School of Electronic Engineering

IP Core Development of PRESENT-80 Lightweight Cipher

Project Portfolio

Alex Doherty
ID Number: 15508783

August 2020

MEng in Electronic and Computer Engineering

Supervised by Xiaojun Wang

IP Core Development of PRESENT-80 Lightweight Cipher

Project Portfolio

Acknowledgements

I would like to thank my project supervisor Xiaojun Wang and second assessor Eamonn Martin for their guidance and assistance in the development of this project. I would also like to thank Billy Roarty for his support in providing the necessary tools to complete this project, particularly during a difficult period of restrictions.

Table of Contents

ACKNOWLEDGEMENTS II

CONFERENCE-STYLE PAPER 7

APPENDIX A LITERATURE REVIEW 15

Preface **16**

APPENDIX B PROJECT PLAN 23

PROJECT DESIGN PLAN 24

Research Question **24**

Project Scope **24**

Design Approach **24**
Design & Implementation 24
Testing & Validation 25
Analysis 25

Confirmation **25**

Detailed Timeline **25**
Original Plan 26
Updated Plan 27

Success Criteria **28**

APPENDIX C RESEARCH LOG 30

Preface **31**

APPENDIX D PROJECT DESIGN & IMPLEMENTATION 44

TOOLS & DEVELOPMENT FLOW 45

Tools **45**
Xilinx Vivado 45
Xilinx SDK 45
Zybo Board 46

Development Flow **47**

AXI Peripheral **49**

Block RAM **49**

DESIGN & IMPLEMENTATION 51

Preface **51**

Introduction **51**

Hardware Development **51**
Introduction 51
Round Based Implementation 51
Pipelined Implementation 53

AXI Peripheral **56**
Finite State Machine 56
AXI Peripheral Creation 57
Edit & Package IP 58
Logical Interface/IP Physical port mapping 60
IP Integration 61

Software Development **63**
Software Driver Application 63
Software Implementation 63
Tester Application 64

Conclusion **64**

VALIDATION 65

Introduction **65**

Test Vectors & Test Benches **65**

Debugging **67**

Conclusion **68**

APPENDIX E TESTING & RESULTS 69

TESTING & RESULTS 70

Introduction **70**

Metrics **70**
Timing 70
Procedure 70
Results 72
Analysis 72
Area 73
Procedure 73
Results 74
Analysis 78
Power 79
Procedure 79
Results 80

Analysis 80

Latency **81**
Procedure 81
Results 81
Analysis 81

Throughput **81**
Procedure 81
Results 82
Analysis 82

Conclusion **82**
Appendices References 83

APPENDIX F SOURCE CODE LISTING 85

SOURCE CODE 86

Conference-Style Paper

IP Core Development of PRESENT-80 Lightweight Cipher

Alex Doherty

Dublin City University

***Abstract*— The PRESENT family of lightweight block ciphers were designed to provide a solution for the use of block ciphers in constrained environments, such as RFID tags and sensors. The area of IoT is an industry which will benefit greatly from lightweight cryptography, allowing for secure communication and storage of data. The PRESENT cipher allows for implementations using either an 80 or 128-bit key. This project focuses on the 80-bit key implementation of the cipher. Two designs of the cipher will be designed in VHDL, one with a focus on area constraints and the other on performance. An implementation is integrated as an AXI peripheral to achieve hardware acceleration of the cipher.**

Keywords—PRESENT, cipher, cryptography, Vivado, Zybo, Zynq, AXI, VHDL, low-resource

I. INTRODUCTION

The Internet of Things (IoT) has led to a rise in the number of devices with internet connectivity and transmitting information between each other. This leads to the need for secure communications for low-resource devices which is achieved through the use of lightweight encryption algorithms. Ciphers such as DES [1], and more recently AES [2] have been used for security, however, these ciphers do not meet the requirements of many low-resource modern devices. As a result, lightweight cryptography has emerged, with ciphers that are suitable for these devices.

PRESENT is an example of a lightweight cipher that has been designed with area and power constraints in mind. It is a hardware optimised block cipher. PRESENT was designed to improve on area utilisation from ciphers such as AES and DES. It supports both 80-bit and 128-bit keys.

This paper presents two designs of the cipher that have been implemented in VHDL and target a Zybo Zynq-7000 system on a chip (SoC) for hardware acceleration. This is achieved through the creation of an AXI peripheral where the programmable logic (PL) and processing system (PS) can communicate. The design can then be invoked by a software driver for hardware acceleration of the encryption. The Zynq SoC contains a dual-core ARM Cortex-A9 processor along with a Xilinx 7-series field programmable gate array (FPGA) logic.

II. EXISTING LIGHTWEIGHT BLOCK CIPHER IMPLEMENTATIONS

A. Survey of Lightweight Block Ciphers

Poschmann [3] presents a lightweight hardware implementation architecture for the DES cipher as well as providing results for the implementation, along with DES variants. DES was chosen as it was designed with a focus on hardware efficiency and is a very well investigated algorithm. PRESENT was designed as a result of this investigation with the intention of designing an even more hardware efficient lightweight cipher.

Mohd et. al [4] have provided a comprehensive comparison of lightweight block ciphers, which includes results for PRESENT among many others, stating that PRESENT is "a good reference for hardware implementations" [4, p.1]. According to various comparative cipher results, on a hardware platform PRESENT is first in least area and area/bit [5], third in throughput and second in throughput/area [5] and second in throughput and throughput/area [6]. PRESENT shows very good results in terms of its low area consumption and throughput, which is shown in the above results and more results that have been compiled by Mohd et. al [4].

B. Existing PRESENT Implementations

Rolfes et. al [7] report on three main lightweight hardware architectures for PRESENT that are of interest. The first of these is the basic round-based architecture. This design is a direct implementation of the top-level algorithm provided in the cipher specification. This is a low area architecture, as this was the focus of the cipher by the designers. Disadvantages to this design include the reduction of the substitution layer which is used to decrease the resource utilization. This leads to an increase in the latency. Another disadvantage is the variation of size in the data-path width requires extra routing and control that will lead to an area overhead.

A parallel architecture is also reported by Rolfes et. al [7] with the purpose of achieving a high throughput. This design uses a pipelined architecture by unrolling the 31 rounds of encryption and cascading the components, such as the substitution and permutation layers (p-layers). A clear disadvantage to this design is that it will result in a significant increase in the resource utilisation.

The final architecture to be discussed by Rolfes et. al [7] is a serialised approach to the PRESENT cipher. This design modifies the round-based architecture to result in a further reduction in the area utilisation. This is achieved by reducing the data being processed to 4 bits. This means there is only a need for 1 substitution-box (s-box) in the substitution layer, which is one of the most area consuming parts of the design due to the 16 parallel s-boxes. Due to only 4 bits being processed each clock cycle, there will be an increase in the computation time. 20 clock cycles are needed for initialisation as well as an additional 15 cycles needed to compute the substitution layer of each round when compared to the round-based implementation. A memory structure with two different modes is used for the load and encryption phases.

A 16-bit architecture has been proposed by Lara-Nino et. al [7] which allows for a reduction in the number of substitution boxes as well as reducing additional logic that is necessary in designs that require a variable data-path width. The main advantages of this design are a low area utilisation and a reduced latency compared to the previous serialised

architecture. A disadvantage to this design is a potential vulnerability to side-channel attacks.

III. Technical Description

A. The PRESENT Cipher

PRESENT is a hardware-optimised block cipher that has been designed with area and power constraints in mind. PRESENT is a 31-round substitution-permutation network (SPN) which has a block size of 64-bits and accepts an 80-bit or 128-bit key. This project focuses on the 80-bit key implementation of the cipher. The operations used in this cipher for encryption are XORs, 4-bit to 4-bit s-boxes and bit permutation. The key schedule provides 64-bit round keys for each round of encryption.

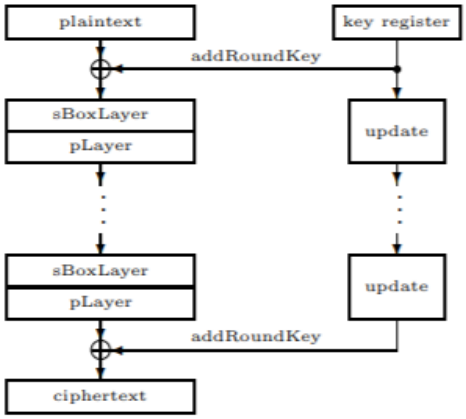

Figure 1: Top-level Algorithmic representation of PRESENT [8]

Figure 1 shows the encryption process of the cipher in an algorithmic representation, with the key schedule parallel to the encryption providing the round keys. The keys are updated by using an SPN. An XOR operation is performed with the round keys and the current state of the 64-bit text in the encryption process.

B. PRESENT-80 Cipher Core

Two hardware cipher cores were developed for PRESENT in VHDL. The first is a round-based architecture, with a focus on low area utilisation and low power consumption. The second architecture developed is a pipelined or parallel architecture which focuses on a high throughput. These VHDL designs are targeting a Zybo Zynq FPGA platform, through which hardware acceleration of the encryption can be achieved.

The round-based cipher is based on the algorithm description provided in the algorithm specification as seen in Figure 1. A loop-based approach is used to save area and power in this design. Reusable blocks in the 31 round structure allows for this design to display low area utilisation and power consumption, as well as a high energy efficiency. This design makes use of 2-1 multiplexers to switch between the load phase and the round computation phase for the plaintext and key inputs. The control logic in this design is implemented using a finite state machine (FSM) and a 5-bit counter to count the rounds. The FSM is also used to switch between the load and encryption phases. As previously mentioned, this design focuses on a low area utilisation and power consumption. Clock gating is a method implemented to reduce the power.

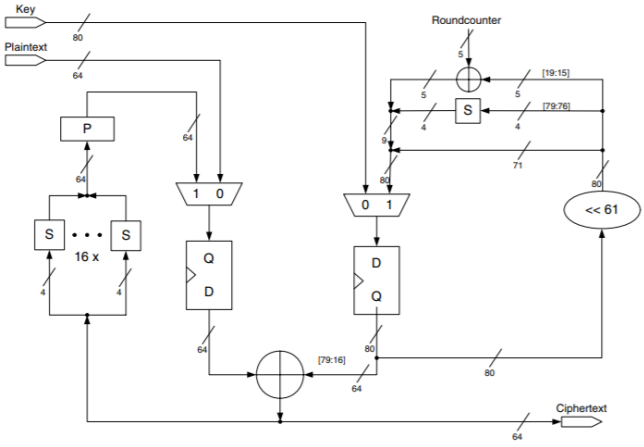

Figure 2: Round-based architecture block diagram [9]

The pipelined cipher has a significant increase in the area utilisation and power consumption when compared to the round-based architecture. This can be attributed to the unrolling of the 31 rounds of encryption and cascading the components, such as the substitution and permutation layers. This is captured in Figure 3, where the first and last rounds are shown after the XOR. This is repeated 31 times, with components being generated for each round. There is a total of 32 64-bit XORs, 496 4-bit s-boxes and 31 64-bit p-layers. The 16 4-bit s-boxes required for the 64-bit text run in parallel. The unrolling of the encryption rounds will lead to a significant increase in the throughput; however, this comes at the cost of a significant increase in the area utilisation also. This design also results in a reduction to the operating frequency.

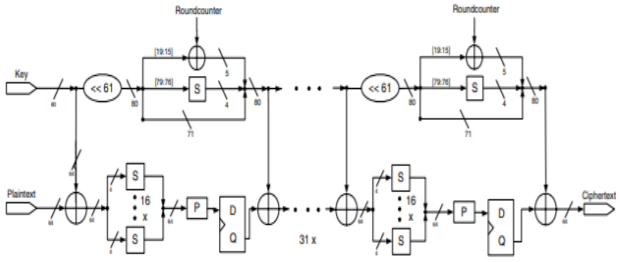

Figure 3: Pipelined architecture data-path [9]

The improvement in the throughput is seen over a larger amount of data, as pipelined architectures allow for an input to be accepted at each clock cycle.

C. Design Validation

The algorithm specification provides 4 test-vectors (plaintext and key) which can be run through the design, to achieve a specified value for the ciphertext. These test-vectors can be set as the input to the system through the use of VHDL testbenches. The test vectors are set as stimuli and a behavioural simulation is run. The output response (ciphertext) to the stimulus can be compared to the expected response from the given test-vectors.

D. AXI Peripheral

The AXI peripheral designed encompasses the RTL design, which has been packaged as an IP, as well as the

BRAM which is used to store the plaintext and ciphertext data and the processing system (PS) which is an ARM Cortex-A9 processor. The AXI peripheral is built to allow for hardware acceleration of the cipher. Hardware acceleration is a key aspect to the PRESENT cipher, as it was designed with hardware optimization in mind.

An AXI-Lite interface is chosen for this project due to its simplicity and low-area utilisation, as area constraints is one of the key focuses of this cipher. Figure 4 shows the full system to be created to allow for the hardware accelerated implementation.

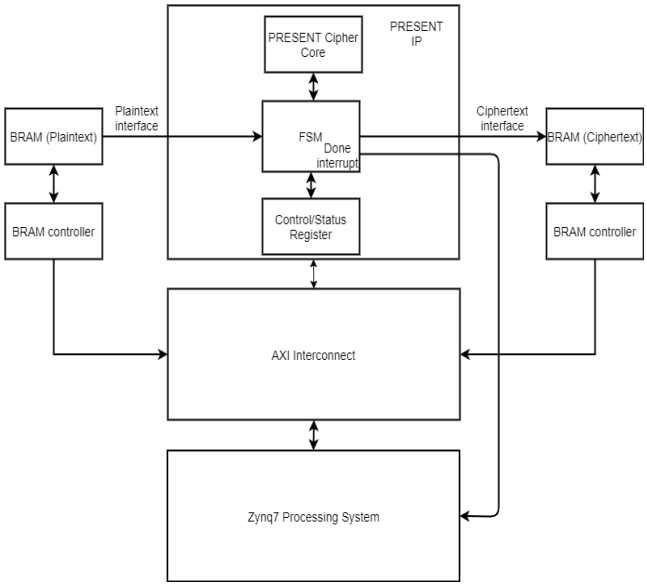

Figure 4: Full AXI peripheral

The IP for the design contains the round-based cipher core, which is controlled by a FSM. The FSM is designed as a Moore FSM, which means that its output values are determined by only its current state. A state diagram for the flow of the FSM is presented in Figure 5.

The control of the round-based core is provided by the FSM, which has logic implemented to set the address for the plaintext and ciphertext BRAM, it sets the load port for the cipher core so that inputs can be accepted and it counts the blocks of text to be encrypted as well as counting the rounds. The FSM is connected to the cipher core, the control/status registers (CSR) and two tiles of BRAM.

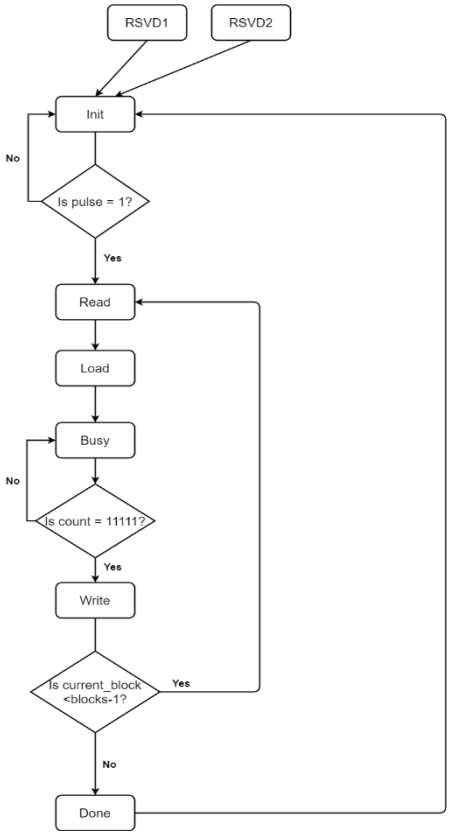

Figure 5: FSM State diagram

The CSR contains information about the number of blocks to be encrypted, a bit which signifies the encryption is complete with a ‘high’ signal and a pulse bit which signifies that encryption can begin when its bit is set to ‘high’. There are also three registers that are used for the storage of the 80-bit key.

There are two BRAM tiles, one to store the plaintext and one to store the ciphertext. Ports have been created in the IP for both the plaintext and ciphertext interfaces, to allow it to connect to the BRAM interfaces ports through a port mapping process.

Vivado has an IP integration feature which can be used to create the connections between the various components of the design, as well as the creation of components such as the BRAM and BRAM controllers. The BRAM is set as true dual port BRAM, which allows for reading and writing of data through two ports, which can be used by the PRESENT IP and the CPU. The memory depth has been set through the AXI BRAM controller to a value of 1024, with a data width of 64 bits. This allows for 1024 blocks of plaintext/ciphertext to be stored in each tile of BRAM.

Once the full design has been created, a bitstream can be generated for the design and exported, so that a software driver can be designed to invoke the hardware.

E. Software Driver

A software driver file has been created in C to invoke the hardware peripheral design from the PS. The development of this driver was in Xilinx SDK, which can be used to program the FPGA as well as write the software driver and test code. During the development of the AXI peripheral, a base address was set for the AXI slaves. The base address and offsets can

be found in an auto-generated file in Xilinx SDK. An interrupt service routine is used to copy the ciphertext data from the BRAM to system memory. The driver also performs some initial configuration for the developed AXI peripheral and writing data such as the key and plaintext values. Similar to the RTL development, the test vectors provided by the algorithm specification can be used for verification of the hardware accelerated design.

IV. Results Obtained

A. *Resource Utilisation Metrics*

The area utilisation can be found using a similar method to the power consumption. Vivado provides a report for the utilisation, which also breaks it down into specifics such as the slice LUTs, slice registers and BRAM tiles.

Table 1: Area utilisation results

Design	Total Slices	Slice LUTs	Slice Registers	BRAM Tiles
Round-based core	70	209	213	0
Pipelined core	979	2815	4227	0
PRESENT IP	172	289	533	0
Full IP peripheral	1077	2423	3127	4

A comparative visualization has been provided in Figure 6, showing the slice utilisation of the different designs using Vivado's floorplanner.

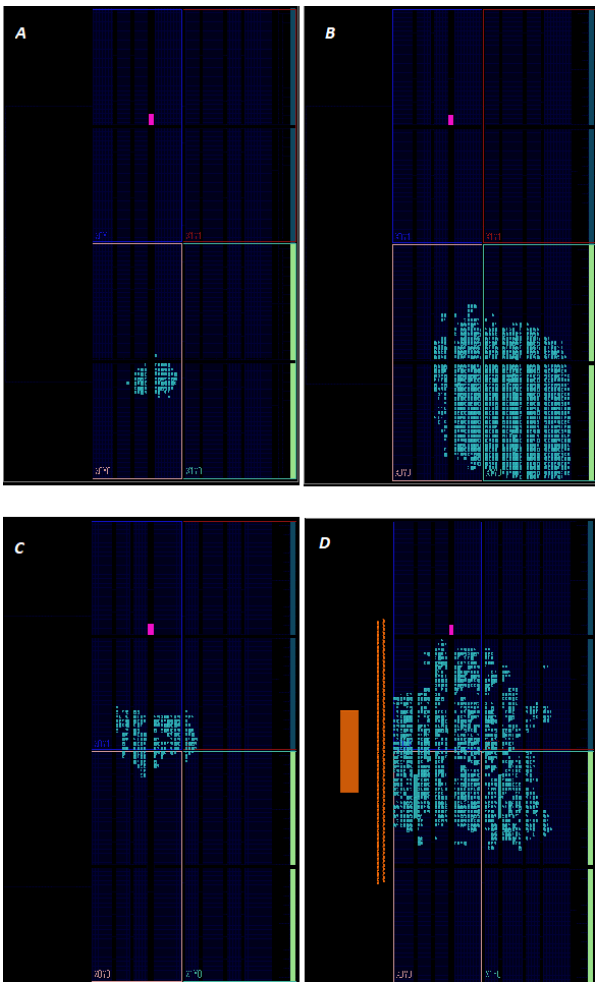

Figure 6: Area utilisation comparison
A: Round-based core. B: Pipelined core. C: Round-based IP. D: Full AXI Peripheral.

B. *Performance Metrics*

The maximum operational frequency of the design is a key metric that needs to be found before the other metrics presented are measured. Using the maximum operational frequency is important for designs that require low resource utilization. It is calculated using timing reports from Vivado, which can be obtained by running implementation for the design and determining the minimum period for the design that it does not fail timing constraints. The designs are synthesised out of context due to IO on the device not being allocated. The operational frequency can then be calculated using (1), where f_{max} represents the max operational frequency and T_{min} represents the minimum period.

$$f_{max} = \frac{1}{T_{min}} \quad (1)$$

In the following results, the round-based (rb) core and implementations using the round-based core will contain the shorted rb. It should also be noted that the full IP peripheral has the PRESENT IP integrated within it.

Table 2: Maximum operational frequency results

Design	Maximum Frequency (MHz)
Rb core	385
Pipelined core	350
Rb PRESENT IP	227
Full IP peripheral	108

Results have been obtained for a software implementation as a means for comparison with the hardware implementation. A modified version of [9] was used to obtain these results, which contains a C implementation of the PRESENT-80 cipher.

With the maximum operating frequency for each of the designs now known, the results for the resource utilization metrics can be determined. Similarly, utilisation reports can be obtained through implementation runs.

The power consumption of the designs can be found by running an implementation run. Post implementation, Vivado provides a power report which gives results of the total power consumption, as well as breaking it down into more specific details such as static and dynamic power.

Table 3: Power consumption results

Design	Total Power (W)
Round-based core	0.119
Pipelined core	0.545
PRESENT IP	0.464
Full IP peripheral	1.702

Latency is the number of clock cycles required to encrypt a block of ciphertext.

