools to Accelerate the Prototyping of Custom 5G Applications in Hardware," 2023 IEEE High Performance Extreme Computing Conference (HPEC), Boston, MA, USA, 2023, pp. 1-6, doi: 10.1109/HPEC58863.2023.10363538.
- [7] J. Craig, "5G NR CP-OFDM PUSCH Generation for RFSoc," Mathworks.com, <https://mathworks.com/matlabcentral/fileexchange/180093-5g-nr-cp-ofdm-pusch-generation-for-rfsoc> (accessed Feb. 09, 2025).
- [8] J. Craig, "CP-OFDM PUSCH Generation for 5G New Radio Transmitter on ZCU216 RFSoc Device Dataset," University of Strathclyde, PURE, 2025, doi: 10.15129/ceaddf01-d2f4-46fa-8894-79ba8fc3004f
- [9] "HDL Coder," Mathworks.com, 2025. <https://mathworks.com/products/hdl-coder.html> (accessed Feb. 09, 2025).
- [10] "Fixed-Point Designer," Mathworks.com, 2025. <https://mathworks.com/products/fixe-point-designer.html> (accessed Feb. 09, 2025).
- [11] 3GPP, "Physical Channels and Modulation," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.211, 2020, version 16.2.0.
- [12] "Wireless HDL Toolbox," Mathworks.com, 2025. <https://mathworks.com/products/wireless-hdl.html> (accessed Feb. 09, 2025).