English Explanation of Thesis Paper: Design and Implementation Based on the LEAIoT Cryptographic Algorithm, for Internet of Things (IoT) Applications

Vasileios Aivaliotis

Contents

1 Introduction **1**

1.1 Modular arithmetic 1

1.2 The LEAIoT encryption algorithm 2

2 System design **4**

2.1 Overall system form and functions 4

2.1.1 System blocks 4

2.1.2 System functions 5

2.2 Design and function of individual units 6

2.2.1 n mod 37 calculator 6

2.2.2 3*3 matrix determinant calculator 8

2.3 Modulo 37 inverse calculator 11

2.4 Matrix inverse calculator 13

2.5 Encryption-Decryption unit 13

2.5.1 Encryption 15

2.5.2 Decryption 16

2.5.3 Use of v_add, v_mul and v_mod during key generation . . . 17

3 Simulation and implementation **19**

3.1 Simulation 19

3.1.1 Symmetric key replacement 19

3.1.2 Matrix-key replacement and inversion 20

3.1.3 Encryption and decryption 21

3.2 Implementation 22

List of Figures

2.1 Overall system design 5

2.2 n mod 37 calculator 7

2.3 32 bit $mod37$ calculator 9

2.4 3*3 matrix determinant calculator 10

2.5 Unit calculating matrix I 11

2.6 2*2 matrix determinant calculation unit 12

2.7 Inverse mod 37 calculator 13

2.8 matrix inverse calculator 14

2.9 v_mul. 15

2.10 v_add. 16

2.11 v_mod. 17

2.12 The encryption-decryption unit 18

3.1 32 bit key replacement 20

3.2 64 bit key replacement 20

3.3 128 bit key replacement 20

3.4 256 bit key replacement 21

3.5 Matrix key inversion 21

3.6 Intermediate stages of matrix key inversion 22
3.7 Encryption-decryption 22
3.8 Encryption and decryption require 8 clock cycles 23

List of Tables

3.1 Results of synthesis in terms of material 22

3.2 Finding the clock period 23

3.3 Clock cycles needed for key generation 24

3.4 Key generation time 24

3.5 Proposed design compared to AES candidates 25

3.6 Throughput/slice comparison 25

Chapter 1

Introduction

This text is an english presentation of my thesis, titled Design and implementation based on the LEAIoT cryptographic algorithm, for Internet of things (IoT) applications, which was originaly writen in greek as part of my studies in the Computer Engineering and Informatics Department at the University of Patras. The goal of this text is to explain the design of the implemented circuit and the reasoning behind it, not to provide a full translation.

The goal of the thesis was to create an FPGA implementation of LEAIoT, a lightweight encryption algorithm presented in [1]. In order to achieve this the VHDL hardware description language and the Vivado design suit were used. The original greek text and the VHDL description of the circuit can be found at `https://nemertes.lis.upatras.gr/jspui/handle/10889/14717`.

1.1 Modular arithmetic

Modular arithmetic is a basic element of many modern cryptographic algorithms, including LEAIoT. It is therefore appropriate to give a brief description of it's properties that were used in the proposed implementation.

The division theorem:

- For every integer a and every positive integer n there exist integers q and r so that $n * q + r = a$.

Integer r is the remainder of a and n and it is symbolised as $a \mod n$ or a modulo n.

For integers a_1 and a_2 it can be proven that:

- $(a_1 + a_2) \mod n = [(a_1 \mod n) + (a_2 \mod n)] \mod n$
- $(a_1 - a_2) \mod n = [(a_1 \mod n) - (a_2 \mod n)] \mod n$
- $(a_1 * a_2) \mod n = [(a_1 \mod n) * (a_2 \mod n)] \mod n$

If a,b and n are integers and $a * b \mod n = 1$, integer b is called the modular inverse of $a \mod n$. Ino order for the modular inverse of an integer a mod n to exist it is necessary for a and n to have a greatest common factor of 1 [2].

In linear algebra matrix B is called the inverse of matrix A mod n if $A * B$ mod $n = EYE$ were EYE is the identity matrix.

The inverse $\mod n$ of a matrix (A) is calculated as follows [3]:

- The inverse $\mod n$ (det') of the determinant of A is calculated.
- The intermediate matrix I is calculated by the formula $I[a][b] = ((-1)^{a+b} * d_{a,b})$ were $d_{a,b}$ is the determinant of the matrix that occures by deleting row a and column b of A.
- The inverse (A^{-1}) of A mod n is calculated as: $A^{-1} = (I^T * det') \mod n$.

1.2 The LEAIoT encryption algorithm

As mentioned above the designed circuit was based on the LEAIoT algorithm. LEAIoT uses two keys: a simple symmetric key n and a matrix K which constitutes the block key. Encryption via LEAIoT is achieved as following:

Let p be the plaintext.

- For each character p_i of p $(p_i * n) \mod 37$ is calculated.
- The text that occures is broken in to blocks (arrays) so that each block can be multiplied with K.

- For each block (b_i), $c_i = K * b_i * l \mod 37$ is calculated. l is the length of n.
- The encrypted text is produced by concatenating all the arrays that occur together.

Decryption is achieved as following:

Let c be the encrypted text, $K' = K^(- 1) \mod 37$ and $l' = l^(- 1) \mod 37$ and $n' = n^(- 1) \mod 37$.

- The encrypted text is broken in to blocks (arrays) so that each block can be multiplied with $K^(- 1) \mod 37$.
- For each block (c), $b_i = K' * b_i * l \mod 37$ is calculated. l is the length of n.
- The arrays that occur are concatenated.
- Each character of the occuring text is multiplied by n'. The result is the plaintext.

Chapter 2

System design

In the following design the user can choose weather key n has a length of 32, 64, 128 or 256 bits. Key K is a 3*3 matrix. Because of the use of mod 37 arithmetic the characters of the plaintext and the elements of K can take values from 0 to 36.

2.1 Overall system form and functions

2.1.1 System blocks

The proposed has the form shown in figure 2.1. It's basic blocks are the following:

- ***ENCR_DECR_ROUND*** This unit is the system's "brain". It conatains hardware that executes all the functions required for encrypting and decrypting messages. It also executes modular multiplication and modular addition and sends the data to the appropriate units during kay generation.
- ***MOD37_CALC***: Calculates n mod 37.
- ***DET_MOD37_CALC*** Using as input a 3*3 matrix, this unit produces it's determinant mod 37 and an intermediate matrx I. Each element of i is calculated as $I[a][b] = ((-1)^{a+b} * d_{a,b})$ were $d_{a,b}$ is the determinant of the matrix that occures by deleting row a and column b of A.