Table 4: Latency Cycles Results

Design	Latency (Cycles)
Round-based core	32
Pipelined core	32

Table 5: Latency Time Results

Design	Latency (ms)
Software core	1.09
Hardware-accelerated core	0.038

Throughput is the rate at which a new output is produced with respect to time. This can also be expressed as the time to encrypt the plaintext to ciphertext in this system. Equation (2) shows the equation to calculate the throughput, which divides the operational frequency by the latency and then multiplies this by the block size. The operational frequency is the maximum operational frequency calculated previously for these designs.

$$\mathrm{Thr} = \frac{\mathrm{f_{op}} \times \mathrm{BSize}}{\mathrm{Lat}} \qquad (2)$$

Table 6: Throughput Results

Design	Throughput (Mbps)
Round-based core	770
Pipelined core	22,400
Hardware-accelerated core	45
Software core	1.8

Table 7: Throughput at 13.56MHz

Design	Throughput (Mbps)
Round-based core	27.1
Pipelined core	840.72

V. Analysis

A. Resource Utilisation Analysis

Table 1 shows the area utilisation results, which is broken down into the total number of slices used, slice look-up tables (LUTs), slice registers and BRAM tiles. A significant increase in the area utilisation is shown in the pipelined core when compared to the round-based core with ~14x more slices used, ~13.5x more slice LUTs used and ~20x more slice registers used. The IP builds upon the round-based core, with extra ports and interfaces added, as well as logic included in the FSM and the AXI interface. This adds to the area utilisation; however, it is still significantly less than the pipelined core. Finally, the AXI peripheral includes the developed round-based IP, as well as BRAMs to store the plaintext and ciphertext, the AXI interconnect and the Zynq PS. The significant increase in the area utilisation comes as no surprise with all the extra components involved in this design. This design uses the highest number of slices and is the only design to use BRAM, however, the pipelined core has a greater utilisation of slice LUTs and slice registers. This shows the main reason why the AXI peripheral was chosen for the round-based core over the pipelined core, which is due to its extremely high area utilisation.

Area utilisation can be an awkward metric to compare its results to related work due to a divide in using slice utilisation to measure area in Xilinx FPGAs and gate equivalents in an application-specific integrated circuit (ASIC). Fan et. al [10] present results for lightweight ciphers on an FPGA, with the results presented in the number of slices. The results presented in table 1 show an increase in the number of slices used for the full IP peripheral implementation of the round-based cipher, however, the round-based core alone does not utilise many slices. The significant difference in area utilisation shown in the results can be attributed to the design of the full IP AXI peripheral.

B. Performance Analysis

The maximum operational frequency results are as expected, with the round-based core displaying the highest frequency due to having the shortest minimum period. The minimum period was expected to increase when adding extra logic for the IP and even further for the full AXI peripheral. The pipelined core has a lower maximum operational frequency than the round-based core. This is the expected result as the pipelined core has a longer critical path. An improvement in the maximum frequency is noted in the designs in this paper when compared to the results compiled by Fan et. al [10].

Table 3 shows the power consumption for each of the designs. The majority of the power consumption for each of these designs is from the dynamic power which mainly results from signals, logic, clocks, and BRAM. Static power is the power consumed by the device when it is powered up, configured with user logic and there is no switching activity. There is a similar static power for all the designs, so the difference in power consumption between these designs mainly comes from the dynamic power. The power consumption follows a similar trend to the area utilisation results, where the designs with a high area utilisation generally also resulted in a higher power consumption. This can be seen in Table 3, where the pipelined core shows a significant increase to the round-based core, with almost 5x the power consumption. The IP core shows ~4x an increase in the power consumption when compared to the round-based core on its own. There is a significant increase in the power consumption in the AXI peripheral when compared to the other designs, including the pipelined core. The AXI peripherals power consumption being ~3x higher than the next closest in terms of power consumption can be attributed to the Zynq PS, which consumed 96% of the dynamic power in the design.

The power consumption of the designs presented when compared to related works such as in [7] and [11] show a significant increase in the power consumption; however, as can be seen in [11], even the static power is significantly lower than that presented in the results of this paper. The platform used for implementations as well as the supplied voltage could both influence the results for the power consumption.

The results for the latency shown in Table 4 show an equal amount of cycles needed for the latency in both the round-based and pipelined core. This was the expected result as they both only need a single clock cycle per round of encryption.

The results for the throughput are shown in tables 6 and 7, where Table 6 shows the throughput for the designs at their maximum operating frequency and Table 7 shows the throughput for the designs at an operating frequency of 13.56MHz, which is an appropriate frequency for RF applications, which is highly applicable to this cipher [12]. Both results show that the pipelined core has a much higher throughput than the round-based core, where area and power constraints are its uppermost focus rather than its throughput. The reason for this is that the pipelined core can accept an input once the previous input has been processed through the first pipeline stage. This allows for a massive increase in the amount of data that can be in the pipeline core at once, at different stages. The results for the hardware-accelerated cipher can be compared to software-based implementations of the cipher, where the hardware-accelerated implementation performed much better in terms of its throughput and latency than the software implementation, which validates the point that it's a hardware optimised cipher.

The results in table 7 for the throughput can be compared to results in [13]. The pipelined architecture in particular shows a significant increase in the resulting throughput, however, most of the designs presented in [13] focus on a low-area implementation rather than high throughput.

VI. Conclusions

This paper has presented implementations of two different designs for the hardware-optimised lightweight block cipher PRESENT. The round-based implementation has been implemented with an AXI peripheral to achieve hardware acceleration of the encryption. A comparison between a software implementation of the cipher and a hardware-accelerated implementation of the cipher shows clear improvement in the performance in the hardware-accelerated implementation.

It is also clear that the resource utilisation of the round-based core shows much better results than the pipelined core. Consequently, the round-based core was chosen to be used for the hardware accelerated version of the cipher, as the increase in the resource utilisation for the pipelined core for the improvement in performance was deemed not worth it due to the priority of area and power constraints for this cipher.

An AXI lite interface was chosen for the design of the AXI peripheral in this project, however, this was at a sacrifice in the performance of the system where a full AXI interface would have allowed for better performance due to its burst access capabilities.

A serialised architecture such as the 16-bit architecture mentioned in II.B could further improve on the resource utilisation results presented here for the round-based implementation of the cipher.

The results presented in this paper show that the PRESENT-80 cipher is a suitable low-resource cipher that provides adequate performance while proving to be an alternative to more resource heavy ciphers such as AES.

References

[1] 'Data Encryption Standard (DES)'. U.S. DEPARTMENT OF COMMERCE, Oct. 25, 1999, Accessed: Aug. 25, 2020. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf.

[2] 'FIPS 197, Advanced Encryption Standard (AES)', p. 51.

[3] A. Y. Poschmann, Lightweight Cryptography: Cryptographic Engineering for a Pervasive World. 2009.

[4] B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, 'A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues', *Journal of Network and Computer Applications*, vol. 58, pp. 73–93, Dec. 2015, doi: 10.1016/j.jnca.2015.09.001.

[5] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, 'A Survey of Lightweight-Cryptography Implementations', *IEEE Design Test of Computers*, vol. 24, no. 6, pp. 522–533, Nov. 2007, doi: 10.1109/MDT.2007.178.

[6] C. De Cannière, O. Dunkelman, and M. Knežević, 'KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers', in *Cryptographic Hardware and Embedded Systems - CHES 2009*, Berlin, Heidelberg, 2009, pp. 272–288, doi: 10.1007/978-3-642-04138-9_20.

[7] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, 'Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents', in *Smart Card Research and Advanced Applications*, Berlin, Heidelberg, 2008, pp. 89–103, doi: 10.1007/978-3-540-85893-5_7.

[8] A. Bogdanov *et al.*, 'PRESENT: An Ultra-Lightweight Block Cipher', in *Cryptographic Hardware and Embedded Systems - CHES 2007*, vol. 4727, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466.

[9] P. Tonkovic, *Pepton21/present-cipher*.

[10] X. Fan, G. Gong, K. Lauffenburger, and T. Hicks, 'FPGA implementations of the Hummingbird cryptographic algorithm', in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Jun. 2010, pp. 48–51, doi: 10.1109/HST.2010.5513116.

[11] L. Batina *et al.*, 'Dietary Recommendations for Lightweight Block Ciphers: Power, Energy and Area Analysis of Recently Developed Architectures', 753, 2013. Accessed: Aug. 30, 2020. [Online]. Available: http://eprint.iacr.org/2013/753.

[12] 'RFID Frequency Bands & Spectrum » Electronics Notes'. https://www.electronics-notes.com/articles/connectivity/rfid-radio-frequency-identification/frequency-bands-spectrum.php (accessed Aug. 25, 2020).

[13] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, 'Lightweight Hardware Architectures for the Present Cipher in FPGA', *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2544–2555, Sep. 2017, doi: 10.1109/TCSI.2017.2686783.

Appendix A
Literature Review

Preface

A few updates have been made to the literature review since it was originally submitted. In section II.B, there has been an extra two paragraphs added in to describe a pipelined architecture, along with a figure which displays the data-path of the pipelined architecture. Section II.C on cryptographic attacks has also been reduced due to the information not being too important in the design and implementation of the project. Sources were added in for the cryptographic attack section as well as figure 5 and table 1, which were originally missing sources. Finally, a sentence was adapted from saying there can be a 'sacrifice' to the security as this was a poor choice of words.

Rolfes et. al, reference [7] in the paper was one of the more relevant and significant sources from these that was used to progress the project. The low-area architecture and parallel architecture were both used to design the cipher.

Bogdanov et. al, reference [4] in the paper was also a key source to progress the project. This contains the cipher specification, which has information such as the top-level algorithm for the cipher, paragraphs on the steps for each component involved in the cipher as well as test vectors provided to validate the functionality of the implementations.

IP Core Development of PRESENT-80 Lightweight Cipher

Alex Doherty

Dublin City University

***Abstract*— The PRESENT family of lightweight block ciphers were designed to provide a solution for the use of block ciphers in constrained environments, such as RFID tags and sensors. The area of IoT is an industry which will benefit greatly from lightweight cryptography, allowing for secure communication and storage of data. The present cipher allows for implementations using either an 80 or 128-bit key. This project will focus on 80-bit key implementations. Multiple implementations of PRESENT will be designed and implemented using Vivado Design Suite and then packaged as an IP core and integrated as an AXI peripheral which can be invoked by software. The implementations of the algorithm will be measured and compared in terms of various metrics to evaluate the performance of the architecture. This literature review looks at prior work which has been done for this cipher, including multiple implementations with results that are compared to each other. Finally, this review relates the work which has been done and how it can be used to plan for and implement the project.**

Keywords—cryptography, cipher, IoT, encryption, Xilinx, decryption, PRESENT, security, plaintext, ciphertext, FPGA, Zybo, Zynq, IP core, latency, throughput, area utilization, power consumption.

I. INTRODUCTION

Present is a hardware optimized block cipher which has been designed with area and power constraints in mind. A block cipher encrypts a block of plaintext, in contrast to a stream cipher which encrypts the plaintext one bit at a time. It has been designed as a block cipher over a stream cipher as block cipher design seems to be better understood than stream cipher design. AES is the preferred choice of cipher for the majority of block cipher applications; however, AES is not suitable for extremely constrained environments. Present is a lightweight cipher which supports an 80-bit and 128-bit key. The motivation for lightweight ciphers in recent years is due to the need for security against attacks in constrained environments. [1] Examples of where these ciphers could be applied are RFID tags, contactless cards, sensors, etc. An area in which lightweight cryptography is important for is the Internet of Things (IoT). In IoT devices, it is common for there to be end-to-end communication. It is important for many of the devices that communication between devices is secure. The issue with lightweight cryptography is that it is not easy to implement the cryptographic functions necessary for the security on constrained devices. There will often be a sacrifice in terms of cryptographic strength when designing a cipher for constrained environments, such as in an IoT device. As a result, one of the challenges to be addressed, and in which there has been a lot of research into in recent years is the creation of a cipher which has adequate security while not having a significant effect on the device performance, the longevity or the cost. For lightweight cryptography, there are different measures which are important for evaluating the lightweight properties. These measures will depend on if the cipher being implemented is hardware-based or software-based. For a hardware-based cipher such as PRESENT, chip size and/or energy consumption are important measures in evaluating the lightweight properties. The use of the lightweight symmetric key algorithm allows for a lower energy consumption, which is advantageous for IoT. [2]

The title of the project I will be implementing is IP Core Development of PRESENT-80 Lightweight Cipher. The aim of this project is to develop an IP core of the PRESENT cipher, using an 80-bit key size on a Xilinx Zynq system on a chip (SoC) [3]. The Zynq SoC contains a dual-core ARM Cortex-A9 processor along with a Xilinx 7-series field programmable gate array (FPGA) logic. The cryptographic algorithm can be designed in the Xilinx Vivado software suite. The implementation of the project will be packaged as an intellectual property (IP) core and integrated as an AXI peripheral which can then be invoked by software for hardware acceleration of the encryption. Certain parameters will be measured to evaluate the performance of the implementation. The metrics that will be measured are hardware area utilization, latency, throughput and power consumption.

Multiple implementations of the cipher will be designed and tested. This will allow for a comparison of the above metrics between the different designs. This means conclusions can be drawn in terms of what the benefits and downsides to the different designs are. The PRESENT cipher employs a round-based architecture, which allows for the reuse of components. This will lead to a reduction in the area utilization and the power consumption. Area utilization and power consumption will often rise and fall together, so an implementation with a high area utilization will likely require a greater amount of power. It is important to realise that this is not always true, but it is usually the case. From this relation, an implementation focusing on a reduction in the area utilization and hence, the power consumption will be designed. Conversely, latency and throughput have a converse relationship with each other, so an implementation with a high amount of latency will lead to a reduction in the throughput. Using this relation, an implementation which focuses on a reduction in latency and hence, an increase in the throughput will be designed.

II. REVIEW & ANALYSIS OF PRIOR WORK

A. Cryptographic Ciphers

The purpose of a cryptographic cipher is to convert plaintext, which is given as an input to the cipher, into ciphertext, which will be the output of the cipher. Keys will be used to perform this conversion along with mathematical operations to encrypt the text. Ciphers perform a series of steps to transform the plaintext into ciphertext and similarly, a

backwards traversal of the encryption steps can be performed to transform the ciphertext into plaintext. Cryptography is important to keep information secure and out of the hands of those it is not intended for.

B. PRESENT

PRESENT is a lightweight block cipher which was developed by Andrey Bogdanov et. al in 2007. It was developed as an alternative to AES, which is a block cipher that is less suitable for use in constrained environments. PRESENT uses a substitution-permutation (SP) network as well as an iterative 31 round structure. The block of plaintext which is the input to the system to be encrypted is 64 bits and the cipher supports both an 80-bit and 128-bit key size input. Within the 31 rounds of encryption that take place in the cipher, substitution, permutation and key mixing are used. The goals and environment of use which are stated in the specification for the PRESENT cipher are that it should be implemented in hardware, the applications should only require moderate security, applications are unlikely to require encryption of large amounts of data, the area utilization is priority followed by power consumption and timing metrics. [4]

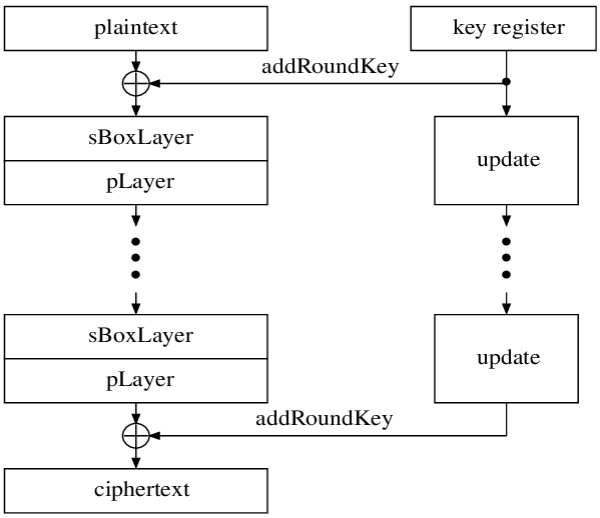

Figure 1: Algorithmic representation of PRESENT cipher [4]

Figure 1 above shows a graphic representation of the PRESENT cipher. This shows a breakdown of different components in the encryption process. The key mixing is performed using an XOR operation, the sBoxLayer block represents the S-box substitution and the pLayer block represents the permutation. The S-box used is a 4-bit to 4-bit substitution, using a substitution table given in the specification. The bit permutation is performed using a table provided by the specification also. The key schedule runs parallel to the encryption process as it needs to provide keys for each round of the encryption. The keys are updated by using a SP network as shown in figure 2.

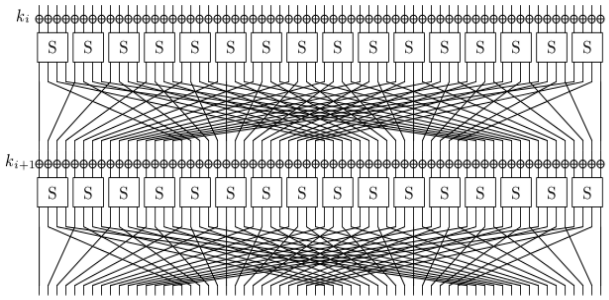

Figure 2: SP Network for PRESENT [4]

The steps taken to update the keys in the key schedule are as follows:

1. Rotate key register by 61 bit positions to the left
2. The left-most four bits are passed through the S-box
3. Round counter value i is exclusive-ored with the 5 bits of the current round counter

C. Cryptographic Attacks

The cipher specification has provided a security analysis for different forms of cryptographic attacks. Differential and linear cryptanalysis is one technique which can be used to obtain the data which has been encrypted. A lower bound to the number of active S-boxes involved in a differential or linear characteristic has been provided. For differential and linear cryptanalysis, theorems have been provided for the cryptanalysis of each. In differential cryptanalysis, advanced techniques can be used by a cryptanalyst to exploit a shorter characteristic, however, even then there is not enough data available to exploit the remaining rounds differential characteristics. A practical confirmation is also provided to confirm that the bound on the number of active S-boxes in the theorem is tight. For linear cryptanalysis, under the assumption that a cryptanalyst would need only to approximate 28 of the 31 rounds in PRESENT to mount a key recovery attack, linear cryptanalysis would then require data requirements that exceed the available plaintext/ciphertext. [4]

Due to PRESENT being almost exclusively bitwise and the permutation operation being semi-regular, this means the development and propagation of word-wise structures will be interrupted by the bitwise operations, making PRESENT less susceptible to a structural attack. [4]

Another form of attack is an algebraic attack, which has better success when applied to stream ciphers. As PRESENT is a block cipher, it is less susceptible to these attacks, however, due to the simple structure of the cipher, it was deemed as important to study the attacks on PRESENT. [4]

D. Existing Implementations

Lara-Nino et. al [5] report on a number of different lightweight hardware architectures for PRESENT. The basic architecture which is presented by Bogdanov et. al [4]. This design closely follows the algorithm specification and the latency is equivalent to the number of rounds.

The next architecture is a low area architecture which reduces the number of substitution boxes, leading to a lower area utilization but an increase in the latency. An illustration of this architecture is shown in figure 3.

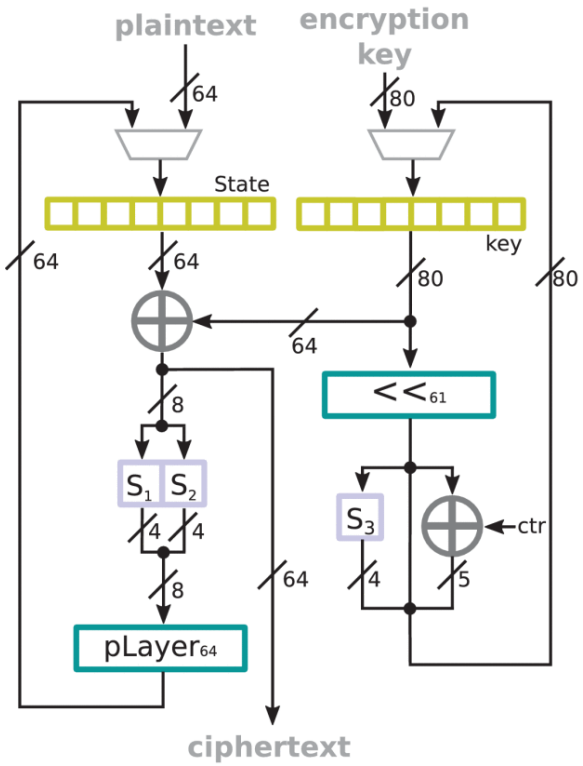

Figure 3: Low area architecture implementation of PRESENT [5]

Two disadvantages for this architecture have been listed, the first being the need for additional logic for routing and control which will lead to area overhead. The second disadvantage is that the reduction of the substitution layer which is used to decrease the resource utilization leads to an increase in the latency.

A 16-bit architecture designed by Andres Lara et. al [6] in 2016 aimed to reduce the data-path width, along with the substitution layer and permutation layer width.

The permutation layer width is reduced from 64-bit to 16-bit, meaning the substitution layer can also utilize a width of 16-bit. This will lead to a reduction in the number of substitution boxes to just 4. In this design, it takes 4 cycles to input the data, 124 to perform the encryption and 4 more to produce the output data. There is a reduced latency in this architecture but there are security concerns as there is a possibility that there are vulnerabilities to side-channel attacks.

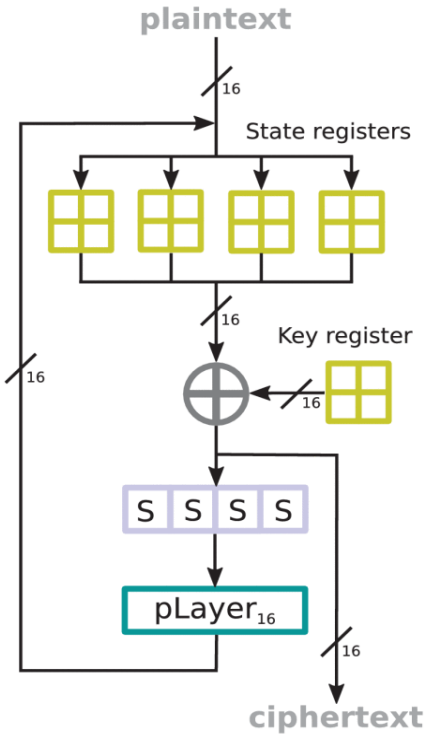

Figure 4: 16-bit architecture of PRESENT [6]

The final architecture reported on that contains an 80-bit key rather than a 128-bit key is one which has been proposed by Lara-Nino et. al [5].

