AES-256 Accelerator for Cadence Processor

Project Number: **22-1-1-2495**

Project Report

Student: Tzvi Steinberg ID: XXXXXXXXX

Student: Shahar Levi ID: XXXXXXXXX

Supervisor: Oren Ganon

Project Carried Out at: University

Contents

List of equations

Optimized design metric: $\frac{clock\ cycles}{area\ utilization}$

Abstract

This paper presents a performance evaluation of a 256-bit version Advanced Encryption Standard encryption and decryption core running on the Xilinx MicroBlaze 32-bit soft-core processor implemented on a Xilinx Kintex-7 KC705 FPGA development board. The AES core, meanwhile, was programmed in Verilog and connected to the processor over the AXI-4 Lite bus architecture. Performance metrics used for the performance evaluation of system include encryption and decryption execution time, resource usage, and power usage. The overarching goal of this project is to demonstrate the optimal division of the AES-256 algorithm between software and hardware, such that performance and resource utilization are optimized in relation to each other. Best results were achieved for designs in which hardware execution of the algorithm was maximized, hardware use was minimized, and transfers between hardware and processor was minimized.

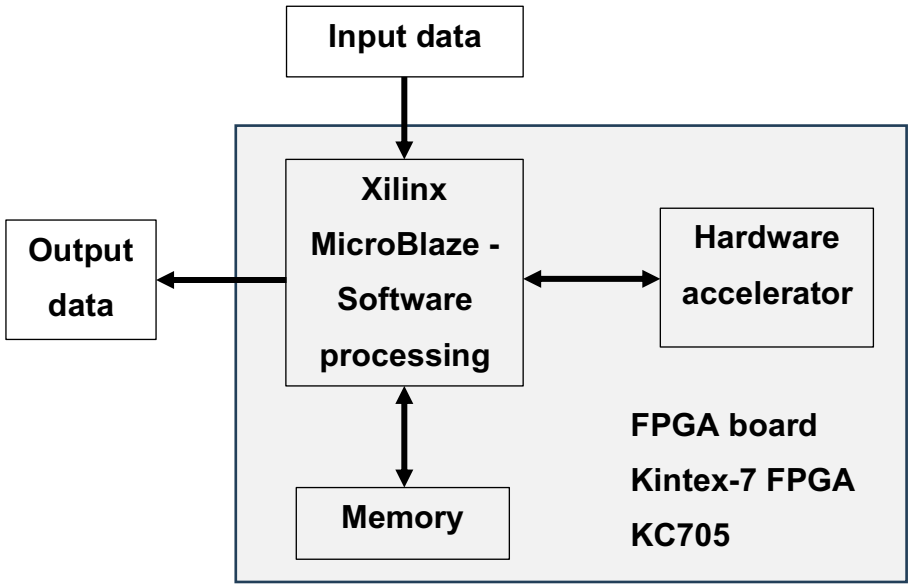

Figure 1: Abstract system diagram

1 Introduction

This project is meant to demonstrate different implementations of the symmetric encryption algorithm AES-256. In each implementation, we applied hardware acceleration in conjunction to the software to varying degrees. Thus, our final goal is to find the implementation which results in the smallest ratio between the performance to area utilization. i.e., minimizing $\frac{clock\ cycles}{area\ utilization}$.

The AES encryption standard has become an industry standard for encrypting information and is used prolifically and frequently for public, private, commercial, and non-commercial applications. Therefore, optimizing its performance speed and area requirements are of great benefit.

To solve this problem, we used the MicroBlaze soft processor on our FPGA board to run the software parts of a given design, as well as measure the clock cycles. When a given segment of the design is meant to run through hardware, the MicroBlaze transfers the data to the custom AES-256 hardware accelerator we created. As such, we wrote and packaged several AES-256 IP blocks and respective C scripts, and then analyzed and compared the results.

From our research, though similar existing projects were found, none were found with both a 256-bit design as well as support for encryption and decryption.

2 Theoretical background

AES is a symmetric encryption algorithm, meaning the encryption key is also used for decryption. AES-256 refers to the size of the 256 bit key, while the inputted and resulting data is 128 bits in length. The algorithm operates based on a substitution-permutation network (SPN) structure. The algorithm works on fixed-size blocks of plaintext and applies a series of transformations to encrypt or decrypt the data. The algorithm can be implemented in several modes, such as ECB, CBC, CTR. Our implementation utilizes the ECB mode as described below.

AES-256 consists of 14 rounds of transformation, with each round performing several operations on the data using a round-specific 128 bit key. Each round consists of four transformation steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey. While their inverses are used for decryption.

- In SubBytes each byte of the data is replaced with a corresponding byte from a substitution table called the S-box.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2: SubBytes

- ShiftRows cyclically shifts the bytes in each row of the data block. This operation provides diffusion and ensures that data in one row affects multiple columns during encryption.

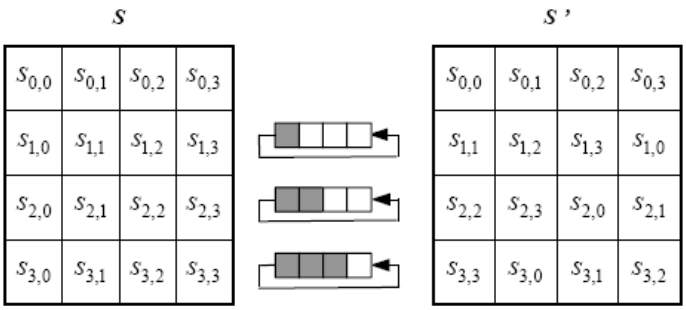

Figure 3: ShiftRows

- The MixColumns step operates on the columns of the data block, combining bytes using a matrix multiplication operation. This step provides further diffusion and improves the algorithm's ability to resist linear attacks.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 4: MixColumns first column

- For AddRoundKey, a bitwise exclusive OR (XOR) operation is performed between the data block and the round key. The round key is derived from the original encryption key using the key expansion algorithm.

