


# Automated performance-based design technique for an efficient LTE PDSCH implementation using SDSoC tool

Mohamed Eladawy<sup>1</sup> | Mahmoud Mostafa<sup>1</sup> | M. Sameh Said<sup>1</sup> | Hassan Mostafa<sup>1,2</sup> 

<sup>1</sup>Department of Electronics and Communications Engineering, Cairo University, Giza, Egypt

<sup>2</sup>University of Science and Technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt

## Correspondence

Hassan Mostafa, University of Science and Technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt.  
Email: hmostafa@uwaterloo.ca

## Summary

System on a chip (SoC) creates massive design challenges for SoC-based designers. The design challenges start from functional, architectural verification complexity and finally meeting performance constraints. In addition, heterogeneity of components and tools introduces long design cycles. The Software-Defined System-on-Chip (SDSoC) developed by Xilinx is used to create custom SoC on a heterogeneous FPGA-CPU platform. The SDSoC tool provides fast, flexible, and short design cycle to develop heterogeneous FPGA-CPU platform. The objective of this paper is to introduce a new automated design technique to build a SoC on a heterogeneous FPGA-CPU platform that meets design requirements using SDSoC tool. In this paper, the typical SDSoC design flow is introduced. In addition, a new automated SDSoC design technique is developed to design SoC on a heterogeneous FPGA-CPU platform on the basis of performance metrics such as area, power, and latency. Design of physical downlink shared channel (PDSCH) in long-term evolution (LTE) is presented as a case study. This paper provides the implementation of the transmitter and the receiver of the PDSCH in LTE using SDSoC tool and selects a platform that meets performance metrics constraints.

## KEYWORDS

FPGA, LTE, PDSCH, SDSoC, SoC, Xilinx

## 1 | INTRODUCTION

Smart homes, automated vehicles, and Internet of things (IoT) are examples of electronic products that are almost involved in every aspect of our lives. The rapid growth of the electronics industry encouraged developers to find faster and more efficient design methods to decrease time-market and development cycle. For such requirement, the system on chip (SoC) design approach is preferred to application-specific integrated circuits (ASICs). The SoC's designer should build a product with an efficient architecture to ensure that system design meets performance requirements. Designing efficient architecture might consume a lot of time with complicated long design cycle. Xilinx announced Software-Defined System-on-Chip (SDSoC) tool targeting developing SoC on a heterogeneous FPGA-CPU platform.<sup>1</sup> The SDSoC is a novel development tool used to create hardware-software co-design on a heterogeneous FPGA-CPU platform. The SDSoC tool makes simpler and shorter development cycle to implement heterogeneous FPGA-CPU platform as well as generate required hardware interface logic to handle the data flow between hardware and software automatically. In addition, SDSoC tool includes system-level profiling and performance analysis capability. The profiling and performance analysis tool includes hardware utilization calculation, latency calculation, and hardware acceleration

improvement estimation. A detailed description of the features of the tool is provided in a user guide.<sup>2</sup> The SDSoC is integrated with a high-level synthesis (HLS) tool, which is used to implement hardware logic on FPGA synthesized from a C/C++ language description. Transforming C/C++ code to an RTL implementation process using HLS is provided previously.<sup>3</sup> In recent decades, the mobile wireless communication system has been changing dramatically. The fast growth of mobile phone market promises the 3rd Generation Partnership Project (3GPP) to develop the long-term evolution (LTE) standard for high-speed wireless communication for mobile devices.<sup>4</sup> The target of the LTE standard is to increase efficiency spectrum utilization, improve system capacity, and increase data rate.<sup>5</sup> In this paper, hardware-software co-design of the LTE physical downlink shared channel (PDSCH) transmitter and the receiver synthesized by SDSoC for heterogeneous FPGA-CPU platform is proposed. The rest of this paper is organized as follows. Section 2 gives an overview about SDSoC tool and explains the proposed automated performance-based design technique. Section 3 illustrates the architecture of the LTE PDSCH transmitter and the receiver chain used in this paper. Section 4 shows the implementation of the LTE PDSCH transmitter and receiver using SDSoC tool. Section 5 shows the experimental results. Section 6 shows the state-of-art and future work. The conclusion is shown in Section 7.

## 2 | SOFTWARE-DEFINED SYSTEM-ON-CHIP

SDSoC is a C/C++ development environment for implementing heterogeneous embedded systems using the Zynq Programmable SoC platform. The SDSoC tool is integrated with an HLS, which is used to implement hardware in an FPGA synthesized from a C/C++ language description. Transforming C/C++ code to an RTL implementation process using HLS is provided in detail in a user guide.<sup>2</sup> The designer specifies certain C/C++ functions to be implemented either as a software function or a hardware-acceleration function to enhance the performance of the system.

In the SDSoC environment, the generation of SoC on a heterogeneous FPGA-CPU platform is controlled by inserting pragmas into the C/C++ source code to guide the system compiler and to control the number of data elements that are transferred to/from the hardware function. All SDSoC pragmas are prefixed with #pragma and should be inserted into C

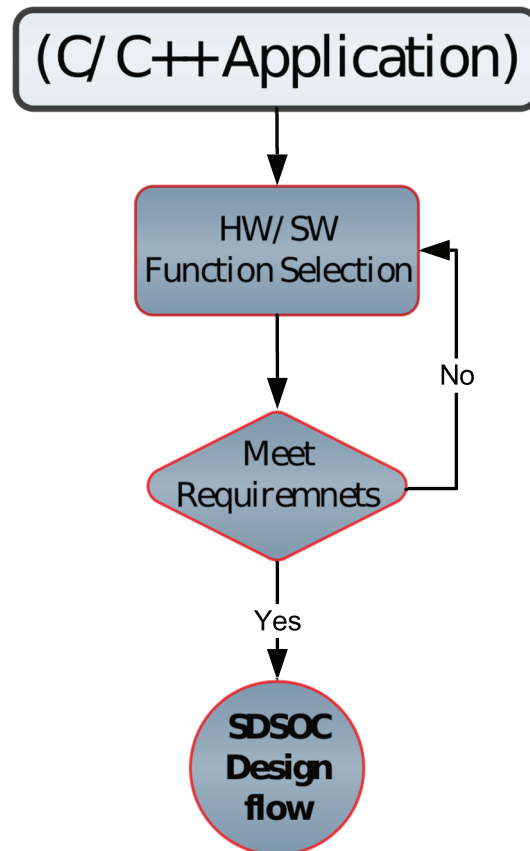


FIGURE 1 Typical HW/SW co-design flow

/C++ source code either immediately before a function declaration or at a function call site depending on the type of pragmas. The pragmas include the following types: data access patterns, data transfer size, memory attributes, data mover type, asynchronous function execution, and hardware/software tracing. All details of the SDSoC pragmas types are provided in a reference guide.<sup>6</sup>

Designing systems consisting of a lot of functions makes it too hard for the designer to determine which functions should be implemented as software function and which functions should be implemented as hardware-accelerator especially when having constraints on the design performance such as area, power, and latency. In Section 2.1, the typical HW/SW co-design flow using SDSoC tool is illustrated. In Section 2.2, a proposed automated SDSoC design technique is introduced to select a SoC platform according to performance metrics constraints.

### 2.1 | Typical HW/SW co-design flow using SDSoC design flow

The typical HW/SW co-design flow using SDSoC is shown in Figure 1. First, the developer should design the application coded in C/C++. Next, the designer should select manually which functions must be implemented as software functions or implemented as hardware-accelerated functions synthesized by Vivado HLS.

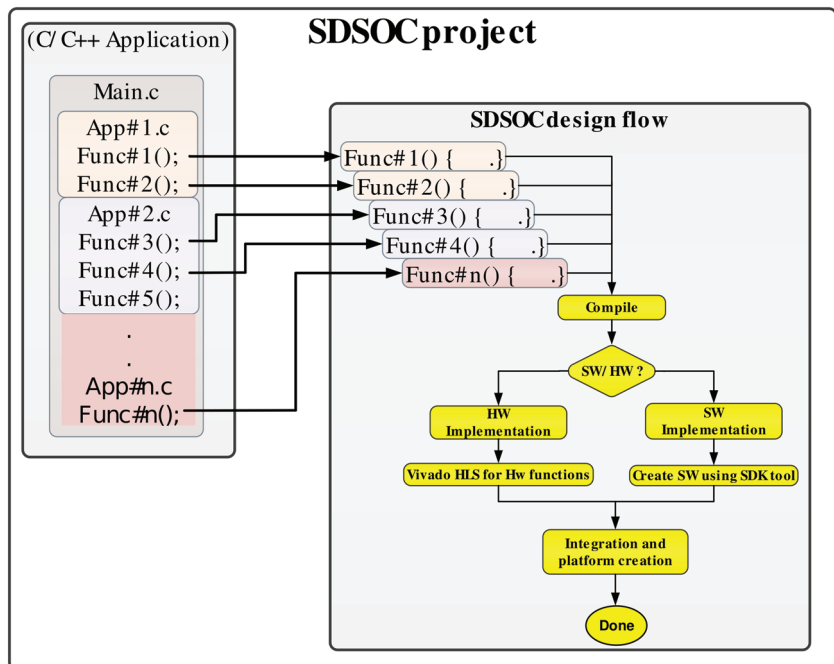


FIGURE 2 Software-Defined System-on-Chip (SDSoC) design flow

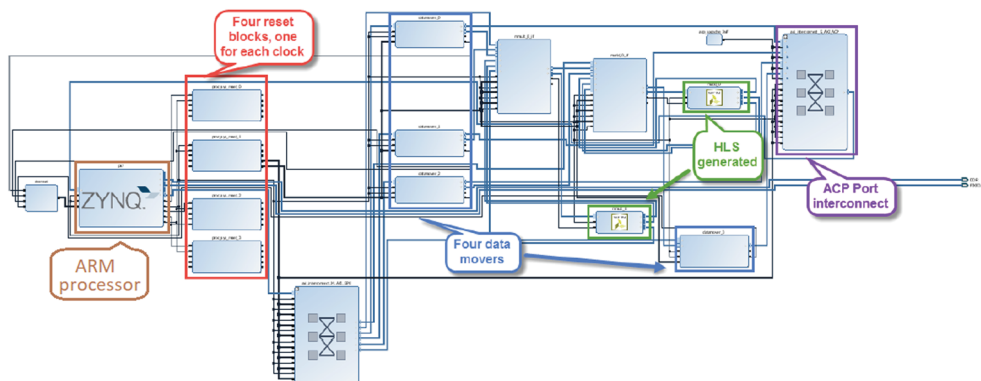


FIGURE 3 Embedded FPGA platform

After refining all C/C++ functions, the SDSoC design flow is executed.<sup>7</sup> The SDSoC design flow is shown in Figure 2. First, all C/C++ functions are compiled, Next, the implementation of the C/C++ functions is partitioned into software implementation functions or hardware-accelerated functions depending on the designer selection.