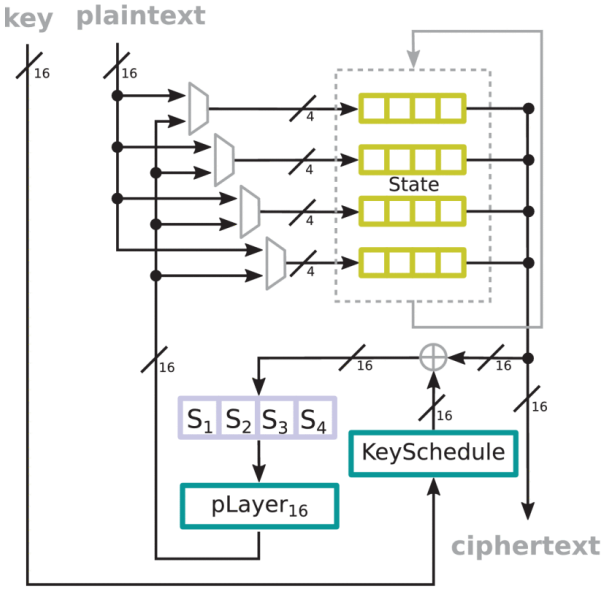

Figure 5: 16-bit architecture with updated key schedule [5]

This proposed architecture is similar to the previous architecture described but with a few key differences. One difference is the alternative method in the generation of the round keys. Four 20-bit registers are used to store the initial key and update its value each round. The round-key to be extracted for each round will consist of the 4 right-most bits from each of the registers. The content of the registers will subsequently be shifted to the right to extract the adequate key for the 16-bit state word being processed. The round key is

updated at the end of each round. This implementation requires five cycles to divide the input key into 16-bit words, but no additional cycles are required for updating the round key as this is performed in parallel with the data processing.

A comparison between the various architectures can be seen in table 1 below.

Table 1: Comparison of Different Hardware Architectures of PRESENT [5]

Name	Area (GE)	Latency	Circuit Count
Basic	1570	33	-
Low-area	1105	250	898
16-bit	576	132	528
New 16-bit	989	133	926

The advantages of the different architectures can be seen from table 1, with the 16-bit implementation obtaining the lowest area utilization, which is measured in gate equivalences. It is noted that for the 16-bit architecture, the area for the key generation module has not been taken into account so the value for the area would increase if the key generation module was considered. The values for the basic architecture are taken from the cipher specification [4], which resulted in the lowest value for the latency, which is measured in cycles. In this architecture, there is one cycle per round, with an extra cycle each for the initial round and to produce the output.

The pipelined cipher has a significant increase in the area utilisation and power consumption when compared to the round-based architecture. This can be attributed to the unrolling of the 31 rounds of encryption and cascading the components, such as the substitution and permutation layers. This is captured in Figure 6, where the first and last rounds are shown after the XOR. This is repeated 31 times, with components being generated for each round. There is a total of 32 XORs, 496 Substitution boxes (S-boxes) and 31 p-layers. The 16 4-bit s-boxes required for the 64-bit text run in parallel. The unrolling of the encryption rounds will lead to a significant increase in the throughput; however, this comes at the cost of a significant increase in the area utilisation also. This design also results in a reduction to the operating frequency.

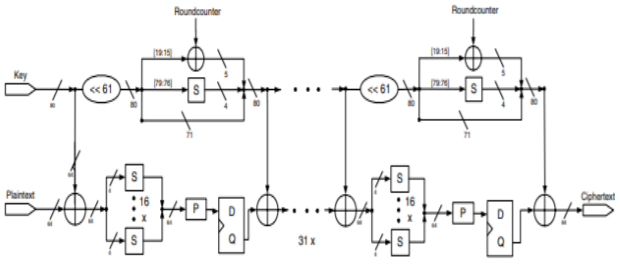

Figure 6: Pipelined architecture data-path [7]

The improvement in the throughput is seen over a larger amount of data, as pipelined architectures allow for an input to be accepted at each clock cycle.

There have been various studies done on comparisons between lightweight ciphers showing that PRESENT performs well in terms of mainly its area utilization and power consumption and also its throughput. From 6 different lightweight ciphers which have been compared, including AES, PRESENT ranks second in terms of its energy consumption per bit as well as ranking first in terms of its area utilization [8]. PRESENT has also ranked first in terms of area utilization per bit when compared to 8 other lightweight ciphers, including block and stream ciphers. PRESENT also ranks third in terms of its throughput and second in terms of its throughput per unit area [9]. In the comparison of various ciphers by Guo et. al [10], PRESENT ranked third in terms of its area utilization and second in throughput per gate equivalence.

E. Relating Prior Work to Project Problem

The architectures and results which have been reviewed in this paper are for 80-bit implementations of the PRESENT cipher, rather than its 128-bit implementation which will not be implemented in this project. The results obtained from prior implementations are a good benchmark for the kind of results that the project should aim to achieve, although they will not be exact due to certain possible environmental differences such as where the cipher is implemented. A Xilinx FPGA for example, may yield slightly different results when compared to an Altera FPGA. The architectures which were reviewed in this paper were all chosen for the 80-bit implementation of the cipher also. This means that the results can be viewed for this and used to help select which architectures should be implemented in this project. It will also be beneficial to keep in mind certain methods of implementation which are more secure. A knowledge of the types of potential attacks on the cipher means it can be built with these attacks in mind. As there are certain implementations that could potentially lead to vulnerabilities to certain attacks, it is important to note this so that this type of design is used for applications which are suited to and can deal with a slight reduction to the security.

III. CONCLUSION

This literature review has presented the lightweight hardware-based block cipher PRESENT. The review focuses on the 80-bit key implementation of the cipher. This cipher will be implemented through hardware so there is a focus on hardware implementations rather than software implementations in this paper. The basic specification and implementation of the cipher has been presented, along with details to the key schedule which runs parallel to the round encryption process. Various different architectures of the cipher have been reviewed to see the advantages and disadvantages of implementing these. This research will allow for an informed decision to be made when choosing the architectures of the cipher to implement and compare. Results from various different sources show that PRESENT has a low area utilization, power consumption and reasonably high throughput. As is mentioned in the cipher specification, it was designed with area and power constraints in mind so it will be important to ensure that these criteria are kept to when implementing the cipher.

REFERENCES

[1] A. Y. Poschmann, *Lightweight Cryptography: Cryptographic Engineering for a Pervasive World*. 2009.

[2] M. Katagi, 'Lightweight Cryptography for the Internet of Things', 2011.

[3] 'Zynq-7000 SoC Data Sheet: Overview (DS190)', p. 25, 2018.

[4] A. Bogdanov *et al.*, 'PRESENT: An Ultra-Lightweight Block Cipher', in *Cryptographic Hardware and Embedded Systems - CHES 2007,*

vol. 4727, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466.

[5] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, 'Lightweight Hardware Architectures for the Present Cipher in FPGA', *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2544–2555, Sep. 2017, doi: 10.1109/TCSI.2017.2686783.

[6] C. A. Lara-Nino, M. Morales-Sandoval, and A. Diaz-Perez, 'Novel FPGA-Based Low-Cost Hardware Architecture for the PRESENT Block Cipher', in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug. 2016, pp. 646–650, doi: 10.1109/DSD.2016.46.

[7] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, 'Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents', in *Smart Card Research and Advanced Applications*, Berlin, Heidelberg, 2008, pp. 89–103, doi: 10.1007/978-3-540-85893-5_7.

[8] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert, 'Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint', in *Cryptographic Hardware and Embedded Systems – CHES 2012*, Berlin, Heidelberg, 2012, pp. 390–407, doi: 10.1007/978-3-642-33027-8_23.

[9] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, 'A Survey of Lightweight-Cryptography Implementations', *IEEE Design Test of Computers*, vol. 24, no. 6, pp. 522–533, Nov. 2007, doi: 10.1109/MDT.2007.178.

[10] J. Guo, A. Poschmann, M. Robshaw, and T. Peyrin, 'The LED Block Cipher', 600, 2012. Accessed: Jan. 26, 2020. [Online]. Available: http://eprint.iacr.org/2012/600.

Table of Figures

Figure 1: Xilinx SDK IDE 22
Figure 2: Digilent Zybo development board 23
Figure 3: Zynq Ap SoC architecture 24
Figure 4: High level design flow [2] 25
Figure 5: AXI peripheral controller interface [3] 26
Figure 6: Dual port BRAM [4] 27
Figure 7: Datapath of round-based PRESENT 29
Figure 8: Key schedule logic 29
Figure 9: Top level file for round-based cipher 30
Figure 10: Datapath for pipelined cipher 31
Figure 11: Encryption process in pipeline stage 32
Figure 12: Generation of pipeline stages 32
Figure 13: State Diagram of designed FSM 33
Figure 14: Interface settings 35
Figure 15: Code to set the key and number of blocks from the slave registers 36
Figure 16: Control of pulse which signals cipher execution to the FSM controller 36
Figure 17: Package IP Tab 37
Figure 18: Block Design stage 39
Figure 19: Developed AXI peripheral with integrated IP core 39
Figure 20: Start time measurement 40
Figure 21: End time measurement 41
Figure 22: Source testbench files 42
Figure 23: Clock generator process 43
Figure 24: Check result process 43
Figure 25: Vivado waveform viewer 44
Figure 26: Timing summary 48
Figure 27: Round based area utilisation floorplan 51
Figure 28: Pipelined area utilisation floorplan 52
Figure 29: IP area utilisation floorplan 53
Figure 30: AXI peripheral area utilisation floorplan 54
Figure 31: Area utilisation floorplan comparison 55

Appendix B
Project Plan

Project Design Plan

Research Question

The aim of this project is to compare and analyse the performance of the various designs that can be used to implement the PRESENT-80 lightweight block cipher.

Project Scope

This project will include:

- Xilinx Vivado Design Suite will be used for synthesis and analysis of HDL designs which will be created using VHDL.
- The Xilinx Zynq-7010 SoC which contains 7-series FPGA logic for the development of an IP core.
- The Xilinx Software Development Kit for embedded software applications that can be used for testing and debugging.
- A software implementation of the cipher will be designed in C using Eclipse as an IDE.
- VHDL is the language that will be used to design the cipher implementations.
- Three different designs will be made for the hardware implementations: a round-based architecture, a pipelined architecture, and a serialised architecture.
- Testbenches will be created to run test vectors, provided by the cipher specification, as inputs to the cipher and ensure the functionality of the design created.
- Xilinx Core Generator software will be used to develop an AXI peripheral.
- Software will be developed to perform the hardware accelerated encryption, using the Zynq board.
- Measurement and analysis of performance and resource usage metrics such as hardware area utilisation, latency, maximum operational frequency, throughput, and power consumption.
- Comparison of the different approaches to implement the cipher.

This project will not include:

- An evaluation of the decryption process for the PRESENT-80 cipher.
- The use of languages other than C and VHDL for the implementation of the design.
- Parallel computation using multiple cipher core instances.

Design Approach

Design & Implementation

The design of this project has three key sections to it: the design of the software implementation of the cipher in C, the design of the hardware implementation of the cipher in VHDL.

The software design of the cipher will target the ARM based processor on the Zynq board. This implementation of the cipher will encrypt the input bits using the PRESENT algorithm without the hardware-acceleration that will be used in other implementations.

There are three different hardware implementations that will be designed for this project. The first one is a round based approach. The second is a pipelined approach, which can be used to achieve a higher throughput but will consequently lead to high area utilisation and power consumption. The

third implementation is a serialised approach that will process 4 bits at a time instead of 64 bits. This approach leads to a reduction in area utilisation and power consumption but there will be a significant increase to the computation time.

Testing & Validation

The main testing and validation for this project can be done using the test vectors provided by the algorithm specification. For the hardware implementations, testbenches can be created to set the test vectors as the inputs to the system and ensure that you get the correct output. The software and full peripheral implementations can be tested and validated by creating test software applications.

Analysis

To analyse the various implementations of the cipher, metrics will be measured to evaluate and compare the implementations. These metrics will relate to the performance (throughput and latency) and the resource utilisation (area utilisation, power consumption) of the cipher. For the hardware implementations, there are automatically generated reports available for area utilisation and power consumption with Vivado. The maximum operational frequency can be obtained from these reports. The latency and throughput can be calculated from the results gained using the testbenches for the cipher.

The ciphers will be compared on the individual metrics as well as a comparison in terms of their resource utilisation and performance. From these results, the advantages and disadvantages of each design can be determined, as well as suitable applications for each design. Varying sizes for the payload will be tested to compare the impact of the hardware acceleration of different sizes for the payload.

Confirmation

Upon consultation with my supervisor, I have established the portions of this project that can be completed remotely without any access to the college. The Zynq board is required for testing the hardware implementation, however the rest of the project can be completed without this board including analysis of the hardware implementation through Vivado. This plan allocates time for testing the implementation using the Zynq board however, this portion of the timeline can simply be ignored as it is not required to complete any other aspects of the project.

Detailed Timeline

A detailed timeline of the activities and tasks that will be undertaken, as well as milestones that will be hit are shown below. The spreadsheet can be found from here, where the timeline can be viewed clearly.

Original Plan

Milestones	Activity	Task	Description	Projected Start Date	Projected Finish Date	Start Date	Finish Date	Extra Details
	Literature Review	Research	Finding sources to be used for the literature review	15/01/20	16/01/20	14/01/20	16/01/20	
		Key Points	A summary of the key points that the literature review will address	17/01/20	17/01/20	16/01/20	16/01/20	
		Write Document		20/01/20	25/01/20	20/01/20	26/01/20	
		Formatting		26/01/20	26/01/20	26/01/20	26/01/20	
Milestone 1		Submission	Literature review submission	26/01/20	26/01/20	26/01/20	26/01/20	
	Presentation	Slide preparation	Creation of Powerpoint slides	10/02/20	14/02/20	10/02/20	14/02/20	
		Presentation preparation	Preparation of timing and what to address	18/02/20	19/02/20	18/02/20	19/02/20	
		Presentation Recording		20/02/20	20/02/20	24/02/20	24/02/20	Recording of presentation moved due to issues with set up in room
Milestone 2		Submission	Submission of slides and video presentation		24/02/20		24/02/20	
Milestone 3		Interview						
	Project Plan	Document Overview	Complete shorter sections of project plan	01/06/20	02/06/20	30/06/20	01/07/20	
		Design Approach	Planning of the design of various implementations & result measuring	02/06/20	04/06/20	02/06/20	04/06/20	
		Detailed Timeline	Plan for a realistic and streamlined completion of tasks and activities	05/06/20	10/06/20	06/06/20	12/06/20	Days missed during this period due to other commitments
		Complete Document	Fill in sections which need updating or have not been completed	11/06/20	14/06/20	13/06/20	14/06/20	
Milestone 4		Submission	Submission of project design plan		14/06/20	14/06/20	14/06/20	
	Implementation	Develop round-based cipher core		10/02/20	16/05/20	10/02/20	17/05/20	Issues with timing led to taking slightly longer
		Validate round-based cipher core		16/05/20	17/05/20	10/05/20	17/05/20	
		Develop AXI peripheral using Xilinx IP		18/05/20	30/05/20	18/05/20	25/05/20	Previous documentation sped up this process
		Document cipher core implementation	Details on VHDL design of cipher core	29/05/20	01/06/20	27/05/20	30/05/20	
		Develop pipelined cipher core		01/06/20	20/06/20	25/05/20	05/06/20	Extra time given as it overlaps with project plan. Re-use of components sped up process
		Validate pipelined cipher core		20/06/20	21/06/20	05/06/20	06/06/20	
		Develop AXI peripheral using Xilinx IP		22/06/20	28/06/20			
		Document cipher core implementation	Details on VHDL design of cipher core	28/06/20	30/06/20			
		Develop serialised cipher core		30/06/20	15/07/20			
		Validate serialised cipher core		15/07/20	16/07/20			
		Develop AXI peripheral using Xilinx IP		16/07/20	20/07/20			
		Document cipher core implementation	Details on VHDL design of cipher core	21/07/20	22/07/20			
		Develop software drivers	Software applications to test and run full IP peripheral	23/07/20	23/07/20			
		Document full IP peripheral	Details on how it was implemented and how it functions	24/07/20	29/07/20			
		Develop software implementation		30/07/20	07/08/20			
		Document software implementation		08/08/20	09/08/20			
		Debugging/Improvements	Clean up code/Minor improvements in presentation and running of code	10/08/20	12/08/20			
	Testing and Analysis	Validate round-based cipher core		16/05/20	17/05/20	10/05/20	17/05/20	
		Validate pipelined cipher core		20/06/20	21/06/20	05/06/20	06/06/20	
		Validate serialised cipher core		15/07/20	16/07/20			
		Validate IP peripherals		12/08/20	15/08/20			
		Develop timing constraints		14/08/20	16/08/20			
		Obtain resource utilisation reports	Reports for power consumption (mw) and area utilisation (LUTs)	16/08/20	18/08/20			
		Develop test application	Software application	18/08/20	20/08/20			
		Compare and analyse results	Graph comparisons, side-by-side result comparisons and explanations of what the results mean	20/08/20	22/08/20			
	Portfolio	Plan and outline conference style paper		01/08/20	03/08/20			
		Write conference style paper		10/08/20	17/08/20			
		Format conference style paper		17/08/20	18/08/20			
		Compile project design & implementation documentation	Put documentation of sections together and ensure all details are included and formatting is okay	20/08/20	22/08/20			
		Compile testing & results documentation		22/08/20	25/08/20			
		Include source code		25/08/20	25/08/20			
		Portfolio formatting		25/08/20	26/08/20			
		Seek feedback from supervisor		25/08/20	25/08/20			
		Review portfolio and update based on feedback		30/08/20	30/08/20			Depends on when previous task is sent & received
Milestone 5		Submit portfolio		31/08/20	31/08/20			

Updated Plan

There were a few significant changes made to the project plan. The main change was that the serialised implementation was removed from the project due to time constraints and the low priority of completing a third implementation of the cipher. The software implementation of the cipher was also not completed due to time constraints. Due to the main purpose of having a software implementation being for the purpose of comparison to the hardware implementation, a C implementation of the cipher was found online [9] and modified slightly to obtain results appropriate for comparison from it. There were also certain tasks which took longer than anticipated and I ran into problems during the process such as the development of the AXI peripheral. Many of the tasks were a bit optimistic and did not anticipate problems that may be run into. Finally, a 2 week period was used for a holiday during the July period which led to a lack of project work being done and overlapped with the development of the AXI peripheral. This should have been planned for better in advance.

The spreadsheet for the updated timeline can be found from here, where the timeline can be viewed clearly.

Milestones	Activity	Task	Description	Projected Start Date	Projected Finish Date	Start Date	Finish Date	Extra Details
	Literature Review	Research	Finding sources to be used for the literature review	15/01/20	16/01/20	14/01/20	16/01/20	
		Key Points	A summary of the key points that the literature review will address	17/01/20	17/01/20	16/01/20	16/01/20	
		Write Document		20/01/20	25/01/20	20/01/20	26/01/20	
		Formatting		26/01/20	26/01/20	26/01/20	26/01/20	
Milestone 1		Submission	Literature review submission	26/01/20	26/01/20	26/01/20	26/01/20	
	Presentation	Slide preparation	Creation of Powerpoint slides	10/02/20	14/02/20	10/02/20	14/02/20	
		Presentation preparation	Preparation of timing and what to address	18/02/20	19/02/20	18/02/20	19/02/20	
		Presentation Recording		20/02/20	20/02/20	24/02/20	24/02/20	Recording of presentation moved due to issues with set up in room
Milestone 2		Submission	Submission of slides and video presentation		24/02/20		24/02/20	
Milestone 3		Interview		06/04/20	06/04/20	06/04/20	06/04/20	
	Project Plan	Document Overview	Complete shorter sections of project plan	01/06/20	02/06/20	30/06/20	01/07/20	
		Design Approach	Planning of the design of various implementations & result measuring	02/06/20	04/06/20	02/06/20	04/06/20	
		Detailed Timeline	Plan for a realistic and streamlined completion of tasks and activities	05/06/20	10/06/20	06/06/20	12/06/20	Days missed during this period due to other commitments
		Complete Document	Fill in sections which need updating or have not been completed	11/06/20	14/06/20	13/06/20	14/06/20	
Milestone 4		Submission	Submission of project design plan		14/06/20	14/06/20	14/06/20	
	Implementation	Develop round-based cipher core		10/02/20	16/05/20	10/02/20	17/05/20	Issues with timing led to taking slightly longer
		Validate round-based cipher core		16/05/20	17/05/20	10/05/20	17/05/20	
		Develop AXI peripheral using Xilinx IP		18/05/20	30/05/20	18/05/20	09/07/20	This task was revisited as it was not implemented as intended originally
		Document cipher core implementation	Details on VHDL design of cipher core	29/05/20	01/06/20	27/05/20	30/05/20	
		Develop pipelined cipher core		01/06/20	20/06/20	25/05/20	05/06/20	Extra time given as it overlaps with project plan. Re-use of components sped up process
		Validate pipelined cipher core		20/06/20	21/06/20	05/06/20	06/06/20	
		Develop AXI peripheral using Xilinx IP		22/06/20	28/06/20	24/06/20	18/07/20	**Task removed from project**
		Document cipher core implementation	Details on VHDL design of cipher core	28/06/20	30/06/20	-	-	**Task removed from project**
		Develop serialised cipher core		30/06/20	15/07/20	-	-	**Task removed from project**
		Validate serialised cipher core		15/07/20	16/07/20	-	-	**Task removed from project**
		Develop AXI peripheral using Xilinx IP		16/07/20	20/07/20	-	-	**Task removed from project**
		Document cipher core implementation	Details on VHDL design of cipher core	21/07/20	22/07/20	-	-	**Task removed from project**
		Develop software drivers	Software applications to test and run full IP peripheral	23/07/20	23/07/20	25/07/20	28/07/20	Extra research was needed for this task
		Document full IP peripheral	Details on how it was implemented and how it functions	24/07/20	29/07/20	29/07/20	04/08/20	
		Develop software implementation		30/07/20	07/08/20	06/08/20	06/08/20	Full development of software not completed due to time constraints. Results found from version found online
		Document software implementation		08/08/20	09/08/20	08/08/20	08/08/20	
		Debugging/Improvements	Clean up code/Minor improvements in presentation and running of code	10/08/20	12/08/20	10/08/20	12/08/20	
	Testing and Analysis	Validate round-based cipher core		16/05/20	17/05/20	10/05/20	17/05/20	
		Validate pipelined cipher core		20/06/20	21/06/20	05/06/20	06/06/20	
		Validate serialised cipher core		15/07/20	16/07/20	-	-	**Task removed from project**
		Validate IP peripherals		12/08/20	15/08/20	12/08/20	15/08/20	
		Develop timing constraints		14/08/20	16/08/20	16/08/20	16/08/20	
		Obtain resource utilisation reports	Reports for power consumption (mw) and area utilisation (LUTs)	16/08/20	18/08/20	16/08/20	18/08/20	
		Develop test application	Software application	18/08/20	20/08/20	18/08/20	20/08/20	Testing not done in single application
		Compare and analyse results	Graph comparisons, side-by-side result comparisons and explanations of what the results mean	20/08/20	22/08/20	21/08/20	23/08/20	
	Portfolio	Plan and outline conference style paper		01/08/20	03/08/20	10/08/20	11/08/20	
		Write conference style paper		10/08/20	17/08/20	21/08/20	25/08/20	
		Format conference style paper		17/08/20	18/08/20	26/08/20	26/08/20	
		Compile Literature Review in portfolio		18/08/20	19/08/20	23/08/20	23/08/20	
		Update project plan for portfolio		19/08/20	19/08/20	29/08/20	29/08/20	
		Finish and compile research log for portfolio		19/08/20	20/08/20	27/08/20	28/08/20	Last entries and tidying it up for portfolio
		Compile project design & implementation documentation	Put documentation of sections together and ensure all details are included and formatting is okay	20/08/20	22/08/20	22/08/20	24/08/20	
		Compile testing & results documentation		22/08/20	25/08/20	20/08/20	24/08/20	
		Include source code		25/08/20	25/08/20	29/08/20	30/08/20	
		Portfolio formatting		25/08/20	26/08/20	20/08/20	31/08/20	
		Seek feedback from supervisor		25/08/20	25/08/20	26/08/20	26/08/20	
		Review portfolio and update based on feedback		30/08/20	30/08/20	29/08/20	30/08/20	Depends on when previous task is sent & received
Milestone 5		Submit portfolio		31/08/20	31/08/20	31/08/20	31/08/20	

Success Criteria

This project will be considered successful if the following criteria are met:

- Completion of the literature review
- Completion of the presentation
- Completion of the project plan
- Completion of at least one hardware design of the cipher
- Successful validation of the cipher using a testbench
- Completion of a software design of the cipher
- Completion of a full peripheral implementation
- Successful development of software driver for AXI peripheral
- Successful gathering of results that can be compared for the different implementations
- Completion of the research paper
- Completion of the portfolio

Appendix C
Research Log

Preface

For many of the papers that have been chosen in the research log, they include proposed architectures or descriptions of architectures for the PRESENT cipher. Some of these architectures were used for the design of the cipher while others were considered during the research stage of the project. These papers were selected as they had realistic architectures that could be built without too much difficulty. There are also many papers that contain relevant results that can be used for the purpose of comparison for the cipher.

