https://doi.org/10.31436/iiumej.v26i1.3328

OPTIMIZING FPGA RESOURCE ALLOCATION FOR SHA-3 USING DSP48 AND PIPELINING TECHNIQUES

AGFIANTO EKO PUTRA*, OSKAR NATAN, JAZI EKO ISTIYANTO

Department of Computer Science and Electronics, Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada, Yogyakarta - Indonesia

**Corresponding author: agfi@ugm.ac.id*

(Received: 13 June 2024; Accepted: 29 October 2024; Published online: 10 January 2025)

ABSTRACT: Deploying SHA-3 on FPGA devices requires significant resource allocation; however, the resulting throughput still needs improvement. This study employs the DSP48 module on the Xilinx FPGA to address this issue and implements an eight-stage pipeline methodology to minimize latency. The implementation design comprises a datapath and controller module, utilizing a Xilinx Artix-7-100T series FPGA as the hardware. This method makes use of FPGA resources like Look-Up Tables (LUT), Look-Up Table Random Access Memory (LUTRAM), Flip-Flops (FF), Block RAM (BRAM), Digital Signal Processing (DSP), Input/Output (IO), and Buffer (BUFG). The system's highest frequency is 107.979 MHz, achieving different throughputs for cryptographic hash functions. Specifically, it performs a throughput of 5.183 Gbps for SHA3-224, 4.895 Gbps for SHA3-256, 3.743 Gbps for SHA3-384, and 2.591 Gbps for SHA3-512.

ABSTRAK: Menggunakan SHA-3 pada peranti FPGA memerlukan peruntukan sumber yang ketara, walaupun daya pengeluaran yang terhasil adalah terhad. Untuk menangani isu ini, kajian ini menggunakan modul DSP48 yang disertakan pada Xilinx FPGA dan melaksanakan metodologi saluran paip lapan peringkat untuk meminimumkan kependaman. Reka bentuk pelaksanaan terdiri daripada laluan data dan modul pengawal, menggunakan siri FPGA Xilinx Artix-7-100T sebagai perkakasan. Kaedah ini menggunakan sumber FPGA seperti Look-Up Tables (LUT), Look-Up Table Random Access Memory (LUTRAM), Flip-Flops (FF), Block RAM (BRAM), Digital Signal Processing (DSP), Input/Output (IO), dan Penampan (BUFG). Kekerapan tertinggi sistem ialah 107.979 MHz, dan ia mencapai daya pemprosesan yang berbeza untuk fungsi cincang kriptografi yang berbeza. Secara khususnya, ia mencapai daya pemprosesan 5.183 Gbps untuk SHA3-224, 4.895 Gbps untuk SHA3-256, 3.743 Gbps untuk SHA3-384 dan 2.591 Gbps untuk SHA3-512.

KEYWORDS: *SHA-3, FPGA, DSP48, pipeline*

1. INTRODUCTION

NIST released SHA-3, the most recent cryptographic standard within the Secure Hash Algorithm family, on August 5, 2015. The algorithm aims to generate secure hash codes for use within computer and network security applications. SHA-3 compresses unlimited data into limited data by employing its functions. SHA-3 is designed based on permutation and has various versions, including SHA3-224, SHA3-256, SHA3-384, and SHA3-512, each with distinct security levels. The core component of SHA-3 is the permutation function that converts input data into SHA-3 hash. To generate a hash, a state register of 1,600 bits arranged in a three-dimensional matrix of 5x5*64 bits is necessary [1].

https://doi.org/10.31436/iiumej.v26i1.3328

An FPGA was used to test the throughput and slices required to implement SHA-3. Direct implementation of SHA-3 on an FPGA involves extensive and complex wiring demanding significant resources despite low throughput [2, 3]. The algorithm is optimized by merging all the compression functions into one, thereby eliminating intermediate states. The resulting compact version is implemented using LUT on the FPGA. Rao et al. found this implementation to increase throughput, although it still significantly utilizes many resources [4]. While SHA-3 provides robust security, its real-world application on FPGA devices presents significant challenges, particularly regarding resource allocation and throughput. Addressing these challenges is crucial for enabling efficient and scalable cryptographic implementations.

The look-up table (LUT) is a critical logic block in FPGA architecture, widely used for implementing specific logic functions. LUT_6, with six inputs and one output, is vital in realizing complex operations like 6-input logic functions or 64-bit asynchronous ROM. Permutation functions are fundamental in cryptographic applications like SHA-3, requiring extensive data manipulations between 64-bit blocks. As a result, 64 LUTs are often needed to execute such operations. While utilizing LUT_6 helps improve the performance and throughput of systems, it also demands significant hardware resources, especially in high-performance designs that aim to minimize area usage and maximize speed. For instance, optimizing SHA-3's round constants generator shows that a more efficient structure can reduce resource consumption while maintaining high throughput [5]. Pipelined architecture can be implemented at the loop level in FPGA designs to reduce idle time and improve efficiency. Simultaneously executing different stages, such as candidate generation and fitness function evaluation, the architecture ensures that multiple data sets are processed in parallel across iterations. This approach enhances resource utilization and allows for faster execution without increasing latency. In the BB-BC algorithm, this pipelining helps minimize resource usage while maintaining high throughput, making it particularly effective for real-time optimization problems and reducing overall computational time [6, 7].

Despite these efforts, a significant gap must be in optimizing resource allocation and power consumption when implementing SHA-3 on FPGAs. While LUT-based methods enhance throughput, they consume substantial resources and power. Previous research has yet to fully address the efficiency challenges in cryptographic implementations, particularly in balancing high throughput with reduced hardware complexity. To address this gap, this study explores using DSP48 blocks for XOR computations and pipeline techniques to minimize resource usage, aiming to improve FPGA efficiency without compromising performance. Consequently, implementing SHA-3 on FPGA demands significant power and offers limited throughput, while optimizing the SHA-3 algorithm on FPGA necessitates compromising resource usage to increase throughput. This article outlines a solution to this issue by incorporating DSP48 to compute XOR and using a pipeline to handle the system, allowing for implementing SHA-3 on FPGA.