The Vivado tool (Xilinx Inc) and Software Development Kit (SDK) tool are the elements of hardware and software system design, respectively. The Vivado tool includes an HLS tool used for creating the hardware system components by transforming the C/C++ code to an RTL implementation. The SDK tool is a software design suite that includes driver support, C/C++ compiler library, and tools for debugging and profiling. Finally, for integration, the necessary communication blocks between hardware and software are generated automatically by the SDSoC tool, and SoC platform is created by SDSoC tool.

The SDSoC tool generates the embedded FPGA SoC platform as shown in Figure 3. This platform allows executing parts of the C/C++ code on the ARM processor as software functions and the other parts of the C/C++ codes on the FPGA as hardware-accelerated functions. The embedded FPGA platform consists of three parts: First, the processing system includes dual-core ARM cortex-A9 processor hardware processor. Second, interfacing logic includes ACP port interconnect and data movers. Third, hardware logic includes HLS generated block.

## 2.2 | Proposed automated performance-based design technique

This section illustrates in detail the proposed automated performance-based design technique using SDSoC tool. The objective of this technique is to determine platforms that achieve performance metrics and to select the platform that achieves the best performance. Figure 4 shows the proposed automated performance-based design technique flowchart.

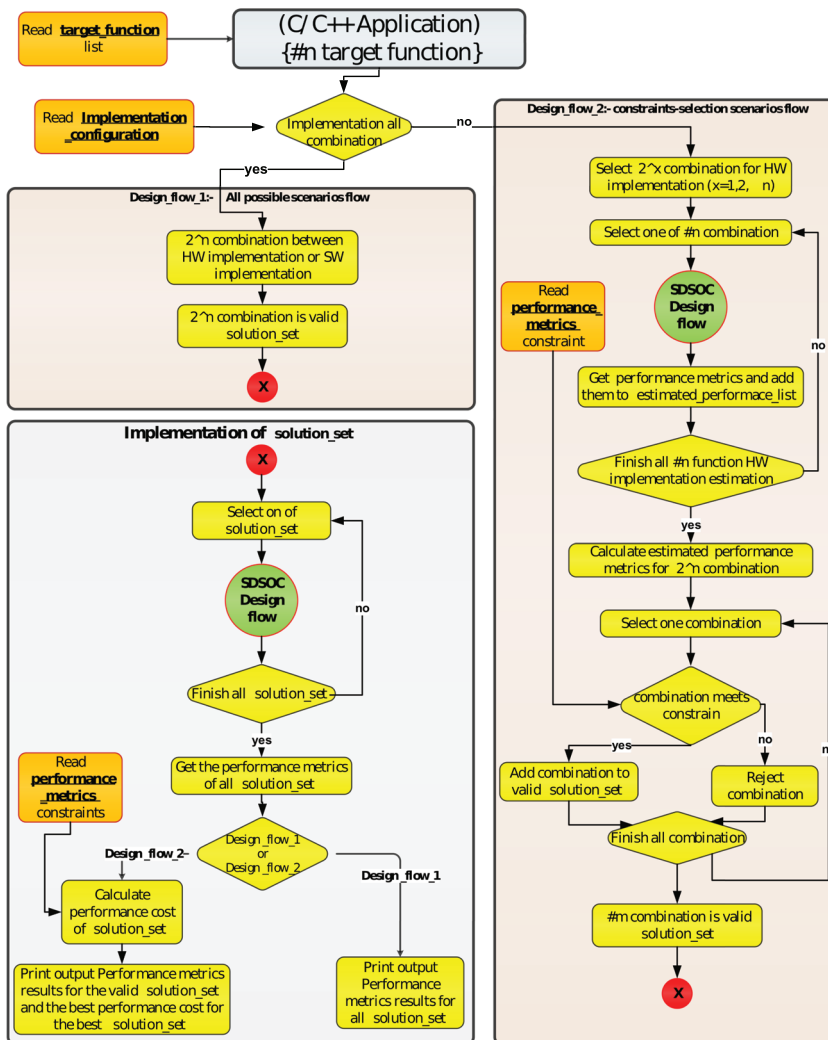


FIGURE 4 Proposed performance-based adaptive design technique

**TABLE 1** Three-function configuration scenario example

Platform Name	fun1	fun2	fun3
platform0	0	0	0
platform1	1	0	0
platform2	0	1	0
platform3	1	1	0
platform4	0	0	1
platform5	1	0	1
platform6	0	1	1
platform7	1	1	1

(0) Software function and (1) hardware-accelerated logic.

**TABLE 2** Estimated performance list for selected combination

Performance Metrics	combination1	combination2	combination4
Hardware utilization	h1	h2	h4
Dynamic power	p1	p2	p4
Latency	l1	l2	l4

The introduced algorithm is fully automated and uses a set of shell scripts for executing the SDSoC tool to generate the required platforms. There are three flow control files defined in the technique used to drive the execution of the shell scripts according to designer requirement. The flow control files are the following: First, the `target_function_list` file contains the modules functions names that were designed to be implemented either as software function or hardware-accelerated function; second, the `implementation_configuration` file used to define the type of FPGA device, operating system, the clock frequency, and type of design flow; third, the `performance_metrics_constrain` file includes constraints on hardware utilization, dynamic power, and latency. The developer should design the application code written in C/C++ and define the  $n$  functions in the `target_function_list` files, which are the functions that could be implemented either as a software function or as a hardware-accelerated function (for example, the designer defines three functions, which are fun1, fun2, and fun3). Next, the shell scripts read all C/C++ files and compile all C/C++ files for any syntax errors or semantic error. Then, the shell scripts read the `implementation_configuration` file. As shown in Figure 4, the technique is divided into two design flows depending on the setting of the `implementation_configuration` file. The first is `design_flow_1` (all possible scenarios flow), and the second is `design_flow_2` (constrained-selection scenarios flow).

The purpose of the `design_flow_1` is to implement all possible scenarios for ( $n$ ) function defined in the `target_function_list` file, thus  $2^n$  combinations between software implementation function or hardware-accelerated function are added to a valid `solution_set` list. For example, the three defined functions in the `target_function_list` generate  $2^3=8$  combinations between software implementation function or hardware-accelerated function as shown in Table 1. The number (1) in the table indicates that this function in current platform is a hardware-accelerated logic, and the number (0) indicates that this function in current platform is a software function. Therefore, platform0 is configuring that all the functions (fun1, fun2, and fun3) are implemented as software functions. Platform1 is configuring that the functions (fun1) is implemented as software functions and other functions (fun2 and fun3) are implemented as a hardware-accelerated function and so on for all possible combinations between software implementation or hardware-accelerated implementation. `design_flow_1` allows exploring the performance metrics of every possible combination between software implementation and hardware-accelerated implementation of  $n$  function and adding the  $2^n$  combinations to a valid `solution_set` list. Sections 4.1 and 4.2 represent the implementation of LTE PDSCH transmitter and receiver as a case study for the `design_flow_1`.

The purpose of the `design_flow_2` is to implement specific possible scenarios that meet performance constraints defined in the `performance_metrics_constrain` file. The performance constraints described in the file are hardware utilization, dynamic power, and latency. The steps of the `design_flow_2` are as follows: (1) Select the ( $2^x$ ) combinations for hardware implementation where ( $x=0,1,..n$ ). For current example, select combinations from Table 1 that generate

**TABLE 3** Estimated performance list for all combinations

Performance Metrics	comb.0	comb.1	comb.2	comb.3	comb.4	comb.5	comb.6	comb.7
Hardware utilization	0	h1	h2	h3=h1+h2	h4	h5=h1+h4	h6=h2+h4	h7=h1+h2+h4
Dynamic power	0	p1	p2	p3=p1+p2	p4	p5=p1+p4	p6=p2+p4	p7=p1+p2+p4
Latency	0	l1	l2	l3=l1+l2	l4	l5=l1+l4	l6=l2+l4	l7=l1+l2+l4

comb., combinations.

platform1, platform2, and platform4. (2) Select one of ( $2^x$ ) combinations from step 1 to be input to the SDSoC design flow shown in Figure 2. (3) After finishing the execution of step 2, get the performance metrics from step 2 and add them to the estimate\_performance\_list. (4) Repeat from step 1 to step 3 until finishing the implementation of all ( $2^x$ ) combinations defined in step 1. For example, the combination1, combination2, and combination4 are implemented, and platform1, platform2, and platform4 are respectively generated. Consequently, the estimate\_performance\_list will be as shown in Table 2, where h1, h2, h4, p1, p2, p4, l1, l2, and l4 are numeric values generated in step 3.

(5) Calculate the estimated performance metrics for all  $2^n$  combinations using the information from estimate\_performance\_list. The estimated performance metrics for all  $2^n$  combinations are a summation of the performance metrics of the  $2^n$  combinations. For the current example, the estimated performance metrics for all  $2^n$  combinations are shown in Table 3. The combination0 is configuring that all the functions (fun1, fun2, and fun3) are implemented as software functions so it has zero latency and constant hardware utilization and constant dynamic power, which are the area and the power consumptions of the ARM processor. The objective of this paper is to study the performance of hardware-accelerated functions synthesized by HLS; therefore, combination0 is considered an ideal case and removed from the estimate\_performance\_list.