The project planning stage was clearer due to the papers that were read in the research stage of the project. It was clear all the steps that needed to be taken to get to the desired outcome of the project by seeing other people who have done similar. This made the outlining of tasks more defined which allows for better planning.

Compiling results for different hardware and software implementations of the cipher that are mainly targeting an FPGA is very useful and helps to analyse whether the performance of my implementation is good or even if these metrics are being measured correctly. Values are given in many papers for the area utilisation, power consumption, latency and throughput of the cipher. The energy usage of the cipher is also another metric that features regularly in papers when measuring the resource utilisation of the cipher. Results could be compared to some of the results published, however, some of the papers contained different measurements for the results that could not be compared to the results obtained, such as gate equivalents (GE) for the area utilisation whereas my results measured the area in slice utilisation.

The cipher specification is also included in the research log which was immediately clear to be one of the most important papers when designing this project. This was used along with some other descriptions of the cipher to originally implement the cipher. The specification also contains the test vectors needed for the cipher to validate its functionality which is imperative.

The continued reading of literature throughout the project ensured that it was very clear what I was designing and implementing. Reading many different possibilities to implement the project gave a wide range of options for designs that could be implemented as well as making me aware of methods that can be used in certain parts of the cipher for its implementation.

Masters Project Research Log

Masters in Electronic and Computer Engineering 2019/2020

Student Name: Alex Doherty
Student ID: 15508783
Project Title: IP core development of PRESENT-80 lightweight cipher

Statement of project problem / research question (maximum 200 words)

This project will involve the design and implementation of the 80-bit key version of the PRESENT lightweight block cipher. This is a hardware optimised cipher, so it will be implemented through hardware. The RTL design will be packaged as an IP and implemented in an AXI4 peripheral that will be designed to achieve hardware acceleration of the cipher. Once the creation of the AXI peripheral is complete, a software driver will be designed to invoke the encryption through the hardware. Results will be obtained for the resource utilisation and performance of the cipher. The resource utilisation incorporates area utilisation and power consumption while the performance is measured by the latency and throughput of the cipher. Similar results for the performance will also be obtained for a software implementation of the cipher so that the performance of the cipher on hardware and software can be compared and analysed. The cipher implementation should be investigated to see if the lightweight aspect of the cipher has been achieved through low resource usage as well as investigating the hardware optimisation of the cipher to see if the results show an improvement in performance through hardware.

PRESENT: An Ultra-Lightweight Block Cipher
A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann and e. al, "PRESENT: An UltraLightweight Block Cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, Berlin, Springer Berlin Heidelberg, 2007, pp. 450466.
http://www.lightweightcrypto.org/present/present_ches2007.pdf
Summary of paper (maximum 100 words)
This paper was released by the authors of the PRESENT lightweight cipher specification. The paper proposes the hardware-optimised block cipher and underlines its focus on power and area constraints. A survey of the existing literature is included at the beginning of the paper, which addresses the performance metrics of similar existing lightweight ciphers. This is followed by a description of the cipher (including its encryption process), a security analysis section and a performance analysis section.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper gives a description of the PRESENT algorithm, so it is very helpful to use for the implementation of its hardware design in HDL or its software design in a language such as C++. The cipher is described in pseudo code as well as there being a section to describe each stage in the algorithm process, along with accompanying diagrams to help describe and visualise the process.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
This paper addresses a round-based implementation of the PRESENT cipher. This implementation focuses on area and power constraints. There is an SP network and 31 rounds, as well as a separate key schedule to provide round keys for the encryption process. While this implementation demonstrates good results for area utilisation and power consumption, the cipher can also be designed for other metrics such as an improved throughput. A benefit of the design is its simplicity, however, this paper takes into account the applications it will likely be used in and suitably designed it.

Lightweight Hardware Architectures for the Present Cipher in FPGA
C. A. Lara-Nino, A. Diaz-Perez and M. MoralesSandoval, "Lightweight Hardware Architectures for the Present Cipher in FPGA," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2544-2555, 2017.
https://ieeexplore.ieee.org/document/7911280
Summary of paper (maximum 100 words)
This paper addresses the most representative designs for this cipher as well as presenting two alternative implementations for the cipher. The proposals for these designs are presented, evaluated, and compared using area, performance, energy and efficiency as the metrics for comparison. The methods for achieving these results are shown as well as the platform performed on. The advantages of the different implementations of the cipher are discussed as well as how these benefits to each design are achieved. There are five different architectures discussed in this paper in total.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper gives a description of different implementations PRESENT algorithm, so it can be helpful to use for the implementation of its hardware design in HDL or its software design in a language such as C++. The iterative and serialised implementations of the cipher described in this paper show alternative methods of designing the cipher, which have different benefits to them when compared to the low-area architecture described in the original algorithm specification.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
The 16-bit architecture shown in this paper has benefits attributed to it such as a reduction in the latency of the system, due to the key not being entered into the circuit so its clock cycles are avoided. Another benefit with this design is that no extra registers are needed to store the key as it can be read from the key space. The serial architecture has a low area utilisation but this is at the sacrifice of a significant increase in the latency of the system.

Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents
Rolfes, Carsten, et al. “Ultra-lightweight implementations for smart devices - security for 1000 gate equivalents.” International Conference on Smart Card Research and Advanced Applications. Springer, Berlin, Heidelberg, 2008.
https://link.springer.com/chapter/10.1007/978-3-540-85893-5_7
Summary of paper (maximum 100 words)
This is another paper which addresses different implementations of the PRESENT block cipher. Results are found for the different implementations and the suitability of the architectures for active and passive smart devices are discussed. The architectures that are outlined in this paper are a round-based architecture, a parallel architecture, and a serialised architecture. The results of the architectures are displayed and compared to some other similar lightweight block ciphers that can be applied to smart devices.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper has similar advantages to the previous papers in the research log as it details different designs in which the PRESENT cipher can be implemented. There is a good description of each of the different designs as well as a data-path block diagram of each architecture, which helps to understand the design significantly.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
The first solution presented in this paper is the round-based architecture that was originally shown in the algorithm specification. This design has high energy efficiency, low area utilisation and low power. The parallel architecture, unlike the previous solution, will lead to a high area utilisation and power consumption. The advantage of this architecture is a high throughput. Finally, the serialised architecture has the advantage of a lower chip area through the reduction in s-boxes by using a 16-bit data-path for the plaintext. The sacrifice made for saving area is an increase in the computation time.

A Survey on Lightweight Block Ciphers for Low-Resource Devices: Comparative Study and Open Issues

B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, 'A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues', Journal of Network and Computer Applications, vol. 58, pp. 73–93, Dec. 2015, doi: 10.1016/j.jnca.2015.09.001.

https://www.researchgate.net/publication/282576166_A_Survey_on_Lightweight_Block_Ciphers_for_Low-Resource_Devices_Comparative_Study_and_Open_Issues

Summary of paper (maximum 100 words)

This paper investigates different lightweight block ciphers which have had a lot of research into them due to their important role in low-resource device security. The paper reviews the current progress that has been made in the lightweight block cipher area along with the direction research could go towards in the future. There was also a focus on the metrics used to compare and analyse these ciphers, as well as an examination on the source of inaccuracies and deviations in reported results.

How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)

This paper looks at a range of lightweight block ciphers, which are similar to PRESENT, as well as PRESENT being mentioned as a good reference for hardware ciphers. Methods of obtaining the results for comparing these ciphers are described, as well as formulas being given, which is useful for the analysis of the performance of the cipher. This paper can also be used for a literature review of the lightweight block ciphers, as it has compiled a good list of results from ciphers that are suitable to compare to the PRESENT cipher.

What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)

This paper has broken down the comparison of ciphers into software and hardware implementations, so that a fair comparison can be made to similar implementations. A list is given of the advantages of many of the different ciphers, for example, PRESENT ranks as the cipher with the least RAM memory and second in code size out of all the ciphers that were researched and evaluated in this paper. The typical cipher design flow diagram is also useful for getting a basic idea of the order of processes in implementing the cipher.

Novel FPGA-Based Low-Cost Hardware Architecture for the PRESENT Block Cipher
C. A. Lara-Nino, M. Morales -Sandoval and A. Diaz -Perez, "Novel FPGA -Based Low -Cost Hardware Architecture for the PRESENT Block Cipher," in 2016 Euromicro Conference on Digital System Design (DSD), 2016.
https://ieeexplore.ieee.org/document/7723612
Summary of paper (maximum 100 words)
This paper presents an FPGA-based implementations of the PRESENT block cipher, along with the results they presented. The target of one of the architectures being presented is a low-area hardware implementation. The architecture presented uses a 16-bit data path as well as four 16-bit full registers. This varies from the presented design in the algorithm specification. The cipher implementation is measured and compared to other current designs that have been reported on in papers. Throughput, latency, and area utilisation are the main metrics for evaluation of performance, resource utilisation and comparison.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper is relevant to my project as it presents alternative methods of implementing the PRESENT lightweight block cipher. Different architectures for this cipher have different benefits to them, so it is useful to be aware of how you could design your implementation so that it performs well in particular metrics. This paper also useful to show the comparison of results between different PRESENT architectures, which can be used as a target for my project to achieve similar results, depending on the focus of the implementation.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
This change in design shown in the presented architecture leads to a reduction in the latency and area utilisation of the design. The reduced data path is also suitable for platforms which have a lower amount of ports. The designs, however, yielded lower results in terms of its throughput when compared to previous implementations of the cipher that are compared to it. This was not the focus of these implementations so it is not a major downside.

Lightweight Cryptography: Cryptographic Engineering for a Pervasive World
A. Y. Poschmann, Lightweight Cryptography: Cryptographic Engineering for a Pervasive World, 2009.
https://www.researchgate.net/publication/220335292_Lightweight_Cryptography_-_Cryptographic_Engineering_for_a_Pervasive_World
Summary of paper (maximum 100 words)
This paper focuses on hardware lightweight ciphers with a low area utilisation. The writer of this paper was involved in the design of the PRESENT cipher, following a review of the DES cipher which is deemed as a cipher with too high area requirements. Results for PRESENT are presented using different design strategies and design platforms. There is a discussion on the area utilisation of block ciphers, and in particular the PRESENT cipher designed as it is compared to the area utilisation of stream ciphers.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper is highly relevant to this project as it was written by one of the designers of the cipher. It discusses encryption routine, with a section dedicated to each of the processes involved in each round of the encryption process. Similarly, the key schedule which provides the keys in each round of the encryption process is explained in detail, with visualisations of this process provided to aid this description for both the 80-bit and 128-bit key versions of the cipher.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
This paper looks at the DES encryption cipher and concludes that a lightweight cipher that can perform similarly but with more of a focus on low area utilisation is required. For this reason, the PRESENT cipher is designed, which yields minimal area footprint, while displaying a strong security when cryptanalysis was performed on the cipher. Reading the details of the creation of the cipher, its design and focus as well as algorithm descriptions provided is beneficial when coming from a designer of the cipher and will be a great help in understanding and implementing the cipher.

A Survey of Lightweight-Cryptography Implementations
T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, 'A Survey of Lightweight-Cryptography Implementations', IEEE Design Test of Computers, vol. 24, no. 6, pp. 522–533, Nov. 2007, doi: 10.1109/MDT.2007.178.
https://www.researchgate.net/publication/3251519_A_Survey_of_Lightweight-Cryptography_Implementations
Summary of paper (maximum 100 words)
This paper surveys a number of lightweight ciphers that could be used for tight-cost and low-resource devices such as RFID tags and smart cards. Different lightweight ciphers have been presented with their results compared to the state of the art results in the field at the time of publishing the paper. There is a focus on the description of PRESENT in this paper, due to contributors from two of the designers of the cipher and its recent release at the time of publish. Both hardware and software implementations of the ciphers are discussed and compared.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper is very useful for enhancing my understanding of the PRESENT cipher. A further understanding of the cipher will help with the implementation of the cipher, particularly for the HDL design of the cipher. The insight from the designers of the cipher is useful. It is also useful to use the results presented in this paper as a benchmark for comparison when results for the cipher have been obtained for this project.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
A detailed description of the cipher has been given, with details given on the reasoning behind some of the decisions made in the design of the cipher. The insight from the designers of the cipher is useful. This paper only described the basic encryption process behind the PRESENT cipher rather than suggesting any alternative implementations.

RAM-Based Ultra-Lightweight FPGA Implementation of PRESENT’
E. B. Kavun and T. Yalcin, ‘RAM-Based Ultra-Lightweight FPGA Implementation of PRESENT’, in *2011 International Conference on Reconfigurable Computing and FPGAs*, Nov. 2011, pp. 280–285, doi: 10.1109/ReConFig.2011.74.
https://ieeexplore.ieee.org/abstract/document/6128590
Summary of paper (maximum 100 words)
This paper presents two different implementations of the PRESENT cipher. These implementations both use RAM blocks in the FPGA for storage to reduce the area utilisation of the designs. The algorithm for the cipher is summarised and then the designs proposed are described in detail. The results are presented for these implementations with a short discussion of the results and how these implementations could be useful in application.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper addresses alternative methods of implementing the cipher with an FPGA that could result in an improvement in the results for the designs implemented. This is highly relevant to this project which will be implemented on an FPGA for hardware acceleration of the cipher. Block diagrams have been provided to aid the understanding of these implementations as well as results that show the impact these designs have had on the cipher.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
These designs use block RAM for storing the state of the data being encrypted as well as the key data instead of registers. This leads to a smaller slice count for the design. The two designs split in their methods here, where one implements the substitution boxes on slices and the other in RAM as a look-up table. The second design will give a lower slice count but an increase in the cycle count due to the extra control logic needed.

Secure Video Streaming with Lightweight Cipher PRESENT in an SDN Testbed'
X. Wang, P. Liu, S. R. Chaudhry, M. Collier, K. Javeed, and M. Yue, 'Secure Video Streaming with Lightweight Cipher PRESENT in an SDN Testbed', *CMC*, vol. 57, no. 3, pp. 353–363, Jan. 2018, doi: 10.32604/cmc.2018.04142.
https://www.researchgate.net/publication/329908601_Secure_Video_Streaming_with_Lightweight_Cipher_PRESENT_in_an_SDN_Testbed
Summary of paper (maximum 100 words)
This paper looks at the security and resource limitations of traditional processors and FPGAs. An efficient design of the PRESENT cipher is implemented for good performance as well as low power consumption. This design is created using Verilog, with a software design also created in C. The performance of hardware and software implementations of the cipher are evaluated with results to show that the hardware architecture proposed is suitable for use in constrained devices.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper is relevant as it implements the PRESENT cipher that is the target cipher of this project. Results for the hardware and software implementation are presented as well as results for its usage in secure video streaming, which shows an application for the cipher. Low latency is important for this design as it is used for video streaming.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
A performance comparison between the software and hardware implementations are shown in this paper. Through the Verilog design the FPGA implementation proved to be more efficient and faster while using fewer resources than the software implementation. One round per clock cycle is also used for this implementation for the encryption and decryption processes. There is also a significant drop in the RAM usage in the hardware implementation.

High-throughput and lightweight hardware structures of HIGHT and PRESENT block ciphers
B. Rashidi, 'High-throughput and lightweight hardware structures of HIGHT and PRESENT block ciphers', *Microelectronics Journal*, vol. 90, pp. 232–252, Aug. 2019, doi: 10.1016/j.mejo.2019.06.012.
https://www.sciencedirect.com/science/article/pii/S0026269218304154
Summary of paper (maximum 100 words)
This paper looks at both the HIGHT and PRESENT ciphers with a proposed alternative substitution-box structure for the PRESENT cipher. This paper focuses on efficient hardware implementations of these ciphers which are both low-cost ciphers. The PRESENT and HIGHT cipher algorithms have been outlined, with details on the changes made to the PRESENT encryption process for the proposed architecture. There is a security analysis followed by results and analysis for both block ciphers.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper proposes an alternative architecture for implementation of the PRESENT cipher. This architecture mainly changes from the basic implementation in its implementation of the substitution-boxes. Simplifications are made to the expression terms for the substitution-boxes to reduce the number of logic gates needed. This is another method of design for the cipher which can be considered. It has different benefits to many of the previous cipher designs mentioned previously.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
The implementation presented is an efficient implementation of the cipher. The results show a high throughput and a low computation time through the use of a loop unrolling technique mentioned in the paper. The hardware consumption has shown acceptable results also when comparing to other related works. The s-boxes achieve a low-cost design for the algorithm with their alternative implementation. The throughput/area results also perform well when compared to other related works.

Compact FPGA implementation of PRESENT with Boolean S-Box
J. J. Tay, M. L. D. Wong, M. M. Wong, C. Zhang, and I. Hijazin, 'Compact FPGA implementation of PRESENT with Boolean S-Box', in *2015 6th Asia Symposium on Quality Electronic Design (ASQED)*, Aug. 2015, pp. 144–148, doi: 10.1109/ACQED.2015.7274024.
https://ieeexplore.ieee.org/abstract/document/7274024
Summary of paper (maximum 100 words)
This paper provides a hardware implementation of the PRESENT cipher on an FPGA. A decision made for the design of this architecture was to implement the substitution box of the cipher using logic circuits and consisting of AND and OR gates instead of look-up tables or block RAMs to implement it. The details of this methodology are explained in the paper along with results and comparison of these results to related works.
How is this paper relevant to solving your project problem or addressing your research question? (maximum 100 words)
This paper is relevant for this project as it is another paper that proposes an alternative architecture with its own advantages to it that can be considered. The design of this architecture is targeted for an FPGA in this paper similarly to the implementation process for this project. The results are presented with a comparison to related works so this can be used to decide if this implementation should be further pursued.
What are the strengths and weaknesses of the solutions/methods/technologies proposed in this paper? (maximum 100 words)
This paper states that as far as they know, their implementation (published in 2015) requires the least amount of FPGA slices when compared to related works. The number of substitution boxes needed in the implementation is reduced by using an 8-bit datapath and using two counters for control logic. The substitution box area was reduced by simple logic using Karnaugh maps and further factorisation, which allows for the Boolean substitution box that results to be used for pipelining.

Appendix D
Project Design & Implementation

Tools & Development Flow

Tools

Xilinx Vivado

Vivado is an Integrated Development Environment (IDE) designed by Xilinx for building, creating and synthesizing HDL projects. These projects can then be implemented on FPGA boards. Vivado Design Suite is the main software that is required to build this project.

Xilinx SDK

Xilinx SDK (software development kit) is the environment which can be used to create applications for Xilinx embedded processors, such as the Xilinx Zynq. Xilinx SDK comes with many features that can be used for development.