- ***MULT_INV_MOD37*** Using as inut either $n \mod 37$ or the determinant of the matrix K $mod 37$ and produces its inverse $mod 37$. The input is chosen via a multiplexer.
- ***MATRIX_INV_CALC*** Using the inverse $\mod 37$ of the determinant of matrix K and the intermediate matrix (I) as input, this unit calculates the inverse of K $mod 37$.
- ***LENGTH_CALC_MOD37 & MULT_INV_MOD37B***: These units calculate the length of n and it's inverse $\mod 37$.

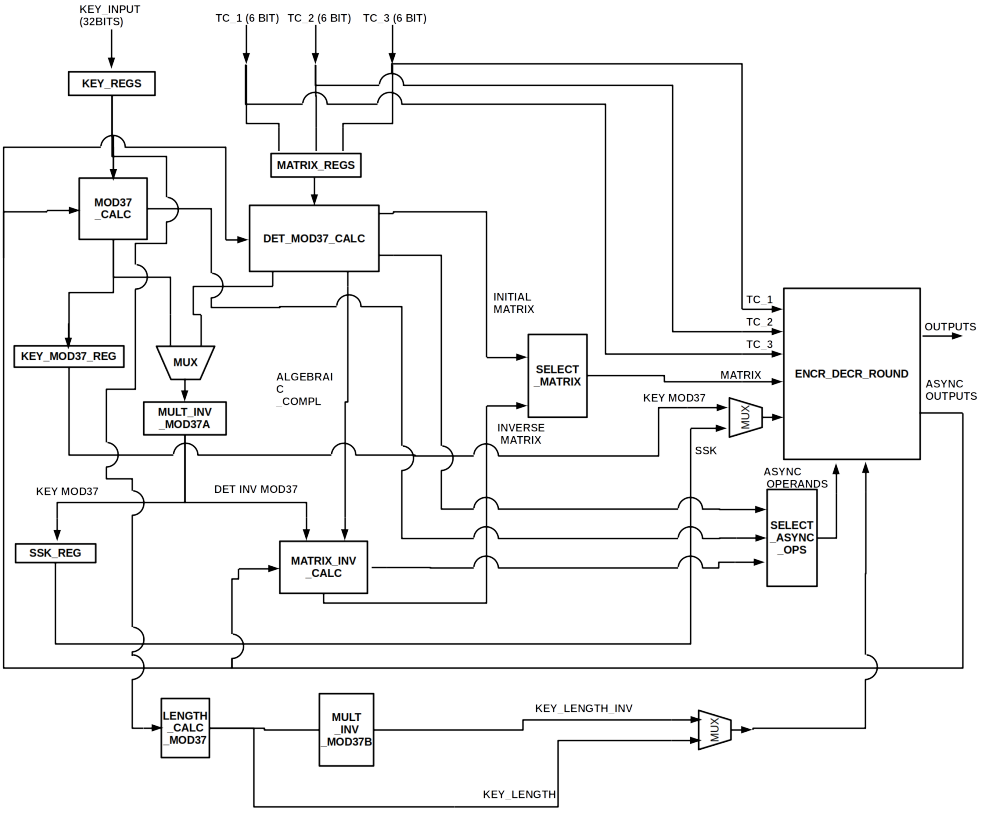

Figure 2.1: Overall system design

2.1.2 System functions

The system can execute four main functions:

- ***Change key n***: The user chooses if the new key will have a length of 32, 64, 128 or 256 bits. The new key n is entered in 32-bit blocks and is stored in the *KEY_REGS*. The content of the *KEY_REGS* enters *MOD37_CALC* were the mod 37 of the key is calculated and is stored in the *KEY_MOD37_REG*. The inverse n' of the key mod 37 is calculated in *MULT_INV_MOD37* and stored in *SSK_REG*. In *LENGTH_CALC_MOD37* and *MULT_INV_MOD37B* the length l of n and it's inverse l' are calculated. When the process is completed '1' is stored in signals **ssk_calc_done ssk_valid**.

- ***Change matrix-key K***: The new K is stored in registers in three cycles. In each cycle one row is writen in the *MATRIX_REGS* through tc_1, tc_2 and tc_3. The determinant of K and the intermediate matrix I are calculated in *DET_MOD37_CALC*. The inverse (det') of the determinant is calculated in *MULT_INV_MOD37A*. I and det' are entered into *MATRIX_INV_CALC* were K' is calculated.

- ***Encryption***: Three characters of the plaintext are entered into *ENCR_DECR_ROUND* through tc_1, tc_2 and tc_3. Matrix K, n mod 37 and l mod 37 are also inserted. The output is the encrypted text.

- ***Decryption***: Three characters of the encrypted text are entered into *ENCR_DECR_ROUND* through tc_1, tc_2 and tc_3. Matrix K', n' and l' are also inserted. The output is the plain text.

Aside from the inputs and outputs shown in figure 2.1 there are inputs that control the length of n and the begining and type of a new process, and outputs that signall the end of replacing a key or the end of encryption and decryption.

2.2 Design and function of individual units

2.2.1 n mod 37 calculator

The this unit's design can be seen on figure 2.2. It's basic element is *mod37_32b*, an asynchronus unit that calculates the mod 37 of a 32 bit number. For keys that are longer than 32 bits, the properties of modular arithmetic that were mentioned abive apply.

The calculation process consists of the following steps:

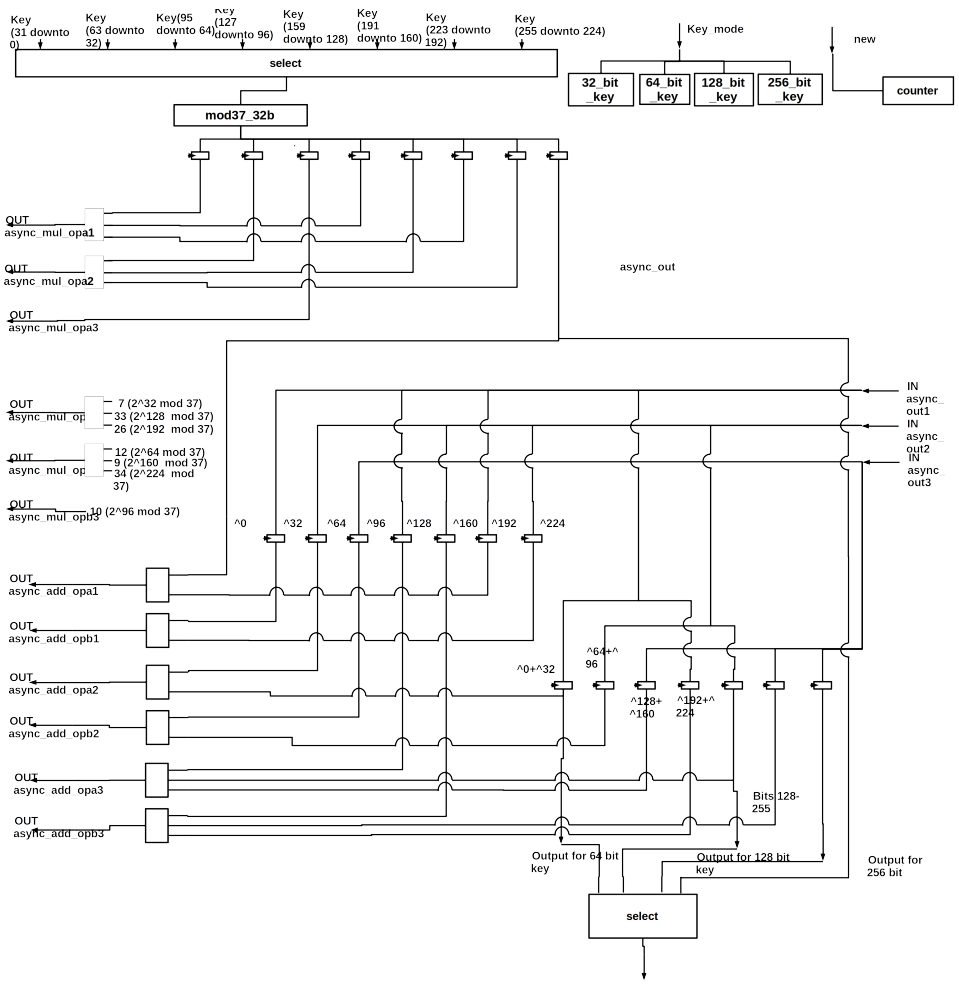

Figure 2.2: n mod 37 calculator

1. The key is broken to 32 bit blocks and the mod 37 of each block is calculated. In the case of the 32 bit key the process is complete.

2. The product mod 37 of each block, except the 32 least significant bits, with a number tha depends on the position of the block in the key is calculated. To avoid a redundancy of material, this calculation takes place in the *ENCR_DECR_ROUND* module, which can complete three mod 37 multiplications per clock cycle.

3. The mod 37 sum of the products mentioned above is calculated in the the *ENCR_DECR_ROUND* module which can complete thee mod 37 additions per cycle.

32 bit *mod*37 **calculator**

This is an asynchronus unit that functions in accordance with the algorithm propsed by Paul Barrett in [4], so if W is an integer then $W \mod 37$ is calculated via the following steps:

1. W is mutiplied by the integer part of $R = 2^{32}/37 = 06EB3E45_{hex}$. The product ($P$) is a 64 bit integer.
2. The 32 least significant bits (P') of P are mulltiplied with 37.
3. $D = W - P''$, were P'' are the 32 leasst significant bits of P'. If $D < 37$ then $D = W \mod 37$. If $37 >= D > 74$ then $D - 37 = W \mod 37$. If $D >= 74$ then $D - 74 = W \mod 37$.

The design of the unit cann be seen in unit 2.3. The multipliers implement Wallace's algorithm ([5]).

2.2.2 3*3 matrix determinant calculator

The input of this unit is a 3*3 matrix (M) and the output is $det \mod 37$ and intermediate matrix I were det is the determinant of M. The determinant is calculated via the relationship $det = M_{11} * D_{11} - M_{12} * D_{12} + M_{13} * D_{13}$ were D_{ab} is the determinant of the 2*2 matrix created by deleting row a and column b of M. By definition $I_{11} = D_{11}$ and $I_{13} = D_{13}$. The calculation follows the steps described bellow.

1. The elements of M are inserted into *alg_compl_mod37* were I is calculated. When I_{11} is calculated, it and M_{11} are send to the encryption-decryption unit were their product $\mod 37$ is calculated and is then stored in a register. $M_{12} * D_{12} \mod 37$ and $M_{13} * I_{13} \mod 37$ are calculated and stored in a similar manner. As it will soon become clear along with I_{12}, D_{12} is also calculated.
2. When I has beeen calculated $M_{11} * I_{11} \mod 37$ and $M_{13} * I_{13} \mod 37$ are sent to the encryption and decryption unit were their sum $\mod 37$ (S) is calculated.
3. $Diff = S - (M_{12} * D_{12} \mod 37)$. If $Diff > 0$ then $Diff = det \mod 37$. It is stored in a register and the calculation is completed.

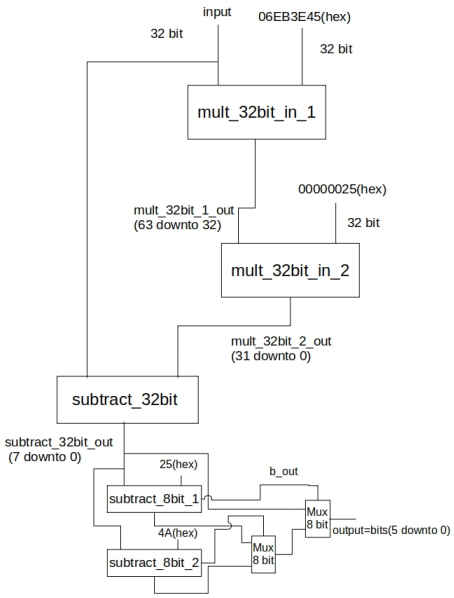

Figure 2.3: 32 bit $mod37$ calculator

4. If $Diff < 0$ then $Diff_1 = (M_{12} * D_{12} \mod 37) - S$ is stored. After this $Diff_f = 37 - Diff_1$ is calculated. $Diff_f$ is the determinatnt $\mod 37$.

5. If $Diff = 0$ the matrix can't be reversed.

Calculating matrix I

The structure of unit *alg_compl_mod37* is shown in 2.5. The elements of the input matrix (M) are stored in registers. To calculate i_{ab}, the elements of the 2*2 matrix that is created by ignoring row a and column b are entered into *det2x2*which calculates the determinant $\mod 37$ of a 2*2 matrix. If $(-1)^{a+b} = 1$, then $I[a][b]$ the output (o) of *det2x2* which is stored in a register. If $(-1)^{a+b} = -1$ Then $I[a][b] = 37 - o$. While calculating i_{12} o is also stored because it is used in calculating the determinant of a 3*3 matrix.