(6) Compare the calculated  $2^n$  combination performance metrics from step 5 against the performance metrics constraints defined in the performance\_metrics\_constrain file. The designer sets the maximum limit of hardware utilization, dynamic power, and latency in the performance\_metrics\_constrain file. Combinations that do not meet the designer's constraints will be rejected; only combinations that passed constraints will be considered in the next steps and added to the valid solution\_set list. Consequently, in this case, the number of valid solution\_set is less than or equal to  $2^n$  depending on designer constraints. Section 4.3 represents implementation of the LTE PDSCH transmitter and receiver as a case study for the design\_flow\_2.

After defining the valid solution\_set list from a  $2^n$  combination for design\_flow\_1 or design\_flow\_2, the implementation of the combinations in the solution\_set list is executed as the following steps: (1) Select one of possible combination from the solution\_set list generated from design\_flow\_1 or design\_flow\_2 to be input to the SDSoC design flow shown in Figure 2. (2) Repeat step 1 until finishing implementation of all combination in the solution\_set list defined in step 1. (3) Get the performance metrics of all solution\_set. If design\_flow\_1 is applied, print the output performance metrics results for all combination in the solution\_set list. (4) If design\_flow\_2 is applied, then read the performance\_metrics\_constrain file and calculate the performance metrics cost (PMC) of all combinations in the solution\_set as described in Equation (1). (5) Print the performance metric for all combinations in the valid solution\_set list, and print the best combination in the solution\_set that achieves target performance, which is the least cost value calculated in step 4.

The PMC is calculated as follows:

$$\text{PMC} = \frac{|\text{area} - \text{areat}|}{\text{areat} * \text{areaw} + |\text{power} - \text{powert}|} + \frac{|\text{latency} - \text{latencyt}|}{\text{latencyt} * \text{latencyw}} \quad (1)$$

where area\_t is target area, area\_w is area weight, power\_t is target power, power\_w is power weight, latency\_t is target latency, and latency\_w is latency weight.

The PMC calculation of solution indicates how far the solution performance is from the target performance. As shown in Equation (1), the PMC equation sets a target of area (hardware utilization), power (dynamic power), and latency design constraint, so the SoC designer sets the target required performance metrics. Also, the PMC equation sets weight of area (hardware utilization), power (dynamic power), and latency design constraints; therefore, the SoC designer is able to decide which parameter is more important and increase its weight, and unimportant parameters' weight could be set to zero weight. For example, if dynamic power is an important metrics in system design, increase

its weight and implement only the solution\_set that generate a heterogeneous FPGA-CPU platform that consume the minimum dynamic power.

### 3 | LONG-TERM EVOLUTION

The LTE standard defines six downlink channels, three channels for controlling information and three channels carrying user data. The control channels are physical hybrid indicator channel (PHICH), physical control format indicator channel (PCFICH), and physical downlink control channel (PDCCH). The data channels are physical broadcast channel (PBCH), physical multicast channel (PMCH), and PDSCH.<sup>8</sup> This paper focus only on the design of PDSCH channel because this channel is used for carrying and processing the user data.

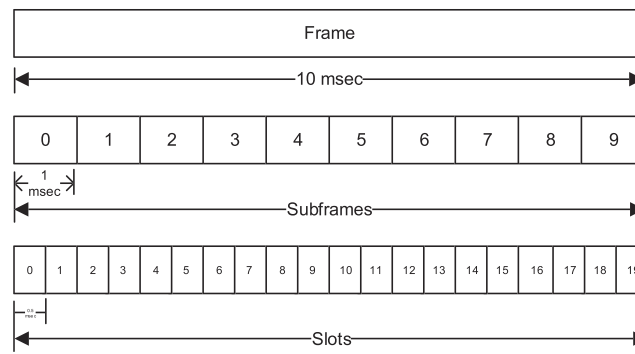
#### 3.1 | LTE frame structure

The LTE physical frame structure defines two types of frame structure in the 3GPP standard,<sup>9</sup> the frequency division duplex (FDD) type and time division duplex (TDD) type. Figure 5 shows the structure of the LTE frame in the FDD mode. The frame has a 10-ms duration, and each frame consist of 10 subframes having 1-ms duration. Each subframe consists of two slots having 0.5-ms duration.

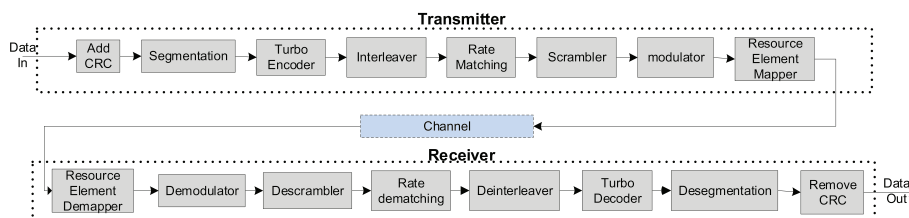
#### 3.2 | LTE PDSCH transmitter and receiver model

The PDSCH is a physical channel that carries user data. The transmitter and the receiver model of single-input single-output (SISO) PDSCH chain<sup>10</sup> is shown in Figure 6. In this paper, The PDSCH chain model is tested over additive white Gaussian noise (AWGN) channel for high signal-to-noise ratio (SNR).

The input data to the LTE PDSCH transmitter chain are called transport block. The transport block flow goes into cyclic redundancy check (CRC) calculation and appending, segmentation, turbo encoding, interleaving, rate matching, scrambling, and modulation followed by the resource element mapper. The data from the transmitter is passed to channel, and then it is fed to the LTE PDSCH receiver. The received input flow goes into the resource element de-mapper, de-modulation, de-scrambling, rate de-matching, de-interleaving, turbo decoding, and de-segmentation followed by CRC calculation and extraction.



**FIGURE 5** Structure of the long-term evolution (LTE) frame in frequency division duplex (FDD) mode



**FIGURE 6** Structure of the long-term evolution (LTE) transmitter and receiver models

### 3.2.1 | CRC addition and CRC removing

CRC is a sequence of redundant bits used for error detection on transport blocks. CRC parity bits are calculated and appended to the transport block. The CRC parity 24A is calculated using CRC generator polynomial. The equation of the CRC24A polynomial generation is  $^{11}g_{CRC24A}=1+D+D^3+D^4+D^5+D^6+D^7+D^{10}+D^{11}+D^{14}+D^{17}+D^{18}+D^{24}+D^{23}+D^{24}$ .

On the receiver side, CRC polynomial is generated using the same CRC polynomial generator. The CRC is checked for any error in the received bits; if there is no error, remove the CRC bits from the transport block; else, if any error is found in a particular block, that transport block is retransmitted.

### 3.2.2 | Segmentation and de-segmentation

Large amount of data bits from transport block should be transmitted at the same time. The transport block is divided into smaller blocks called code blocks. The LTE standard defines that the minimum code block size is 40 bits and the maximum code block size is 6144 bits.<sup>11</sup> If input bits to the code block segmentation are less than 40 bits, then add filling bits. If input bits to the code block segmentation are larger than 6144 bits, then perform segmentation of input bits and append other 24 CRC bits of type 24B to the end of each code blocks.

On the receiver side, the de-segmentation block performs the inverse operation of segmentation block. CRC is checked for any error in the received code block bits; if any errors are found in a particular block, the transport block is retransmitted; else, if there is no error, remove the CRC from the code block and concatenate the multiple code blocks to form the transport block frame.

### 3.2.3 | Channel coding and channel decoding

The channel coding used in the PDSCH chain is turbo coding.<sup>11</sup> Turbo encoder with constant coding rate  $1/3^{12}$  is used for input data coding as shown in Figure 7. The scheme of the turbo encoder is parallel of recursive systematic convolution (RSC) encoder separated by internal code interleaver. The input goes into the first RSC encoder, and after interleaving, it feeds a second RSC encoder. The multiplexing and puncturing block accepts inputs and generates coded bits. The turbo interleaver permutes the indices of the input bits to improves the turbo code performance.

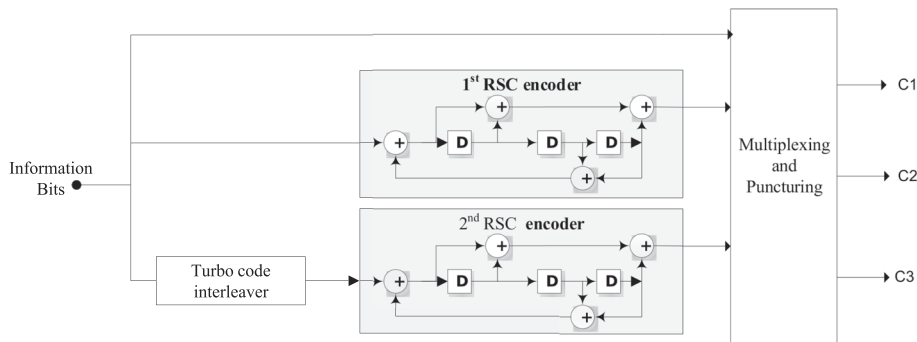


FIGURE 7 Turbo encoder block diagram

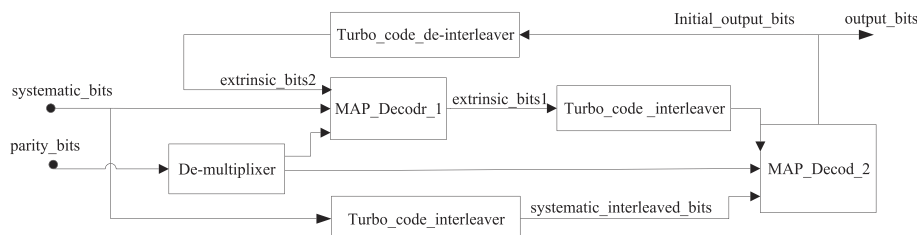


FIGURE 8 Turbo decoder block diagram



Turbo decoder block accepts input from the de-interleaver block, then performs turbo decoding using a sub-log-MAP (Max-Log-MAP) algorithm to decoded input bits to output bits. Figure 8 shows the block diagram of the turbo decoder.<sup>13</sup> The turbo decoder consists of maximum a posteriori (MAP) decoder block, demultiplex block, turbo\_interleaver block, and turbo\_de-interleaver block.