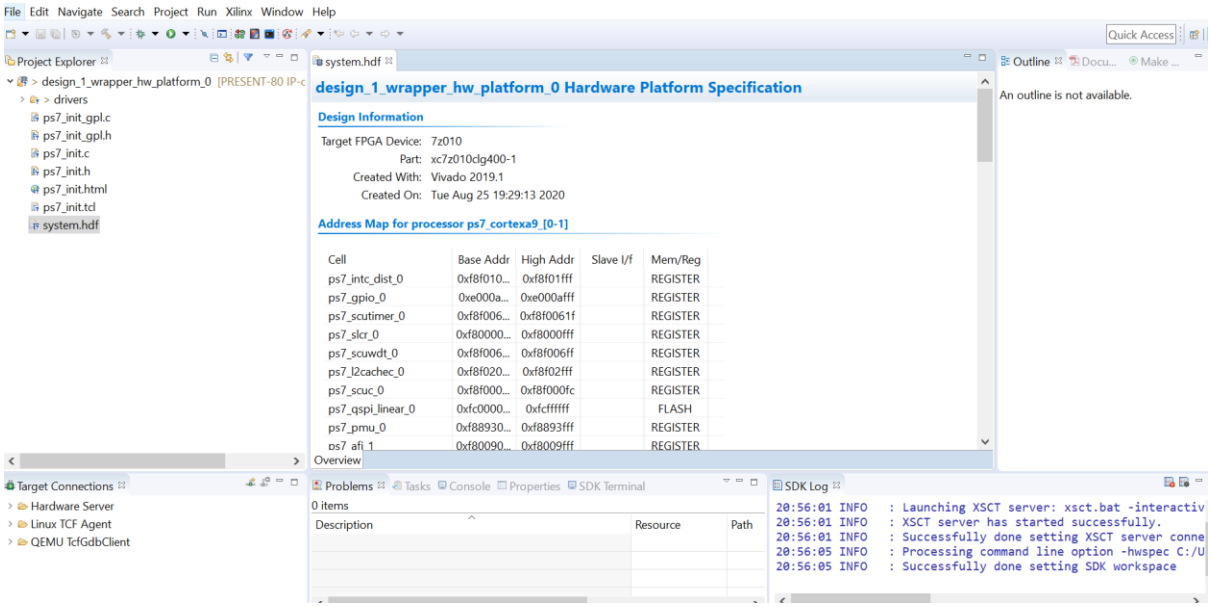

Figure 1: Xilinx SDK IDE

The project explorer will display the files such as the source and header files in a tree format. This allows for easy navigation and visualisation of the hierarchy and structure in the directory. There is a text editor which can be used for the writing and editing of software code. This has support for C and C++ syntax. The code that has been developed can be compiled using the GCC compiler provided, which is suitable for compilation of both C and C++ code. There are different perspectives available in the SDK, which serve a different purpose. There is the development perspective, which is suitable for writing and editing of code as well as a debug perspective, which can be used for stepping through individual lines of code, setting breakpoints and deducing the function of the program.

The FPGA can be programmed from the Xilinx SDK, so that the hardware and software configuration can be uploaded and run on the board. The serial console can be setup at the bottom of the SDK environment to communicate with the serial interface of the board. This can be used for printing out messages, which can be useful for debugging and verification of the program.

Zybo Board

The board that is being used for this project is the Digilent Zybo development board. The Zybo board contains a dual core ARM Cortex-A9 processor as well as a 7-series FPGA board.

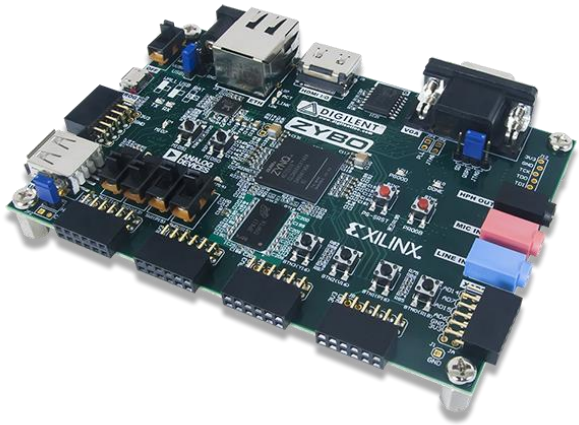

Figure 2: Digilent Zybo development board

Some of the features this board consists of includes:

- 650Mhz dual-core Cortex-A9 processor
- Zynq-7000 FPGA
- 512MB DDR3 memory
- 128Mb Serial Flash with SPI interface
- On-board JTAG programming and a UART to USB converter

The Zynq All Programmable System-on-chip (AP SoC) is divided into two subsystems, the Processing System (PS) and the Programmable Logic (PL). The Zynq AP SoC architecture can be seen in Figure 3*:* below, with the PS highlighted in green and the PL highlighted in yellow. The peripheral controllers are connected to the processors as slaves and contain readable/writable registers that are addressable in the processor's memory. The PL is connected to the interconnect as a slave. Designs can make use of multiple cores in the FPGA fabric that contain addressable control registers. [19]

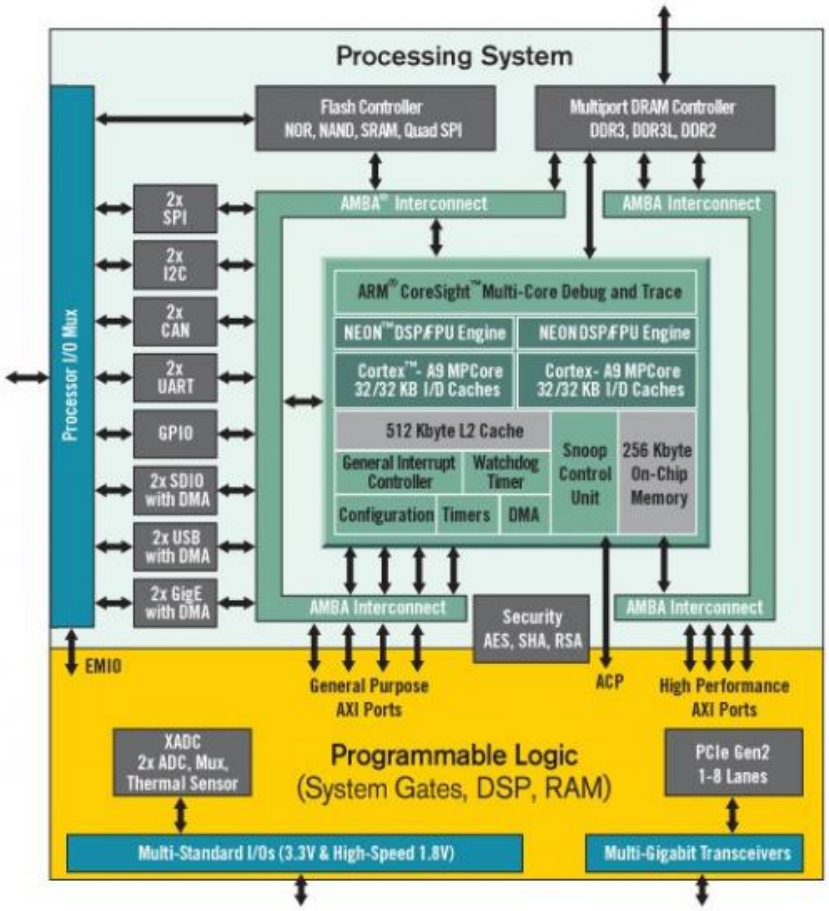

Figure 3: Zynq Ap SoC architecture

Development Flow

There are a number of steps involved in the development flow for implementing a hardware design on an FPGA. These steps are outlined in the below headings.

Step 1: Design Creation

The first step is design creation, which involves creating a project in the integrated synthesis environment (ISE). The various files needed for the design of the project can be created here and user logic can be added to these files using HDL. These files can interact with and use each other in the design by modularising the system and breaking it up into individual components. The input and output ports of these components can be used for the interaction between files. The register-transfer level (RTL) development is done in Vivado using the text editor.

Step 2: Functional Verification

This next step in the process is vital to the development process for the cipher. Running a behavioural simulation in Vivado can allow for the debugging and verification of the design being created. To do this, testcases need to be provided in a testbench to drive the simulation. This can allow you to observe and analyse the response of the design created to the stimulus input to it. A waveform viewer will also be displayed upon simulation of the design, which can help with analysis of the signals and the overall design.

Step 3: Synthesis

The RTL source code and the physical design constraints are transformed into a gate-level representation using the synthesis process. Once the design has been synthesised, you can analyse the results by opening reports that have been generated and use the synthesised design.

Step 4: Implementation

Once synthesis is complete, an implementation of the design can be created. The implementation will place and route the netlist onto device resources, within the logical, physical and timing constraints of the design. A high level design flow has been provided in the Vivado Implementation Guide documentation that can be seen in Figure 4 below.

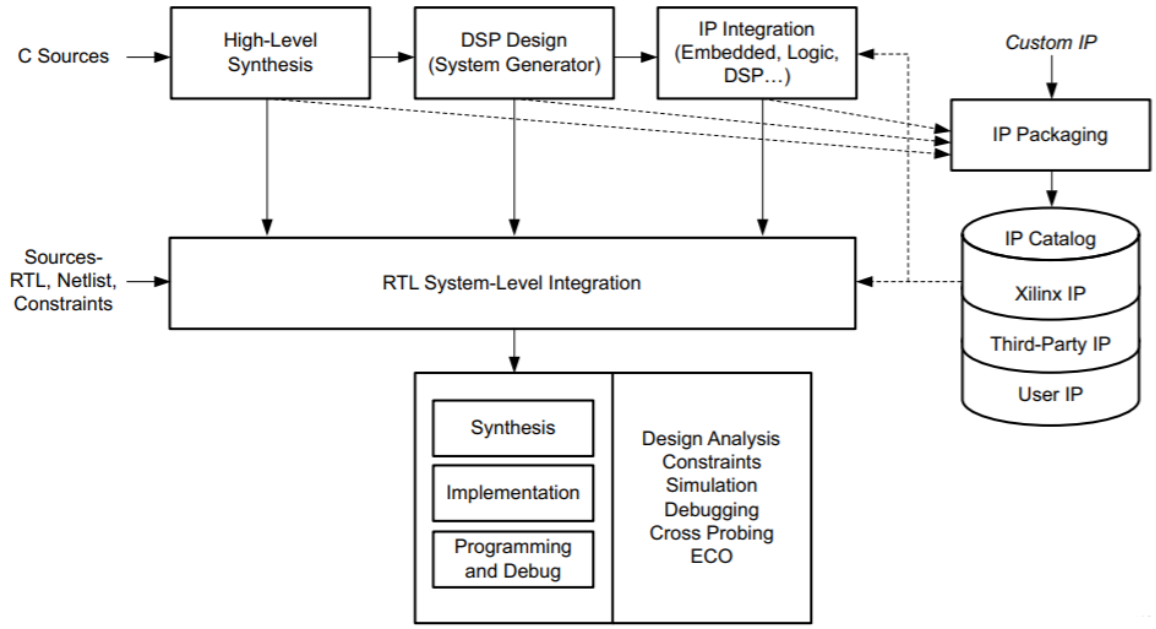

Figure 4: High level design flow [19]

There is a floorplan viewer available which allows you to manually place blocks of logic in an FPGA and select the appropriate connectivity of logic for your design. Floorplanning can increase performance as well as reduce route delays.

Step 5: IP Packaging & Integration

The IP packager tool can be used for packaging the IP. The design can be packaged for import into the IP integrator tool. This will then allow for the IP to be available for selection in the Vivado IP catalogue. A GUI block designer view is provided in the Vivado toolset that can be used to connect the Zynq processing system with the packaged IP through an AXI interconnect.

Step 6: Hardware Exportation

The hardware design can be exported to the SDK for a software application to be created. The hardware and software can be uploaded to the board from here.

Step 7: Bitstream Generation

The bitstream is a file which contains programming information for an FPGA.

Step 8: Programming FPGA

The generated bitstream file can now be loaded onto the FPGA.

AXI Peripheral

The AXI peripheral is designed as a communication interface, with a range of features such as address/control and data phases and separate and independent read and write channels. The AXI4-Lite is a subset of the AXI protocol and will be the focus of this project. The AXI4-Lite contains a register-like structure and data accesses use the full data bus width of either 32 or 64 bits.

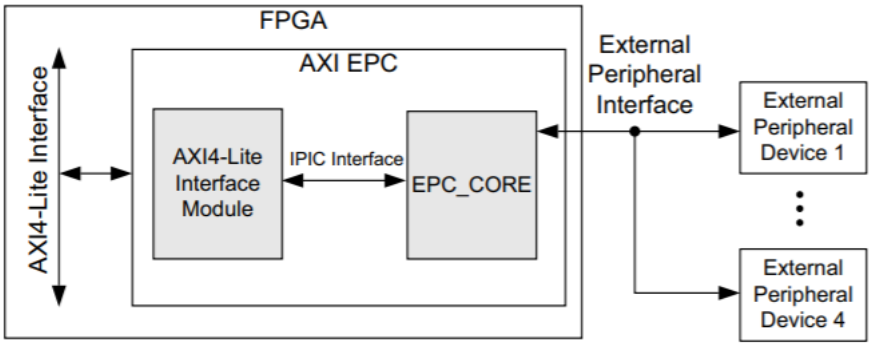

Figure 5: AXI peripheral controller interface *[20]*

Figure 5 gives an example of an AXI External Peripheral Controller interface. While this is not exactly the design that will be used for this project, it is useful to understand the general design and connections between devices. [20]

Block RAM

Block RAM (BRAM) is one of the key components in the design of the project to run the cipher on an FPGA using an AXI peripheral. BRAM is used to store lots of data, which can be useful to store the input or output of the cipher as a lot of data may need to be encrypted through the cipher. BRAM allows you to store data internally on the FPGA, rather than externally through an SD card or another device used for storage. BRAM can come in a single port or dual port configuration, however, both configurations act similarly. The difference is that dual port BRAM has a second port available for reading and writing data. This can be seen in Figure 6 below. [21]

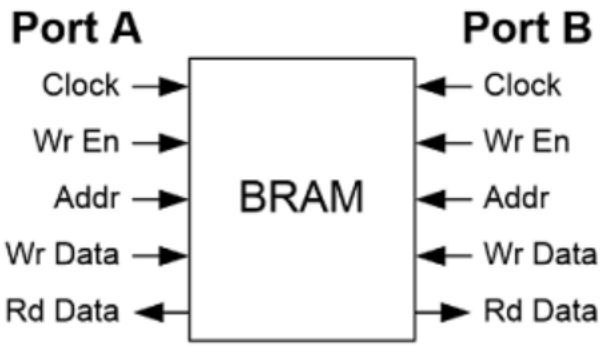

Figure 6: Dual port BRAM [21]

Design & Implementation

Preface

I would like to thank and give credit to Michael Morgan for his creation of a guide for the creation of a full AXI peripheral for hardware acceleration of a cipher. This allowed for the research and implementation process for this section of the development go smoothly, as the general method described was useful for guidance when creating the AXI peripheral for this project.

Introduction

This section will detail the different designs that have been implemented for this project, explaining how the design works and how it was developed. First, there was the hardware development which began with the RTL development in VHDL. This was used to design the round-based core and the pipelined core.

Hardware Development

Introduction

This section will walk through the steps taken to develop the hardware side of this project. General steps will be laid out to complete this process, which can be applied to your project, with suitable steps taken to allow it to suitably apply to the project being developed. Following these steps will allow you to develop RTL design of a system, package it as an IP and connect it to a Zynq PS for use of the hardware to run the project. Following the RTL development of the ciphers, the round-based core is packaged as an IP and an AXI peripheral is created. This process is described with all the components involved to create the full connectivity needed for hardware acceleration of the cipher.

Round Based Implementation

This implementation has been designed with area and power constraints in mind, as was intended with the design of the cipher. This design intends to reduce area and power utilisation without a compromise in the necessary security required for applications in which lightweight ciphers will be used.

This implementation of the PRESENT-80 cipher core contains five main VHDL files that implement the design. Modular, reusable components are created for this design, which allows them to be re-used in other implementations of the cipher. Modular components are created for the key schedule, the substitution-box(s-box) layer, the s-box and the permutation layer(p-layer). This also allows for components, such as the substitution box, to be generated using the generate statement in VHDL.

The PRESENT cipher uses a substitution-permutation network (SPN) to encrypt the text. SPNs are commonly used in block ciphers for a strong level of encryption. SPNs are suitable for low-area designs due to the shift and permutation operations having extremely low area utilisation. The reason for this is that the substitution simply substitutes the block of bits with another block of bits while permutation rearranges or permutates the bits following the substitution.

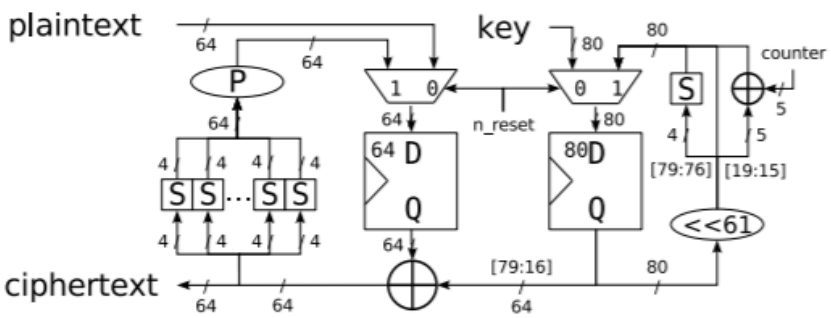

Figure 7: Datapath of round-based PRESENT

This round-based architecture is based off the design that is described in the algorithm specification. [8] The state of the encrypted text and the key are two elements which use memory in the design.

The key schedule runs to provide round keys for each round in the encryption process. This process uses XOR and substitution. The key schedule module stores the round keys in a key register. A component of the s-box is contained within the key schedule, where the keys are passed in 4 bits at a time. The key register is rotated by 61 bits to the left. The leftmost 4 bits of the rotated key are passed into the s-box and the value of the round counter is XORed with bits of the key register. This process can be seen in Figure 8.

```
24      process(keyIn, rotated_key)
25          begin
26          rotated_key <= keyIn(18 downto 0) & keyIn(79 downto 19);
27
28          keyOut(19 downto 15) <= rotated_key(19 downto 15) XOR round_counter;
29          keyOut(75 downto 20) <= rotated_key(75 downto 20);
30          keyOut(14 downto 0) <= rotated_key(14 downto 0);
31          end process;
32
33           SB: sbox
34           Port map (
35                     dataIn => rotated_key(79 downto 76),
36                     dataOut => keyOut(79 downto 76)
37                     );
```

Figure 8: Key schedule logic

The s-box uses simple assignment logic to mix up the data coming in by following the key provided from the algorithm specification that is shown in Table 1.

Table 1: Substitution Box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

The SPN is responsible for the encryption of the text through the initial mixing of a round key with the state using the XOR operation, followed by the substitution using sixteen parallel 4-bit blocks. The permutation is subsequently performed for the 64 individual bits. The permutation operation, similar to the substitution, was achieved by using individual signal assignment operations. There is likely a formula to perform this operation in a loop, as there seems to be a pattern, for reduced lines of code in the permutation layer module, however, this would not have any significant effect on the system and I was not able to figure one out so it was left as the individual assignments.

Following the 31 rounds of encryption, a final mixing of a round key is performed using the XOR operation. This process is known as key whitening. The top file in this design is used to connect the whole encryption process together. It does this by connecting the outputs and inputs to each other, where appropriate. It also is used to reset the values when the reset value is set and set the output cipher text when the encryption is complete. A load input value is used to tell the cipher core when to load the plaintext and key values for the 31 rounds of encryption. Finally, the round counter is also used in this module to indicate the round of encryption. This value is passed into components, such as the key schedule. The ciphertext is set when the round counter reaches 0, which is after its final round of encryption, so that the value of the state of the text can be fully updated upon the clock cycle.

```
76      text_rKey_added <= inText XOR key_s(79 downto 16);
77
78      process( clock)
79      begin
80          if rising_edge(clock) then
81              if reset='1' then
82                  round_counter <= "00001";
83                  inText <= textIn;
84                  key_s <= key;
85                  textOut <= (others => '0');
86              elsif load = '1' then
87                  inText <= next_pLayer_state;
88                  key_s <= next_key;
89                  round_counter <= std_logic_vector(unsigned(round_counter)+1);
90
91                   case round_counter is
92                      when "00000" => textOut <= text_rKey_added;
93                      when others => textOut <= (others => '0');
94                  end case;
95              end if;
```

Figure 9: Top level file for round-based cipher

Pipelined Implementation

This implementation of the cipher focuses on a higher throughput. The higher throughput comes with a sacrifice as it leads to a high area utilisation and power consumption. This implementation [7] contains 32 XORs, 496 S-boxes and 31 p-layers with 31 s-boxes and 31 XORS for the key scheduling.

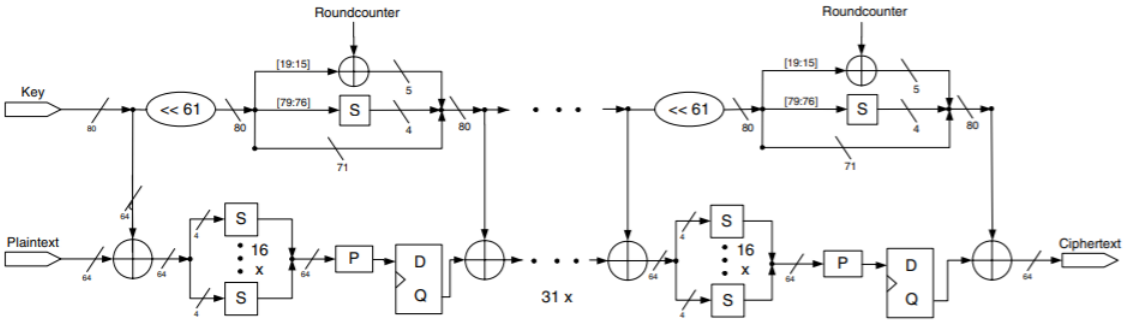

Figure 10: Datapath for pipelined cipher

This implementation creates 31 units for each round of encryption that are cascaded. This is shown in Figure 10 where the first and final rounds of encryption are shown, with the middle rounds representing a similar process but using the output of the previous round as the input to the current round. This cascaded architecture leads to greater area utilisation, but it also allows for a significant increase to the system throughput. The project structure for this implementation is similar to the structure for the round-based implementation, using modular components that can be reused for each round of encryption.

The same files can be used for the key schedule, the s-box, the s-box layer and the p-layer, but there will be a pipe generated for each of the 30 stages of the pipeline. A pipeline stage file is used to control the functionality in each of these stages.