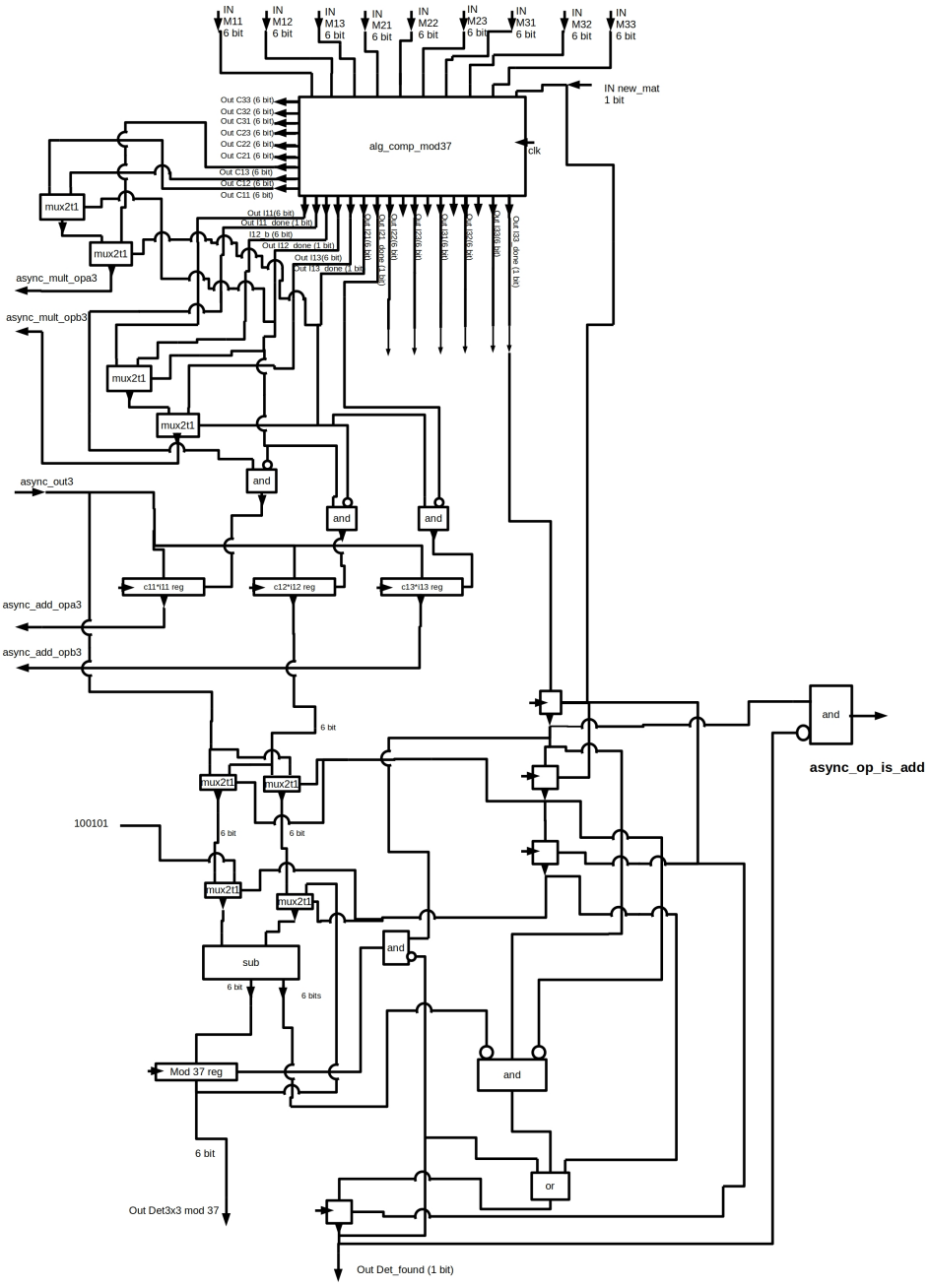

Figure 2.4: 3*3 matrix determinant calculator

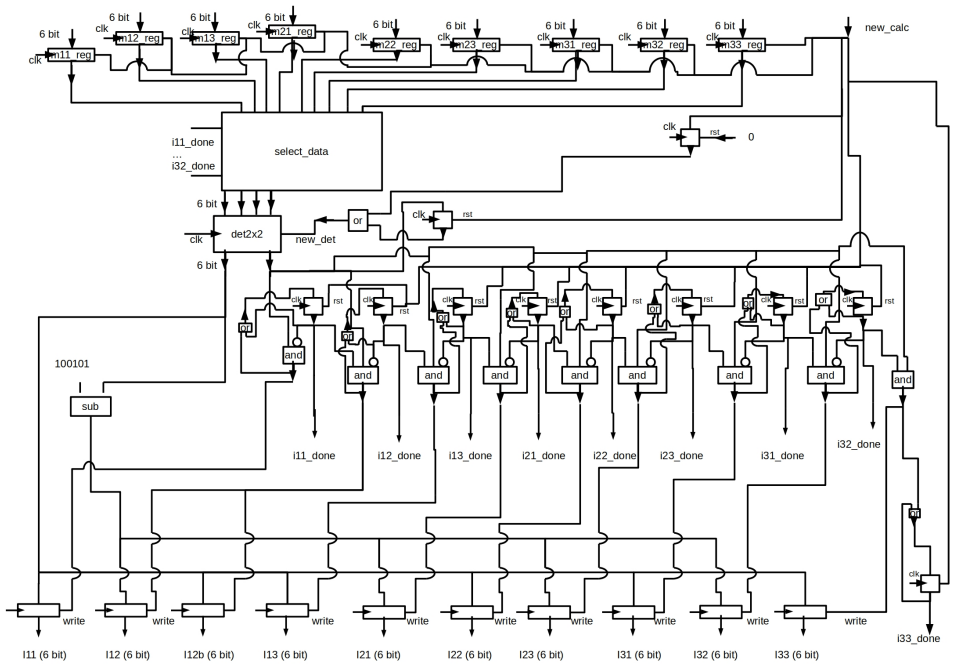

Figure 2.5: Unit calculating matrix I

Determinant of a 2*2 matrix As shown in 2.6, the inut of this unit is a 2*2 matrix (E) and the output is $det(E) \mod 37$. Initialy, the elements of E are stored in register. The products $p_1 = e_{11} * e_{22} \mod 37$ and $p_2 = e_{12} * e_{21} \mod 37$ are calculated in the encryption-decryption unit and then stored in registers. The difference $d_1 = p_1 - p_2$ is calculated. If $d_1 >= 0$ then d_1 is the determinant of E, so it is stored in the output register and the operation is completed. If $d_1 < 0$ then $d_2 = p_2 - p_1$ is calculated and stored in the output register. Then $d' = 37 - d_2$ is calculated and stored in the output register, completing the operation.