The MAP decoder is a decoder designed using Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm. The BCJR algorithm is an algorithm for error-correcting codes defined as trellises. The BCJR algorithm calculates forward probabilities, backward probabilities, and smoothed probabilities based on channel information.<sup>14</sup>

The operation of turbo decoder is done as follows: (1) The MAP\_decoder\_1 accepts the systematic\_bits and parity\_bits then generates the extrinsic\_bits1 (extrinsic bit is a soft estimate bits do not contain any information). (2) The extrinsic\_bits1 bits are interleaved and generates extrinsic\_interleaved\_bits1; also, the systematic bits are interleaved and generates systematic\_interleaved\_bits. (3) The MAP\_decoder\_2 accepts extrinsic\_interleaved\_bits1, systematic\_interleaved\_bits, and parity\_bits, then generates the initial\_output\_bit. (4) The initial\_output\_bit is passed to the de-interleaver, and the extrinsic\_interleaved\_bits2 is generated. (5) The MAP\_decoder\_1 accepts the systematic\_bits, the parity\_bits, and the extrinsic\_interleaved\_bits2, then generates the extrinsic\_bits1. The steps from step 1 to step 5 are repeated iteratively until the bit error rate is zero. At the end of the process, the output\_bits bits are generated according to threshold operation.

### 3.2.4 | Interleaver and de-interleaver

Interleaving is the process of reordering data so that successive bunch of data is distributed over a larger sequence of data to reduce the effect of burst errors. Using interleaver increases the performance of the error protection decoder to correct the burst error.<sup>11</sup> Error protection coding process cannot correct the errors that occur in groups, so using of interleaver allows reducing such error. At the receiver side, the de-interleaver block reverses the operation of interleaver.

### 3.2.5 | Rate matchings and rate de-matching

The LTE turbo encoder has a fixed coding rate of 1/3. The communication standard added feature for adapting the throughput on the basis of the channel conditions.<sup>11</sup> In degraded channels, smaller coding rates are used to increase the number of error-correction bits and vice versa. Rate matching is used to arrive to any desired rate by repeating or puncturing. In case of reducing the encoding rate lower than 1/3, repeat the turbo coder output bit. In case of increasing rate higher than 1/3, puncture (remove) some of the turbo coder output bits.

The rate matching block<sup>15</sup> consists of three sub-block\_interleaver blocks, bit\_collection block, and bit\_selection block. The sub-block\_interleaver is based on the classic row-column interleaver with 32 columns and a 32 intracolumn permutation. The rate matching block accepts three input streams from turbo encoder. The bits of each stream are written row by row into a matrix with 32 columns. After a column permutation, bits are read out from the matrix column by column. The column permutation of the sub-block interleaver is given as follows: [0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30,1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,31]. The bit\_collection block accepts input from three sub-block\_interleaver and generates output depending on coding type. Finally, the bit\_selection block receives inputs from bit\_collection block to skip dummy bits and generate the required output bits with proper size.

On the receiver side, the rate de-matching block<sup>16</sup> consists of three bit\_de-interleaver blocks and one bit\_de-selection block. bit\_de-selection block accepts input bits and divide it to three outputs; each output has a length equal to the code block length. Three bit\_de-interleaver blocks accept outputs from the bit\_de-selection block. Each one of the bit\_de-interleaver block accepts input from bit\_de-selection block with a length equal to the code block length. The bit\_de-interleaver block inverse the operation of sub-block\_interleaver.

### 3.2.6 | Scrambler and de-scrambler

The scrambler is a block that pseudo-randomly changes the values of bits into a data block, thus ensuring that the interference is randomized for each different cell or to introduce security as part of an encryption procedure. The data bits are scrambled with a sequence that is unique to each cell by initializing the sequence generators in the cell on the

basis of the physical cell identity. The scrambler consists of two parts: pseudo-random sequence generation and bit multiplication. The pseudo-random sequences are defined by a Gold sequence with length equal to 31.

On the receiver side, the de-scrambler function, the same Gold sequence generator of scrambler is used to invert the scrambling operation. The de-scrambler block operates on the LLR outputs of the demodulator, converting the Gold-sequence bits into either 1 or  $-1$  values.

### 3.2.7 | Modulator and de-modulator

The modulator accepts groups of input bits and maps them to specific constellation symbols, according to the specified modulation method. The LTE standard supports quadrature phase shift keying (QPSK), quadrature amplitude modulation (16QAM), and 64QAM modulation schemes types for the LTE PDSCH. Multiple modulation schemes allow adaptive modulation on the basis of channel conditions. When the SNR is high, denser constellations (ex:-64QAM) are used to increase the throughput. However, when the SNR is low, modulation schemes with more intersymbol separation should be used to reduce the throughput and decrease the bit error rate.

On the receiver side, the de-modulator block de-maps QPSK symbol, 16-QAM symbol, and 64-QAM symbol to bits according to the modulation method specified.<sup>17</sup>

### 3.2.8 | Resource element mapper and resource element de-mapper

The resource element mapper is a time-frequency representation of data organized as the resource grid. The resource grid is a two-dimensional map of symbols in the horizontal axis (time domain) and sub-carrier on the vertical axis (frequency domain).<sup>10</sup> The placement of data within the resource grid is important and depends on which subframe is in use. There are five types of data placed in a resource grid: First, the PDSCH signal, which carries user data. Second, the cell-specific reference signal (CRS), which is used by the receiver for estimating the channel frequency response and cross-channel effects. Third, the PDCCH signal, which carries important information for processing (eg, modulation scheme). Fourth, the primary synchronization signals (PSS) data and secondary synchronization signals (SSS) data, which are used to determine the frame timing and cell identification. Fifth, the broadcast channel (BCH) signal, which carries the master information such as cell bandwidth.

On the receiver side, the role of a resource element de-mapping is to extract PDSCH, CRS, etc, from the resource grid and feed each type of data to the corresponding next stage. The PDSCH receiver chain should accept PDSCH input from the resource grid to complete the processing of data in the receiver chain.

## 4 | IMPLEMENTATION OF LTE PDSCH TRANSMITTER and RECEIVER

This section explains the implementation of the LTE PDSCH transmitter and LTE PDSCH Receiver on a heterogeneous FPGA-CPU platform by applying the proposed performance-based adaptive design technique using SDSoC tool. The LTE PDSCH transmitter and LTE PDSCH receiver are written using C programming language and integrated into the main function including test-bench function to verify them. In Section 4.1, the LTE PDSCH transmitter functions are implemented using `design_flow_1`; similarly, in Section 4.2, the LTE PDSCH receiver functions are implemented using `design_flow_1`. In Section 4.3, the LTE PDSCH receiver and receiver functions are implemented using `design_flow_2`.

### 4.1 | LTE PDSCH transmitter implementation

The purpose of this section is to implement all possible scenarios for the LTE PDSCH transmitter function. Each scenario generates an embedded FPGA platform, which depends on the implementation of the LTE PDSCH transmitter subfunctions, either software function or hardware-accelerated function synthesized by HLS. As shown in Figure 6, the LTE PDSCH transmitter consists of eight subfunctions, which are CRC\_addition, segmentation, turbo\_encoder, interleaver, rate\_matching, scrambler, modulator, and resource\_element\_mapper. As LTE PDSCH transmitter consists of eight subfunctions, 256 ( $2^8=256$ ) combinations are generated to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS. Table 4 shows all possible configuration

**TABLE 4** LTE PDSCH transmitter subfunction configuration scenario

Platform Name	RE Mapper	Modulator	Scrambler	Rate Matching	Interleaver Encoder	Turbo	Segmentation Addition	CRC
Tx0	0	0	0	0	0	0	0	0
Tx1	0	0	0	0	0	0	0	1
Tx2	0	0	0	0	0	0	1	0
Tx3	0	0	0	0	0	0	1	1
Tx4	0	0	0	0	0	1	0	0
Tx5	0	0	0	0	0	1	0	1
Tx6	0	0	0	0	0	1	1	0
Tx7	0	0	0	0	0	1	1	1
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
Tx250	1	1	1	1	1	0	1	0
Tx251	1	1	1	1	1	0	1	1
Tx252	1	1	1	1	1	1	0	0
Tx253	1	1	1	1	1	1	0	1
Tx254	1	1	1	1	1	1	1	0
Tx255	1	1	1	1	1	1	1	1

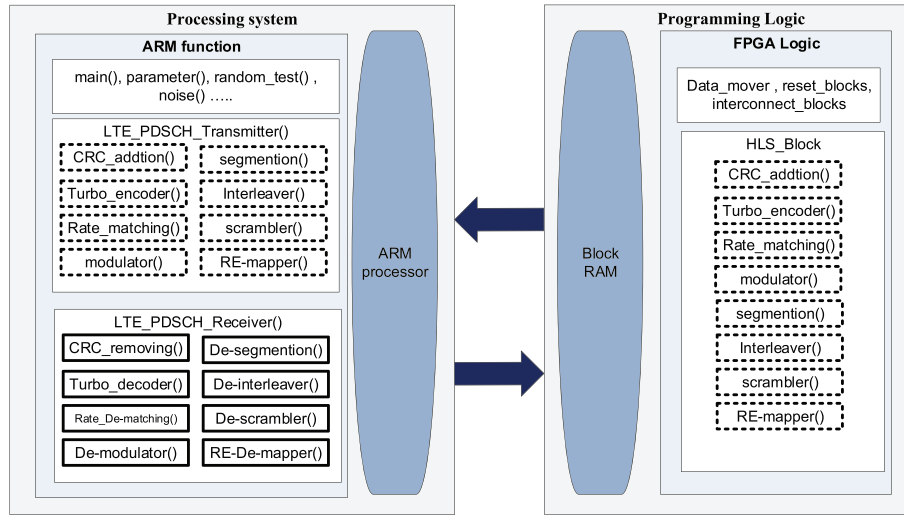
(0) Software function and (1) hardware-accelerated logic.