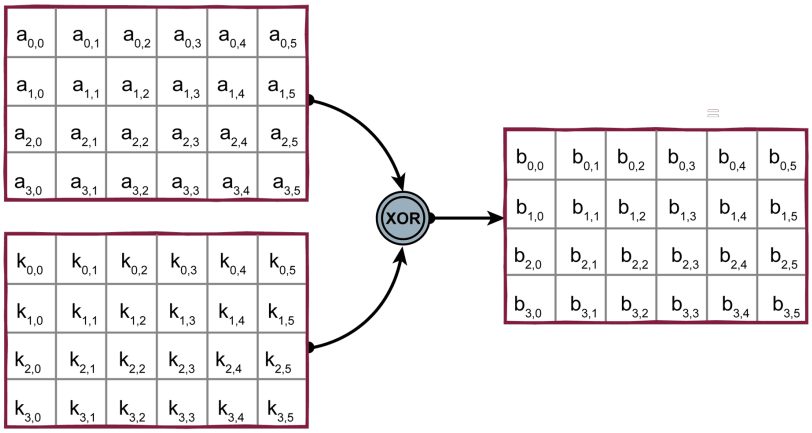

Figure 5: AddRoundKey

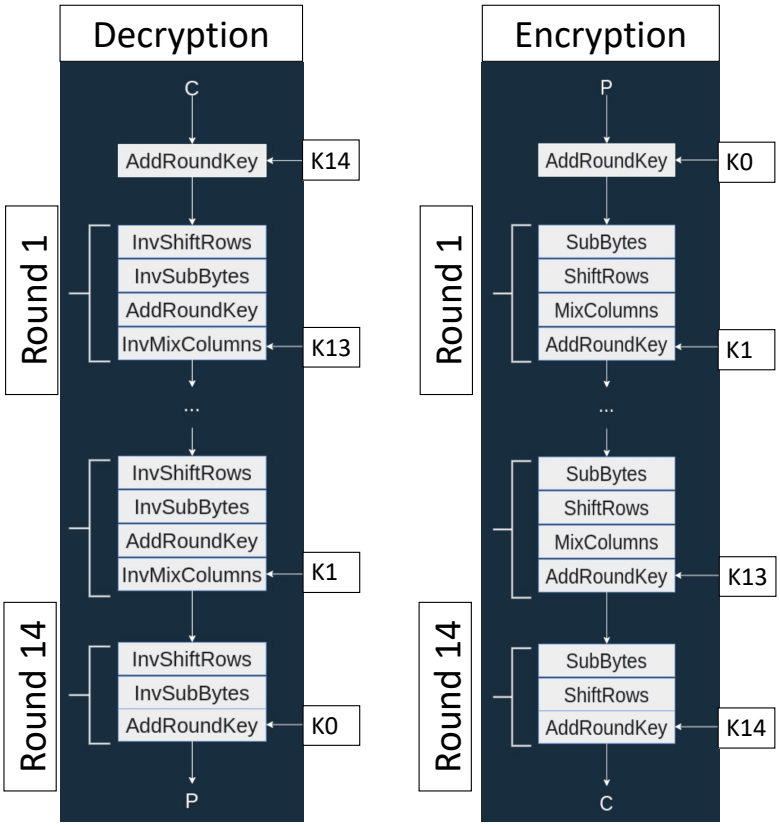

Figure 6: Encryption and Decryption flow

The MicroBlaze soft processor is a configurable and customizable 32-bit RISC (Reduced Instruction Set Computer) processor core designed by Xilinx. It is a soft processor, which means it can be implemented on Xilinx FPGAs and programmed to perform various tasks. It features a range of peripherals and interfaces that allow it to interact with external devices. These peripherals can include UART (Universal Asynchronous Receiver-Transmitter) for serial communication, Timer for clock cycle monitoring, AXI interface for interfacing with external AXI configured blocks, and custom IP blocks. The availability of these peripherals enables seamless integration of the MicroBlaze processor into larger embedded systems.

Alternative algorithm implementations are possible. However, emphasis should be placed on algorithms with intensive, iterative calculations. For instance, the cyclic redundancy check (CRC) algorithm, which includes many mathematical operations. [8]

3 Simulation

Simulations of the Verilog programmed hardware were done using the Vivado 2021.1 application. Simulations of the final platform were run from the Vitis SDK 2022.2 application.

The simulations of the first, second, and fourth implementations show the signals in the AES core for a scenario comprised of four stages. In the first stage, the plaintext:

0x3243f6a8885a308d313198a2e0370734

is encrypted with the key:

0x2b7e151628aed2a6abf7158809cf4f3c762e7160f38b4da56a784d9045190cfe

The resulting cypher-text:

0x1a6e6c2c662e7da6501ffb62bc9e93f3

is then used in the second stage with the same key, however this time in decryption mode. We see that the resulting deciphered text is the original plaintext as expected. The same process is shown in the third and fourth stages, with the difference being that decryption is done first to show correctness in the opposite direction as well.
The simulation of the third implementation shows part of an encryption round, to demonstrate correct computation between each round.

Verilog simulations:

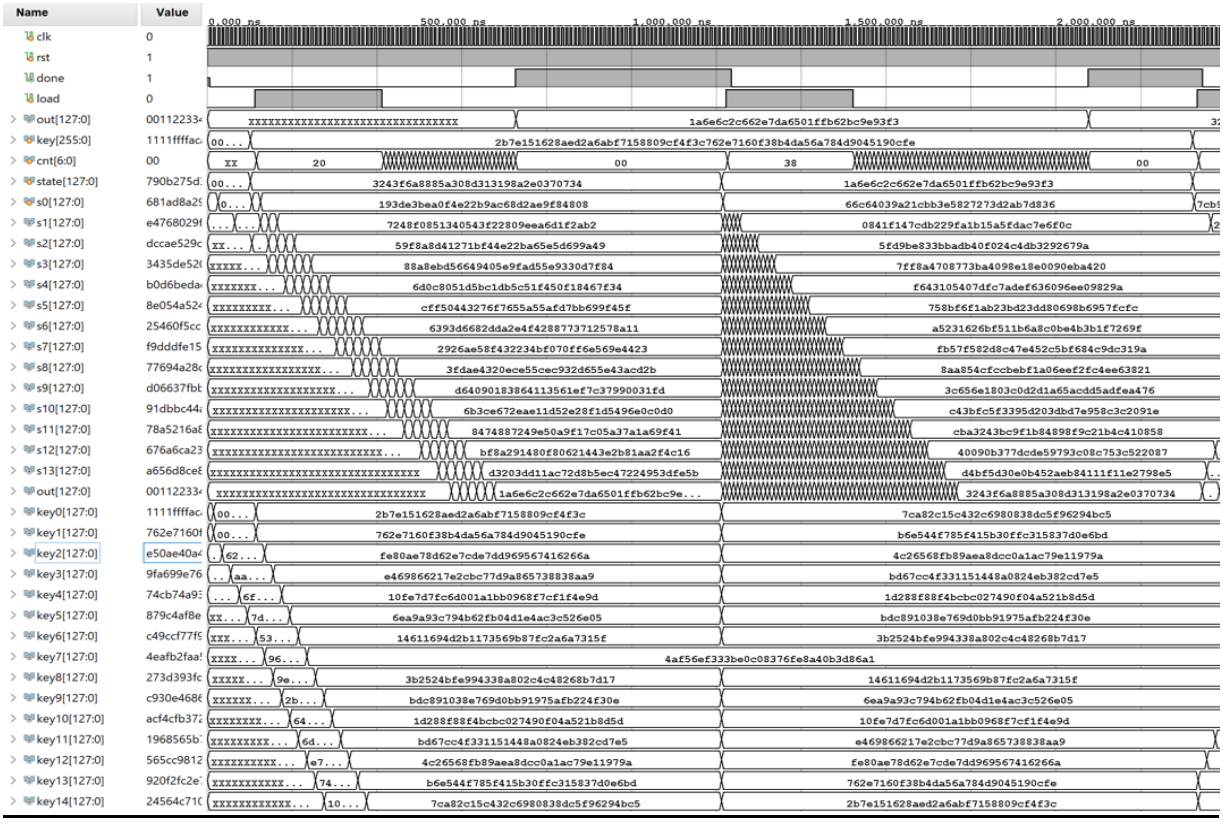

Figure 7: Simulation of first implementation (1/2)