2.3 Modulo 37 inverse calculator

Using as input a 6 bit number (a) between 0 and 36 and returns b for which $(a * b) \mod 37 = 1$. As shown in 2.7, a systen of logic gates allows the unit to identify a and b is chosen by a system of multipexers.

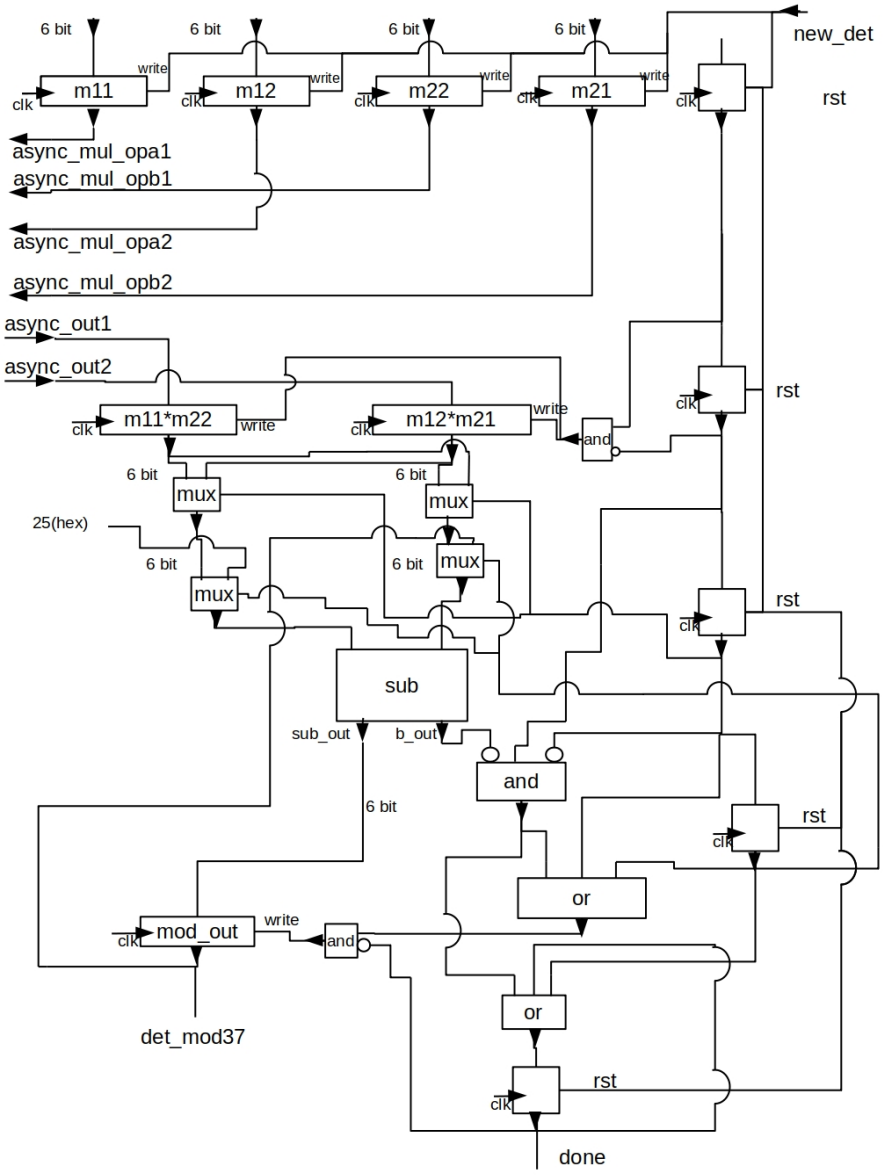

Figure 2.6: 2*2 matrix determinant calculation unit

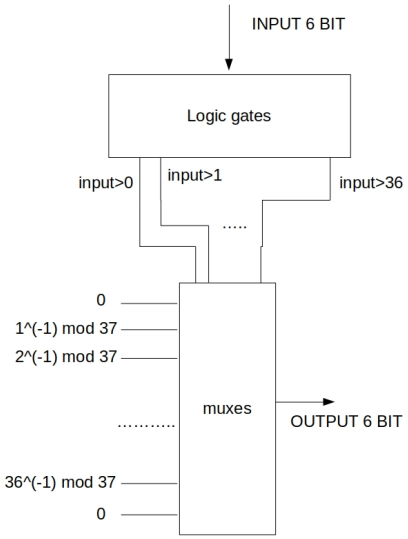

Figure 2.7: Inverse mod 37 calculator

2.4 Matrix inverse calculator

As shown in figure 2.8 this unit uses intermediate matrix I and the inverse mod 37 of the determinant of a matrix (M) as input, and produces the inverse (M') of M using the formula $M' = [I^T * (det(M)^{-1} \mod 37)] \mod 37$. The elements of each row of M' are calculated by multiplying the elements of the same row of I^T with the inverse of the determinant. The multiplication is carried out at the encryption decryption unit which can complete thee operations per clock cycle so the inversion is completed in three cycles.