2. RESEARCH METHOD

The DSP48 is a resource employed for carrying out specialized operations, specifically those related to the computational chain in DSP48 containing add/subtract units linked to multipliers. It is also connected to the final add/remove/accumulate engine. When used on FPGAs, it leads to high throughput and power efficiency [8]. DSP48 can also optimize complex operations like 2D convolution. Additionally, DSP48 can handle multiplication and accumulation within the block itself, reducing the need for extra logic, such as adders [9]. In this research, DSP48 enables more efficient resource allocation on the FPGA by reducing the need for additional logic components like LUTs and external adders [10, 11]. In this research,

https://doi.org/10.31436/iiumej.v26i1.3328

CascadeDSP will be used for the implementation. CascadeDSP is a technique in which multiple DSP blocks are connected in a cascading configuration, allowing them to work together to handle more complex operations that require higher bit-width or multiple stages of computation. Furthermore, DSP48 blocks are integrated into the pipeline architecture designed to minimize latency while maintaining high throughput, ensuring that operations are executed parallel to enhance overall system performance. Using the DSP48 slices, the system optimizes the processing of large data blocks required by the SHA-3 algorithm, achieving both efficiency and high performance in the cryptographic function's FPGA implementation.

A pipeline architecture can be utilized to reduce the delay time in SHA-3 implementation. The pipeline functions so that various execution stages can be conducted simultaneously on different data. The pipeline architecture is expected to enhance the utilization of FPGA resources without compromising computational speed [6]. In this system, the pipeline architecture is employed to reduce the latency of implementing the SHA-3 cryptographic algorithm. The pipeline operates by dividing the SHA-3 permutation function into several stages, such as Theta, Chi, and Iota, and allowing different operations to be executed in parallel across multiple stages [12, 13]. This design ensures that while one stage processes a data block, the next stage can begin processing the subsequent block, optimizing the data flow and reducing idle time. Specifically, the system utilizes an eight-stage pipeline, which includes internal registers within the DSP48 slices at critical points in the Theta and Chi stages to synchronize data and ensure efficient processing [14]. The Theta stage uses external serial registers to ensure data arrives promptly. In contrast, the Rho and Pi stages do not require pipelining as they perform more straightforward rotation and permutation operations. Implementing this pipeline structure achieves higher throughput without compromising execution speed, making it ideal for high-performance applications like SHA-3.

The system design incorporates three modules: input, processing, and output, which are depicted in Figure 1. The input module receives data serially and transmits it in parallel to the next module. This module receives data from a PC through a USB-UART interface. The GUI on the PC sends serial data to the FPGA, which the input module receives and then converts from serial to parallel format. This parallel data is crucial because it is easier to process in the subsequent stages of the system, especially in high-throughput cryptographic operations like SHA-3. The processing module comprises a data storage unit for temporarily holding data from the input module and the results generated by the SHA-3 permutation function. In the data storage unit, input data is stored before processing by the SHA-3 permutation function, which handles the system's core cryptographic operations. The SHA-3 permutation function unit performs crucial stages of the SHA-3 algorithm, including Theta, Rho, Pi, Chi, and Iota, which collectively transform the input data into its final cryptographic hash.

The controller unit plays a vital role in governing the overall flow of operations within the system. It ensures proper timing and coordination of data movement between the input, processing, and output modules, allowing smooth and efficient operation. After the data has been processed, the output from the processing module is transferred to the output module, which then transmits the serialized data back to the PC. The input and output modules communicate at 115,200 bits per second, ensuring fast and reliable data transmission between the FPGA and the external system. This high-speed communication enables efficient real-time processing and transmission of cryptographic data, optimizing system performance for practical applications.

https://doi.org/10.31436/iiumej.v26i1.3328

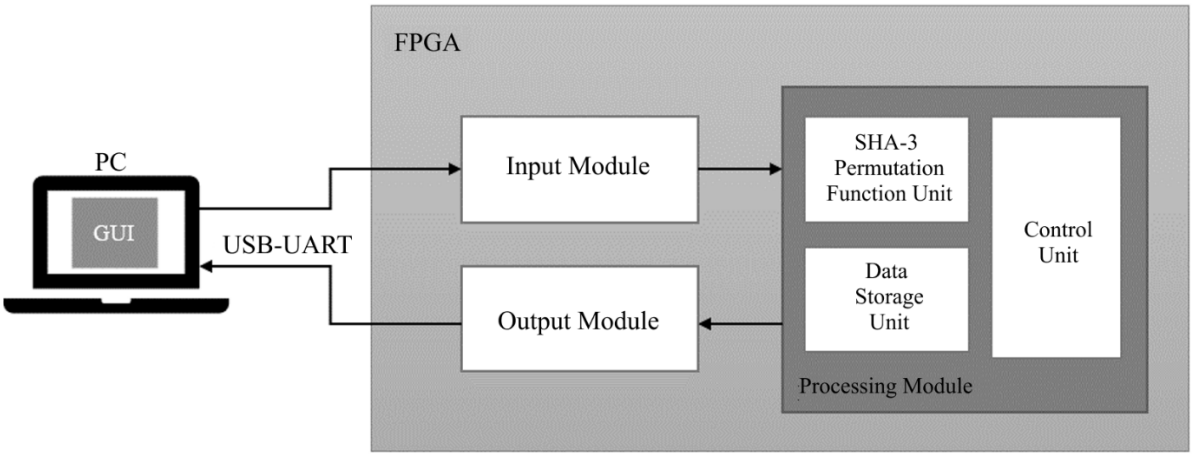

Figure 1. System block diagram

2.1. Design of the Top Layer System (Top Level Design)

The top-level design is the highest-level module in the hierarchy of all the designed modules. This design comprises four main components: UART_RX and UART_TX act as data receivers and senders to the PC via the UART serial port; the datapath is the system's core that performs SHA-3 operations, and the controller unit maintains regulation of system operations. Figure 2 shows the top-level block diagram. The top-level design features input and output ports. The input port comprises 'CLK100MHZ' linked to the FPGA clock generation pin and 'UART_TXD' connected to the serial communication pin for receiving data from the PC. The output port, 'UART_RXD,' transmits serial data to the PC via the relevant pin.

