EPiC Series in Computing

Volume 58, 2019, Pages 36–44

Proceedings of 34th International Conference on Computers and Their Applications

Optimization of Advanced Encryption Standard (AES) Using Vivado High Level Synthesis (HLS)

Luka Daoud[1, 2], Fady Hussein[1, 3], and Nader Rafla[1, 4]

[1] Electrical and Computer Engineering
Boise State University, Boise, ID 83725
[2] LukaDaoud@u.boisestate.edu
[3] FadyHussein@u.boisestate.edu
[4] nrafla@boisestate.edu

Abstract

Advanced Encryption Standard (AES) represents a fundamental building module of many network security protocols to ensure data confidentiality in various applications ranging from data servers to low-power hardware embedded systems. In order to optimize such hardware implementations, High-Level Synthesis (HLS) provides flexibility in designing and rapid optimization of dedicated hardware to meet the design constraints. In this paper, we present the implementation of AES encryption processor on FPGA using Xilinx Vivado HLS. The AES architecture was analyzed and designed by loop unrolling, and inner-round and outer-round pipelining techniques to achieve a maximum throughput of the AES algorithm up to 1290 Mbps (Mega bit per second) with very significant low resources of 3.24% slices of the FPGA, achieving 3 Mbps per slice area.

keywords: Advanced Encryption Standard, AES, High Level Synthesis, HLS, Optimization, High throughput, Low area resources, Zynq, FPGA.

1 Introduction

Advanced Encryption Standard (AES) is a standardized algorithm approved by the National Institute of Standards and Technology (NIST) [16]. It has been adopted by numerous applications ranging from data servers to low-power hardware embedded systems to ensure data secrecy and privacy. However, AES-based block cipher is computationally intensive and time demanding for software implementation on general purpose processors, which leads to hardware acceleration of the AES algorithm on application-specific integrated circuit (ASIC) or reconfigurable hardware devices such as field programmable gate arrays (FPGAs). It is known that current embedded systems may depend on dedicated hardware accelerators for data encryption and decryption.

The AES encryption algorithm consists of several rounds of encryption and each round is comprised of three main layers to apply data confusion through nonlinear transform and data diffusion by mixing the data state. The algorithm for each round takes the state array and, after applying a round encryption, returns an updated state array. The implementation

and optimization of such complex functions in Hardware Description Languages (HDL) is time consuming and not easily optimized. In order to achieve an efficient design with less effort, High-Level Synthesis (HLS) procedures are applied. HLS is an automated process that accepts a system design created in a high level language, such as C or C++, and then generates a Register Transfer Level (RTL) design describing the behavior of the system. HLS plays a vital role in the design process by reducing the effort of HDL design and debugging, and providing flexibility in the final hardware implementation to meet design constraints set by the developer.

In this paper, we present the implementation of AES using Vivado High Level Synthesis [19] and evaluate the performance of the proposed architecture. The design is implemented on the Xilinx Zynq-7000 SoC FPGA chip of the ZedBoard prototyping board [20]. The standard AES-128 block cipher consists of a full 10 rounds of data permutation and mixing. Each round was optimized and pipelined to achieve high throughput with minimum area cost. Our proposed AES design was only implemented by look-up tables (LUTs) and flip flops (FFs) without including any block RAM (BRAM) or DSP slices of the FPGA. Therefore, our design may be appealing to low cost and high throughput applications. Additionally, our proposed HLS implementation of the standard AES-128 is compared to previous implementations on FPGAs. The rest of this paper is organized as follows: Background and the related works are explored in Section 2, an overview of HLS and relevant techniques are represented in Section 3, implementation of the AES block cipher algorithm and its optimization is described in Section 4 and compared to previous work as well. Finally, Section 5 concludes the work done and gives suggestions for future work.

2 Background and Related Work

The AES is a symmetric block cipher [16] and its fundamental operations are performed on byte-level field over the Galois Field GF(2^8) [15], where the input block is divided into a set of 8-bit vectors. The algorithm encrypts and decrypts a 128-bit block of data by repeatedly applying the same round transformation and using a secret key. The key size can be either 128, 192, or 256 bits and is chosen based on the preferred security level. The different versions of AES are known as AES-128, AES-192, or AES-256, respectively with the key size and the corresponding number of rounds, 10, 12, and 14, respectively. AES-128 is well known and supported by most hardware implementations. So, in this paper, we present the implementation of the AES-128 on FPGA using HLS.