The pipeline stage has similar functionality to the top-level file in the round based core, however, in the round-based core, the 64-bit input accepts the plaintext and the 64-bit output gives the ciphertext. A pipe stage is generated for each of the 31 rounds, so it will run 31 times to account for each round of the encryption process. In this pipeline stage, the 64-bit input accepts the current state of the plaintext after the number of rounds of encryption it has been through and the output gives the state of the plaintext after going through the round of the encryption that the pipe represents. There is no need for a round counter in this pipeline module as it only accounts for one round at a time. The main process contained in this module is shown in Figure 11.

```
77      RoundEncryption: PROCESS (clock)
78      BEGIN
79          IF rising_edge(clock) THEN
80              IF reset = '1' THEN
81                  round_counter <= roundCounter;
82                  inText <= textIn;
83                  textOut <= (OTHERS => '0');
84                  key_s <= keyIn;
85              ELSE
86                  inText <= textIn;
87                  key_s <= keyIn;
88                  textOut <= next_pLayer_state;
89              END IF;
90          END IF;
91      END PROCESS;
```

Figure 11: Encryption process in pipeline stage

The top-level module is used as a controller for the usage of the components. Each stage of the pipeline is instantiated in a for- generate loop, by creating a port mapping which passes in signals from a vector. This is shown in Figure 12.

```
37  pipe: for i in 0 to 30 generate
38      pipe_stage : Pipeline_stage
39      PORT MAP(
40          textIn => state_s(i),
41          textOut => state_s(i+1),
42          keyIn => key_s(i),
43          keyOut => key_s(i+1),
44          roundCounter => std_logic_vector(to_unsigned(i+1,5)),
45          clock => clock,
46          reset => reset
47      );
48  end generate pipe;
```

Figure 12: Generation of pipeline stages

Casting is needed for addition in VHDL, which is used to increment the number of rounds. Due to the round counter being a logic vector, it needs to be cast to unsigned to perform the addition operation and then cast back to a logic vector. Arrays have been created to store the 64-bit text at the state of encryption it is at, as well as another array to store the round keys. Finally, this module will set the output ciphertext for the cipher. This implementation also reduces the number of top level ports compared to the round-based implementation.

AXI Peripheral

It is important to be familiar with the process of development of the cipher, from creating the project to running the cipher on the FPGA. A familiarity with this process will make the development a lot quicker and more efficient.

Following the RTL development of the cipher core, the design will be packaged as an IP and an AXI peripheral will be created. RTL designs can be developed for the cipher and also to simulate and validate the cipher cores before creating an AXI IP for the design. For the development of the full AXI peripheral, the round-based cipher, which has been described previously, was packaged as an IP for use in it.

Finite State Machine

The finite state machine controller designed is imperative to the design for the control of the connection between the round-based cipher core and the control/status registers. The figure below describes the states contained in the FSM and the flow of the encryption process through the transition of states.

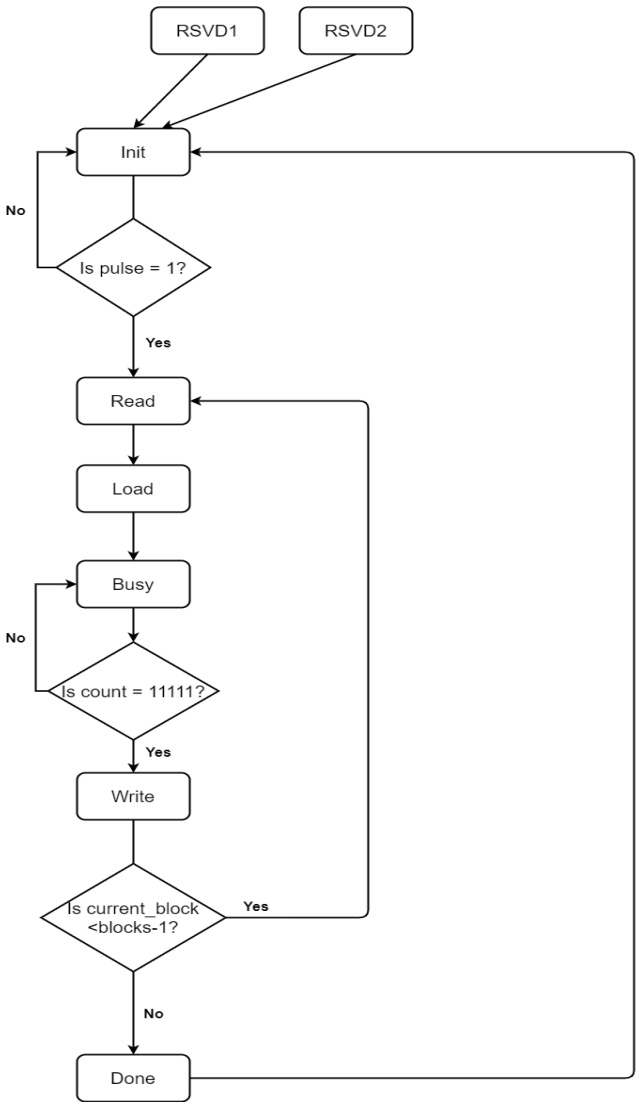

Figure 13: State Diagram of designed FSM

This design uses a Moore FSM. In a Moore FSM, its output values are determined by only its current state, whereas in a Mealy FSM, its output values are determined by both its current states and the values of its inputs. There is a total of eight states in this FSM, which have been described below.

INIT: This is the initial state of the FSM which transitions to the next state upon a high pulse which signifies that the execution of the cipher can proceed when the bit is set to '1'.

READ: This state is used to read the block of 64-bit plaintext from the designated plaintext BRAM.

LOAD: The 64-bit block will now be loaded into the connected cipher core.

BUSY: The FSM will enter this state during the 31 rounds of encryption.

Write: This state is used to write the block of 64-bit ciphertext to the designated ciphertext BRAM. The state will return to the READ state if the current block is not the last block.

Done: This state signifies that the encryption process has been complete and the resulting ciphertext has been written to the BRAM. The FSM will now return to the INIT state.

RSVD1: Reserved state. State will return to INIT.

RSVD2: Reserved state. State will return to INIT.

The above is a brief description of the logic applied in each state, however, the full logic can be seen in the code in Appendix F, where most of the logic for the states is making signal assignments and setting bits to '1'.

This FSM module also contains logic for the BRAM interface, such as static ties to zero for several signals such as the write enable. Registers are also used for the purpose of counting. There is a round counter to count the round of the encryption process, a block counter to count the blocks being encrypted and an address counter to store the BRAM address at which the plaintext block is read from and also the address that the ciphertext block will be written to. The round counter logic is not assigned to the cipher core however, as a round counter is already contained there. This is an area that could be improved on, as having two individual counters in the AXI peripheral for the same count is not the best design, however, the design for the round-based core was designed before taking this into account and it does not increase the complexity of the design any significant amount. Finally, the FSM controller module contains control/status register (CSR) to cipher core signals that are routed for the connection of data.

AXI Peripheral Creation

Once the hardware development of the design has been completed, we can package it as an IP. A full AXI peripheral will be designed including the round-based PRESENT IP which will interface with the Zynq PS. The interface type is set to Lite, the interface mode is set to Slave, the data width is 32 bits and the number of registers is 7. These were the details set for this project; however, these can be adjusted to suit the needs of the project being worked on. From here, the IP will be edited so that our custom design for the IP can be created.

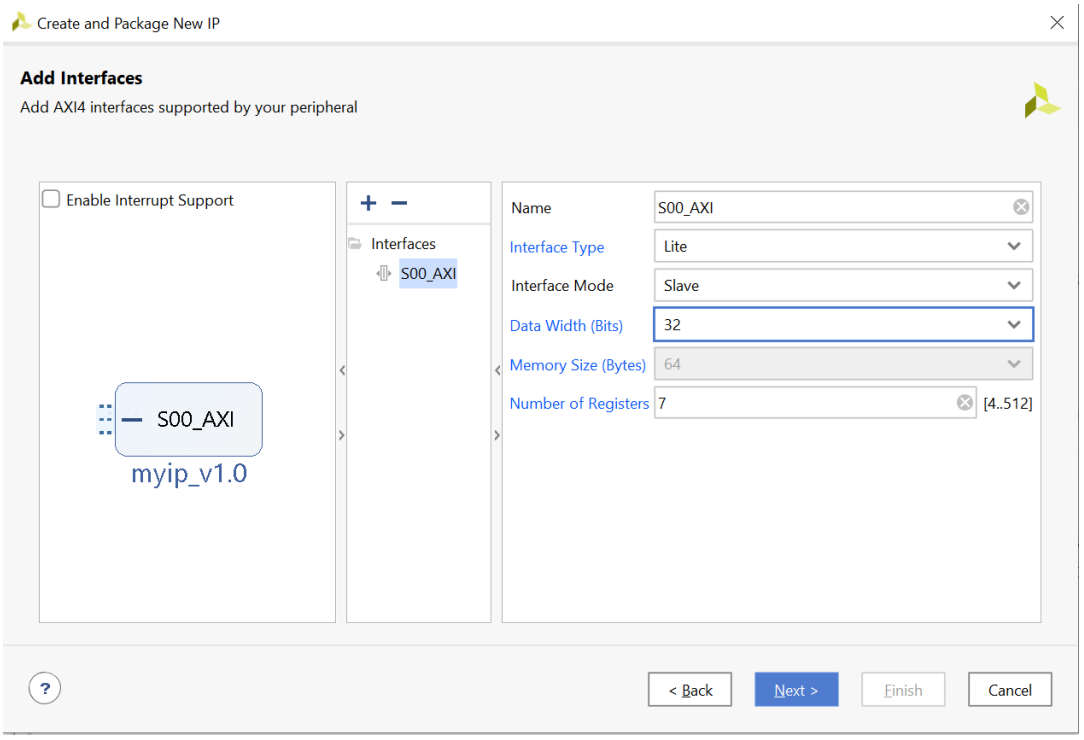

Figure 14: Interface settings

Edit & Package IP

The IP can now be edited in the IP packager, in a new Vivado window that will open. Two files have been created that can be seen in the 'Sources' pane. One is the AXI interface and the other is the top level RTL file, which contains the AXI bus interface as a component. RTL files can be added to this project to set the inputs and outputs of the system. The components can be instantiated in the top level file, which will be labelled as the name of your IP.

This top level file for the IP already contains ports for the AXI slave bus interface, however, we need to add in extra ports to interface to the plaintext and ciphertext BRAM that will be created as well as a done interrupt pin. This file should already be populated with a component declaration along with a port map for the AXI interface module. This should similarly be done for the FSM and the top-level cipher core. Appropriate signals have been created for connecting the AXI slave interface to the FSM controller and the cipher core. These are then connected through the port mappings. The AXI clock and reset which have been provided are used for the design to run on. This allows for synchronisation between the cipher design and the AXI fabric.

The AXI slave interface does not need a lot of modification to the file that has been generated. Firstly, the 80-bit key needs to be stored in the slave registers to provide the key for the FSM controller. The slave registers could only be created as 32-bit registers, so 3 registers need to be used to store the full key. Slave registers 4, 5 and bits 0-15 of register 6 were concatenated together to provide the key to the FSM controller. A port was created for the key for this purpose, as well as a port for the blocks and the pulse.

```
141     key <= slv_reg4 & slv_reg5 & slv_reg6(15 downto 0);
142     blocks <= slv_reg0(11 downto 1);
```

Figure 15: Code to set the key and number of blocks from the slave registers

Similar is done for the control register, which contains the number of blocks in bits 1 to 11, as can be seen in Figure 15 above.

The final bit of logic to be created in this file is for the pulse which is a single bit that is also contained in the control register. This bit controls the execution of the cipher, which can proceed when the bit is set to '1'. This bit remains set as '1' until it is manually cleared.

```
447     process(S_AXI_ACLK) is
448     begin
449         if ( S_AXI_ARESETN = '0' ) then
450             pulse_ff <= '0';
451         else
452             pulse_ff <= slv_reg0(0);
453         end if;
454     end process;
455
456     pulse <= (NOT pulse_ff) AND slv_reg0(0);
```

Figure 16: Control of pulse which signals cipher execution to the FSM controller

Once editing of these files is complete, they can be saved and the 'Package IP' window should be opened. From here the changes can be merged and the IP can be packaged so that the PRESENT core can be integrated into the AXI peripheral.

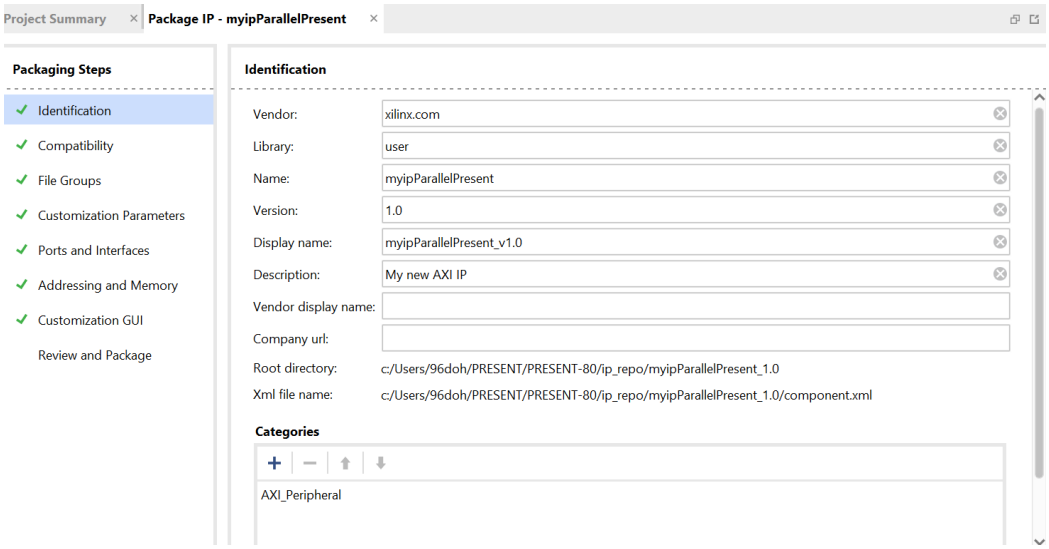

Figure 17: Package IP Tab

Logical Interface/IP Physical port mapping

The final stage of the IP development is the packaging of the IP and in particular the mapping of the interface's logical ports with the IPs physical ports. After making these changes to the IP, the changes will need to be merged in the Package IP window. This needs to be done for any of the packaging steps that does not have a tick beside it. The Ports and Interfaces tab should contain all the new ports that were set up for the IP, as well as the AXI ports which are grouped together.

The BRAM interfaces need to be mapped logically with the interface. To achieve this, two new BRAM interfaces need to be created. These two BRAM interfaces will be for the plaintext and the ciphertext. The interface definition should be set to bram_rtl when creating these interfaces and the mapping should follow the tables Table 4 and Table 5 below. Table 2 and Table 3 have a brief description of the function of each of the ports for the interface and the IP.

Table 2: Plaintext Physical Ports

IPs Physical Ports	Description
pt_e	Clock gating check
pt_read_data	Read plaintext data
pt_wr_data	Write plaintext data
pt_we	Write enable
pt_addra	BRAM address
pt_clk	Clock
pt_reset	Reset

Table 3: Ciphertext Physical Ports

IPs Physical Ports	Description
ct_e	Clock gating check
ct_read_data	Read ciphertext data
ct_wr_data	Write ciphertext data
ct_we	Write enable
ct_addra	BRAM address
ct_clk	Clock
ct_reset	Reset

Table 4: Plaintext BRAM Port Mapping

Interfaces Logical Ports	IPs Physical Ports
EN	pt_e
DOUT	pt_read_data
DIN	pt_wr_data
WE	pt_we
ADDR	pt_addra
CLK	pt_clk
RST	pt_reset

Table 5: Ciphertext BRAM Port Mapping

Interfaces Logical Ports	IPs Physical Ports
EN	ct_e
DOUT	ct_read_data
DIN	ct_wr_data
WE	ct_we
ADDR	ct_addra
CLK	ct_clk
RST	ct_reset

Once the mapping has been completed, the IP is ready to be packaged/re-packaged.

IP Integration

A new project is now created for the IP integration step to build the full AXI peripheral. This project is what will be used to design the full AXI peripheral for the round-based PRESENT IP. The Zynq processing system (PS) needs to be configured to allow software to be executed on it. The block designs feature as part of the IP Integrator in Vivado can be utilised to configure and connect the PS using a GUI representation to display the connections. You can add IPs to the block design using the '+' button. This allows you to search the IP repository where you can add the Zynq PS and the IP you have created for the cipher. A run block automation suggestion will appear when the Zynq PS has been added to the block design, which will create connections to DDR memory and fixed I/O. Following this, the connection automation suggestion should be clicked. These two processes combined will assist in creating a system by making internal connections between different blocks and connections to external interfaces.

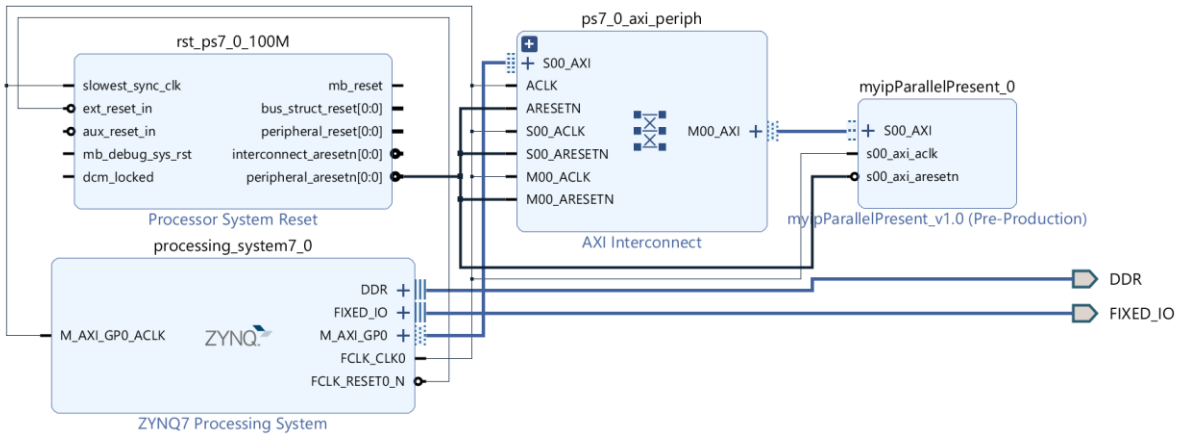

Figure 18: Block Design stage

From this point, the BRAM needs to be added and connected to the design appropriately. An AXI BRAM controller and a Block Memory Generator should both be added and connected as can be seen in Figure 19: Developed AXI peripheral with integrated IP core below.

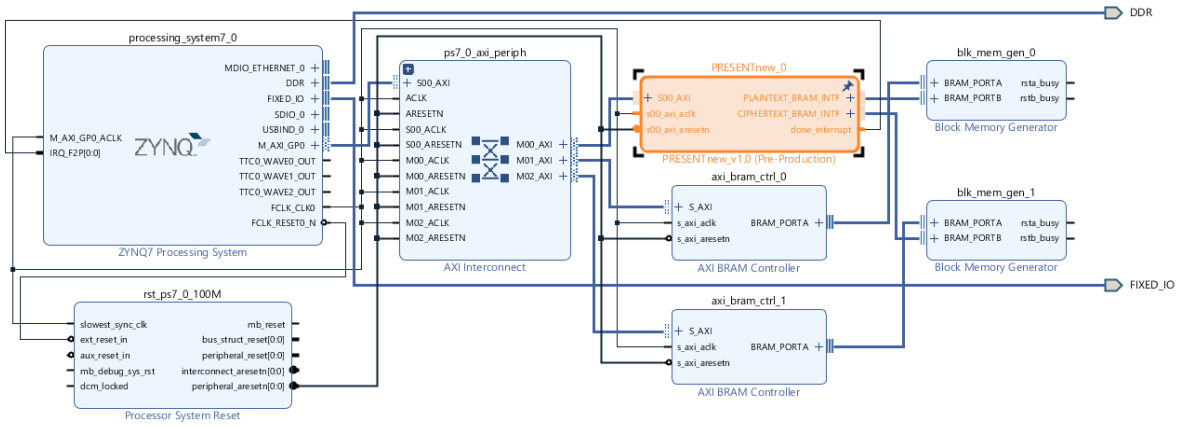

Figure 19: Developed AXI peripheral with integrated IP core

These blocks will need to be created for both the plaintext BRAM as well as the ciphertext BRAM. The BRAM port A of the BRAM controller needs to be connected to the BRAM port A of the memory generator. Port B of the Block Memory Generators should be connected to the plaintext and ciphertext interfaces that were previously created and setup in the IP. The done_interrupt port of the IP is connected to the IRQ_F2P port of the Zynq PS, which enables the 16-bit shared interrupt port from the programmable logic (PL).

It is important to validate the design by clicking the validate design button which will run a comprehensive check on the design for any errors or potential issues. Once the design has been completed, with appropriate connections for the Zynq PS created, a round of synthesis and implementation should be run. Output products need to be generated and then a HDL wrapper can

be created. This will create a .vhd file with appropriate VHDL code to be utilised as a wrapper for the design. It will automatically be set as the top file in the project and inherit the design as part of it by creating a component of the design.

A bitstream for the design can now be generated for the FPGA. The final step to perform before developing a software application to drive the program on the FPGA is to export the hardware platform information, including the bitstream. To do this, you must export the hardware using Vivado. The 'Include bitstream' checkbox should be checked when doing this step.

Following the above steps that have been described, a full hardware implementation including an AXI IP will have been developed. The system will be ready to be imported into the Xilinx SDK where a software application can be designed to drive the hardware.

Software Development

Software Driver Application

A software driver file has been created in C to invoke the hardware peripheral design from the PS. The development of this driver was in Xilinx SDK, which can be used to program the FPGA as well as write the software driver and test code. A source and header file were created for the driver. There are three main functions to the driver, the first is the interrupt service routine (ISR) which is used to copy the ciphertext data from the BRAM to the system memory. "`xparameters.h`" contains information for addresses of parameters that have been set in the peripheral creation. A setup function is used to set up and initialise the ISR, the general interrupt controller (GIC) and the key value. The GIC initialisation requires the "`XScuGic.h`" file. Here all the functions of this file can be seen with detailed comments here and in its source file to explain the purpose of the various functions and the arguments these functions accept. Finally, the key value is set within this function by writing to the appropriate slave register addresses that were set for the key in the IP development. A function is used to begin the encryption process. The plaintext is set from the appropriate BRAM block. The pulse in the control register is then set allowing the encryption to begin. The output is set to the values resulted for the ciphertext. The number of blocks is set to the size value and the pulse is reset.