CRC, cyclic redundancy check; LTE, long-term evolution; PDSCH, physical downlink shared channel.

scenarios to implement the LTE PDSCH transmitter subfunctions. The number (1) in the table indicates that this subfunction in current platform is a hardware-accelerated logic, and the number (0) indicates that this subfunction in current platform is a software function. For example, in Tx6 platform, the segmentation subfunction and turbo\_encoder subfunction are implemented as a hardware-accelerated logic, and other LTE PDSCH transmitter subfunctions are implemented as a software function.

#### 4.1.1 | Configurable embedded FPGA platform for LTE PDSCH transmitter

Figure 9 shows the proposed configurable embedded FPGA platform for the LTE PDSCH transmitter. The configurable embedded FPGA platform consists of processing system and programming logic. The processing system side consists of fixed implementation functions used for integrating and verifying the operation of the LTE PDSCH transmitter function. Examples for fixed implementation functions, the main function of which is integrating all functions together, are the random\_test function used to generate random information bits and the noise function used to generate AWGN noise. On the LTE PDSCH transmitter platform, all of the LTE PDSCH receiver functions configured to be implemented as a fixed software function because they are used in integrating and verifying the LTE PDSCH transmitter/receiver chain. In addition, the processing system consists of the LTE PDSCH transmitter function, which consists of the configurable implementation functions. The term configurable means that each subfunctions of the LTE PDSCH transmitter function is implemented as software function or hardware-accelerated function synthesized by HLS according to the configuration in Table 4. The programming logic side consists of a fixed implemented interconnect hardware logic used to handle the data flow between processing system and programming logic. The fixed interconnect logics are generated by the SDSoc tool and depend on the number of connection ports between software functions and hardware-accelerated functions synthesized by HLS. In addition, the programming logic consists of LTE PDSCH transmitter functions implemented using HLS. As described in Section 4.1, each subfunction of LTE PDSCH transmitter function is



**FIGURE 9** Configurable embedded FPGA platform for the long-term evolution (LTE) physical downlink shared channel (PDSCH) transmitter

implemented as software function or hardware-accelerated function synthesized by HLS according to the configuration in Table 4.

## 4.2 | LTE PDSCH receiver implementation

The purpose of this section is to implement all possible scenarios for the LTE PDSCH receiver function. Each scenario generates an embedded FPGA platform, which depends on the implementation of the LTE PDSCH receiver subfunctions, either software function or hardware-accelerated function, synthesized by HLS. As shown in Figure 6, the LTE PDSCH receiver consists of eight subfunctions, which are `CRC_removing`, `de-segmentation`, `turbo_decoder`, `de-interleaver`, `rate_de-matching`, `de-scrambler`, `de-modulator`, and `resource_element_de-mapper`. As LTE PDSCH receiver consists of eight subfunctions, 256 ( $2^8=256$ ) combinations are generated to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS. Table 5 shows all possible configuration scenarios to implement the LTE PDSCH receiver subfunctions. The number (1) in the table indicates that this subfunction in current platform is implemented as a hardware-accelerated logic, and the number (0) indicates that this subfunction in current platform is implemented as a software function. For example, in Rx6 platform, the `de-segmentation` subfunction and `turbo_decoder` subfunction are implemented as a hardware-accelerated logic, and other LTE PDSCH receiver subfunctions are implemented as a software function.

### 4.2.1 | Configurable embedded FPGA platform for LTE PDSCH receiver

Figure 10 shows the proposed configurable embedded FPGA platform for the LTE PDSCH receiver. The configurable embedded FPGA platform consists of processing system and programming logic. The processing system side consists of fixed implementation functions used for integrating and verifying the operation of the LTE PDSCH transmitter function. On the LTE PDSCH receiver platform, all of the LTE PDSCH transmitter functions configured to be implemented as a fixed software function because they are used in integrating and verifying the LTE PDSCH transmitter/receiver chain. In addition, the processing system consists of the LTE PDSCH receiver function which consists of the configurable implementation functions. The term configurable means that each subfunction of the LTE PDSCH receiver function is implemented as software function or hardware-accelerated function synthesized by HLS according to configuration in Table 5.

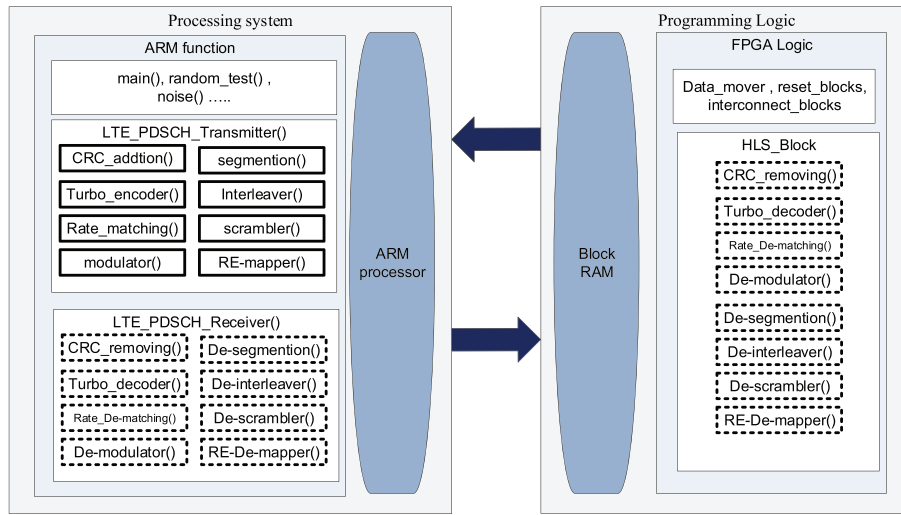
The programming logic side consists of a fixed implemented hardware logic used to handle the data flow between processing system and programming logic. In addition, the programming logic consists of configurable logic which is the LTE PDSCH receiver functions implemented using HLS. As described in section 4.2, each subfunction of LTE

TABLE 5 LTE PDSCH receiver subfunction configuration scenario

Platform Name	RE De-mapper	De-modulator	De-scrambler	Rate De-matching	De-interleaver Encoder	Turbo Decoder	De-segmentation	CRC Removing
Rx0	0	0	0	0	0	0	0	0
Rx1	0	0	0	0	0	0	0	1
Rx2	0	0	0	0	0	0	1	0
Rx3	0	0	0	0	0	0	1	1
Rx4	0	0	0	0	0	1	0	0
Rx5	0	0	0	0	0	1	0	1
Rx6	0	0	0	0	0	1	1	0
Rx7	0	0	0	0	0	1	1	1
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
Rx250	1	1	1	1	1	0	1	0
Rx251	1	1	1	1	1	0	1	1
Rx252	1	1	1	1	1	1	0	0
Rx253	1	1	1	1	1	1	0	1
Rx254	1	1	1	1	1	1	1	0
Rx255	1	1	1	1	1	1	1	1

<sup>a</sup> . (0) Software function and (1) hardware-accelerated logic.

<sup>b</sup> : CRC, cyclic redundancy check; LTE, long-term evolution; PDSCH, physical downlink shared channel.



**FIGURE 10** Configurable embedded FPGA platform for the long-term evolution (LTE) physical downlink shared channel (PDSCH) receiver

**TABLE 6** Constrained solution example

Model Example	Performance_constraints								
	Area (LUT)			Power (watts)			Latency (clock cycle)		
	Limit	Target	Weight	Limit	Target	Weight	Limit	Target	Weight
LTE PDSCH transmitter	21 700	2000	1	0.22	0.1	1	10 <sup>9</sup>	10*10 <sup>6</sup>	1
LTE PDSCH receiver	35 000	5200	0	0.249	0.249	0	20*10 <sup>6</sup>	1.22*10 <sup>6</sup>	1

<sup>a</sup>LTE, long-term evolution; LUT, look-up tables; PDSCH, physical downlink shared channel.

PDSCH receiver function is implemented as software function or hardware-accelerated function synthesized by HLS according to configuration in Table 5.

### 4.3 | Constrained solution

This section introduces examples for implementation of the LTE PDSCH transmitter and LTE PDSCH receiver under user constraints conditions. The design flow applied is the design\_flow\_2 described in Section 2.2. Table 6 shows an example of design constraints applied for implementation. As shown in the table, sets of constraints are as follows: The first is the maximum limit of area, power, and latency. Only solutions that passed maximum limit constraints will be added to valid solution\_set. The second is the target and weight of area, power, and latency. Both of them are used to calculate PMC as shown in Equation (1).

## 5 | RESULTS AND COMPARATIVE STUDIES

There are different tools used to implement heterogeneous FPGA-CPU SoC platforms and to plot different performance figures. First, a set of shell scripts are used to run the automated SDSoC tool and to implement heterogeneous FPGA-CPU SoC platforms and print the performance results in a text file. The SDSoC tool is integrated with the HLS tool used for transforming the C/C++ code to an RTL implementation. Also, SDSoC tool is integrated with SDK tool that includes driver support, C/C++ compiler library, and used for debugging and profiling. Finally, the performance figures are plotted using MATLAB tool. This section focuses on studying the performance of hardware-accelerated functions synthesized by HLS. Therefore, the first solution on Table 4, which is Tx0 platform, and the first solution on Table 5, which is Rx0 platform, are assumed an ideal case. Tx0 and Rx0 solutions are considered as an ideal case because all of the

LTE PDSCH transmitter/receiver subfunctions are implemented as a software function. In this case, the hardware-accelerated logics are not generated, and the generated platform has a constant area, which is the ARM processor area, and has a constant power consumption, which is the power consumption of the ARM processor. The Xilinx ZYNQ ZC702 device was used for the implementation. It consists of dual ARM Cortex A9 core as the processing system and XC7Z020-CLG484-based FPGA as the programming logic.<sup>18</sup> The hardware-accelerated functions are synthesized and implemented at frequency of 100 MHz.