2.1 AES Structure

Figure 1 shows the different phases of the standard AES algorithm. It starts with an initial round followed by a number of standard rounds and ends with a final round. Each standard round, intermediate cipher, has four different operations to scramble and non-linearly transform the data. The intermediate results is called a state and represented as a 2-D matrix notation of 4×4 of bytes.

SubBytes: is an invertable non-linear transformation. It is a substitution process of each byte of the input state by another one from a predefined table (S-box). The size of the table is 256 different elements of bytes. The design criteria of the S-box values is to be resistant against the known differential and linear crypto-analysis. Each possible element of the S-box is generated by computing the multiplicative inverse in GF(2^8) and then applying an affine transformation.

ShiftRows: is an operation of cyclically shifting (rotating) each row with a different offset.

MixColumns: is an operation on the different columns by performing a polynomial multipli-

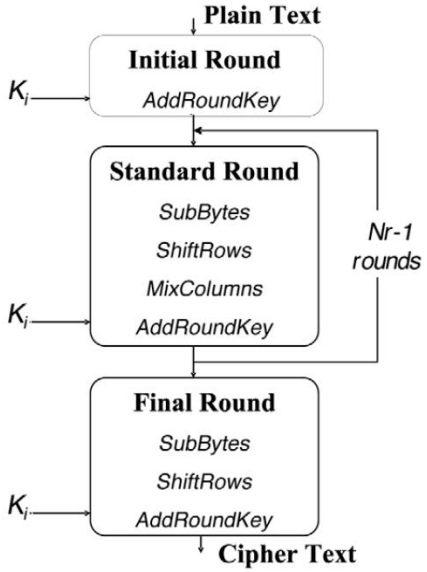

Figure 1: AES design block [15]

cation in GF(2^8)

AddRoundKey: is an operation of bit-wise XORing the round key (sub-key) with the current state. Each round key is derived from the previous sub-key. This requires the encryption algorithm to schedule the key for each round.

The key schedule takes the original input key and derives a sub-key for each round. For AES-128, the number of rounds is 10, and 11 sub-keys are derived, each of 128 bits. The sub-key derivation is computed recursively and the first sub-key is the original input key. The AES key schedule is word-oriented of word size = 32 bits. Figure 2 shows the AES key schedule for 128-bit key size, where the purpose of the function g() is not only to add non-linearity to the key schedule but also to remove symmetry in the AES, and RC[i] is the round coefficient that varies from round to round.

In the standard AES-128, the initial round is done by applying AddRoundKey, i.e., XORing the key with the input data. Then, it is followed with 10 repeated rounds and each round performs SubBytes, ShiftRows, MixColumns and AddRoundKey. The final round is slightly different by dropping the MixColumns function.

2.2 Related Work

A considerable amount of literature has been proposed and evaluated on the implementation of the AES algorithm on FPGA [2, 4, 5, 7], and ASIC [13, 14, 17]. Most of the implementations feature high speeds and high costs suitable for high-end applications only. Early AES designs were mostly straightforward implementations of various loop unrolled and pipelined architectures. Recent implementations of the AES on FPGA demonstrate better utilization of FPGA resources using dedicated on-chip memories implementing S-boxes and DSP slices [6].

In [4], the authors studied and compared the performance of implementations of different candidates of the AES encryption techniques on FPGA and evaluate their suitability for FPGA-