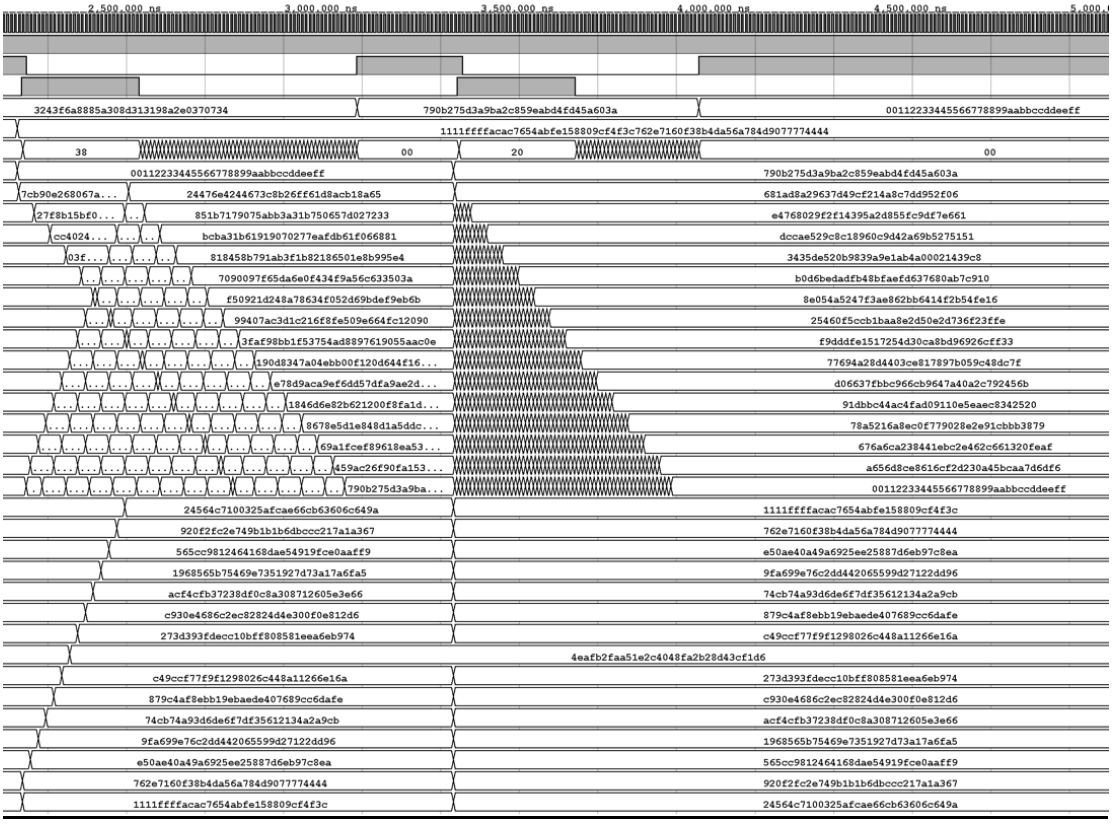

Figure 8: Simulation of first implementation (2/2)

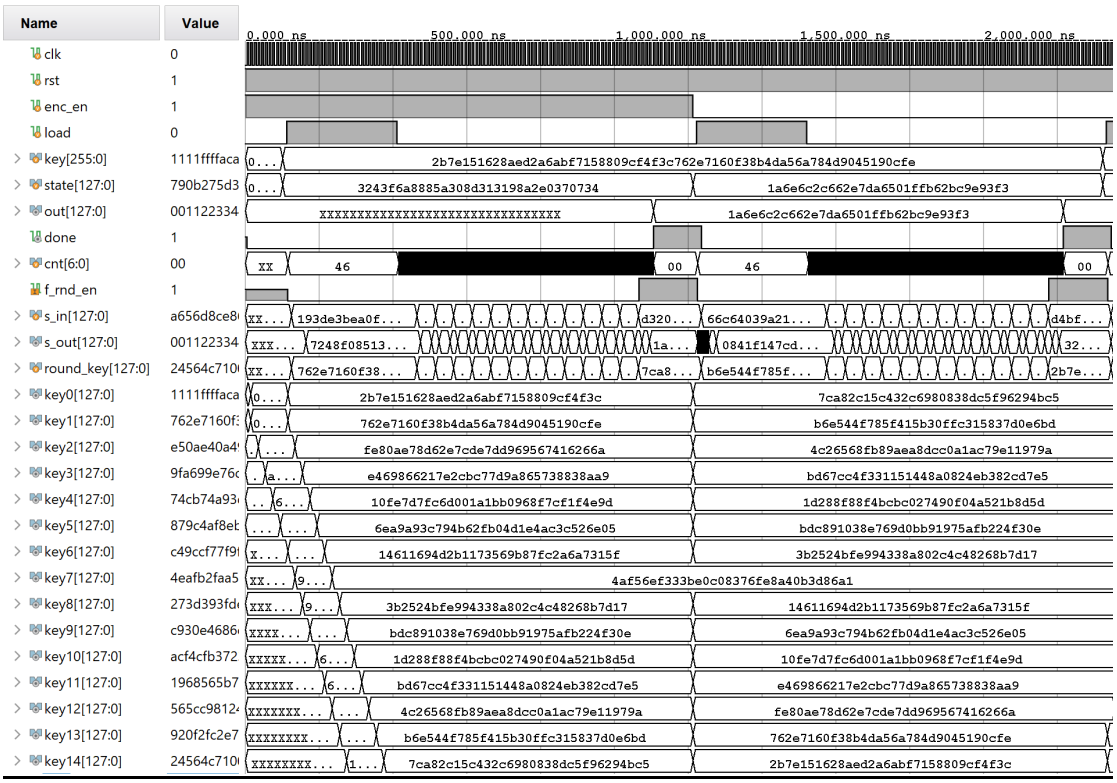

Figure 9: Simulation of second implementation (1/2)