2.5 Encryption-Decryption unit

The structure of encryption-decryption unit can be seen in figure 2.12. It's inputs are a 3*3 matrix with 6 bit elements (m11,m12,...,m33), three characters

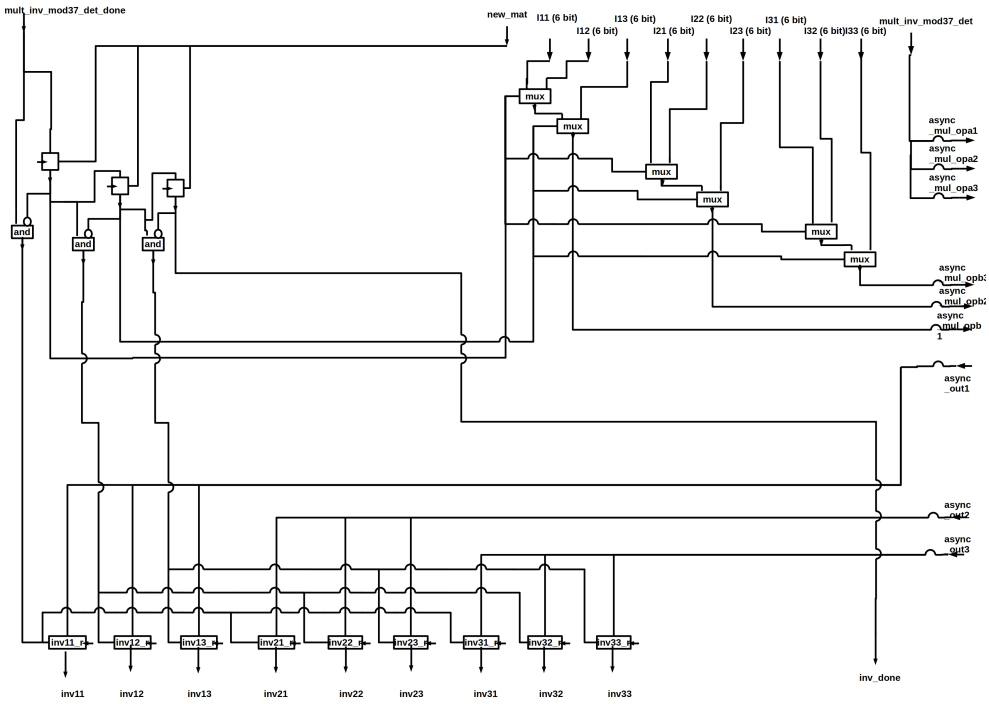

Figure 2.8: matrix inverse calculator

(tc_1, tc_2, tc_3)of the text tha is to be decrypted or encrypted (6 bit each), signal s_ko which is either key $n \mod 37$ or it's inverse $n^{-1} \mod 37$ (6 bits), signal k_lo which is either the length of the key $l \mod 37$ or it's inverse $l^{-1} \mod 37$ (6 bits)and three single bit signals that control the start of a new operation, and weather the new operation is encryption or decryption. Additionaly throuh input ports async_add_opa1, async_add_opa2, async_add_opa3, async_add_opb1, async_add_opb2, async_add_opb3, async_mul_opa1, async_mul_opa2, async_mul_opa3, async_mul_opb1, async_mul_opb2 and async_add_opb3, data that ned tobe added or multiplied during key generation are inserted. Signal async_op_is_add controls weather the opperation that is executed during key generation is addition or multiplication.

Key elements of this unit are v_add, v_mul and v_mod. v_add consists of three parallel adders. v_mul consists of three parallel Wallace multipliers. v_mod consists of three smaler versions of the unit that calculates $\mod 37$ using Barrett's algorithm. Depending on weather the operation beiing executed is addition or multiplication he inputs of v_mod are selected between the outputs of v_add and v_mul.

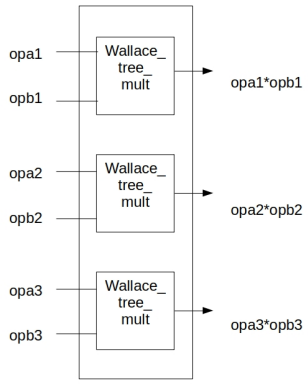

Figure 2.9: v_mul.

2.5.1 Encryption

In the first clock cycle tc_1, tc_2 and tc_3are multiplied by sk_o, in v_mult. The results are used as inputs by v_mod which calculates mod 37. The outputs of v_modare stored in the output registers. In the three next cycles each row of the input matrix is multiplied element by element with the content of the output registers in v_mul. The modulo 37 of is calculated in v_mod and the result is stored in the matrx registers (mr11,...,mr33). In the next two cycles the collumns of the matrix whose elements are stored in (mr11,...,mr33) are added to each other in v_add, their remainder is calculated in v_mod and the result is stored in the output registers. Finaly the content of the output registers is multiplied mod 37 with kl_o, the result is stored in the output registers and a signal marking the end of the process is generated.

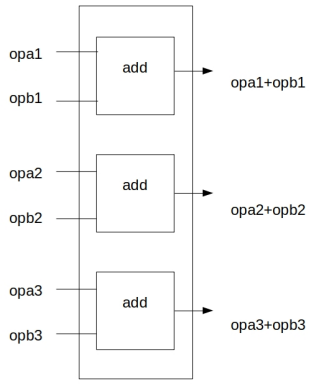

Figure 2.10: v_add.

2.5.2 Decryption

In the first three cycles the rows of the matrix are multiplied element by element with array [tc_1, tc_2, tc_3]. the modulo 37 of each product is calculated and the results are stored in the matrix registers (mr11,...,mr33). In the next two cycles the collumns of the matrix whose elements are stored in (mr11,...,mr33) are added to each other in v_add, their remainder is calculated in v_mod and the result is stored in the output registers. The content of the output registers is multiplied with k_lo and the remainder of the product mod 37 is stored in the output registers. The content of the output registers is multiplied by s_ko, the product mod 37 is stored in the output registers and a signal marking the end of the process is generated.

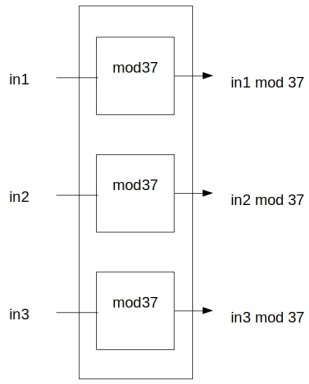

Figure 2.11: v_mod.

2.5.3 Use of v_add, v_mul and v_mod during key generation

During encryption, decryption and key generation there are many mod 37 additions and multiplication. In order to avoid hardware redundancy the encrytpion-decryption unit allows the rest of the units to "borrow" it's modular arithmetic hardware. During key generation the encryption-decryption unit takes as input the data tha must be processed through input ports async_add_opa1, async_add_opa2, async_add_opa3, async_add_opb1, async_add_opb2, async_add_opb3, async_mul_opa1, async_mul_opa2, async_mul_opa3, async_mul_opb1, async_mul_opb2 and async_add_opb3. Input signal async_op_is_add determines weather the operation that is executed is addition or multiplication. The results are returned to the other units via output ports async_out1, async_out2 and async_out3.