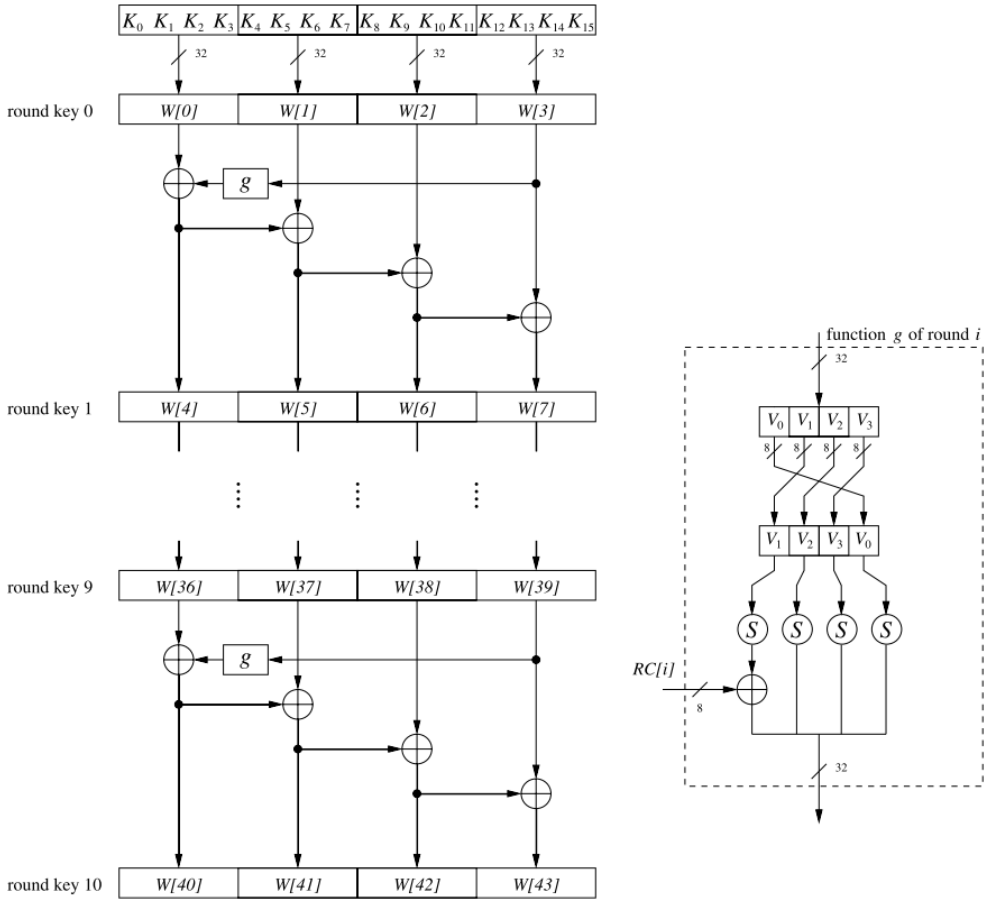

Figure 2: Key expansion for each round [15]

based implementations. They focused on time performance and the encryption throughput. Authors in [2] presented a hardware implementation of the AES algorithm developed for an external data storage unit in a dependable application and optimization of the encryption algorithm to meet the needs of the target application. In [7], authors focused in optimizing the AES algorithm to suit small embedded applications or low power consumption devices. They achieved a throughput of 121 Mbps at a maximum frequency of 153 MHz targeting small area design and lower energy consumption per processed block.

However, HLS has been used to optimize implementations of the cryptography protocols in hardware [8, 11]. There are few works that uses HLS to implement the AES algorithm on FPGAs [1, 9, 12, 18].

In [1], authors investigated various optimizations of the C-based AES implementation into hardware using C2R [3] methodology for co-processor synthesis. These implementations included baseline hardware design, BRAM-based architecture, a pipelined scheme, and an optimized architecture for performance and area. In [12], the authors explored different hardware implementations of the AES using HLS directives and memory partitioning optimizations. In [18], authors explored four different implementation methods of the AES using Vivado HLS

for different types of the substitution table.

3 Vivado High Level Synthesis

Systems and hardware architectures are generally designed using HDL which models the operation in RTL primitives. On the other hand, high-level languages can efficiently model systems and applications with less effort and better ease of configurability. In order to reduce the amount of design time necessary, high-level synthesis tools are used for transforming systems modeled in a high-level language into a RTL implementation that can be synthesized into FPGAs [10]. HLS not only reduces the effort of HDL design and debugging but also provides flexibility in the final hardware implementation to meet design constraints.

The Xilinx Vivado HLS tool [19] transforms a C specification design, such as C, C++, and System-C, into a RTL design. The tool goes through several phases to achieve an optimized design. The HLS tool schedules logic operations for each clock cycle and assigns hardware resources for each scheduled operation. From the flow of the design, the control logic is extracted create a finite state machine which sequences the operations in the RTL design.