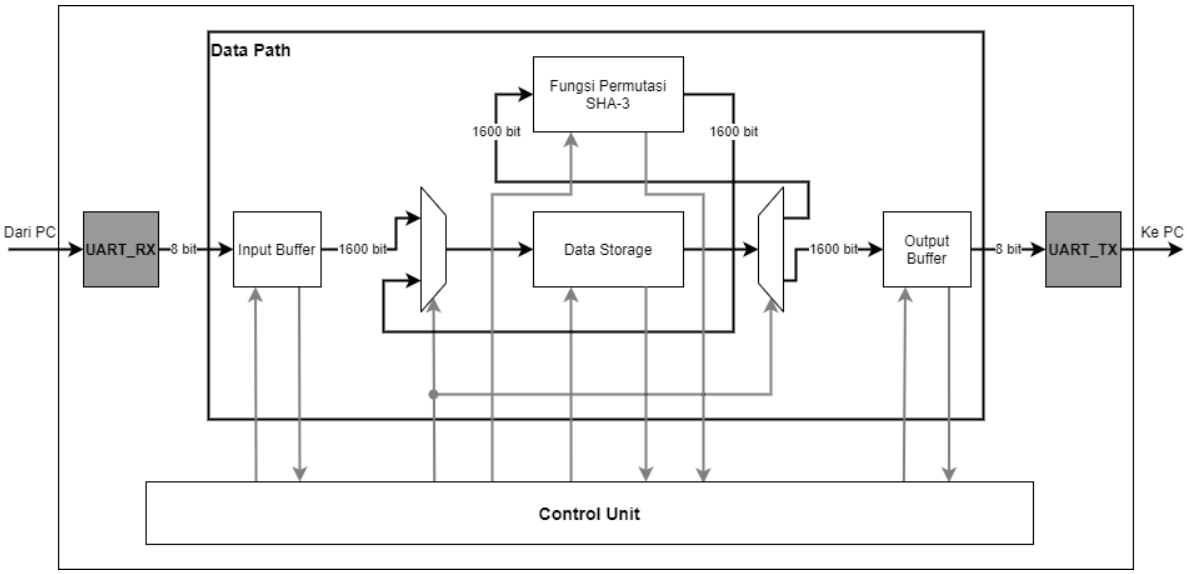

Figure 2. Top-level design block diagram

2.2. Datapath Module Design

The datapath module processes input data from the PC to generate new data via the SHA-3 permutation function operation. It comprises the input buffer, output buffer, data storage, and SHA-3 permutation function modules. The SHA-3 permutation function module, depicted in Figure 3, shall consist of five SHA-3 permutation function stages [1].

a. "Theta Step (θ): ($0 \leq x,y \leq 4$,)"

https://doi.org/10.31436/iiumej.v26i1.3328

$$C[x]=A[x,0]\oplus A[x,1]\oplus A[x,2]\oplus A[x,3]\oplus A[x,4]; \quad (1)$$

$$D[x]=C[(x-1)]\oplus ROT(C[(x+1),1]); \quad (2)$$

$$A[x,y]=A[x,y]\oplus D[x]; \quad (3)$$

b. "Rho Step" ("ρ")": $(0 \le x,y \le 4)$"

$$A[x,y]=ROT(A[x,y,],r[x,y]); \quad (4)$$

c. "Pi Step " ("π")": $(0 \le x,y \le 4)$"

$$B[y,(2x+3y)]=A[x,y]; \quad (5)$$

d. "Chi Step " ("χ")": $(0 \le x,y \le 4)$"

$$A[x,y]=B[x,y]\oplus(NOT(B[(x+1),y])AND(B[(x+2),y])); \quad (6)$$

e. "Iota Step " ("ι")":"

$$A[0,0]=A[0,0,z]\oplus RC[i]; \quad (7)$$

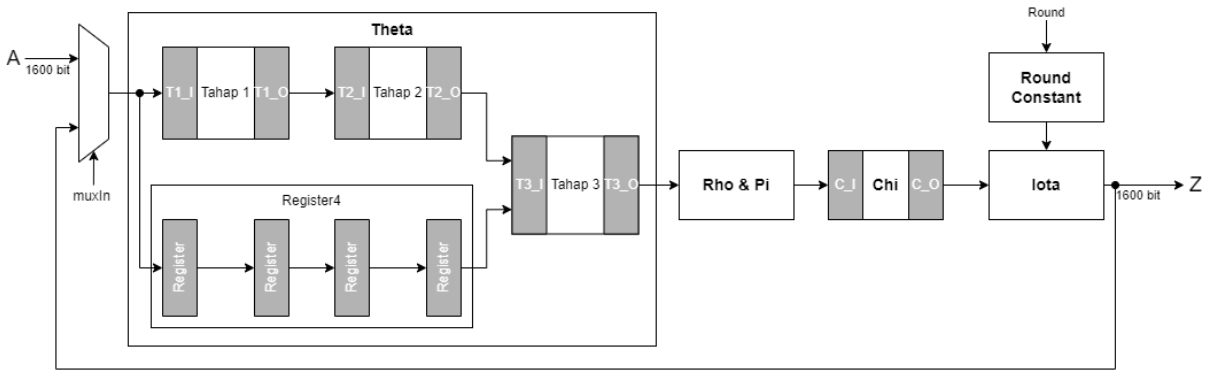

Figure 3. Block diagram of the SHA-3 permutation function

The DSP48 module executes logical equation operations through three stages - Theta, Chi, and Iota. The Theta stage comprises three sub-stages - the first sub-stage processes equation (1), the second sub-stage processes equation (2), and the third sub-stage processes equation (3). Notably, for equations involving more than two inputs for a single output, including equation (1) in the Theta stage and equation (6) in the Chi stage, the DSP48 employs a cascading method for connectivity. The methodology involves linking the output pin 'PCOUT' (Figure 4) of one DSP48 to the input pin 'PCIN' of another DSP48. The functionality of the 'PCOUT' pin is identical to that of the 'P' output pin on the DSP48, which has been explicitly configured for this cascading technique. This method was chosen due to the unique pins and pathways on the Xilinx FPGA DSP48, which allow for more efficient and rapid interconnection between DSPs in a single column compared to ordinary output pins [8].

The pipeline implementation in this research comprises eight stages that employ DSP48 internal registers at every input and output of stages 1, 2, and 3 Theta and Chi stages, as presented by the blue block in Figure 3. In the Theta module, we added four serial external registers that equalize the data arrival upon entering the Theta stage. The Rho and Pi modules lack pipelining because their sole purpose is rotation operations, which do not necessitate a DSP48. Similarly, the Iota module lacks pipelining as it merely executes an XOR operation on the A[0,0] matrix. This implementation mandates eight input data; each cycle inputs one data per module. One of these input data constitutes the complete data for SHA-3, which is 1,600 bits in size.