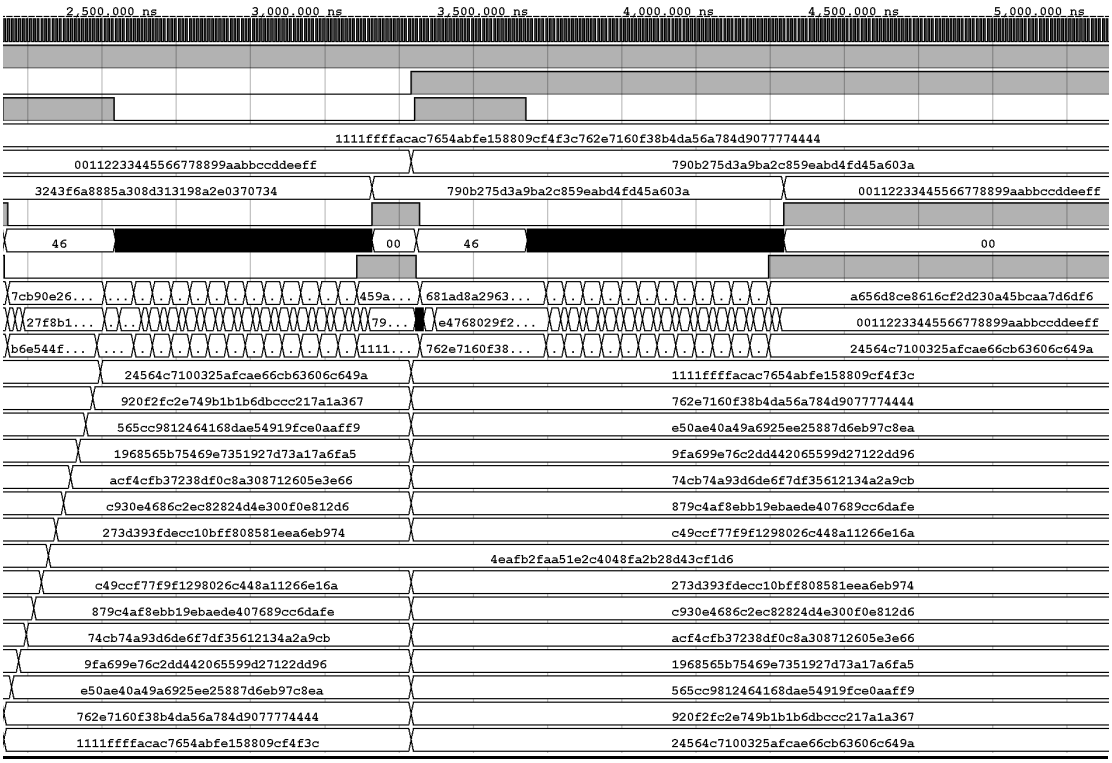

Figure 10: Simulation of second implementation (2/2)

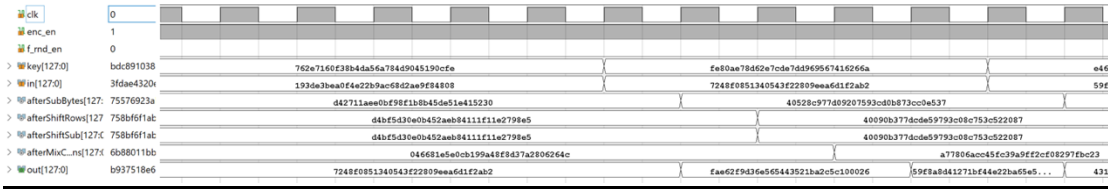

Figure 11: Simulation of third implementation (1/2)

Figure 12: Simulation of third implementation (2/2)

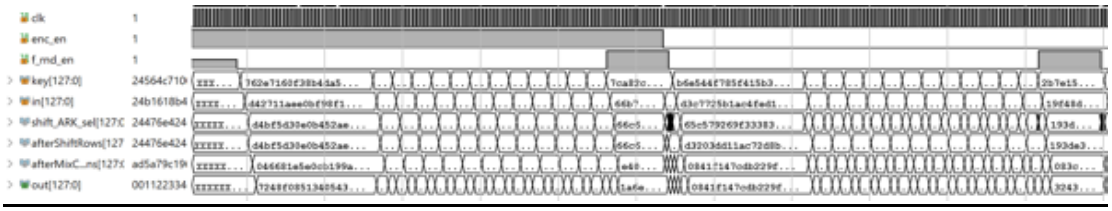

Figure 13: Simulation of fourth implementation (1/2)

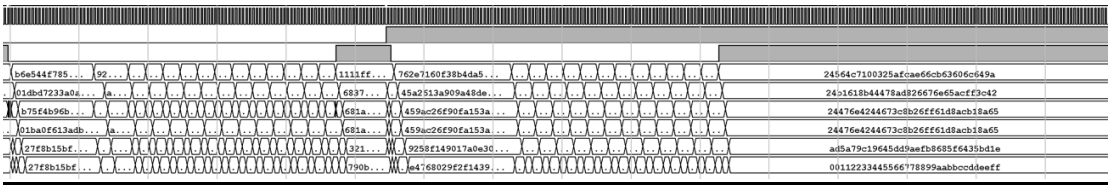

Figure 14: Simulation of fourth implementation (2/2)

4 Implementation

As can be seen in the block diagram of the system, the design is composed of two main parts: The Vivado designed hardware, and the Vitis run software. Considerations in design included: Which AES cores to implement, which AXI protocol to implement, what monitoring methods to use, and what overall system simulation to run.

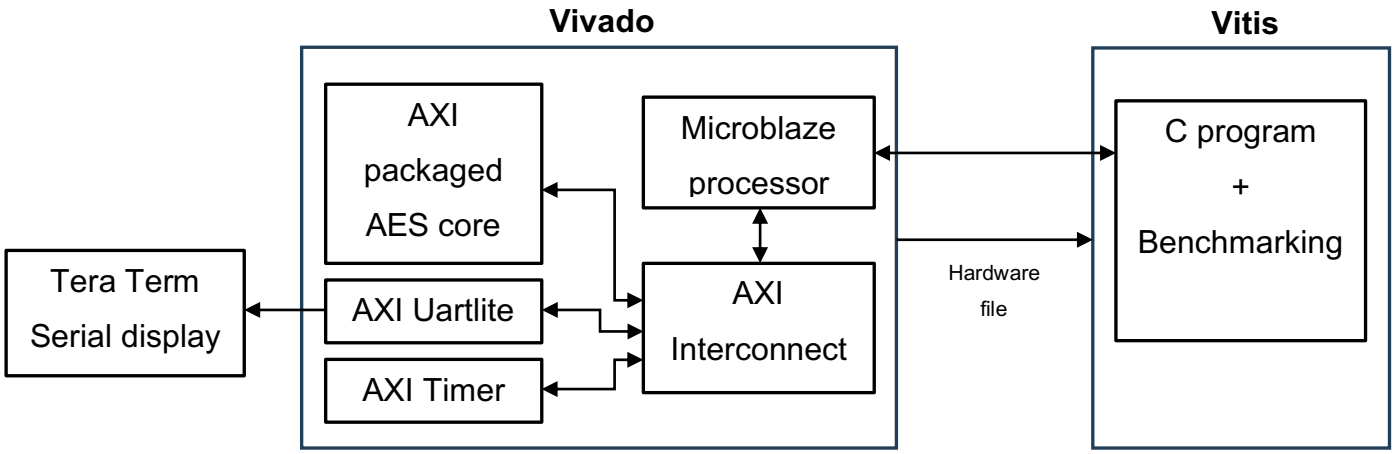

Figure 15: The System

4.1 Hardware Description

The hardware, designed using the Vivado application, consists of the Xilinix Kintex-7 KC705 FPGA development board on which several blocks were placed:

1. The MicroBlaze soft processor which runs the C software and is the master of the design's AXI bus interface.
2. The Clock Wizard supplies a $100MHz$ clock signal to the system using the FPGA's built-in clock.
3. The AXI interconnect interfaces between the MicroBlaze and its various AXI peripheries. The AXI-4 Lite was chosen over the AXI-4 Full due to reduced area usage as well increased simplicity.
4. The "EncDec" custom IP is the encryption/decryption block, packaged for the AXI interface.