The Vivado HLS tool synthesizes the C functions into blocks within the RTL hierarchy. The top-level function arguments are synthesized, as appropriate, into I/O ports accompanied by appropriate hand-shaking signals. The HLS tool allows the hardware developers to analyze and optimize the design. Based on default behavior, constraints, and any optimization directives; the tool creates an optimal RTL implementation of the high-level design. The tool then generates a set of synthesis reports which are used to analyze the implementation and, hence, several optimizations are applied to meet the design constraints.

In this work, the AES encryption algorithm was implemented in C-language. The algorithm was synthesized and co-simulated by the Vivado HLS tool to check the functionality of the RTL design. Then, the design was analyzed and optimized to achieve higher throughput.
The throughput, T_p, in this paper is calculated as:

$$T_p = \frac{Block\_Size}{N_C * T_{CLK}} \tag{1}$$

, and the throughput to area ratio is calculated as:

$$K = \frac{T_p}{A} \tag{2}$$

where $Block\_Size$ is the size of a block in bits, i.e., 128 bits, N_C is the number of clock cycles necessary to encrypt a single block, T_{CLK} is the maximum delay path, and A, area, is the number of slices from the Vivado utilization report.

4 AES Implementation Optimization

In this section, we will explain our implementation and optimization of the AES using Vivado HLS. We will explore different HLS optimizations: SW-based baseline implementation, Key Expansion-based implementation, and High throughput-based optimization.

The AES-128 encryption algorithm is composed of 10 rounds and the round is a set of functions as described in Section II. The HLS tool synthesized the AES-128 top function into a RTL block and each sub-function into a sub-block instantiated into the top-level design. On the other hand, the top-function arguments are synthesized into input/output (I/O) ports. The

tool allows us to choose the handshaking protocol to be implemented onto the I/O ports of the designed block(s). I/O ports can be implemented as streaming data to/from a FIFO, or as reading/writing data to/from a memory. There are other handshaking protocols which can be implemented, as necessitated by the design. In our implementation, the design was synthesized to accommodate receiving a stream of data. The output data is synthesized to allow communication with a dual-port memory.

SW-based Implementation: This scheme is the baseline AES algorithm designed for software implementation, where the key extension is executed first before starting the encryption process. The purpose of the key expansion module is to generate the 11 different extended keys each is 128-bit size. In this version, all loops are rolled and no optimization is applied. This implementation led to an encryption of one block in 2556 clock cycles using 154 slices of the FPGA.

```
// SW-based Encryption Function
void AES_Encrypt(unsigned char state[16], unsigned char msg[16], unsigned char
    key[16]){

for (int i = 0; i < 16; i++)
    state[i] = msg[i];

// Rounded key Generation
unsigned char expanded_key[176];
key_expansion(key, expanded_key);

add_round_key(state, key);

for (int j = 0; j < round_cnt; j++) {
    sub_bytes(state);
    shift_rows(state);
    mix_columns(state);
    add_round_key(state, (expanded_key + (16 * (j + 1))));
  }

// Final round
sub_bytes(state);
shift_rows(state);
add_round_key(state, (expanded_key + 160));
}
```

Key Expansion-based Implementation: We modified the C-based code of the AES to merge the key expansion with the encryption round. So, the algorithm starts the encryption process and the extended key is generated in parallel while execution each round. The modified code was synthesized with the default set of the HLS tool. This implementation led to an encryption of one block in 2176 clock cycles using 165 slices of the FPGA.

High Throughput-based Optimization: We applied our optimization directives to the key Expansion-based design to achieve high performance encryption.