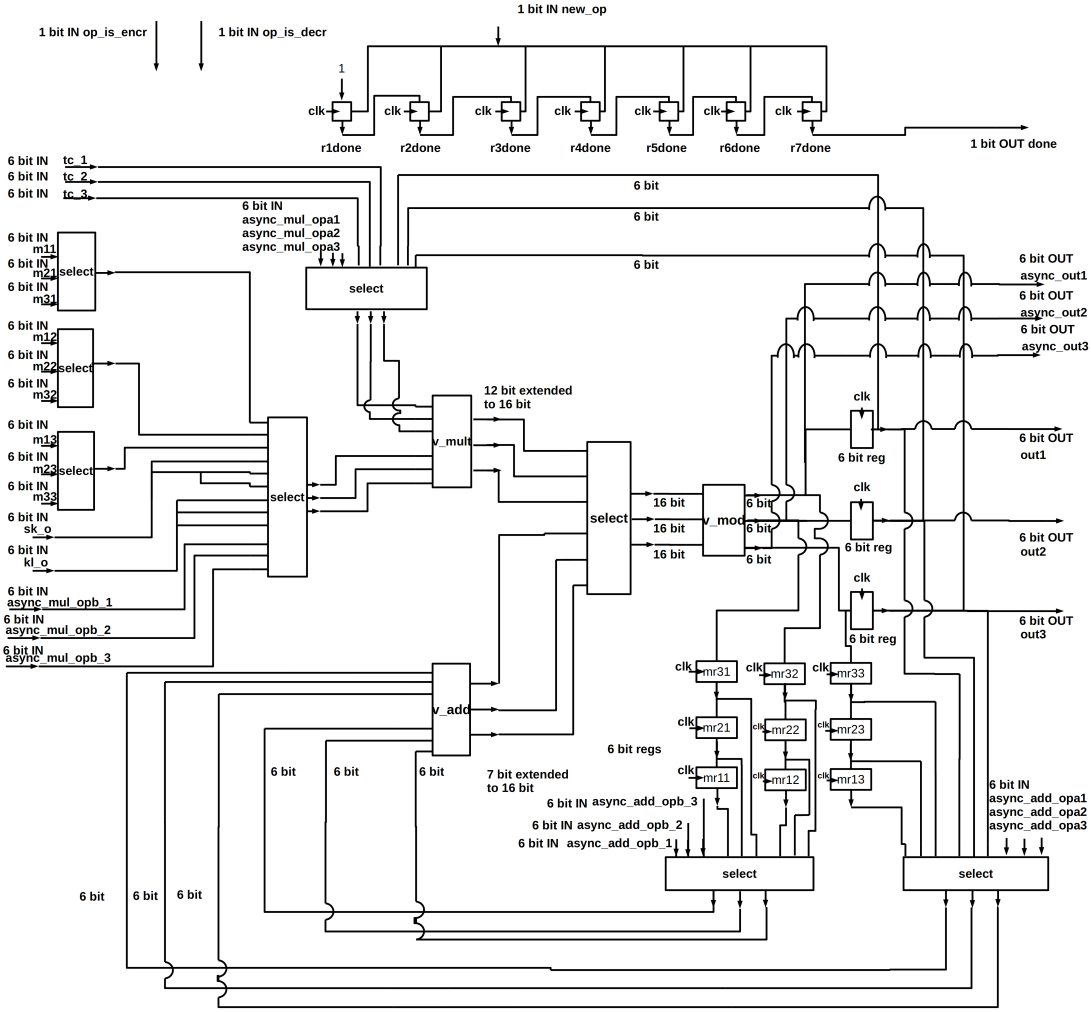

Figure 2.12: The encryption-decryption unit

Chapter 3

Simulation and implementation

The system whose design is described in the previous chapter was described in the VHDL language. The ghdl simulator was used to simulate it's behaviour. For synthesis and implementation Xilix's Vivado webpack 2018.3 was used.

3.1 Simulation

3.1.1 Symmetric key replacement

In the following figures the processing of changing replacing the symmetric key (*private_key*) of 32, 62, 128 and 256 bits.

$00003039_{hex} (= 12345_{dec})$ was used as the 32 bit key. As shown in reffig:inv32, operation 4_{hex} is run first. This is controled by signal *new_op_type*. Operation 4_{hex} replaces the 32 least significant bit's of *private_key*. After this operation 0_{hex}, which produces the inverse (ssk) mod 37 of the key is run. After the operation is completed the signals ssk_calc_done and ssk_valid take 1 as their value. In this particular example 00003039_{hex} was used as *private_key* and the resultinfg ssk was 11_{hex}.

In the case of a 64 bit *private_key*, operations 4_{hex} and 5_{hex} are run first to renew bits [0-31] and [32-63] of *private_key*. After those are completed operation 0_{hex} is run, as shown in 3.2. The inversion process lasts for more clock cycles than for a 32 bit key because of the extra cloc cycles needed in calculating the modulo 37 of a 64 bit key.

A similar process is also followed for 128 and 256 bit keys as can bei seen in figures 3.3 and 3.4. In the case of a 128 bit key operations $4_{hex}...7_{hex}$ are run before the inversion of the key. In the case of a 256 bit key operations $4_{hex}...B_{hex}$ are run before the inversion of the key.