## 5.1 | LTE transmitter implementation results

This section shows the results of implementation of all possible configuration scenarios shown in Table 4. Two hundred and fifty-six projects are generated for the LTE PDSCH transmitter to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS.

### 5.1.1 | Hardware utilization of LTE PDSCH transmitter implementation

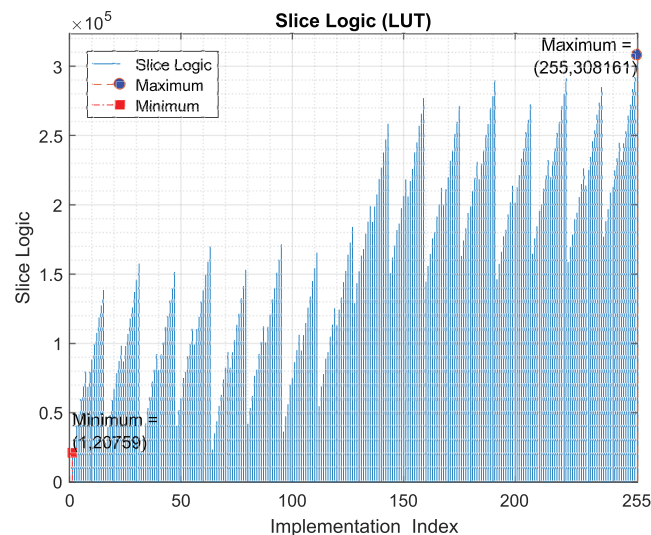
Figure 11 shows the hardware utilization of the generated platforms of the synthesized hardware. The hardware utilization is the sum of the number of look-up tables (LUT), number of flip-flops, and number of muxes. The Tx1 platform has the minimum hardware utilization. The Tx1 platform is configuring that CRC\_addition block is implemented as hardware-accelerated logic, and other LTE PDSCH transmitter subfunctions are implemented as software functions. The Tx255 platform has the maximum hardware utilization. The Tx255 platform is configuring that all of the LTE PDSCH transmitter subfunctions are implemented as hardware-accelerated functions.

### 5.1.2 | Latency of LTE PDSCH transmitter implementation

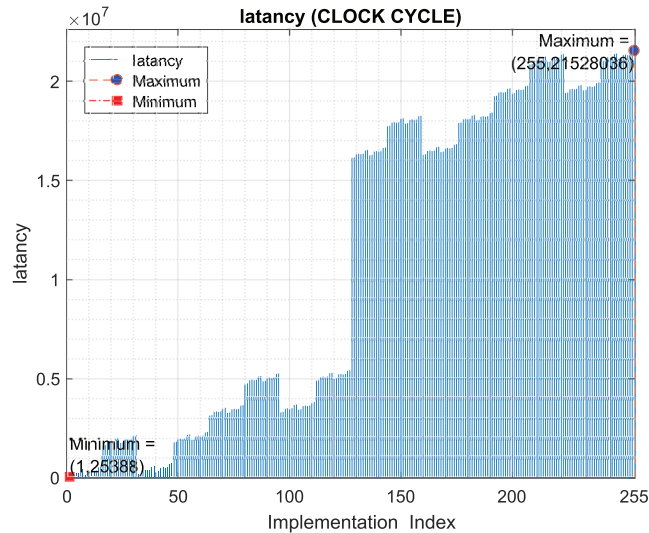
Figure 12 shows the latency calculation of the generated platforms of the synthesized hardware. The latency measures the delay in numbers of the clock cycle. The Tx1 platform has the minimum latency calculation. The Tx1 platform is configuring that CRC\_addition block is implemented as hardware-accelerated logic and other LTE PDSCH transmitter subfunctions are implemented as software functions. The Tx255 platform has the maximum latency calculation. The Tx255 platform is configuring that all of the LTE PDSCH transmitter subfunctions are implemented as hardware-accelerated functions.

### 5.1.3 | Dynamic power of LTE PDSCH transmitter implementation

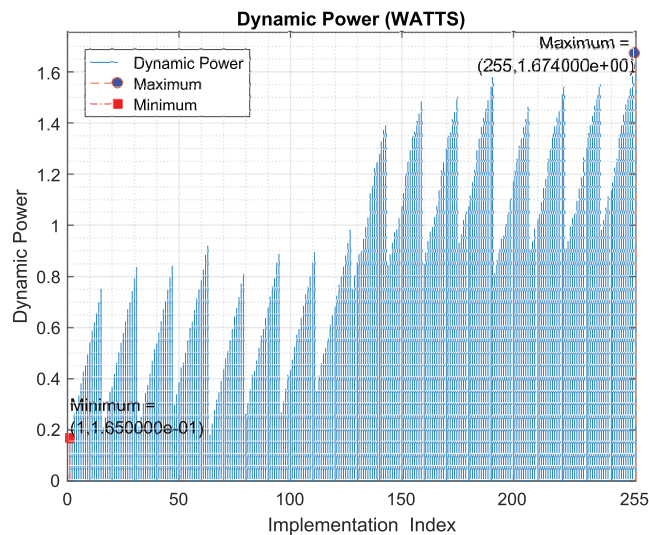
Power is an important metric for any communication system. For FPGA platform, power calculation includes the power consumption in the ARM processor and the static power calculation. The dynamic power calculation is focused only in



**FIGURE 11** Hardware utilization of long-term evolution (LTE) physical downlink shared channel (PDSCH) transmitter implementation



**FIGURE 12** Latency calculations of long-term evolution (LTE) physical downlink shared channel (PDSCH) transmitter implementation



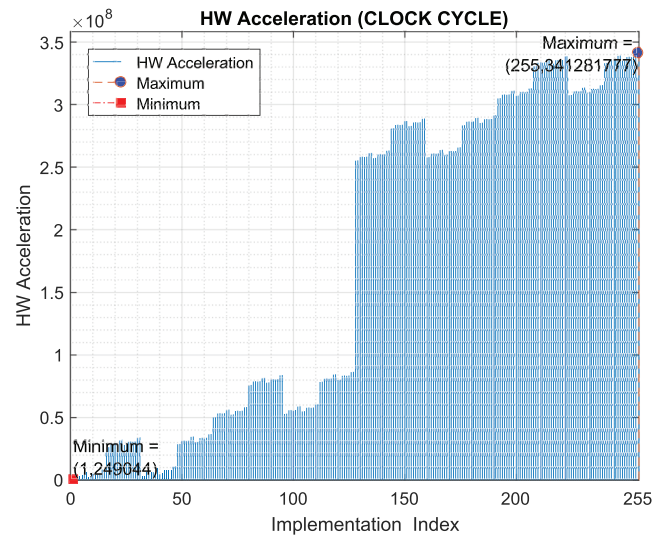
**FIGURE 13** Dynamic power consumption of long-term evolution (LTE) physical downlink shared channel (PDSCH) transmitter implementation

the power calculations; therefore, the ARM processor power and the static power are subtracted from the total power consumptions. Figure 13 shows the dynamic power consumption from the generated platforms of the synthesized hardware. The Tx1 platform has the minimum dynamic power consumption. The Tx1 platform is configuring that CRC\_addition block is implemented as hardware-accelerated logic and other LTE PDSCH transmitter subfunctions are implemented as software functions. The Tx255 platform has the maximum dynamic power consumption. The Tx255 platform is configuring that all of the LTE PDSCH transmitter subfunctions are implemented as hardware-accelerated functions.

#### 5.1.4 | Hardware acceleration of LTE PDSCH transmitter implementation

Hardware acceleration is a performance metric defined by SDSoC tool. Hardware acceleration is the number of clock cycles improvement in execution of the system if a function is implemented as a hardware-accelerated function in the programming logic. Figure 14 shows the hardware acceleration calculations from the generated platforms of the synthesized hardware. The Tx1 platform has the minimum hardware acceleration calculation. The Tx1 platform is configuring that CRC\_addition block is implemented as hardware-accelerated logic and other LTE PDSCH transmitter





**FIGURE 14** Hardware acceleration calculations of long-term evolution (LTE) physical downlink shared channel (PDSCH) transmitter implementation

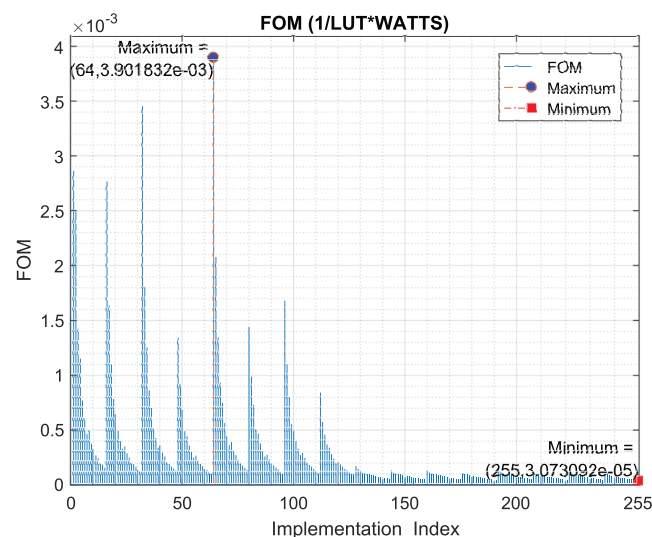
subfunctions are implemented as software functions. The Tx255 platform has the maximum hardware acceleration calculation. The Tx255 platform is configuring that all of the LTE PDSCH transmitter subfunctions are implemented as hardware-accelerated functions.