The aim of the design is to obtain maximum throughput by unrolling loops and applying pipeline with initiation interval = 1. For achieving high throughput, some arrays were fully partitioned and dependence directives were applied to overcome loop-carry dependencies. The most effective optimization is merging the key expansion with the encryption round. In this case, some Pragrmas were applied to clear dependency between the expanded keys and the encryption round. For comparison purposes, we constrained the HLS tool to synthesize the embedded memory blocks into FPGA slices. The following code shows some of the applied Pragmas to achieve high throughput.

```
// Throughput-based Encryption Function
void AES_Encrypt(unsigned char state[16], unsigned char msg[16], unsigned char
    key[16]){

unsigned char ExtendedKey[16];
#pragma HLS ARRAY_PARTITION variable=ExtendedKey complete dim=1

for (int i = 0; i < 16; i++){
#pragma HLS UNROLL
     state[i] = msg[i];
     ExtendedKey[i] = key[i];
  }

for (int j = 0; j <= round_cnt; j++) {
#pragma HLS PIPELINE

    add_round_key(state, ExtendedKey);

// create a region to set false dependence of the extended key
    {
        key_expansion(ExtendedKey, j);
#pragma HLS UNROLL
#pragma HLS DEPENDENCE variable=Extendedkey inter false
    }

    sub_bytes(state);
    shift_rows(state);

    if (j != round_cnt)
        mix_columns(state);
    else
        add_round_key(state, ExtendedKey);
  }
}
```

This solution is able to execute the encryption of one block within 19 clock cycles using 431 slices (3.24%) of the FPGA. This optimized solution is at the expense of the FPGA's resources.

It can be noted that the throughput difference between the throughput-based implemenation and the other two implementations are so vast. This indicates that by simply implementing high-level code into a HLS tool and then optimizing in a way which is beneficial for design constraints can speed up development time. Thus, by using HLS, rapid optimization can be accomplished with results similar to dedicated HDL designs.

The proposed optimization was compared to previous HLS implementations in the literature. The area utilization, in slices, maximum achieved frequency, in Mhz, throughput, in Mbps, and throughput to area ratio, in Mbps/slice, are shown in Table 1 for the Throughput-based implementation and the previous works. Our proposed optimization in HLS achieved higher throughput per area and less number of slices is required to implement the proposed architecture.

Table 1: AES hardware implementation comparison.

Implementation	**Area**				**Max Freq MHz**	**Mbps**	$\frac{\textbf{Mbps}}{\textbf{slice}}$
	LUT	**FF**	**BRAM**	**SLICE**			
[1]	14588	5328	80	7670	144	1530	0.2
[12]	-	-	-	646	-	1393	2.2
Ours	1417	830	0	431	192	1290	3

The synthesized RTL design of the throughput-based optimization was exported to Vivado and design implementation was completed for further analysis. The experiments were conducted using Xilinx Zynq-7000 SoC, Zedboard, XC7Z020-1CLG484C [20] along with the Xilinx Vivado Design suite and SDK 17.4. The FPGA fabric runs normally on 100 MHz (used in this context), but can be configured up to 192 MHz, the maximum frequency of the encryption module.

5 Conclusion

In this work, we explored the standard AES encryption and its implementation into a Xilinx ZedBoard with the Zynq-7000 SoPC. This work focused on the encryption aspect of AES-128, but the decryption part could easily be implemented and tested as well. The AES was initially coded in a high-level language and was then implemented with Xilinx Vivado High Level Synthesis. The Xilinx HLS tool enabled us to quickly realize our design and make optimizations which greatly increased throughput of the AES algorithm. HLS also offers the potential to allow for hardware benchmarking in early design stages and for in-depth analysis of a design's resource usage versus high-level code placement.

The most successful optimization implemented in our design was the pipelining of the function's for-loops besides unrolling loops and computing the extended key on the fly during the encryption process, which reduced the initiation interval and allowed for concurrent execution of operations within loops and functions.

The encryption throughput of the proposed AES in HLS observed to be 1.26 Gbps. This rapid development and optimization of HLS-ready code shows that HLS can be used to increase a designer's productivity by applying directives such as pipelining, array shaping, and port mapping to their new and existing designs. A designer is thus able to see a moderate improvement without the need to design RTL with traditional, and time consuming, HDL languages.

Some future work would include further optimization of the AES algorithm. Adaptation for HLS can be achieved through writing optimized code; both with standard HDL and manual implementation of a pipelined structure. The future work would also be implemented into the same prototyping board for fair comparison. Additional future work may also include the utilization of the AXI interface currently existing within the Xilinx Zynq-7000 series of FPGAs. The on-board processor is able to use the FPGA for hardware acceleration, as opposed to complete implementation of the AES algorithm in the FPGA.