Software Implementation

As a hardware version of the cipher has been developed, it would be useful to also create a software implementation of the cipher. C is an appropriate language for the design of the cipher as it is a relatively low-level language, when compared to other languages I have practiced such as Java. This means the bitwise operations will be easier to perform for modifying individual bits and registers. Due to time constraints for this project, a software implementation was not created; however, a software implementation was not the focus of this project. The main purpose for a software implementation is for a comparison with the hardware implementations, so a PRESENT implementation in C was found and modified to obtain the results needed for comparison. A modified version of [9] was used to obtain these results.

```
240     //Measure performance from start of encryption
241     clock_t start, end;
242     double cpu_time_used;
243     start = clock();
```

Figure 20: Start time measurement

```
249     end = clock();
250     cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

Figure 21: End time measurement

The latency and throughput can be calculated through the use of this code, which uses the time.h library. The CLOCKS_PER_SEC returns the number of clock ticks per second. Using this equation, the computing time for the encryption is calculated. Throughput can also be calculated through the use of this information. The encryption process was repeated for 1000 blocks of data, which will allow for a more accurate measurement of the throughput using a larger amount of data. An average can then be determined from this information to get the throughput of the software implementation.

Tester Application

This application has been designed to call the software driver and measure the execution time of the cipher encryption so that the performance of the cipher can be determined. There are separate functions to perform the hardware test alone or to test the performance of the implementation. The hardware test function will return a 1 or a 0, depending on if the result returns as the expected output.

Conclusion

This section outlines the design of the different PRESENT-80 implementations that have been created in VHDL. Snippets of code have been provided to show how they were implemented in the design. Following this, an IP was created and packaged for use in the full AXI peripheral. The AXI peripheral was designed with an FSM controller to interface with the control/status registers, the round-based core and the plaintext and ciphertext BRAM. Completion of the AXI peripheral means the design can be programmed to the FPGA where it can be tested using Xilinx SDK. The software driver was written in C using the Xilinx SDK as an IDE. This is used to invoke the hardware implementation on the FPGA. A software version of the cipher is also presented that is modified to get appropriate results.

Validation

Introduction

This section will look at the methods of validation for this project. Designing and building the project are important steps but validation is just as important. If the cipher does not work as intended, there is no use for it and validation is the step we can take to ensure it functions correctly.

The RTL design for the cipher has been implemented at this stage. To ensure that it operates as intended, given test vectors for the plaintext and key should be used to produce a specified ciphertext. These test vectors will be used in the system through a testbench. The purpose of a testbench is to provide a stimulus to a unit under test (UUT). The testbench can be used to apply the test vectors to the design.

Test Vectors & Test Benches

The algorithm specification provides four test vectors for the PRESENT-80 cipher. These test vectors give sample inputs for the plaintext and the key, with an output for the ciphertext given for the combination that will result if the cipher has been implemented correctly. Through the use of these test vectors, provided the correct output results from each of them, we can be almost certain that the algorithm has been implemented correctly.

Test vectors can be found from other sources also, which can be used to further validate the functionality of the cipher implementation. These test vectors should only be used to verify the functionality once the test vectors that have been provided in the algorithm specification are producing the expected output. The AVRCryptoLib [22] have provided a large amount of test vectors that were used in development of the library. These can be used as a further source for validation of the implementation.

Testbenches are created for the testing of individual files as well as the overall system.

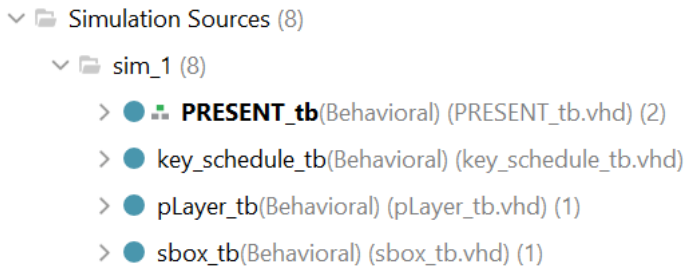

Figure 22: Source testbench files

Figure 22 shows the testbenches that were developed for the initial implementation of the cipher. Unlike, the pipelined architecture, there were testbenches created for the pLayer, s-box and key schedule to ensure they worked correctly. These files were also used in the pipelined implementation, so I knew that they worked correctly. A testbench was created for connecting to the top-level file of the system, with a separate stimulus generator file which was used to set values for the inputs to the

system, as well as generating a clock. The process for generating the clock can be seen in Figure 23 below.

```
31      -- Clock generator
32      Clock1: process
33          variable clktmp : std_logic := '1';
34      begin
35          while DONE /= true and FAILED /= true loop
36              wait for CLOCK_PERIOD/2;
37              clktmp := not clktmp;
38              Clock <= clktmp;
39          end loop;
40          wait;
41      end process;
```

Figure 23: Clock generator process

The values for the plaintext and key are set in a process for setting the inputs and checking the output of the system. The reset is also set to high when each new test of the system is run. An if statement is used to check if the ciphertext result is correct and will print a pass or fail statement to the console, depending on the result. This process is used repeatedly for the number of test vectors applied to the system. This could be done in a looped process by retrieving the test vectors from arrays, however, this does not make any significant difference other than a slight improvement to the tidiness of the code so this method was not applied.

```
76          CheckResult: process
77          begin
78              wait for 20ns;
79              reset <= '1';
80              load <= '1';
81              plaintext <= x"0000000000000000";
82              key <= x"00000000000000000000";
83              --reset <= '0';
84              wait for CLOCK_PERIOD;
85              reset <= '0';
86              wait for 600ns;
87
88              wait for 20ns;
89              wait for 20ns;
90              load <= '0';
91              if ciphertext = x"5579C1387B228445" then
92                  report "Test vector 1 passed";
93              else
94                  report "Test vector 1 failed";
95              end if;
```

Figure 24: Check result process

Figure 24: Check result process displays this process of checking the result of the ciphertext after setting the plaintext and key to the test vectors.

Debugging

Debugging of the project is an important aspect to the building of this project. During the RTL implementation of the project, problems were encountered that lead to having to analyse and debug the project to figure out what is going wrong. The waveform viewer is a key feature in Vivado that can be used for the debugging of the project. This tool displays the top-level input and output ports, signals created in the testbench as well as signals that you can add to it throughout the time of the simulation running, as shown in Figure 25.

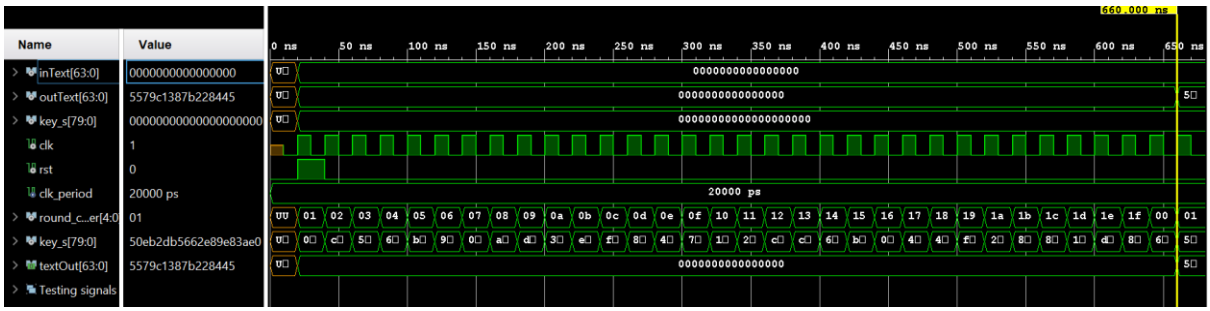

Figure 25: Vivado waveform viewer

To add signals to the waveform, you can open the 'Scope' tab beside the 'Sources' tab, and from here you can select the different files in the project and add them to the waveform from the objects window by right-clicking the signal and selecting 'Add to wave window'. Using the scope and object window can also be useful for debugging the project and finding out where the issue is occurring, however, it is usually easier to add it to the window to get a visual display of the signal over the simulation time.

Another useful method for debugging is by using print statements. An example of this usage is shown in the following hello world VHDL source where line prints are shown. [23] Appropriately placed print messages are useful for showing what the program is doing and when it is doing it. They can be used in both procedural and testbench code and the statement will be printed out and displayed in the TCL console.

Some common errors that can occur in simulation include getting the incorrect output result, an X value or a U value. An incorrect result for the output of the system is the trickiest to debug, as there are a range of reasons as to why your design may not have worked correctly. If the output of the system is incorrect, one of the signals used to get the result at some point in the system will be incorrect. If you can find which signal is incorrect, you can find where the error in the design/code is located.

An X value occurs when you have multiple drivers for a single signal. If there is a different value being driven to the signal, the value will be unknown and set as an X. This is usually quite an easy error to fix, as you just need to review your code for the lines that include signal that has the X value and solve the error from there.

A U value means that the signal has been uninitialized. This is a quick issue to solve as you just need to set a value to initialise the signal. The signal will usually start as U as can be seen in the waveform, but this is not an issue at this point in the simulation.

The ability to use the simulation features to debug is important. You can restart the simulation, set the simulation run time and step through the simulation using the simulation features provided. These are key features to be used when running and debugging a simulation.

Conclusion

Using testbenches, test vectors, and various methods of debugging, we can now conclude that the cipher is working correctly according to the test vectors provided by the algorithm specification. This section has demonstrated the method of creating a testbench, appropriately connecting it to the design and running a simulation with the test vectors driving the design and checking for the correct ciphertext output.

Some of the methods for debugging which were used during this project have been outlined, as well as some useful tips for making the best possible use of the waveform feature provided by Vivado. Upon completion of the validation and confirmation of the cipher's functionality, this allows the project to move onto the next section which will address the packaging of the design into an IP and the creation of the AXI peripheral.

Appendix E
Testing & Results

Testing & Results

Introduction

This section details the results that have been obtained for the various implementations of the cipher, as well as describing the process to achieve these results. Results have been obtained in two categories: resource utilisation and performance. Resource utilisation refers to the area utilisation while performance refers to the operational frequency, power consumption, latency, throughput. A brief description has been provided in Table 6 of the different designs so that it is clear which design is being referred to.

Table 6: Module Descriptions

Design	Description
Round-based core	Round-based cipher core
Pipelined core	Pipelined/parallel cipher core
PRESENT IP	IP for round-based design
Full peripheral	Full AXI peripheral with round-based IP

Metrics

Timing

Procedure

Timing is an important metric to gather information of as it is needed for the operational frequency for the clock. It is important to be able to work out the maximum operational frequency as this will allow for best results in terms of the area utilisation and power consumption.

To obtain the timing reports necessary for this analysis, we must first create a clock with specified constraints to it, in particular its clock period. These timing constraints will be used when Vivado runs synthesis and implementation, which can allow an operational frequency to be determined.

To set up these timing constraints, you can select the Edit Timing Constraints option which is underneath both the Synthesis and Implementation section in the Flow Navigator. This will open a timing constraints window where you can create a clock and save the constraints in a .xdc file. You can manually edit this file or you can use the Timing Constraints window for the creation and editing of the clock. This will automatically enter the adjustments you have made into the timing constraints file when you edit it from the GUI. The .xdc file contained the lines of code as shown below.

```
create_clock -period 4.000 -name clk -waveform {0.000 1.000} -add [get_ports clock]

set_property HD.CLK_SRC BUFGCTRL_X0Y0 [get_ports clk]
```

The first line creates a clock with a period of 4 nanoseconds (ns), named as clk which enters the system through the clock port. This gives the system an operational frequency of 250MHz and the default

duty cycle of 50% is used as it has not been specified as any different percentage. Upon running synthesis and implementation, Vivado will report back on if these timing constraints have been met, as shown in the Figure 26 below.

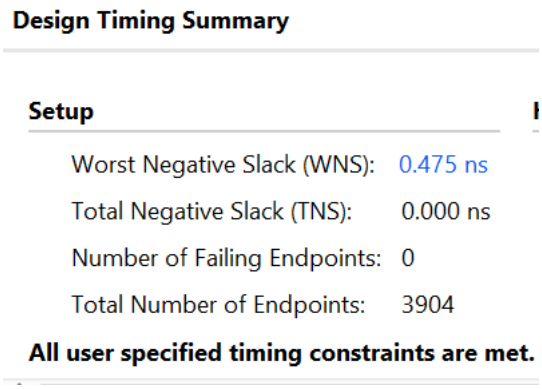

Figure 26: Timing summary

The second line in the timing constraints file refers to the out of context mode in which the design is being implemented. This will route the design to a higher level than the top-level I/O ports of the design. Running the implementation out of context will simplify the timing analysis, however, since the designs are being run out of context, there is no need to specify input/output delays as there is no interfacing with another device that would cause these delays. Without a location on the die to begin analysing the clock paths, the tools cannot perform a skew analysis. This will make timing analysis inaccurate. The second constraint attaches the clock to the die, and the clock can propagate to placed and routed logic. [24] To run the implementation out of context, the following command should be entered into the TCL console before running the implementation. 'synth_1' is the default name set for a synthesis run, however, this can be replaced in the command with the name used for your synthesis run.