### 5.1.5 | FoM of LTE PDSCH transmitter implementation

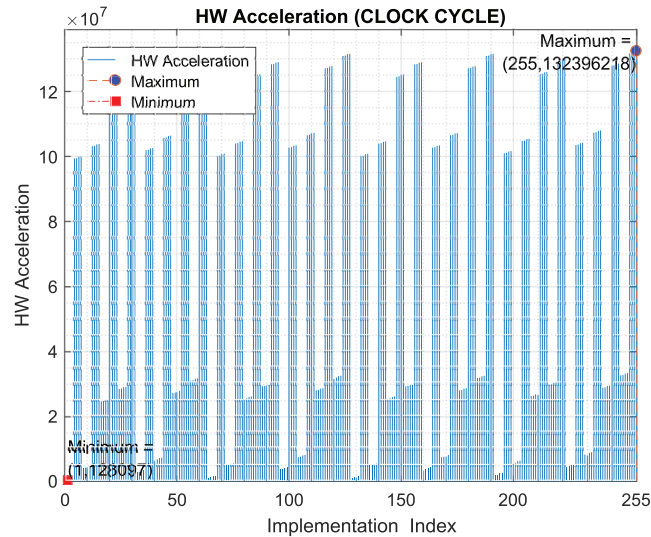
To know the platform that achieves the best performance, figure of merit (FoM) metric was defined in Love et al<sup>8</sup> as follows:

$$\text{FoM} = \frac{(\text{acceleration})}{(\text{area})}(\text{power})(\text{latency}). \quad (2)$$

Equation (2) shows that the acceleration effect is directly proportional to the FoM performance and shows that area, power, and latency effect is reversely proportional to FoM performance. FoM shows that studying hardware acceleration, area, power, and latency metrics alone is not sufficient to implement a platform that achieves the best performance. Figure 15 shows the FoM calculations of the generated platforms of the synthesized hardware. The Tx64



**FIGURE 15** Figure of merit (FoM) of long-term evolution (LTE) physical downlink shared channel (PDSCH) transmitter implementation



**FIGURE 16** Hardware utilization of long-term evolution (LTE) physical downlink shared channel (PDSCH) receiver implementation

platform has the best FoM calculation. The Tx64 is configuring that segmentation block is implemented as hardware-accelerated logic and other LTE PDSCH transmitter subfunctions are implemented as software functions. The Tx255 platform has the worst FoM calculation. The Tx255 platform is configuring that all of the LTE PDSCH transmitter subfunctions are implemented as hardware-accelerated functions.

## 5.2 | LTE receiver implementation results

This section shows the implementation results of all possible configuration scenarios shown in Table 5. Two hundred and fifty-six projects are generated for the LTE PDSCH receiver to cover all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS.

### 5.2.1 | Hardware utilization of LTE PDSCH receiver implementation

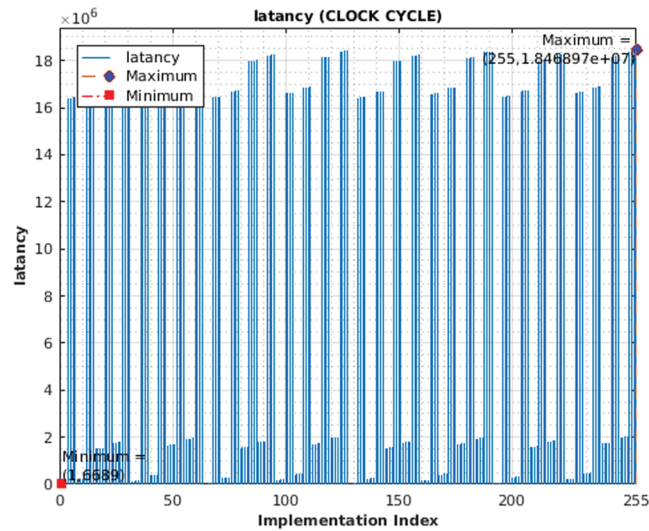
Figure 16 shows the hardware utilization from the generated platforms of the synthesized hardware. The Rx1 platform has the minimum hardware utilization. The Rx1 platform is configuring that CRC\_removing block is implemented as hardware-accelerated logic and other LTE PDSCH receiver subfunctions are implemented as software functions. The Rx255 platform has the maximum hardware utilization. The Rx255 platform is configuring that all of the LTE PDSCH receiver subfunctions are implemented as hardware-accelerated functions.

### 5.2.2 | Latency of LTE PDSCH receiver implementation

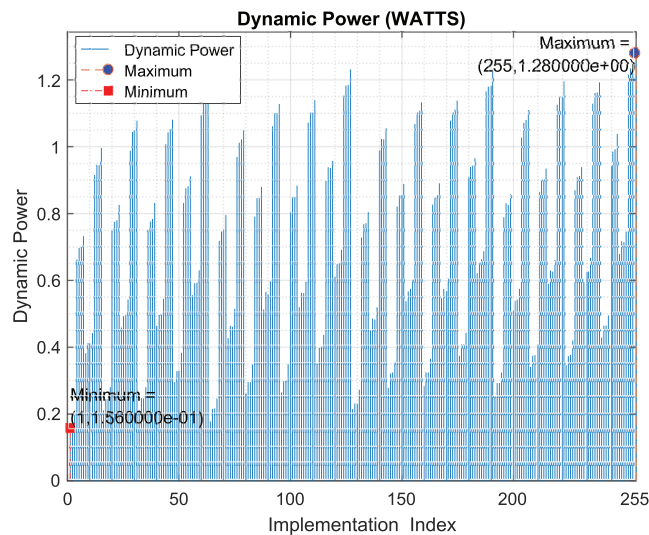
Figure 17 shows the latency calculation of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum latency calculation. The Rx1 platform is configuring that CRC\_removing block is implemented as hardware-accelerated logic and other LTE PDSCH receiver subfunctions are implemented as software functions. The Rx255 platform has the maximum latency calculation. The Rx255 platform is configuring that all of the LTE PDSCH receiver subfunctions are implemented as hardware-accelerated functions.

### 5.2.3 | Dynamic power of LTE PDSCH receiver implementation

Figure 18 shows the dynamic power consumption of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum dynamic power consumption. The Rx1 is configuring that CRC\_removing block is implemented as hardware-accelerated logic and other LTE PDSCH receiver subfunctions are implemented as software functions. The



**FIGURE 17** Latency calculations of long-term evolution (LTE) physical downlink shared channel (PDSCH) receiver implementation



**FIGURE 18** Dynamic power consumption of long-term evolution (LTE) physical downlink shared channel (PDSCH) receiver implementation

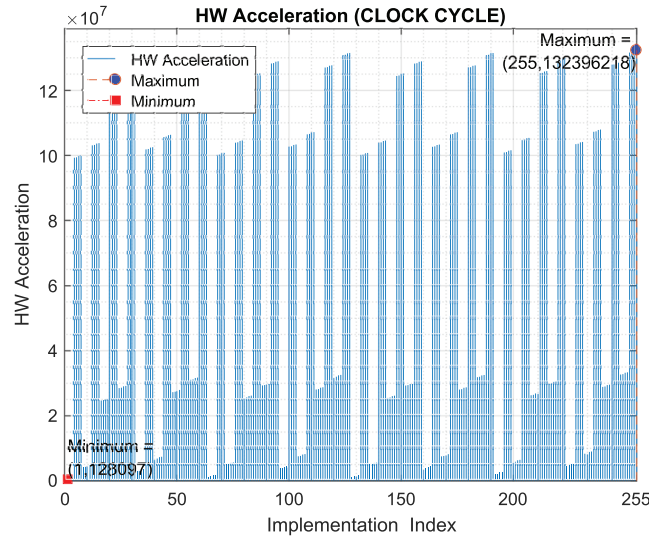
Rx255 platform has the maximum dynamic power consumption. The Rx255 platform is configuring that all of the LTE PDSCH receiver subfunctions are implemented as hardware-accelerated functions.

### 5.2.4 | Hardware acceleration of LTE PDSCH receiver implementation

Figure 19 shows the hardware acceleration calculations of the generated platforms of the synthesized hardware. The Rx1 platform has the minimum hardware acceleration calculation. The Rx1 is configuring that CRC\_removing block is implemented as hardware-accelerated logic and other LTE PDSCH receiver subfunctions are implemented as software functions. The Rx255 platform has the maximum hardware acceleration calculation. The Rx255 platform is configuring that all of the LTE PDSCH receiver subfunctions are implemented as hardware-accelerated functions.

### 5.2.5 | FoM of LTE PDSCH receiver implementation

Figure 20 shows the FoM calculations of the generated platforms of the synthesized hardware. The Rx1 platform has the best FoM calculation. The Rx1 is configuring that CRC\_removing block is implemented as hardware-accelerated logic



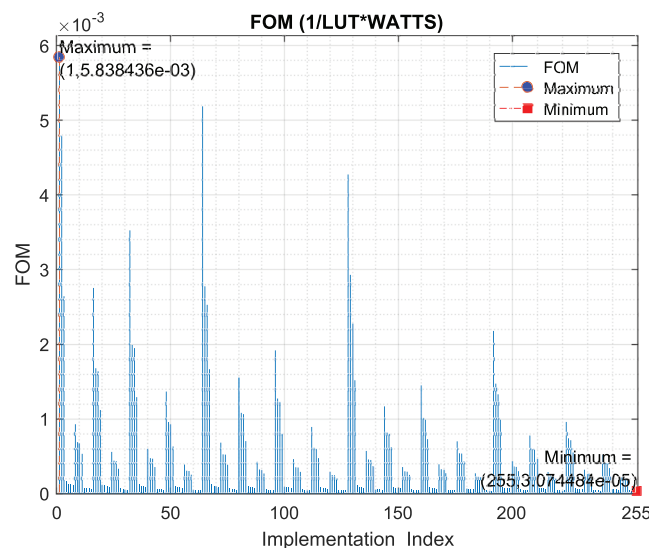
**FIGURE 19** Hardware acceleration calculations of long-term evolution (LTE) physical downlink shared channel (PDSCH) receiver implementation

and other LTE PDSCH receiver subfunctions are implemented as software functions. The Rx255 platform has the worst FoM calculation. The Rx255 platform is configuring that all of the LTE PDSCH receiver subfunctions are implemented as hardware-accelerated functions.