https://doi.org/10.31436/iiumej.v26i1.3328

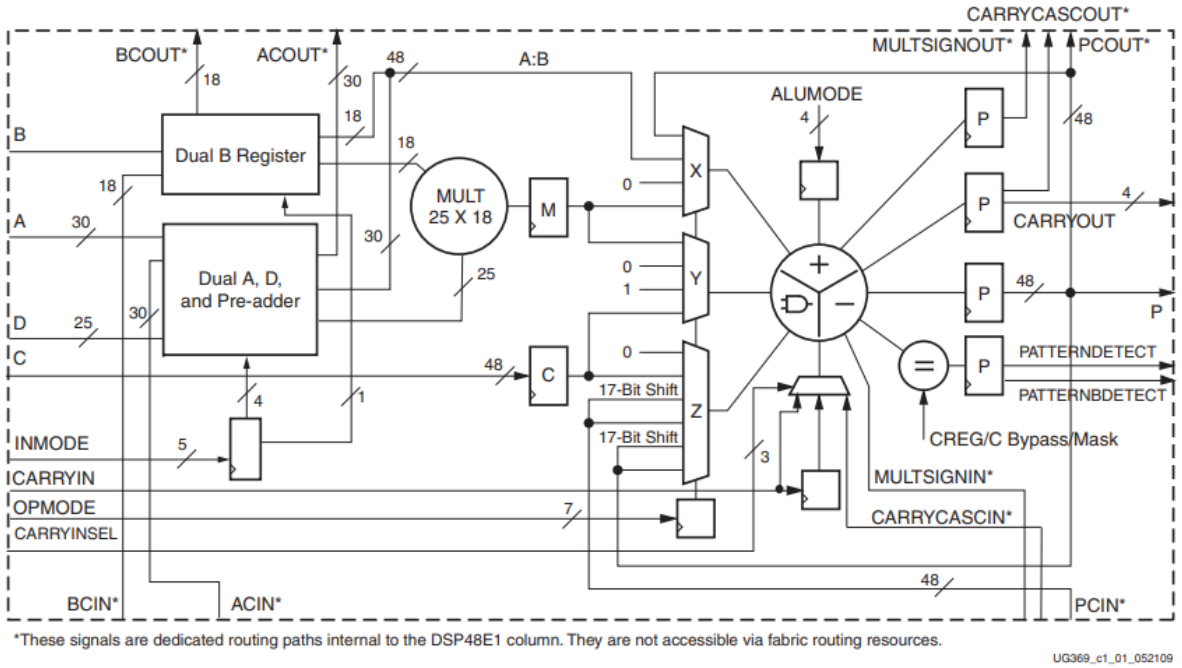

Figure 4. DSP48 block (Xilinx, 2018)

2.3. Controller module design

The controller module is implemented using a Finite State Machine (FSM), as depicted in Figure 5. Every condition in the FSM triggers the modules along with the MUX and DEMUX included in the data path module with the corresponding commands mentioned in Table 1.

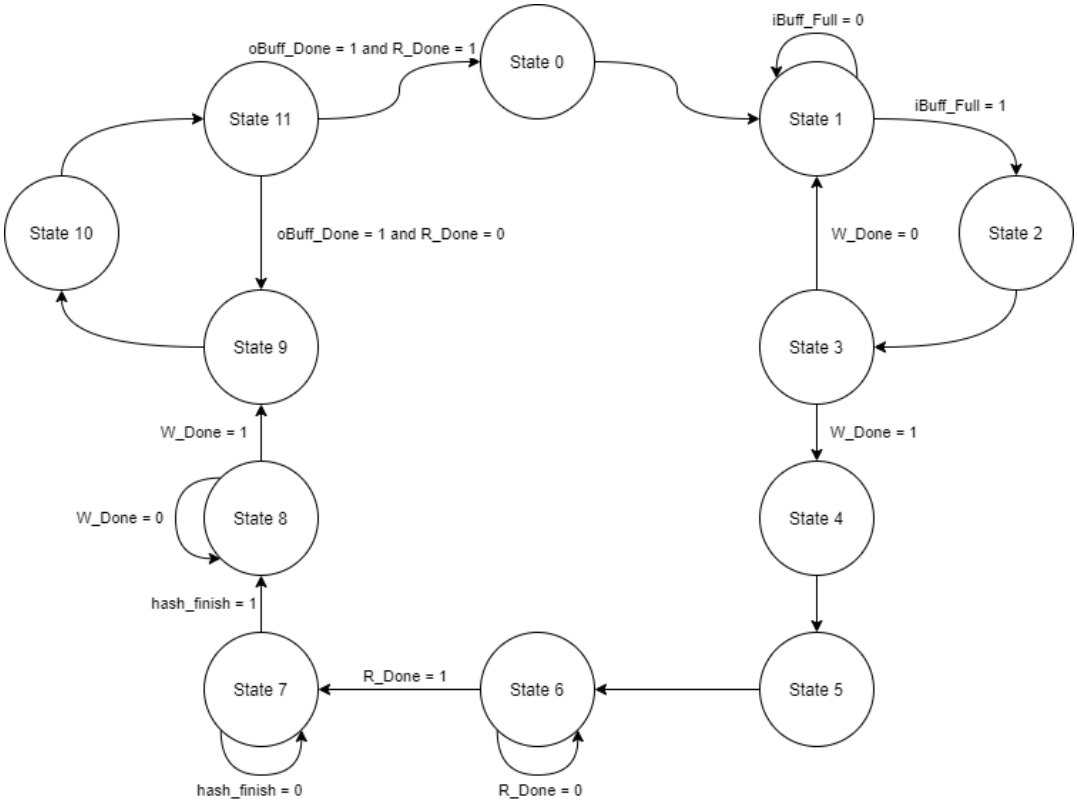

Figure 5. Controller module state diagram

https://doi.org/10.31436/iiumej.v26i1.3328

Table 1. Condition of control signals in each state

Signal/State	Reset_I	Reset_DS	Reset_FP	Reset_O	W_EN	R_EN	muxData
0	1	1	1	1	0	0	0
1	0	0	1	1	0	0	0
2	0	0	1	1	1	0	0
3	1	0	1	1	0	0	0
4	0	0	1	1	0	1	0
5	0	0	1	1	0	1	0
6	0	0	0	1	0	1	0
7	1	1	0	1	0	0	0
8	1	0	0	1	1	0	1
9	1	0	0	1	0	1	1
10	1	0	0	1	0	0	1
11	1	0	0	0	0	0	1

The control signals in Table 1 have the following functions:

1. Reset_I: Reset the input buffer module
2. Reset_DS: Reset the data storage module
3. Reset_FP : Reset SHA-3 permutation function module
4. Reset_O: Reset the output buffer module
5. W_EN: Store data in the data storage module
6. R_EN: Read data from the data storage module
7. muxData: Set MUX and DEMUX to determine the data to be stored in the storage module, whether from the PC input data or the SHA-3 process results.