As a future work, different modes of encryption for AES to encrypt successive blocks of data including counter (CTR), cipher block chaining (CBC), cipher feedback (CFB), and output feedback (OFB) can be implemented and optimized using HLS and compared to their counterpart RTL implementations on FPGA.

References

[1] Sumit Ahuja, Swathi T Gurumani, Chad Spackman, and Sandeep K Shukla. Hardware coprocessor synthesis from an ansi c specification. *IEEE Design & Test of Computers*, 26(4):58–67, 2009.

[2] Marko Mali—Franc Novak—Anton Biasizzo. Hardware implementation of aes algorithm. *Journal of Electrical Engineering*, 56(9-10):265–269, 2005.

[3] C2R Compiler. C2r compiler, 2018. http://www.cebatech.com/.

[4] Andreas Dandalis, Viktor K Prasanna, and Jose DP Rolim. A comparative study of performance of aes final candidates using fpgas. In *International workshop on cryptographic hardware and embedded systems*, pages 125–140. Springer, 2000.

[5] Ashwini M Deshpande, Mangesh S Deshpande, and Devendra N Kayatanavar. Fpga implementation of aes encryption and decryption. In *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on*, pages 1–6. IEEE, 2009.

[6] Saar Drimer, Tim Güneysu, and Christof Paar. Dsps, brams and a pinch of logic: new recipes for aes on fpgas. In *Field-Programmable Custom Computing Machines, 2008. FCCM'08. 16th International Symposium on*, pages 99–108. IEEE, 2008.

[7] Panu Hamalainen, Timo Alho, Marko Hannikainen, and Timo D Hamalainen. Design and implementation of low-area and low-power aes encryption hardware core. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 577–583. IEEE, 2006.

[8] HS Jacinto, Luka Daoud, and Nader Rafla. High level synthesis using vivado hls for optimizations of sha-3. In *IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 563–566. IEEE, 2017.

[9] Makoto Kotegawa, Keisuke Iwai, Hidema Tanaka, and Takakazu Kurokawa. Optimization of hardware implementations with high-level synthesis of authenticated encryption. *Bulletin of Networking, Computing, Systems, and Software*, 5(1):26–33, 2016.

[10] L. Daoud, D. Zydek, and H. Selvaraj. A Survey of High Level Synthesis Languages, Tools, and Compilers for Reconfigurable High Performance Computing. In *Advances in Systems Science*, pages 483–492. Springer, 2014. , DOI: 10.1007/978-3-319-01857-7_47.

[11] Muhammad Latif, HS Jacinto, Luka Daoud, and Nader Rafla. Optimization of a quantum-secure sponge-based hash message authentication protocol. In *IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 984–987. IEEE, Aug 2018.

[12] Rodrigo Schmitt Meurer, Tiago Rogerio Muck, and Antonio Augusto Frohlich. An implementation of the aes cipher using hls. In *Computing Systems Engineering (SBESC), 2013 III Brazilian Symposium on*, pages 113–118. IEEE, 2013.

[13] Sumio Morioka and Akashi Satoh. An optimized s-box circuit architecture for low power aes design. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 172–186. Springer, 2002.

[14] Sumio Morioka and Akashi Satoh. A 10-gbps full-aes crypto design with a twisted bdd s-box architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(7):686–691, 2004.

[15] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[16] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal information processing standards publication*, 197(441):0311, 2001.

[17] Ingrid Verbauwhede, Patrick Schaumont, and Henry Kuo. Design and performance testing of a 2.29-gb/s rijndael processor. *IEEE Journal of Solid-State Circuits*, 38(3):569–572, 2003.

[18] Masashi Watanabe, Keisuke Iwai, Hidema Tanaka, and Takakazu Kurokawa. High-speed implementation of encryption circuit using a high-level synthesis tool. *Bulletin of Networking, Computing, Systems, and Software*, 3(1):63–66, 2014.

[19] Xilinx Inc. *Vivado Design Suite: High-Level Synthesis*, July, 2018.

[20] Xilinx Inc. *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 User Guide*, June, 2018.