### 5.3 | Constrained solution example results

This section shows the implementation results to get the best heterogeneous FPGA-CPU SoC platform that meets constraints examples in Section 4.3. The design\_flow\_2 illustrated in Section 2.2 is applied. Equation (1) is used for calculating cost and selecting the best implementation that meets the design constraints. Table 7 shows the implementation results after applying the target constraints in Table 6. The valid\_solution\_set column in Table 7 shows the valid solution\_set that meets the maximum limit design constraints in Table 6. Next, the performance metrics cost is calculated for each valid solution\_set using Equation (1). Finally, the solution with the minimum calculated PMC value that meets the constraints is selected.

For LTE PDSCH transmitter chain, there are three configuration scenarios that met the required performance constraints in Table 6. The Tx1, Tx2, and Tx32 configuration scenarios are added to the valid\_solution\_sets list; then,



**FIGURE 20** Figure of merit (FoM) of long-term evolution (LTE) physical downlink shared channel (PDSCH) receiver implementation

**TABLE 7** Constrained solution example result

Model Example	Performance_constraints_results		
	valid_solution_set	PMC	Best PMC
LTE PDSCH transmitter	Tx1	9.0320	Tx1
	Tx2	13.9202	
	Tx32	10.1172	
LTE PDSCH receiver	Rx1	0.87540	Rx33
	Rx2	0.527322	
	Rx3	0.4027311	
	Rx4	2.16794	
	Rx16	1.141814	
	Rx32	0.119827	
	Rx33	0.00540819	

<sup>a</sup>LTE, long-term evolution; PDSCH, physical downlink shared channel; PMC, performance metrics cost.

the PMC for each configuration scenarios are calculated, and then Tx1 was selected as the best configuration scenario that meets the constraints because it generated the minimum PMC calculation. For LTE PDSCH receiver chain, there are seven configuration scenarios that met the required performance constraints in Table 6. The Rx1, Rx2, Rx3, Rx4, Rx16, Rx32, and Tx33 configuration scenarios are added to the valid\_solution\_sets list; then, the PMC for each configuration scenarios are calculated, and then Rx33 was selected as the best configuration scenario that meets the constraints because it generated the minimum PMC calculation.

## 6 | STATE-OF-THE-ART AND FUTURE WORK

Designing using SDSoC tool helps SoCs designers by introducing a simple design environment. In addition, SDSoC design environment makes integration and verification of co-design heterogeneous FPGA-CPU faster and more efficient. This paper introduces a new design technique used to design a heterogeneous FPGA-CPU SoC platform that meets predefined performance metrics. This design technique introduces a new idea to develop adaptive heterogeneous FPGA-CPU SoC platform according to design environment status.

Adaptive design implementation dependent on performance requirement could be developed and used to reimplement the SoC platform during run time. For example, the designer may develop SoC-based products depending on environmental variables (eg, availability of sunlight). The designer may develop SoC platform consuming high power assuming the availability of sunlight for recharging batteries. Let us assume for some reason that the developed SoC-based product have to be worked in a new environment where sunlight is not available; the SoC platform could be reimplemented during run time to rebuild a new SoC platform, consuming less power. Dynamic partial reconfiguration (DPR) techniques<sup>19</sup> could be integrated and used to reconfigure the FPGA according to design environment status. For example, a platform with the best performance metrics is loaded initially to the FPGA. In case of low-power mode, the platform with the minimum dynamic power performance metrics is loaded to the FPGA using DPR. In addition, this design technique may lead to the development of FPGA-CPU SoC platform with partially upgrading capability. For example, part of the LTE PDSCH chain has a fixed architecture in every LTE update release (eg, CRC calculation), so this module may be implemented as a hardware-accelerated logic. In the other hand, part of the LTE PDSCH chain has an adjustable architecture in every LTE update release (eg, resource element mapper), so this module may be implemented as a software function because software could be upgraded easily.

## 7 | CONCLUSION

Automated design technique is used to implement multiples of heterogeneous FPGA-CPU SoC platforms using SDSoC tool. This automated method is used in exploring all possible scenarios between software implementation and hardware-accelerated implementation generated using HLS. In addition, this automated method is used to implement

a platform that achieves performance metrics such as area and power. This paper answers the questions of what platform and what implementation, whether hardware or software, is best suited for more efficient platform. Also, this paper answers the question of what SoC platform to be implemented for best overall performance by introducing the FoM metric. In addition, PMC helps to develop a platform that achieves performance metrics requirement. This new design technique may lead to make quantum leap in the design of heterogeneous FPGA-CPU SoC platform by integrating performance design constraint requirement in the design cycle. Although there are a lot of advantages of using the SDSoC tool, however, there are many challenges that should be remembered by the designer. Some of these challenges are summarized as follows: First, the SDSoC tool is based on C/C++ GNU and does not perform any code optimizations. Second, there are elements of C/C++ GNU not supported by the SDSoC tool to be implemented as a hardware-acceleration function (for example, C preprocessor code). Third, any slight code modifications affects the architecture of the hardware acceleration function. Fourth, the designer should be familiar with SDSoC development environment flow. Finally, the designer should be aware of HLS skills to design platforms achieving optimal performance.

## ACKNOWLEDGMENTS

The authors would like to thank the Opto-Nano-Electronics laboratory (ONE Lab), Department of Electronics and Communications Engineering, Cairo University, Cairo, Egypt, and the supporting and funding agencies. This work was supported in part by Cairo University, Zewail City of Science and Technology, AUC, STDF, Intel, Mentor Graphics, ITIDA, SRC, ASRT, NTRA, and MCIT.

## ORCID

Hassan Mostafa  <https://orcid.org/0000-0003-0043-5007>

## REFERENCES

1. Sekar C, Hemasunder. Tutorial t7: designing with Xilinx SDSoC. In: 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID); Hyderabad, India; 2017:xl-xli.
2. Inc. Xilinx. SDSoC environment user guide (ug1028). [https://www.xilinx.com/support/documentation/boards\\_and\\_kits/zc702\\_zvik/ug850-zc702-eval-bd.pdf](https://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf); 2016.
3. Inc. Xilinx. Vivado design suite , tutorial high-level synthesis (ug871). [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2014\\_2/ug871-vivado-high-level-synthesis-tutorial](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug871-vivado-high-level-synthesis-tutorial); 2014.
4. 3GPP. ITU-R confers IMT-advanced (4G) status to 3GPP LTE.
5. Hashim HA, Abido MA. Location management in LTE networks using multi-objective particle swarm optimization. *Comput Netw.* 2019;157:78-88.
6. Inc. Xilinx. SDx Pragma Reference Guide (ug1253). <https://www.xilinx.com/cgi-bin/docs/rdoc?v=2018.3;d=ug1253-sdx-pragma-reference.pdf>; 2018.
7. Rodriguez M, Magdalenom E, Perez F, Garcia C. Automated software acceleration in programmable logic for an efficient NFFT algorithm implementation: a case study. *Sensors.* 2017;17(4):694.
8. Love R, Kuchibhotla R, Ghosh A, et al. Downlink control channel design for 3GPP LTE. In: 2008 IEEE Wireless Communications and Networking Conference; 2008; Las Vegas, NV, USA:813-818.
9. TS36.211 3GPP. Evolved Universal Terrestrial Radio Access (EUTRA); Physical Channels and Modulation (release 10); 2012.
10. Zarrinkoub H. Understanding LTE with MATLAB; 2014.
11. GPPTS36212. Evolved Universal Terrestrial Radio Access (EUTRA); Multiplexing and Channel Coding (release 9); 2012.
12. Kaur R, Chopra S. Iterative decoding of turbo codes. *Int J Sci Eng Res.* 2013;4(6).
13. Kenea JD, Kulatb KD. Soft output decoding algorithm for turbo codes implementation in mobile Wi-Max environment. *Int Conf Commun Comput Sec (ICCCS).* 2012;4(6):666-673.
14. Eladawy M, Kamaleldin A, Mostafa H, Said S. Performance evaluation of turbo encoder implementation on a heterogeneous FPGA-CPU platform using SDSoC. In: 2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT); 2017; Alexandria, Egypt:1072-1092.
15. Wu J, Liu Y, Wang R. Research on the rate matching of LTE. 2015;1-4.
16. Bukris M, Gazit I. Rate matching and de-rate matching for an LTE transport channel. US Patent; 2010.

17. Zhu D, Mathews VJ, Detienne DH. A likelihood-based algorithm for blind identification of QAM and PSK signals. *IEEE Trans Wireless Commun.* 2018;17(5):3417-3430.
18. Inc Xilinx. Zc702 Evaluation Board for the Zynq-7000 XC7Z020, All programmable SoC user guide. [https://www.xilinx.com/support/documentation/boards\\_and\\_kits/zc702\\_zvik/ug850-zc702-eval-bd.pdf](https://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf); 2015.
19. Sadek A, Mostafa H, Nassar A, Ismail Y. Towards the implementation of multi-band multi standard software-defined radio using dynamic partial reconfiguration. 2017;30(17):e3342.

**How to cite this article:** Eladawy M, Mostafa M, Sameh Said M, Mostafa H. Automated performance-based design technique for an efficient LTE PDSCH implementation using SDSoC tool. *Int J Commun Syst.* 2020;33:e4202. <https://doi.org/10.1002/dac.4202>