The explanation of the state conditions in Table 1 are:

1. State 0: Reset all modules
2. State 1: Waiting for data input from PC.
3. State 2: Stores one SHA-3 data (1,600 bits) in the data storage module
4. State 3: Proceed to the next state if it has received 8 SHA-3 data; otherwise, it will return to state 1.
5. State 4 and 5: Initiate the data reading from the data storage module.
6. State 6: Initiate SHA-3 algorithm operation
7. State 7: Waiting for the SHA-3 process to complete
8. State 8: Save the SHA-3 result data to the data storage module
9. State 9 and 10: Initiate reading of one data from the data storage module
10. State 11: Returns to state 0 if all the SHA-3 output data has been sent to the PC; otherwise, it will return to state 9.

https://doi.org/10.31436/iiumej.v26i1.3328

2.4. GUI (Graphical User Interface) Design

The user interface displays the data sent to the FPGA and verifies whether the FPGA output matches the reference result (refer to Figure 6). To input the data for transmission, users select a text file containing eight datasets, with each dataset comprising three components: the first indicates the SHA-3 hash size of the input data under testing (224, 256, 384, or 512 bits); the second denotes the SHA-3 data to be dispatched to the FPGA; and the third represents the hash reference for comparison with the hash output from the FPGA. After the file has been successfully read, the second piece of information is merged and transmitted to the FPGA via the UART serial terminal. Once the program receives the hash result from the FPGA, it will be displayed through the interface.

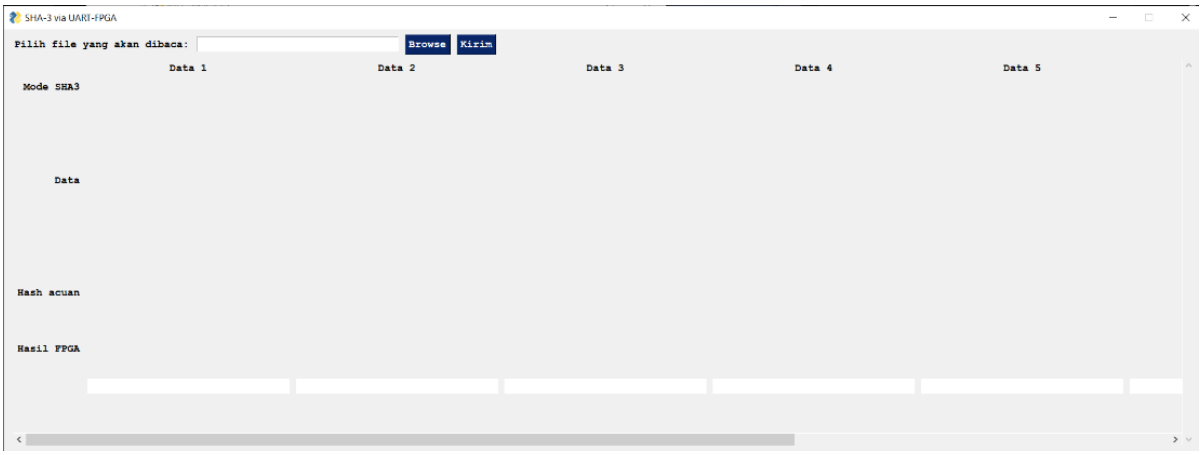

Figure 6. The GUI (Graphical User Interface) Design

3. RESULTS AND DISCUSSION

3.1. Testing the SHA-3 Permutation Module

The first step in testing the core of this research, the permutation function module, is a simulation using Xilinx Vivado software to ensure that the SHA-3 process and pipeline are functioning correctly and producing output following the reference data. We entered SHA-3 data for each clock eight times (eight data) to test. The SHA-3 mode determines the output data size. The module's outcomes are evaluated against the reference results to determine if it operates correctly. The input and reference data are sourced from the CSRC's sample SHA-3 data website [15].

3.2. Synthesis and Implementation Results for Top-Level Design

The top level is created through VHDL design, followed by synthesis and implementation without resource and area optimization. This process generates RTL schematics on an FPGA and reports detailing timing, resource, and power estimations. The resources employed by the top-level design are LookUp Table (LUT), accounting for 8.11% (5,140 out of 63,400), LUTRAM, accounting for 4.21% (800 out of 19,000), and Flip-flop (FF), accounting for 2%. 72% of the total 126,800 components were successful, with 16.67% (22.5 out of 135) of the BRAM, 59.17% (142 out of 240) of the DSP, 1.43% (3 out of 210) of the IO, and 9.38% (3 out of 32) of the BUFG not functioning correctly. The system requires 0.446 W power, as per the timing report. The system's Worst Negative Slack (WNS) value is 0.739 ns, the Worst Hold Slack (WHS) value is 0.026 ns, and the Worst Pulse Width Slack (WPWS) value is 4.020 ns.

https://doi.org/10.31436/iiumej.v26i1.3328

The system's maximum beat frequency can be determined by subtracting the designed clock period value from WNS [16], resulting in a maximum frequency of 107.979 MHz. To calculate throughput, the quantity of input SHA-3 data must be multiplied by the system's maximum frequency and then divided by the total cycles required to complete the SHA-3 hash process [17]. The input data quantity is calculated based on r bitrate, as it is the fundamental input of SHA-3. The bit rate for each SHA-3 mode differs, with SHA3-224 having a bit rate of 1,152 bits, SHA3-256 with 1,088 bits, SHA3-384 with 832 bits, and SHA3-512 with 576 bits. In this system, performing SHA-3 operations requires 192 cycles with eight input data. The throughput value for each SHA-3 mode in this system is 5.183 Gbps for SHA3-224, 4.895 Gbps for SHA3-256, 3.743 Gbps for SHA3-384, and 2.591 Gbps for SHA3-512.

3.3. Controller module synthesis and implementation results

After the synthesis and implementation process is carried out at the top-level design, the controller module as a system work control module produces a circuit that uses units such as FDRE and LUT. FDRE is a D Flip-flop with a clock enabled [18, 19].

3.4. Results of synthesis and implementation of the Datapath module

After synthesizing and implementing the top-level design, the data path module as a data processing module produces a circuit that uses units such as Output_Buffer, DEMUX, FunctionPermutation_SHA3, Input_Buffer, and Data_Storage. In this circuit, two LUTs are connected to the input of the Output_Buffer and Input_Buffer modules.

3.5. Synthesis and implementation results of the permutation function module

The SHA-3 operation module includes the SHA-3 permutation function stages modules Module_Chi, Module_Theta, Module_Iota, and Module RoundConstant. Module_RhoPi is not included in this circuit because the module is only a data transfer process, so the circuit implementation must change the data path between Module_Theta and Module_Chi without requiring any resources. This module also has a clock counter circuit and uses blocks such as CARRY4, LUT, and FDCE. CARRY4 is usually used to create an enumerator circuit [20].