5. The counter mode implemented in the AXI Timer core was used to determined total clock cycles and execution time of the system encryption and decryption processes.
6. The AXI Uartlite core was used to monitor the system.

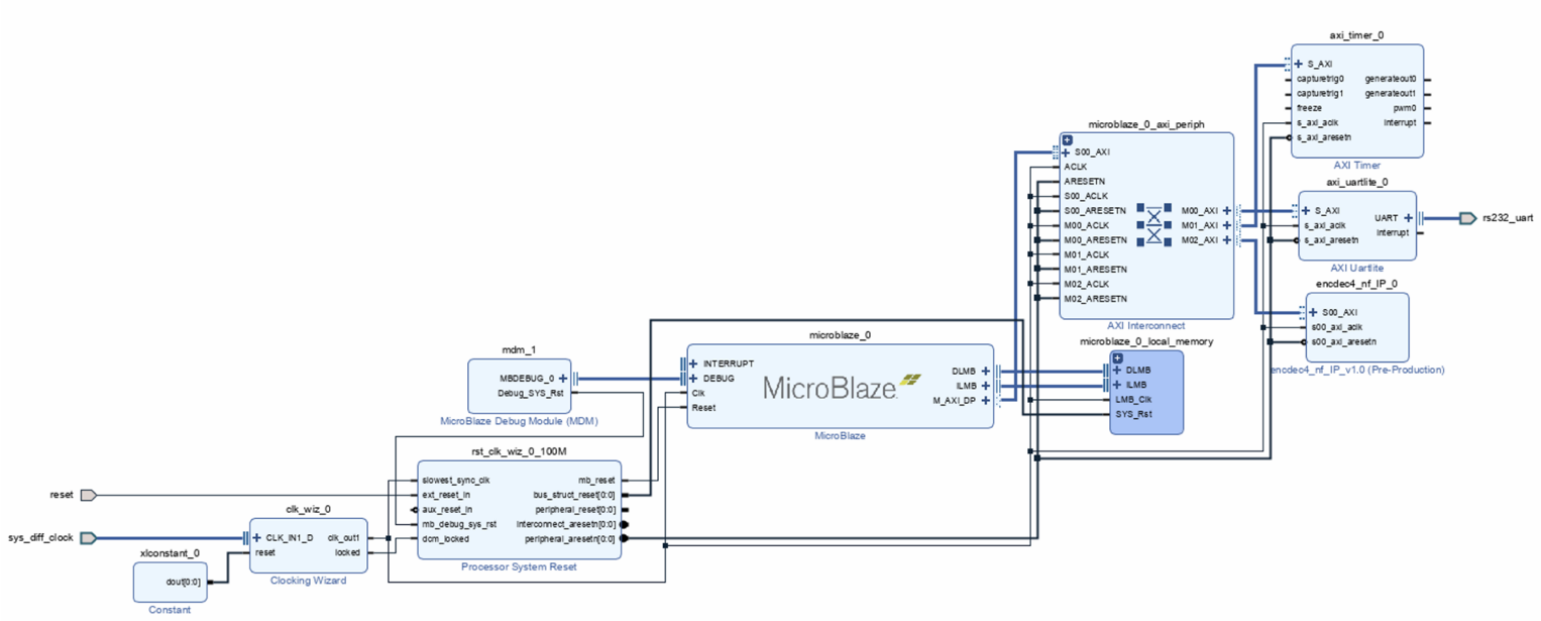

Figure 16: Hardware system

4.2 Software Description

The software is comprised of both Verilog and C programming languages.

The AES cores and AXI interface files were written using the hardware description language Verilog on the Vivado platform. Four implementations of the AES core were created:

1. The first core implements the entire AES algorithm in hardware. The module receives as input a 128-bit plain/cypher text and 256-bit key, as well as load, reset, and encryption/decryption control bits. The module returns a 128-bit output and a done bit. The core is comprised of several modules implementing key-expansion, an encryption/decryption round, each of the four transformations, and a final round. Each round is a separate module such that the data flows in one direction, and therefore, this version best allows for pipelining.

2. In the second implementation, the 14 rounds are achieved using one module which was modified to implement both the normal and final rounds using a control bit. This design takes advantage of no pipelining to greatly reduce area and power consumption.
3. The third implementation consists only of the Verilog implementing the previous hybrid round. With the key expansion no longer included. Thus, swapping a 128-bit round key for the 256-bit key and adding a final round control bit. This design thus reduces area use by moving the key expansion from the Verilog HDL to the C code.
4. The fourth is just as the third, excluding the SubBytes transformation module. Due to the previous design having an S-Box LUT in both the C and verilog code, this design aims to improve area utilization by implementing the SubBytes transformation in the C code. Therefore, the the S-Box LUT is eliminated from the Verilog designed hardware.

The AXI interface code defines the number of registers used for data transfer between the IP and the interconnect, their addresses on the MicroBlaze's side, as well as their connection to the IP IO ports. See appendix C for more.

C code is used for the program running on the MicroBlaze soft processor. The code is written and run using the Vitis application. In each version, an array of 32-bit incrementing numbers is created. This array is then encrypted or decrypted and displayed to demonstrate correctness before the opposite operation is applied with the resulting array being transformed back to its original state. Three separate programs were written:

1. The first one is designed for the first and second AES cores. For these versions, the C program sends a 128-bit text, 256-bit key, and relevant control signals. Thereafter, the output is read from the IP core.
2. The second version is designed for the third AES implementation. In this version, the 256-bit key is expanded to create 14 round keys which are then sent with the corresponding state to the IP, with the output being the

next state. This is repeated until the next state is the final output of the encryption/decryption process.

3. The third version is designed for the fourth implementation, and is similar to the previous version, save for the SubBytes transformation being done prior to sending to the core.