```
set_property -name {STEPS.SYNTH_DESIGN.ARGS.MORE OPTIONS} -value {-mode out_of_context} -objects [get_runs synth_1]
```

Alternatively, you can use the GUI to set the synthesis in out of context mode before running it. Once the implementation has been run, there will be a number of reports available with information about the run, specific to certain details. The report that is most relevant to this timing analysis is the 'impl_1_route_report_timing_summary_0' report. This contains the information required to determine the maximum operating frequency, which can be calculated by using equation (1).

$$f = \frac{1}{T} \tag{1}$$

T represents the period which has been set in the timing constraints, however, we need to find the minimum period in which the system can operate without breaching the timing constraints to then determine the maximum operating frequency. The timing report contains a wealth of information, including clock constraint issues, whether the target frequency has been met, the maximum delay in the design, the combinational latch loops and more. The maximum delay and whether the target frequency have been met are the key details for this timing analysis, as if the target frequency has not been met, there will need to be an increase in the time period. A slack of 0.475ns is reported for the time period of 4ns, so the timing constraints can be adjusted by subtracting the slack from the period, determining the new period to set. This is shown in (2).

$$T_n = T_{-1} - t_s \tag{2}$$

The reason for the slack is due to the tools only running as hard as needed to meet the timing constraints. A trial and error process using this method can be used to determine the maximum operational frequency. After each run of the implementation with timing constraints, the slack can be subtracted from the clock period to give the new clock period. This process is repeated over and over until the minimum clock period, and therefore, the maximum operational frequency has been found.

Due to this method of timing analysis being performed out of context, this allows for the usage of the best possible routing and resources. This is an ideal scenario; however, this would usually not be the case. Similarly, this analysis neglects input/output timing delays for the top-level of the design, as well as uncertainty not being accounted for. One last condition to note is that when the tools are calculating the total slack, it uses the worst conditions [24].

Results

The following results shown in Table 7 were obtained for the various hardware implementations, using the above methodology to achieve these results.

Table 7: Maximum operational frequency results

Design	Minimum Period (ns)	Maximum Frequency (MHz)
Round-based core	2.6	385
Pipelined core	2.85	350
PRESENT IP	4.4	227
Full peripheral	9.3	108

Analysis

From the results obtained for the timing analysis, the round-based cipher core is shown to have a shorter period (2.6ns) in comparison to the pipelined cipher core (2.85ns). This results in a higher maximum frequency for the round-based cipher than the pipelined implementation. The result of the round-based cipher core having a higher frequency than the pipelined cipher comes as a bit of a surprise, however, the timing reports suggest that the reason for a higher frequency result in the round-based cipher is due to the critical path for the pipelined cipher being between the pipelined

stages rather than the full path. The full peripheral has the largest minimum period as was expected while the IP showed a lower frequency than the cipher core designs.

Area

Procedure

The area utilisation in the system is another metric that is of significant importance to consider and measure when building the cipher. We can get results for the area utilisation from Vivado generated reports post synthesis and implementation. These reports give a breakdown of the area utilisation in different areas, such as the slice utilisation which is further broken down to slice LUTs (look-up tables) and slice registers, I/O utilisation, DSPs (digital signal processors) and block RAM (BRAM) tiles.

A decision on the frequency used for analysis of the designs must be made. To decide the operating frequency, the maximum frequency at which the designs can operate at are used as was determined in the timing section. This is an important detail as it will have an influence on the amount of resources that are consumed by the design.

The results for the I/O utilisation and DSPs are not important for this analysis as there is no interfacing the device with other devices, so the number of I/O is not particularly important in that regard. Similarly, there is no usage of DSPs in this design so the results in this section can be ignored.

The results below for utilisation in the system are represented in both total slice utilisation and slice LUT utilisation for analysis of the system. It is also important to note that the report will tell you the number of slices to be utilised, however, it does not specify if the slice has been fully utilised. This means we cannot get a perfect representation of the utilisation in the system, but it will give us a good idea.

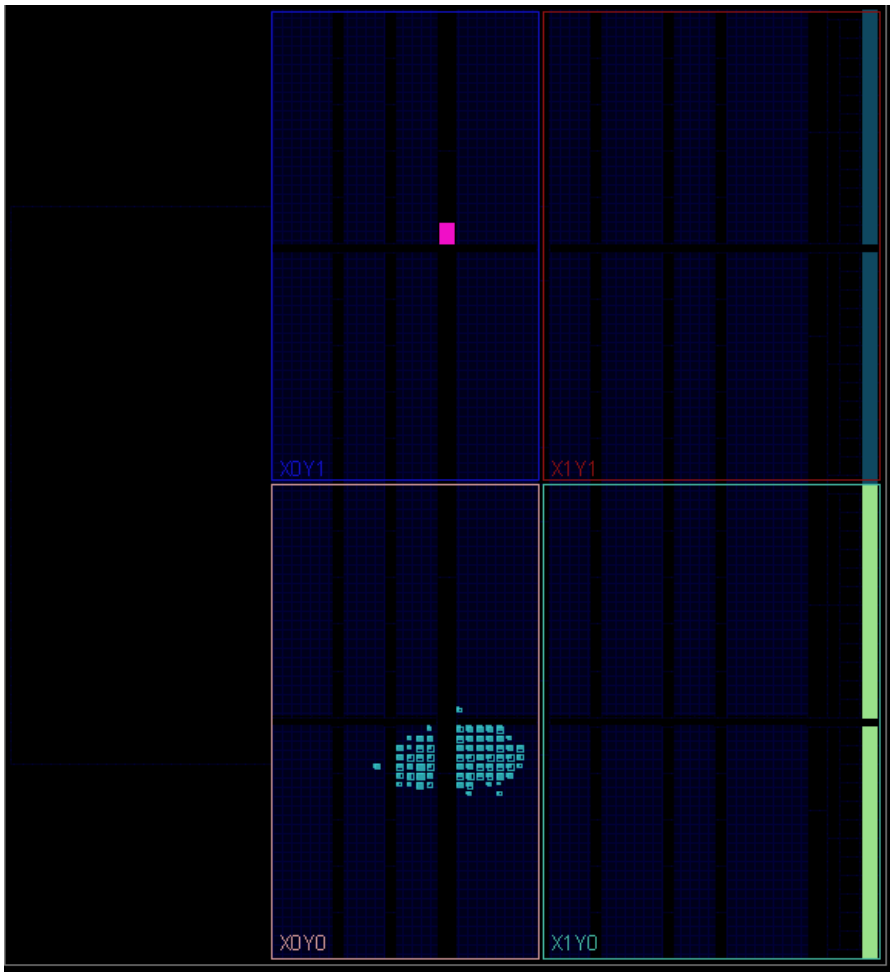

Figure 27: Round based area utilisation floorplan

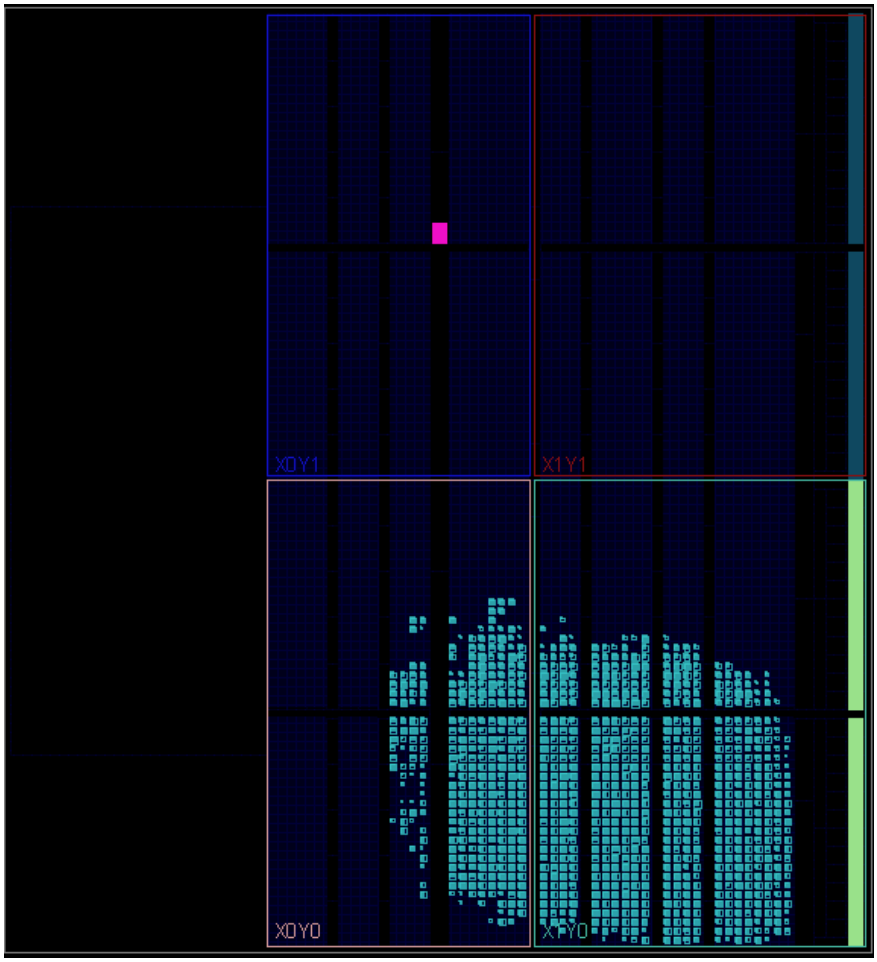

Figure 28: Pipelined area utilisation floorplan

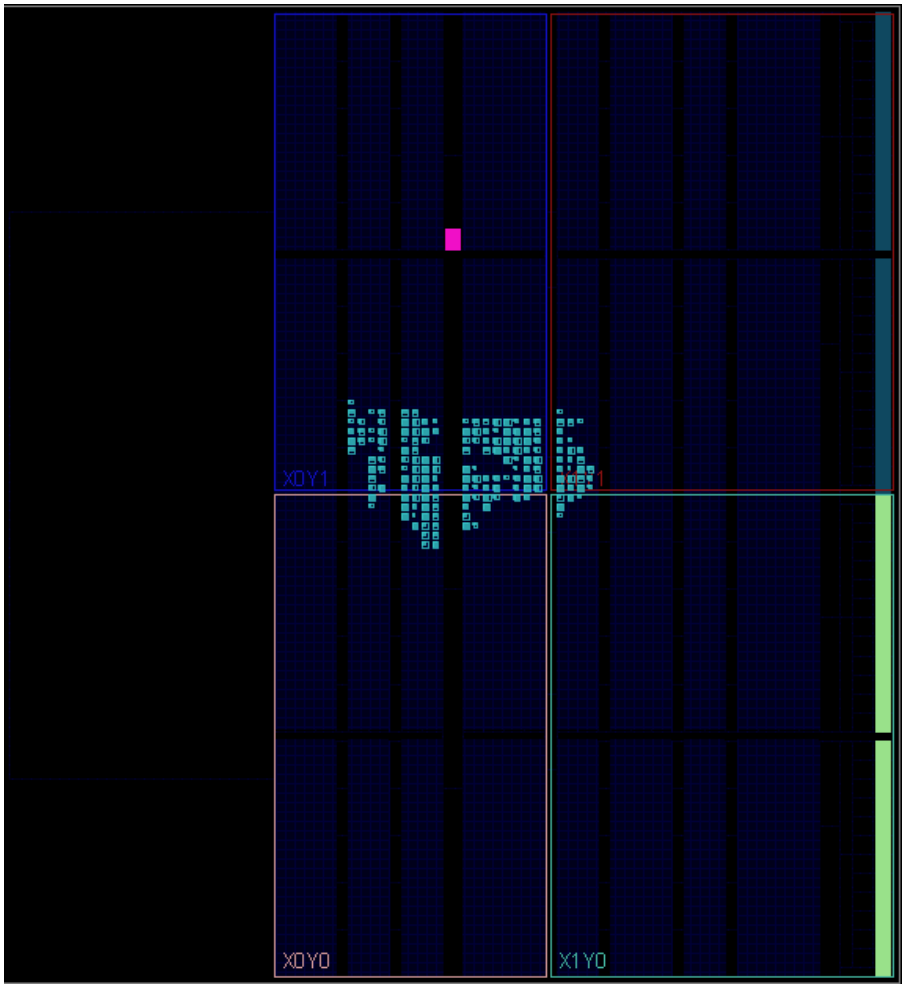

Figure 29: IP area utilisation floorplan

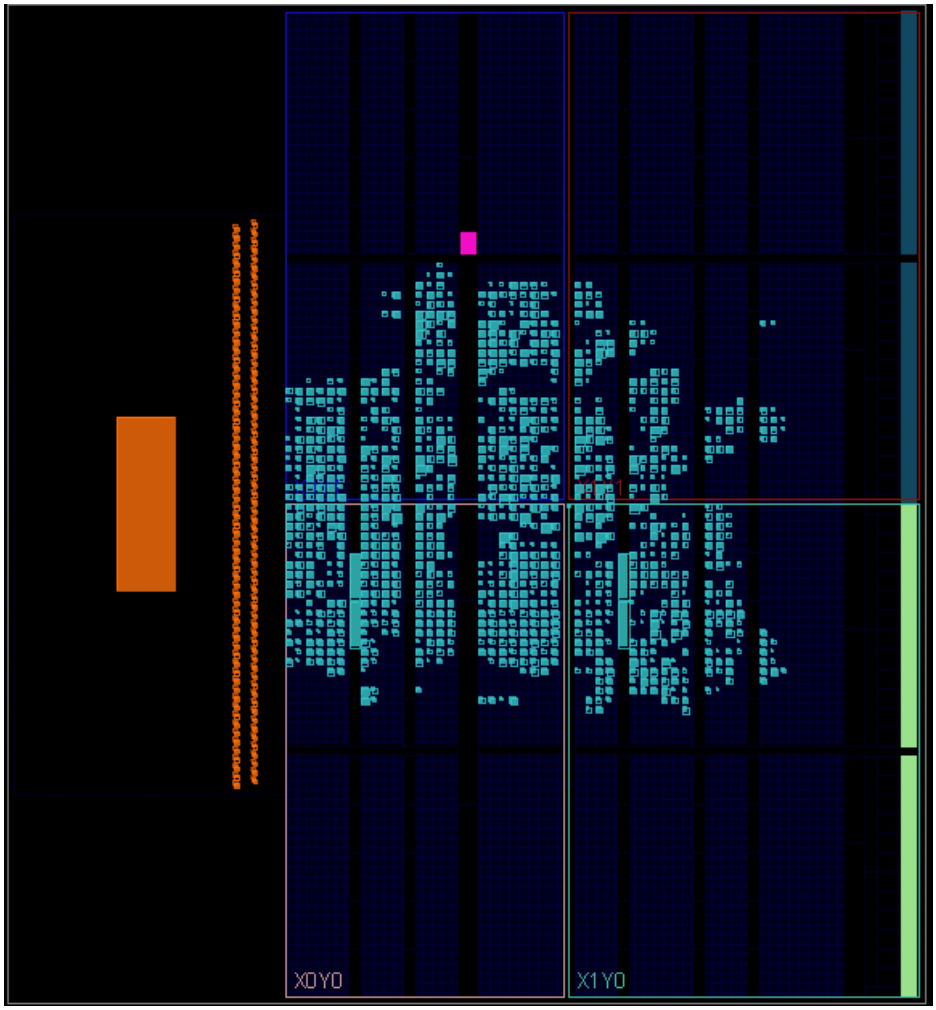

Figure 30: AXI peripheral area utilisation floorplan

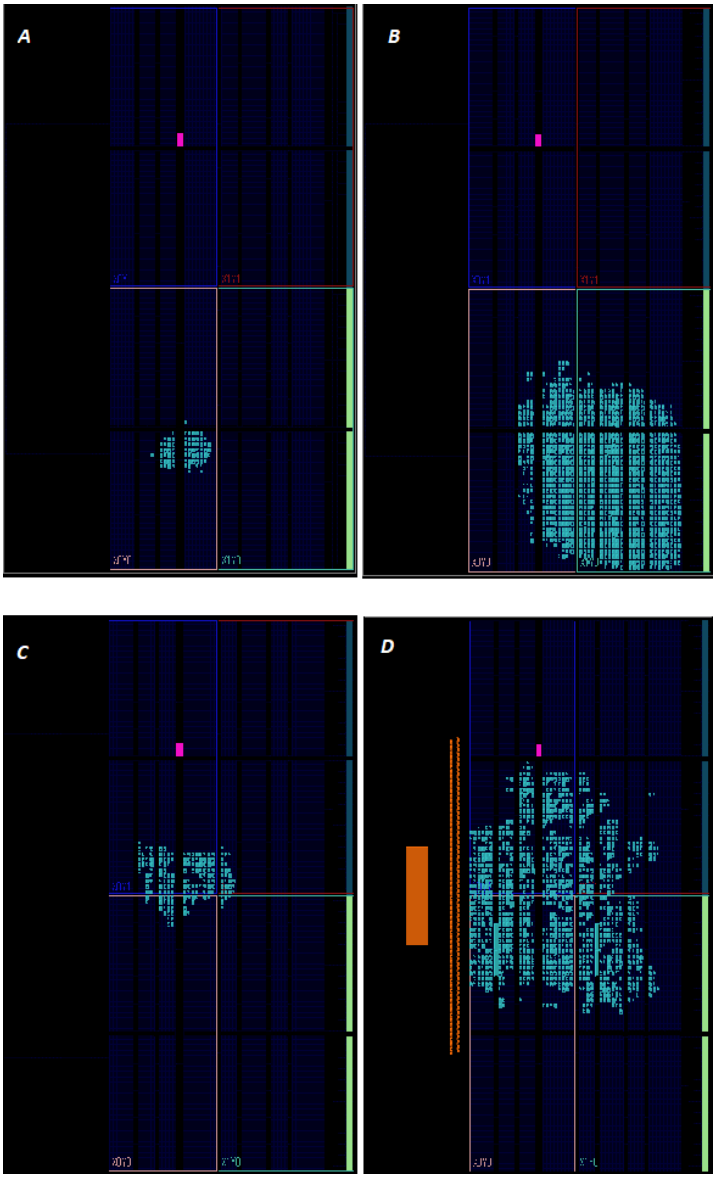

Figure 31: Area utilisation floorplan comparison

Table 8: Area Utilisation Results

Design	Total Slices		Slice LUTs		Slice Registers	
	Used	% Utilisation	Used	% Utilisation	Used	% Utilisation
Round-based core	70	1.59	209	1.19	213	0.61
Pipelined core	979	22.25	2815	15.99	4227	12
PRESENT IP	172	3.90	289	1.64	533	1.51

Design	Total Slices		Slice LUTs		Slice Registers		BRAM Tiles	
	Used	% Util.	Used	% Util.	Used	% Util.	Used	% Util.
Full peripheral	1077	24.47	2423	13.77	3127	8.88	4	6.66

Analysis

The results measured and provided in the above section show a significantly larger area utilisation for the pipelined cipher core in comparison to the round-based cipher core, which has quite a low area

utilisation. The pipelined architecture uses ~14x the number of slices that the round-based architecture uses. These were the expected results, as the pipelined implementation will provide a significant improvement in performance at the sacrifice of its area utilisation. A ~14x increase in the slice utilisation was shown in the pipelined cipher when compared to the round-based architecture.

A visualisation of the area utilisation for the two architectures, as well as the IP and AXI peripheral can be seen in Figure 27, Figure 28, Figure 29 and Figure 30 respectively. A side-by-side comparison has also been provided in Figure 31, which is useful for visualising the difference between them. It is immediately evident from these figures that the pipelined architecture has a significant increase in the area utilisation to the round-based architecture, while the AXI peripheral displays the BRAM in orange on the left and has a large area utilisation also.

The IP shows an increase in the utilisation for the number of slices, slice LUTs and slice registers when compared to the round-based architecture, however, it displays a utilisation that is less than the pipelined architecture. This is not surprising as the pipelined architecture was expected to have a significantly large area utilisation, due to amount of area used in each round of encryption. There is over 6x the number of slices used in the AXI peripheral as there is used in the IP. The AXI peripheral uses 4 BRAM tiles, out of a total of 60 available so there is no issue in the design here. While the full AXI peripheral has relatively close results to the pipelined cipher, it should be noted that this was for the round-based architecture and there is a significant increase in the area utilisation from the round-based cipher core and the IP results presented. The increase in area can be attributed to the two BRAMs, the AXI interconnect and in particular, the PS.

Power

Procedure

Power consumption is the next metric to be measured and analysed for the designs. Power consumption will have a general correlation with area utilisation, as a lower power consumption will usually mean there is a low area utilisation also. However, this is not always true, and it is important to measure and analyse both metrics to see how the design functions.

Results for power consumption can be obtained similarly to the previous metrics, through auto-generated reports after implementation of the design. The power will be broken down into the consumption by clocks, signals, logic, I/O, BRAM, etc. It will also break down the results for the power into two sections: static power and dynamic power.

Static power is the power consumed by the device when it is powered up, configured with user logic and there is no switching activity. It consists of device static power and design static power. Device static power is the power consumed by the device when it is powered up without the user logic. The devices operating environment and, in particular the junction temperature influences this power. Design static power refers to the devices power consumption due to the user logic without any switching activity. [25]

Dynamic power refers to the power generated due to the switching logic and routing used in your design. The round-based cipher uses a loop-based approach, which leads to a reduction in the power consumption in the design. This is a direct implementation of the PRESENT algorithm provided by the specification. According to Bogdanov et. al [8], PRESENT is designed with area and power constraints uppermost in mind, so a direct implementation of the algorithm provided in the cipher should lead to a low power implementation of the cipher.

This power estimation runs the designs under ambient device conditions and also uses the maximum operational frequency for the designs. The device/environment settings can be chosen by selecting the 'Report Power' option from the Flow Navigator. The results shown below were achieved under these conditions.

Results

Table 9: Dynamic Power Breakdown

Design	Clocks		Signals		Logic	
	(w)	(% of total)	(w)	(% of total)	(w)	(% of total)
Round-based core	0.010	33	0.011	39	0.008	28
Pipelined core	0.091	20	0.211	47	0.148	33
PRESENT IP	0	0	0.159	43	0.211	57

Design	Clocks		Signals		Logic		BRAM	
	(w)	(% of total)	(w)	(% of total)	(w)	(% of total)	(w)	(% of total)
Full peripheral	0.011	1	0.005	<1	0.004	<1	0.009	1

Table 10: Power Consumption Analysis Results

Design	Total Power (W)	Static Power (W)		Dynamic Power (W)	
		(w)	(% of total)	(w)	(% of total)
Round-based core	0.119	0.090	75	0.029	25
Pipelined core	0.545	0.095	17	0.451	83
PRESENT IP	0.464	0.094	20	0.370	80
Full peripheral	1.702	0.121	7	1. 581	93

Analysis

The above results displayed in tables Table 9 and Table 10 show a breakdown of the dynamic power consumption, as well as the breakdown of the power consumption for each design in terms of the static and dynamic power respectively. The power consumption analysis followed a similar trend to the area utilisation results, as was expected. This shows a significant increase in the power consumption by the pipelined cipher, which uses ~4.5x the amount of power and ~16x the amount of dynamic as the round-based cipher core. The IP for the round-based core builds upon the files which have been created for that design, so it was expected to see an increase in the power from the round-based power consumption.

The increase in the dynamic power is a better representation and more in line with the significant increase in the area for the pipelined cipher compared to the round-based cipher. The static power for both designs is similar, with a slight increase in the static power for the pipelined cipher. The signals and logic in the pipelined architecture are what lead to the increase in the dynamic power. This is expected as the pipelined cipher is an 'unrolled' version of the architecture with 31 repeated units.

The AXI peripheral, as expected, had the largest amount of power consumption by a significant amount. This contains the IP, as well as the BRAM and the Zynq PS. The Zynq PS was the main source of power in this design, using 96% of the dynamic power for this design. The AXI peripheral has a power consumption that is slightly over 3x more than the second highest design (pipelined cipher).

Latency

Procedure

Latency is the first metric that is used to measure the performance of a block cipher. The latency of the system is the number of clock cycles required for the system before it can accept its next input. A system can accept a new input on every clock cycle; however, it is defined as the time it takes to propagate from input to output. Using this information, we can determine the latency of the design by running a simulation of the design. Using the waveform viewer, we can easily count the clock cycles from the input to output.

Results

Table 11: Latency Results

Design	Latency (Cycles)
Round-based PRESENT	32
Pipelined PRESENT	32

Table 12: Latency Timing Results

Design	Latency (ms)
Software core	1.09
Hardware-accelerated core	0.038

Analysis

Both designs use the same number of clock cycles to perform the encryption. This was the expected result as the number of cycles should mirror the number of rounds of encryption.

The software core shows a higher latency than the hardware-accelerated implementation. This result is expected as the PRESENT cipher was designed with hardware optimisation in mind, so it is good that the results confirm this.

Throughput

Procedure

Throughput is the rate at which a new output is produced with respect to time. This can also be expressed as the time to encrypt the plaintext to ciphertext in this system. (3) shows the equation to calculate the throughput (Thr), which divides the operational frequency (f_{op}) by the latency (Lat) and then multiplies this by the block size (BSize). The operational frequency is the maximum operational frequency calculated previously for these designs.

$$\mathrm{Thr} = \frac{\mathrm{f_{op}} \times \mathrm{BSize}}{\mathrm{Lat}} \quad (3)$$

Results

Table 13: Throughput Results

Design	Throughput (Mbps)
Round-based core	770
Pipelined core	22,400
Hardware-accelerated core	45
Software core	1.8

Table 14: Throughput results at 13.56 MHz

Design	Throughput (Mbps)
Round-based core	27.1
Pipelined core	840.72

Analysis

The results for the throughput are shown in Table 13 and Table 14, where Table 13 shows the throughput for the designs at their maximum operating frequency and Table 14 shows the throughput for the designs at an operating frequency of 13.56MHz, which is an appropriate frequency for RF applications, which is highly applicable to this cipher [12]. Both results show that the pipelined core has a much higher throughput than the round-based core, where area and power constraints are its uppermost focus rather than its performance. The reason for this is that the pipelined core can accept an input once the previous input has been processed through the first pipeline stage. This allows for a massive increase in the amount of data that can be in the pipeline core at once, at different stages.

Conclusion

These results show better performance in terms of resource utilisation for the round-based core when compared to the pipelined core; however, the pipelined core performs significantly better in its performance in terms of its throughput. The software implementation of the ciphers performance does not show good results, but the hardware-accelerated implementation shows significant improvement on these results showing that the cipher performs much better as a hardware design.

Appendices References

[1] 'Data Encryption Standard (DES)'. U.S. DEPARTMENT OF COMMERCE, Oct. 25, 1999, Accessed: Aug. 25, 2020. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf.

[2] 'FIPS 197, Advanced Encryption Standard (AES)', p. 51.

[3] A. Y. Poschmann, *Lightweight Cryptography: Cryptographic Engineering for a Pervasive World*. 2009.

[4] B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, 'A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues', *Journal of Network and Computer Applications*, vol. 58, pp. 73–93, Dec. 2015, doi: 10.1016/j.jnca.2015.09.001.

[5] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, 'A Survey of Lightweight-Cryptography Implementations', *IEEE Design Test of Computers*, vol. 24, no. 6, pp. 522–533, Nov. 2007, doi: 10.1109/MDT.2007.178.

[6] C. De Cannière, O. Dunkelman, and M. Knežević, 'KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers', in *Cryptographic Hardware and Embedded Systems - CHES 2009*, Berlin, Heidelberg, 2009, pp. 272–288, doi: 10.1007/978-3-642-04138-9_20.

[7] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, 'Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents', in *Smart Card Research and Advanced Applications*, Berlin, Heidelberg, 2008, pp. 89–103, doi: 10.1007/978-3-540-85893-5_7.

[8] A. Bogdanov *et al.*, 'PRESENT: An Ultra-Lightweight Block Cipher', in *Cryptographic Hardware and Embedded Systems - CHES 2007*, vol. 4727, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466.

[9] C Software code: P. Tonkovic, *Pepton21/present-cipher*.

[10] X. Fan, G. Gong, K. Lauffenburger, and T. Hicks, 'FPGA implementations of the Hummingbird cryptographic algorithm', in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Jun. 2010, pp. 48–51, doi: 10.1109/HST.2010.5513116.

[11] L. Batina *et al.*, 'Dietary Recommendations for Lightweight Block Ciphers: Power, Energy and Area Analysis of Recently Developed Architectures', 753, 2013. Accessed: Aug. 30, 2020. [Online]. Available: http://eprint.iacr.org/2013/753.

[12] 'RFID Frequency Bands & Spectrum » Electronics Notes'. https://www.electronics-notes.com/articles/connectivity/rfid-radio-frequency-identification/frequency-bands-spectrum.php (accessed Aug. 25, 2020).

[13] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, 'Lightweight Hardware Architectures for the Present Cipher in FPGA', *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2544–2555, Sep. 2017, doi: 10.1109/TCSI.2017.2686783.

[14] M. Katagi, 'Lightweight Cryptography for the Internet of Things', 2011.

[15] 'Zynq-7000 SoC Data Sheet: Overview (DS190)', p. 25, 2018.

[16] C. A. Lara-Nino, M. Morales-Sandoval, and A. Diaz-Perez, 'Novel FPGA-Based Low-Cost Hardware Architecture for the PRESENT Block Cipher', in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug. 2016, pp. 646–650, doi: 10.1109/DSD.2016.46.

[17] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert, 'Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint', in *Cryptographic Hardware and Embedded Systems – CHES 2012*, Berlin, Heidelberg, 2012, pp. 390–407, doi: 10.1007/978-3-642-33027-8_23.

[18] J. Guo, A. Poschmann, M. Robshaw, and T. Peyrin, 'The LED Block Cipher', 600, 2012. Accessed: Jan. 26, 2020. [Online]. Available: http://eprint.iacr.org/2012/600.

[19] 'Zybo Reference Manual [Reference.Digilentinc]'. https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual?_ga=2.56286312.271517191.1595941507-1987321947.1593002395 (accessed Jul. 28, 2020).

[20] ‘AXI External Peripheral Controller (EPC) v2.0 LogiCORE IP Product Guide (PG127)’, p. 53, 2016.
[21] ‘What is a Block RAM in an FPGA? For Beginners.’ https://www.nandland.com/articles/block-ram-in-fpga.html (accessed Aug. 23, 2020).
[22] ‘MattiasBuelens/avr-crypto-lib’, *GitHub*. https://github.com/MattiasBuelens/avr-crypto-lib (accessed Aug. 10, 2020).
[23] CSEE UMBC, ‘Hello World Example’. https://www.csee.umbc.edu/portal/help/VHDL/samples/hello_world.vhdl (accessed Aug. 12, 2020).
[24] M. Morgan, ‘research_paper/IEEEtran.cls · master · Michael Morgan / masters-project’, *GitLab*. https://gitlab.com/mickmorgan/masters-project/-/blob/master/research_paper/IEEEtran.cls (accessed Aug. 17, 2020).
[25] ‘Quiescent Power’. https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xpa_c_quiescent_power.htm (accessed Aug. 18, 2020).

Appendix F
Source Code Listing

Source Code

Round-based Cipher Core

https://github.com/Xela96/PRESENT-80/tree/master/PRESENT/round-based%20core

Pipelined Cipher Core

https://github.com/Xela96/PRESENT-80/tree/master/PRESENT/pipelined%20core

AXI Peripheral

https://github.com/Xela96/PRESENT-80/tree/master/PRESENT/AXI%20peripheral

Software Development

https://github.com/Xela96/PRESENT-80/tree/master/PRESENT/software