The implementation of the DSP48 module, which is the core of this research, consists of DSPCascade4, DSPCascade2, and DSP48 modules. The DSPCascade4 module consists of four DSP48s, as shown in Figure 7; the DSPCascade2 module consists of two DSP48s, as shown in Figure 8, and the DSP48 Module consists of one DSP48, as shown in Figure 9. These modules are then arranged to be able to perform SHA-3 operations. The details of DSP48 usage in the SHA-3 permutation function stage modules are shown in Table 2.

Table 2. Details of DSP48 usage

Step	Total modules used	DSP48 content in one module	DSP48 Total
Step 1	7 * DSPCascade4	4	28
Step 2	7 * ModulDSP48	1	7
Step 3	35 * ModulDSP48	1	35
Chi	35 * DSPCascade2	2	70
		Total	142

https://doi.org/10.31436/iiumej.v26i1.3328

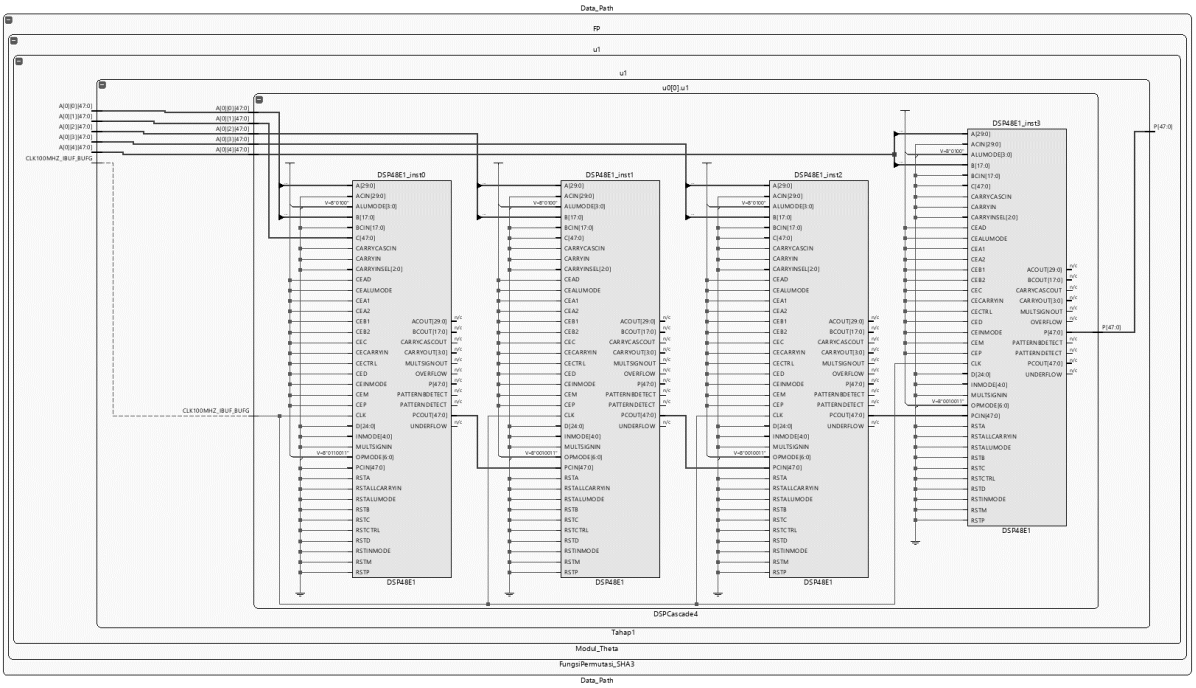

Figure 7. The DSPCascade4 implementation

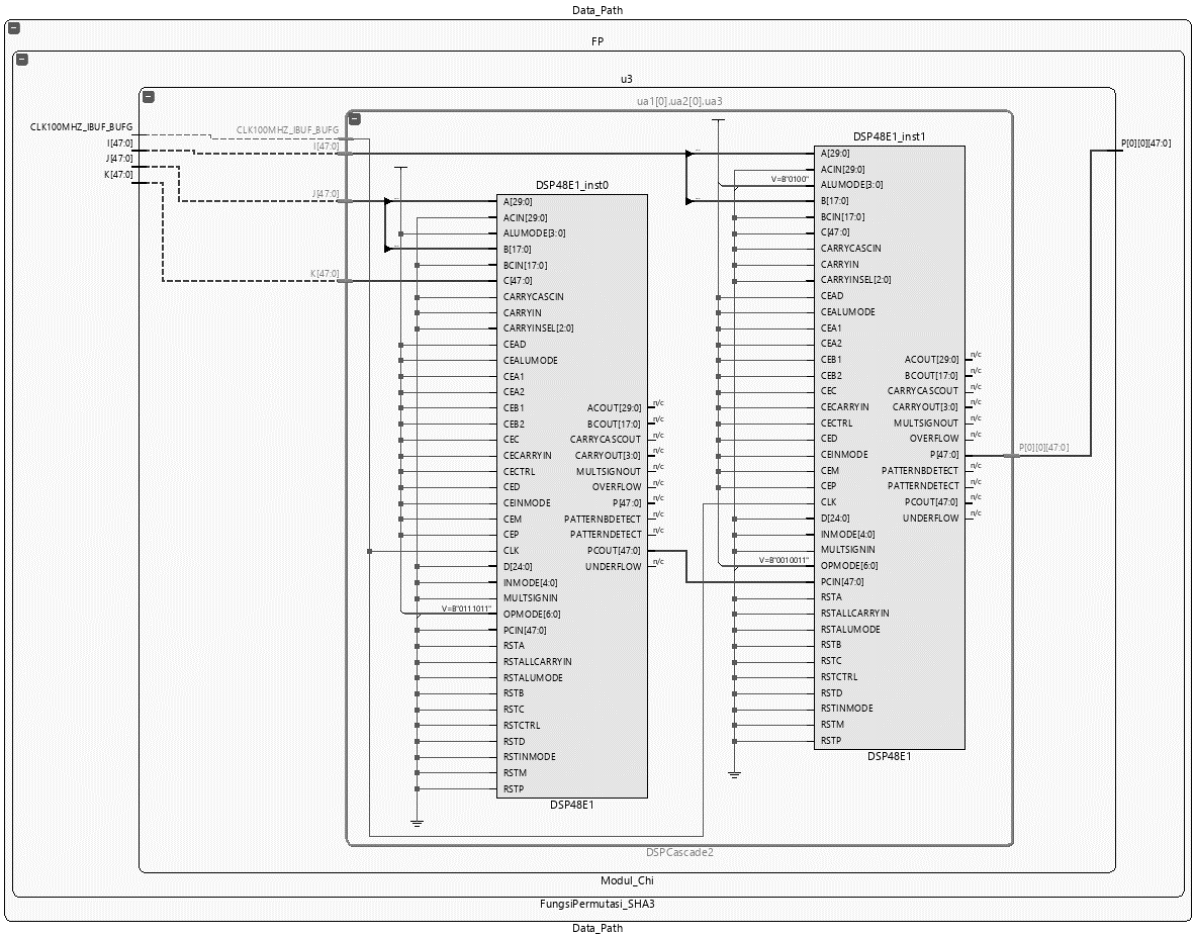

Figure 8. The DSPCascade2 implementation

https://doi.org/10.31436/iiumej.v26i1.3328

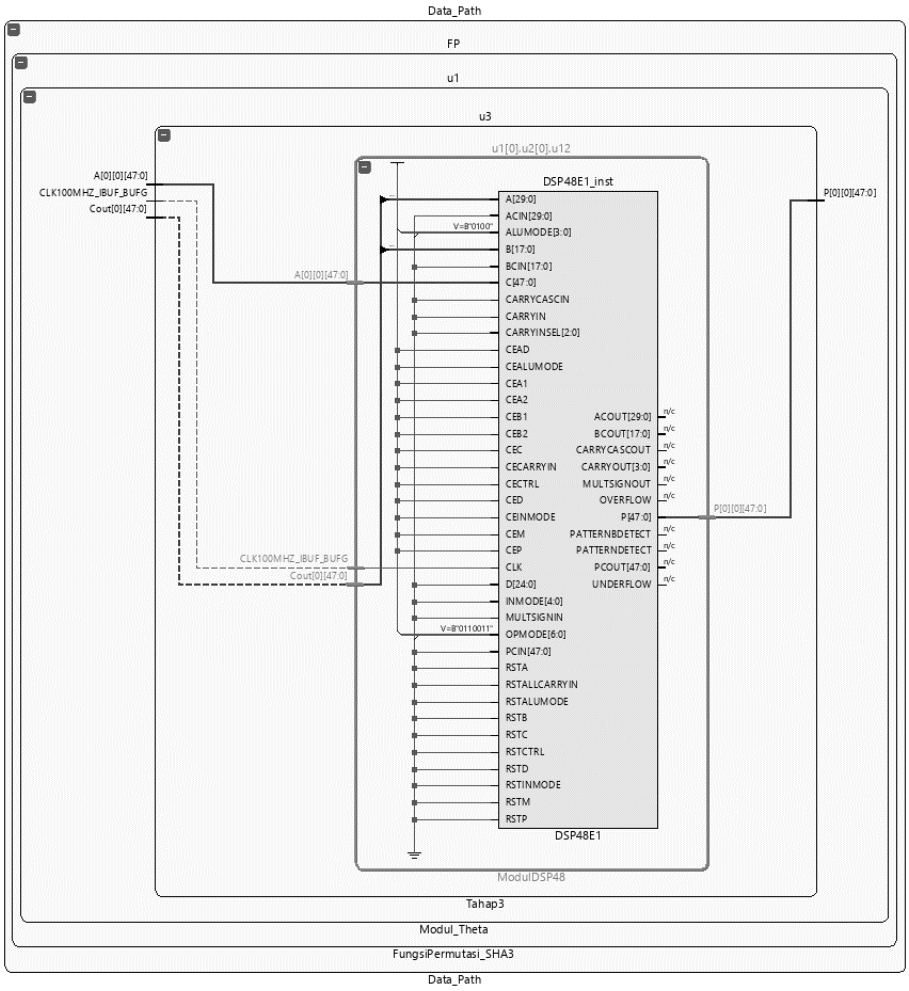

Figure 9. The implementation of the DSP48 Module.

3.6. Testing implementation results on FPGA

FPGA implementation testing occurs after all modules have successfully functioned. The GUI program was designed to enable easy monitoring of input data and output results generated by the FPGA, allowing for comparison with previously established reference results, especially in SHA-3 256, as shown in Table 3.

Table 3. Comparison Throughput between previous research

References	**FPGA**	**Throughput (Gbps)**
F. Kahri et al.[21]	Virtex 2	0,73
	Virtex 4	1,06
	Virtex 5	2,30