Special C libraries used include:

1. Xtmrctr.h – Used for AXI Timer configuration and control. Such as setting, resetting, starting and ending the timer count.
2. Xparameters.h – Maps the addresses on the Kintex board. Used to define the various IP's addresses for writing and reading.
3. Xil_io.h – Contains the interface for I/O components in the IPs. Functions we used include: Xil_Out32 for writing, Xil_In32 for reading, and Xil_printf for printing to the Uartlite.

5 Analysis of results

In the process of evaluating the different implementations, several benchmarking parameters were observed: Area utilization, power consumption, and cycle count.

For full table of results, see appendix A.

A 400·32-bit input was used. For details, see appendix B.

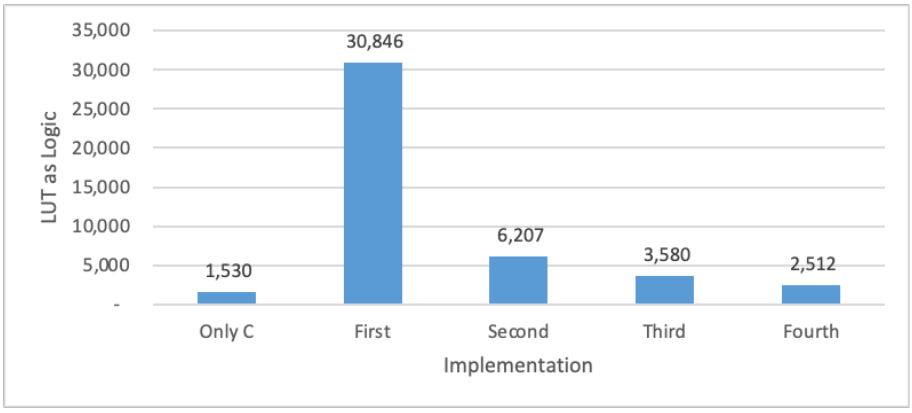

Figure 17: Area utilization comparison

1. Area utilization analysis: In figure 17, we see results matching our expectations. The first design occupied the most area, it being a full hardware design. A marked improvement was achieved with the second by using eliminating 13 of the *round* modules at the expense of pipelining compatibility. Further improvement is seen with the third and fourth designs with the removal of the *key expansion* module for both, and *SubBytes* module for the fourth. Finally, the purely software implementation saw the best results with only the base system of the MicroBlaze and monitoring environment.

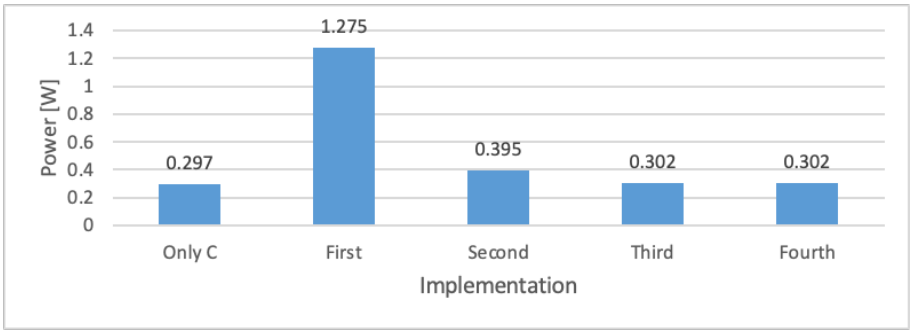

Figure 18: Power consumption comparison

2. Power consumption analysis: In figure 18, we see an expected trend of more hardware equating to more power usage. However, the third and fourth designs both had the same power usage. This is most likely due to their difference being mostly non-switching, and therefore not related to dynamic power. If there is a difference, it is negligible.

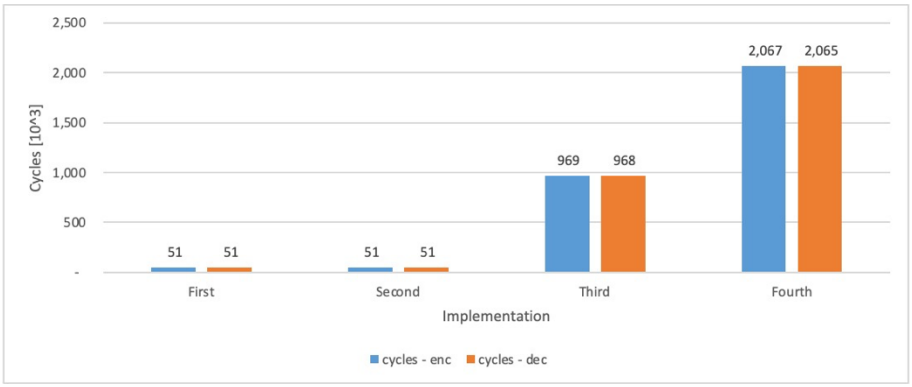

Figure 19: Clock cycle count comparison for non-C designs

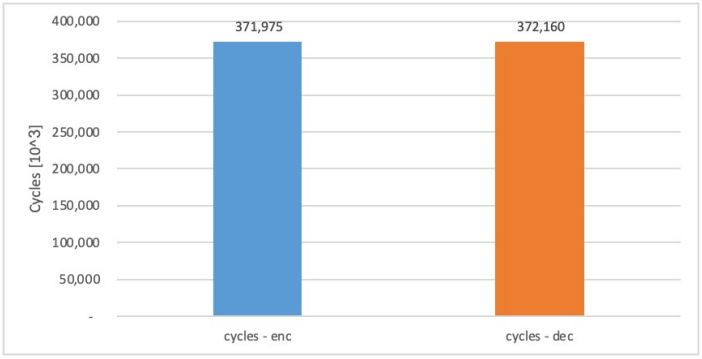

Figure 20: Clock cycle count for C design

3. Cycle count analysis: In figures 19 and 20, the clock cycle count for each design is seen for both encryption and decryption. The small differences between encryption and decryption are due to small differences in their respective algorithms. The first and second designs achieve the best results. This is due to maximizing hardware implementation of the design, while minimizing the use of the AXI-4 bus for data transfers. The third design saw a decrease in hardware use for the algorithm, while also increasing bus interface use. The fourth design further decreased hardware use for the algorithm.

   Overall, we see a clear trend of the performance improving as more of the algorithm is executed through hardware.

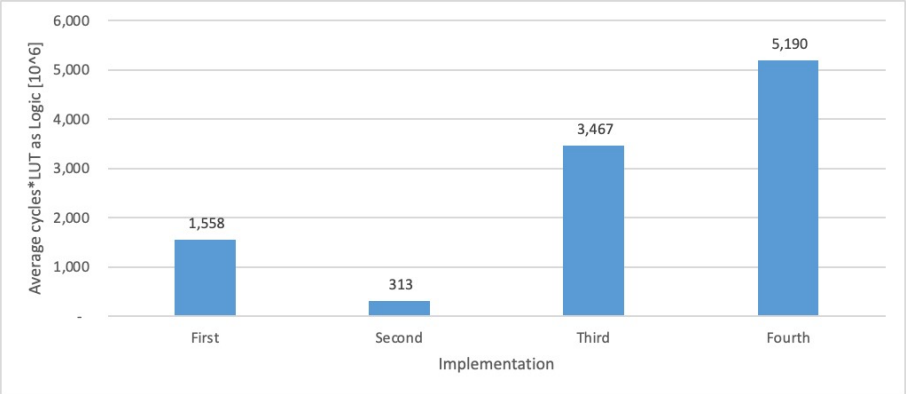

Figure 21: Optimization results - cycle count * area use

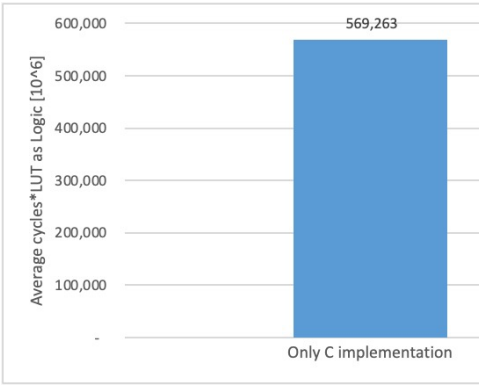

Figure 22: Optimization results in C design - cycle count * area use

4. Using the optimization equation defined in the introduction, there is a clear choice in design. We see that the second implementation was the best overall design, while the C implementation did drastically worse. While the third and fourth designs aimed for an improvement in area usage, they were not able to achieve this do to the high cycle count cost of increased data transfers. Finally, the first design did not take advantage of its loop unrolling to reduce cycle count, and therefore the increased area consumption was of no benefit.

   Lastly, the difference in power use between the second design and the third, fourth and C designs was small. Particularly in relation to its optimization score.

6 Conclusions and further work

In implementing our design, several goals were not met. TIE Extension functionality and pipelining were not implemented, and encryption and decryption in one clock cycle per 128-bit block was not achieved.

Due to the time required for each instruction done in software, in addition to the time required in transferring data to and from the IP cores, improvements in the AES core speed did not benefit the overall system. Future work should focus on several issues to remedy this situation:

1. TIE custom instructions: Upon analysis of the results, TIE was found to be crucial to the design. Customizing the AES instructions to a 128-bit architecture would increase speed in both software operations and sending/receiving data from the IP core.
2. AXI improvements: Using some more area in order to swap the AXI-Lite with the AXI-Full interface would increase our transfer bus width from 32-bits to 256-bits. This would greatly reduce the required number of data transfers. Furthermore, packaging the AES cores with the output set as a master port would allow for elimination of data reading instructions from the software.
3. Pipelining: Taking advantage of increased software execution speeds, pipeline could be implemented in the first design. This can be done either through synchronous reading and writing to the AES core, or by creating a FIFO stack in memory with which to transfer the data. From the analyzed results, achieving pipelining of five encryption/decryption processes at a time would cause the first design to surpass the second in ranking.
4. Key expansion storage, for the ECB AES algorithm, adjustments can easily be made to check whether the key received is identical to the previous one. In such a case, key expansion can be skipped, reducing encryption and decryption time.

7 Project Documentation

Our documentation is saved on Github [7]. There, all annotated Verilog and C files are stored and categorized by implementation.
Explanatory notes and simulation results are uploaded there as well.

8 References

Papers:

[1] Umer Farooq, "Comparative Analysis of Different AES Implementation Techniques for Efficient Resource Usage and Better Performance of an FPGA", Journal of King Saud University - Computer and Information Sciences 29, March 2016

User's Guide:

[2] "Vivado Design Suite AXI Reference Guide", UG1037 (v4.0) July 15, 2017

[3] "KC705 Evaluation Board for the Kintex-7 FPGA User Guide", UG810 (v1.9) February 4, 2019

Links:

[4] Johnny Wilson, "Microblaze-and-AES-Encryption-Core-Performance-Benchmarking", https://github.com/festrada68/Microblaze-and-AES-Encryption-Core-Performance-Benchamarking

[5] Online AES encryption/decryption tool, https://the-x.cn/en-us/cryptography/Aes.aspx

[6] Xilinx community support forum, https://support.xilinx.com/s/topiccatalog?language=en_US

[7] Project documentation, Github, https://github.com/tzvins/AES_256_hardware_acceleration.git

[8] Accelerating CRC algorithm, Accelerating algorithms in hardware, https://www.embedded.com/accelerating-algorithms-in-hardware/

Appendix A. **Comprehensive results used for analysis**

	LUT as Logic	power w	length	cycles - enc	cycles - dec
only c	1530	0.297	100	92,993,825	93,039,875
			200	185,987,650	186,079,750
			300	278,981,475	279,119,625
			400	371,975,300	372,159,500
			600	557,962,950	558,239,250
First	30846	1.275	100	12,925	12,925
			200	25,450	25,450
			300	37,975	37,975
			400	50,500	50,500
			600	75,550	75,550
			656	82,564	82,564
Second	6207	0.395	100	12,925	12,925
			200	25,450	25,450
			300	37,975	37,975
			400	50,500	50,500
			600	75,550	75,550
			656	82,564	82,564
			660	83,065	83,065
Third	3580	0.302	100	248,665	248,315
			200	488,815	488,115
			300	728,965	727,915
			400	969,115	967,715
			600	1,449,415	1,447,315
Fourth	2512	0.302	100	523,065	522,715
			200	1,037,615	1,036,915
			300	1,552,165	1,551,115
			400	2,066,715	2,065,315
			600	3,095,815	3,093,715
			604	3,116,397	3,114,283

Table 1: Results

Implementations' utilization for different implements:

- Microblaze only and C implement for the algorithm:

Name	Slice LUTs (203800)	Slice Registers (407600)	F7 Muxes (101900)	Slice (50950)	LUT as Logic (203800)	LUT as Memory (64000)	Block RAM Tile (445)	Bonded IOB (500)	IBUFDS (480)	BUFGCTRL (32)	MMCME2_ADV (10)	BSCANE2 (4)
N mb_wrapper	1668	1545	109	651	1530	138	32	5	1	4	1	1
mb_i (mb)	1668	1545	109	651	1530	138	32	0	1	4	1	1

- First Implementation:

Name	Slice LUTs (203800)	Slice Registers (407600)	F7 Muxes (101900)	F8 Muxes (50950)	Slice (50950)	LUT as Logic (203800)	LUT as Memory (64000)	Block RAM Tile (445)	Bonded IOB (500)	IBUFDS (480)	BUFGCTRL (32)	MMCME2_ADV (10)	BSCANE2 (4)
N mb_wrapper	31560	14764	8173	416	9054	30846	714	32	5	1	5	1	1
mb_i (mb)	31560	14764	8173	416	9054	30846	714	32	0	1	5	1	1

- Second Implementation:

Name	Slice LUTs (203800)	Slice Registers (407600)	F7 Muxes (101900)	Slice (50950)	LUT as Logic (203800)	LUT as Memory (64000)	Block RAM Tile (445)	Bonded IOB (500)	IBUFDS (480)	BUFGCTRL (32)	MMCME2_ADV (10)	BSCANE2 (4)
N mb_wrapper	6921	7571	685	2452	6207	714	45	5	1	5	1	1
mb_i (mb)	6921	7571	685	2452	6207	714	45	0	1	5	1	1

- Third Implementation:

Name	Slice LUTs (203800)	Slice Registers (407600)	F7 Muxes (101900)	Slice (50950)	LUT as Logic (203800)	LUT as Memory (64000)	Block RAM Tile (445)	Bonded IOB (500)	IBUFDS (480)	BUFGCTRL (32)	MMCME2_ADV (10)	BSCANE2 (4)
N mb_wrapper	3718	2406	653	1193	3580	138	32	5	1	4	1	1
mb_i (mb)	3718	2406	653	1193	3580	138	32	0	1	4	1	1

- Fourth Implementation:

Name	Slice LUTs (203800)	Slice Registers (407600)	F7 Muxes (101900)	Slice (50950)	LUT as Logic (203800)	LUT as Memory (64000)	Block RAM Tile (445)	Bonded IOB (500)	IBUFDS (480)	BUFGCTRL (32)	MMCME2_ADV (10)	BSCANE2 (4)
N mb_wrapper	2650	2280	146	971	2512	138	32	5	1	4	1	1
mb_i (mb)	2650	2280	146	971	2512	138	32	0	1	4	1	1

Appendix B. Demonstrating the linear trend of design performance

Due to constraints in available memory, the array length was limited. Therefore, performance analysis was performed on a few different sizes of data to show linear behavior. This is seen in the figures below:

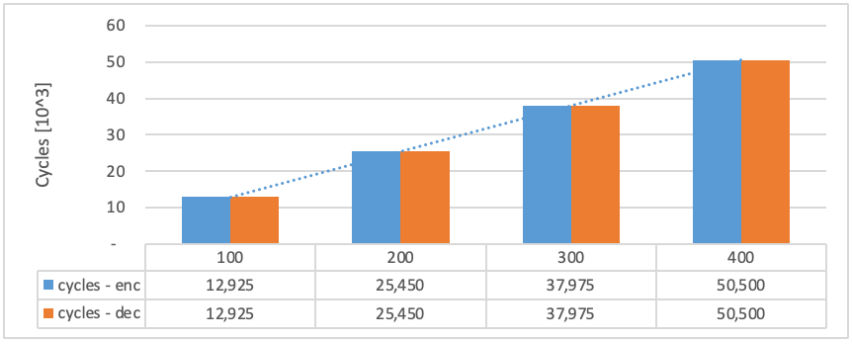

Figure 23: Clock cycle count as function of data size for first and second designs

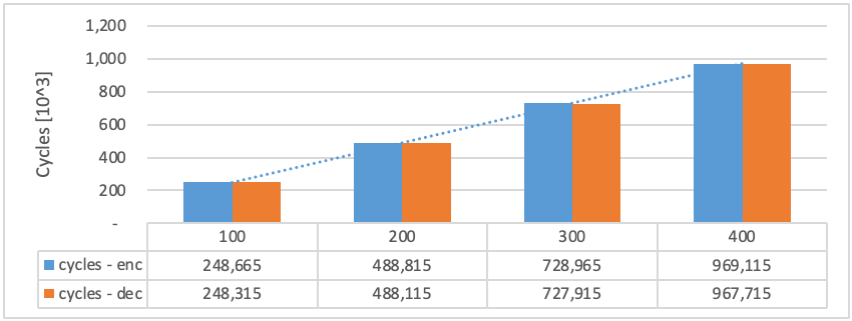

Figure 24: Clock cycle count as function of data size for third design

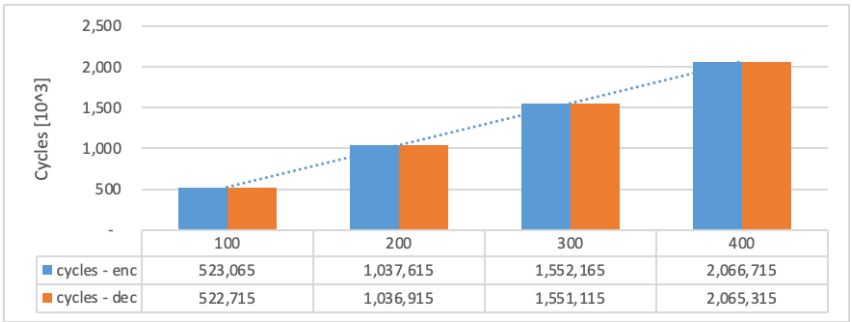

Figure 25: Clock cycle count as function of data size for fourth design

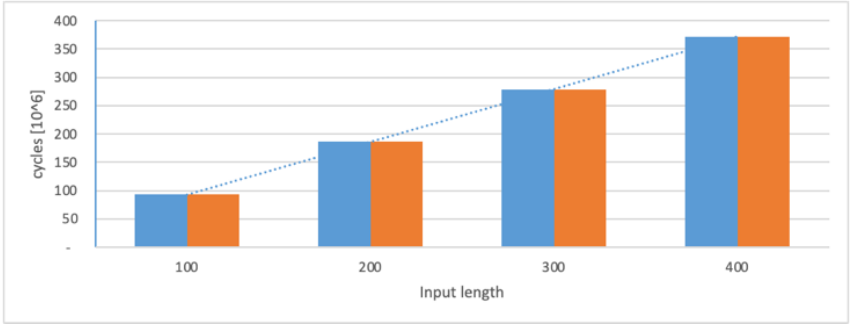

Figure 26: Clock cycle count as function of data size for C design

Appendix C. Core packaging and control signals

In table 2, the input/output of each AES core is further elaborated. Among them:

- Load bit – used to verify that the data starts getting process once all data has arrived.
- Done bit – can be used to check whether output is ready.
- Enc_en bit – specifies whether to encrypt or decrypt.
- Final round bit – specifies in third and fourth round whether round is the final round.

Registers in the AXI-4 Lite packaging are 32-bits each. Therefore, each design uses:

- 4 registers for the 128-bit input
- 4 registers for the 128-bit output
- 8 registers for the 256-bit key – design 1 & 2
- 4 registers for the 128-bit round key – design 3 & 4
- 1 register for input control signals
- 1 register for output control signals

Design	1	2	3	4
Input	128-bits	128-bits	128-bits state	128-bits
Output	128-bits	128-bits	128-bits	128-bits
Key/round key	256-bits	256-bits	128-bits round-key	128-bits round-key
Load	1-bit	1-bit	-	-
Reset	1-bit	1-bit	-	-
enc_en	1-bit	1-bit	1-bit	1-bit
Final round	-	-	1-bit	1-bit
Registers	18	18	13	13

Table 2: IO bits for each design