	Virtex 6	3,81
	Virtex 7	4,2
B. Jungk [22]	Virtex 5	0,86
	Virtex 6	1,07
References	**FPGA**	**Throughput (Gbps)**
I. San and N. At [23]	Virtex 5	0,501
J.-P. Kaps et. al. [24]	Virtex 5	0,075
	Virtex 6	0,136
This Research	Artix-7-100T	4,895

https://doi.org/10.31436/iiumej.v26i1.3328

Based on Table 3, this research demonstrates a significant improvement in throughput compared to previous studies using Virtex FPGAs, achieving 4.895 Gbps with the Artix-7-100T, a low-cost and moderate-performance FPGA. Notably, this throughput surpasses even the highest reported in Kahri's study with Virtex 7 (4.2 Gbps). It significantly outperforms Jungk's Virtex 6 (1.07 Gbps) and San and J.P Kaps' results by an even more substantial margin. Despite being a cost-effective FPGA, the Artix-7-100T performs better than the higher-cost and high-performance Virtex devices, showcasing the efficiency of this research's design and the potential to deliver high throughput in budget-sensitive applications.

(a)

(b)

Figure 10. FPGA implementation test results with data 1 - 5 (a) and 6 - 8 (b)

Test data is uploaded into the program using an eight-piece file containing three elements: SHA-3 mode, input data, and reference hash. The data that has been successfully read is transmitted to the FPGA by pressing the 'Send' button. Following the output data transmission by the FPGA through UART, as demonstrated in Figure 10, the outcomes will be promptly shown. Testing was conducted twice with eight different test data for each test, and the results verified the accurate execution of this system.

4. CONCLUSION

The SHA-3 algorithm was successfully implemented on the Artix-7-100T FPGA using the DSP48 unit and eight pipeline stages. Resources, including LUTs (equivalent to 8.11% of 63,400), LUTRAM (equivalent to 4.21% of 19,000), FF (equal to 2.72% of 126,800), BRAM (equivalent to 16.67% of 135), DSP48 (equivalent to 59.17% of 240), IO (equivalent to 1.43% of 210), and BUFG (equivalent to 9.38% of 32) were utilized for the implementation. The system's overall power requirement is 0.446 W, while the maximum frequency that can be applied to the system is 107.979 MHz. Additionally, the system produces a throughput of 5.183 Gbps for SHA3-224, 4.895 Gbps for SHA3-256, 3.743 Gbps for SHA3-384, and 2.591 Gbps for SHA3-512.

ACKNOWLEDGEMENT

This work was partially supported by the Department of Computer Science and Electronics, Universitas Gadjah Mada, under the 2024 publication funding year.

REFERENCES

[1] Paar, C. dan Pelzl, J., 2010, "SHA-3 and The Hash Function Keccak", Understanding Cryptography — A Textbook for Students and Practitioners, Available: http://www.crypto-textbook.com/.

[2] Honda, T., Guntur, H. dan Satoh, A., 2014, "FPGA implementation of new standard hash function Keccak," 2014 IEEE 3rd Global Conference on Consumer Electronics, GCCE 2014, Available: DOI:10.1109/GCCE.2014.7031105.

[3] Ahmad, I, and Das, A.S., 2005, Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs, Computers and Electrical Engineering 31 (2005) 345–360.

[4] Rao, M., Newe, T. dan Grout, I., 2014, "Efficient high-speed implementation of secure hash algorithm-3 on Virtex-5 FPGA", Proceedings - 2014 17th Euromicro Conference on Digital System Design, DSD 2014, Available: DOI:10.1109/DSD.2014.24.

[5] Sideris, A.; Dasygenis, M., 2023, Enhancing the Hardware Pipelining Optimization Technique of the SHA-3 via FPGA. Computation 2023, 11, 152. https://doi.org/10.3390/computation11080152.

[6] Augoestien, N.G. dan Putra, A.E., 2015, "Purwarupa Perangkat Keras untuk Eksekusi Algoritma AES Berbasis FPGA", IJEIS (Indonesian Journal of Electronics and Instrumentation Systems), [Online] Available: DOI:10.22146/ijeis.7644.

[7] M. Psarakis, A. Dounis, A. Almabrok, S. Stavrinidis, and G. Gkekas, "An FPGA-Based Accelerated Optimization Algorithm for Real-Time Applications," Journal of Signal Processing Systems, vol. 92, no. 10, pp. 1155-1176, Oct. 2020. doi: 10.1007/s11265-020-01522-5.

[8] Kundi, D. e. S. dan Aziz, A., 2016, "A low-power SHA-3 designs using embedded digital signal processing slice on FPGA", Computers and Electrical Engineering, Available: DOI:10.1016/j.compeleceng.2016.04.004.

[9] W. Wang and G. Sun, "A DSP48-Based Reconfigurable 2-D Convolver on FPGA," 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), Jishou, China, 2019, pp. 342-345. doi: 10.1109/ICVRIS.2019.00089.

[10] E. Walters, "Array Multipliers for High Throughput in Xilinx FPGAs with 6-Input LUTs," Computers, vol. 5, no. 4, p. 20, Sep. 2016. doi: 10.3390/computers5040020.

[11] I. M. Alexandru, A. Grama, L. Viman, and D. Pitica, "FFT Radix2 Core Implemented on FPGA with DSP48 Slices," 2018 IEEE 24th International Symposium for Design and Technology in Electronic Packaging (SIITME), Iasi, 2018, pp. 109-113. doi: 10.1109/SIITME.2018.8599234.
[12] A. Sideris, T. Sanida, and M. Dasygenis, "Hardware acceleration design of the SHA-3 for high throughput and low area on FPGA," Journal of Cryptographic Engineering, vol. 14, no. 2, pp. 193-205, Jun. 2024. doi: 10.1007/s13389-023-00334-0.
[13] Z. Han, J. Jiang, J. Xu, P. Zhang, X. Zhao, D. Wen, and Y. Dou, "A high-throughput scalable BNN accelerator with fully pipelined architecture," CCF Transactions on High Performance Computing, vol. 3, no. 1, pp. 17-30, Mar. 2021. doi: 10.1007/s42514-020-00059-0.
[14] Sundal, M. dan Chaves, R., 2017, "Efficient FPGA Implementation of the SHA-3 Hash Function", Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI, [Online] Available: DOI:10.1109/ISVLSI.2017.24.
[15] CENTER, CSR, 2021, "Cryptographic Standards and Guidelines," Available: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values,
[16] Xilinx, 2020, "UltraFast Design Methodology Guide for the Vivado Design Suite," Xilinx, Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2020_1/ug949-vivado-design-methodology.pdf.
[17] Athanasiou, G.S., Makkas, G.P. dan Theodoridis, G., 2014, "High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm", ISCCSP 2014 - 2014 6th International Symposium on Communications, Control and Signal Processing, Proceedings, Available: DOI:10.1109/ISCCSP.2014.6877931.
[18] A. E. Putra, O. Natan, and J. E. Istiyanto, "Enhancing CNN Deployment Efficiency on Mobile Devices with FPGA-based Accelerators: Memory-based Convolution and Resource Optimization," ICIC Express Letters Part B: Applications, vol. 15, no. 10, pp. 989-997, Oct. 2024.
[19] Xilinx Inc., 2020, "Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide", Xilinx,. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug953-vivado-7series-libraries.pdf.
[20] Z. Hakim, O. Natan, R. H. Sari, J. T. Amael, A. Dharmawan and J. E. Istiyanto, "6-DOF Arm Robot Control Using Open-Source FPGA," 2024 9th International Conference on Control and Robotics Engineering (ICCRE), Osaka, Japan, 2024, pp. 157-161
[21] F. Kahri, B. Bouallegue, M. Machhout, R. Tourki, "An FPGA Implementation and Comparison of the SHA-256 and Blake-256", 14th international conference on Sciences and Techniques of Automatic control, & computer engineering, Sousse, Tunisia 2013, pp 152-157.
[22] B. Jungk, "Evaluation Of Compact FPGA Implementations For All SHA-3 Finalists", Third SHA-3 Candidate Conference, March, 2012.
[23] I. San, N. At, "Compact Keccak Hardware Architecture for Data Integrity and Authentication on FPGAs", Information Security Journal: A Global Perspective, Taylor & Francis, 21:5, 231-242, 2012.
[24] J.-P. Kaps, P. Yalla, K.K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, "Lightweight Implementations of SHA-3 Finalists on FPGAs", SHA-3 Conference, March 2012.