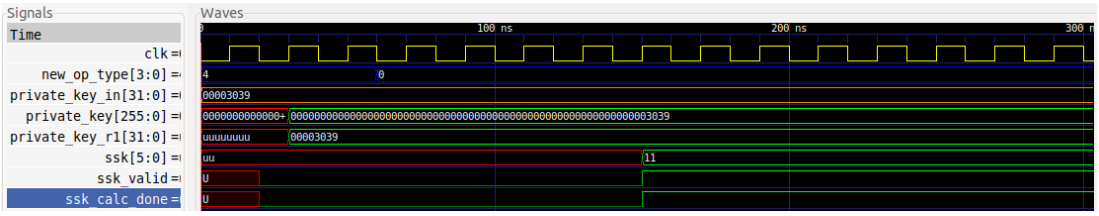

Figure 3.1: 32 bit key replacement

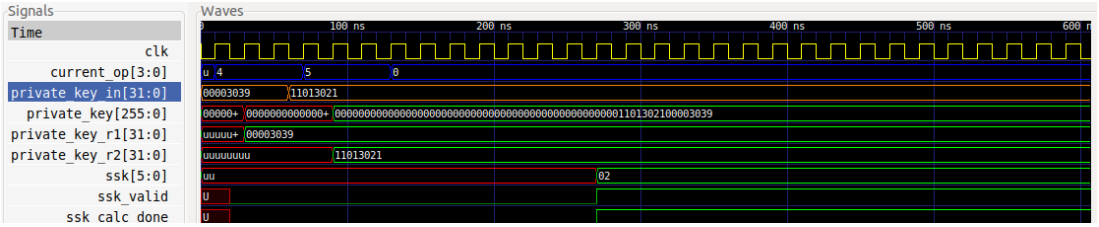

Figure 3.2: 64 bit key replacement

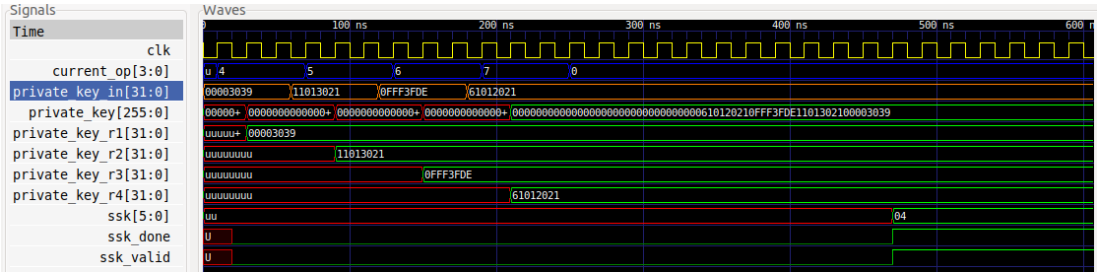

Figure 3.3: 128 bit key replacement

3.1.2 Matrix-key replacement and inversion

In this section $m = \begin{bmatrix} 1 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}_{hex}$ becomes the new matrix-key.

As shown in figure 3.5, processes C_{hex}, D_{hex} and E_{hex} are executed first, in order to write the rows f the matrix to registers whose outputs are m11,...m33.

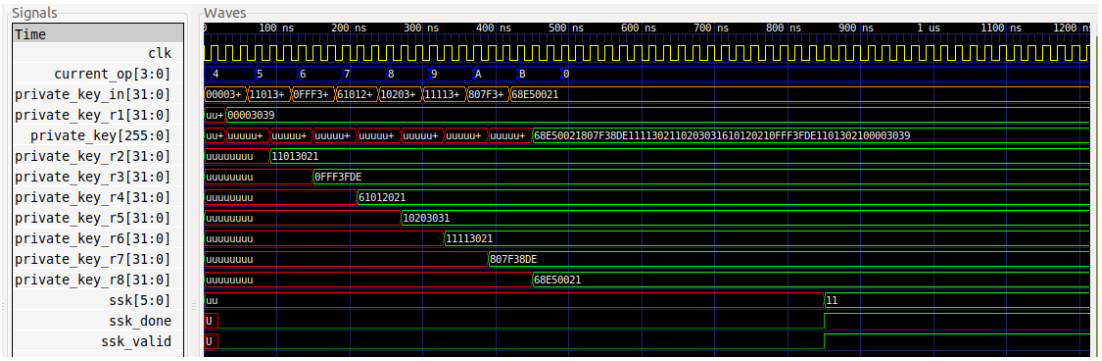

Figure 3.4: 256 bit key replacement

Figure 3.5: Matrix key inversion

Process 1_{hex} is executed to inverse the matrix-key. The inverted matrix's elements are stored in registers with outputs: inv11,...inv33.

Intermediate matrix I and the determinant of the matrix key can be seen in figure 3.6.

3.1.3 Encryption and decryption

Figure 3.7 shows the encyption and decryption of four blocks of plaintext whose synthetic vaues are: [$0D_{hex}$, 13_{hex}, $0F_{hex}$], [06_{hex}, 06_{hex}, 09_{hex}], [03_{hex}, 05_{hex}, $1D_{hex}$] and [$1B_{hex}$, $1C_{hex}$, $1E_{hex}$]. $00003039_{hex0} = 12345_{dec}$ was the symmetric

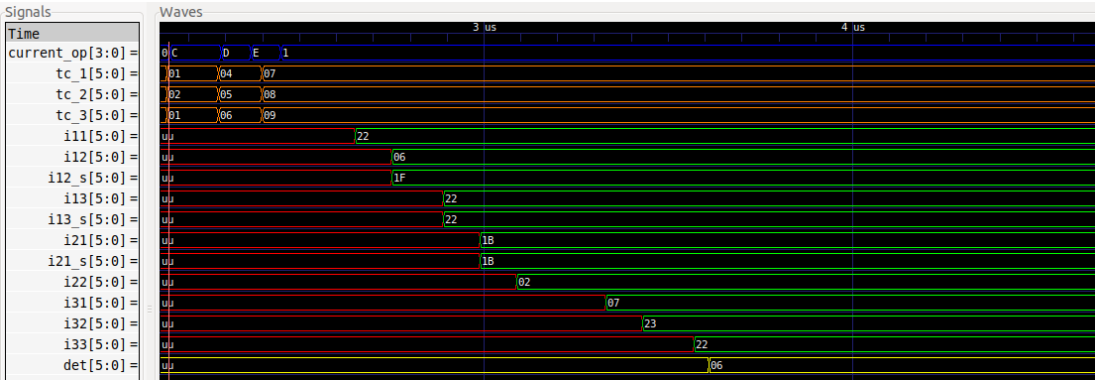

Figure 3.6: Intermediate stages of matrix key inversion

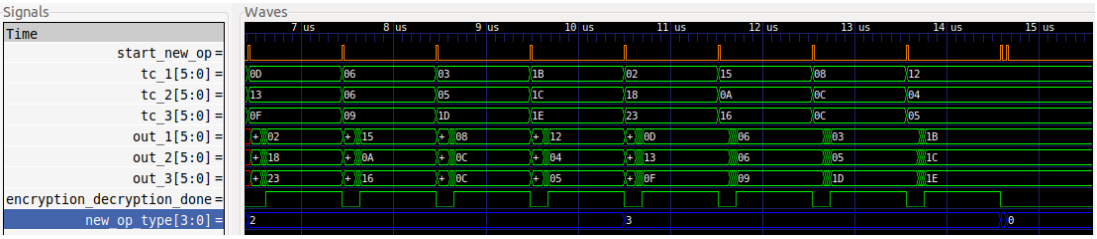

Figure 3.7: Encryption-decryption

key. The matrix-key was:

$$\begin{bmatrix} 1 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}_{hex}.$$

Encryption is process 2_{hex} and decryption is process 3_{hex}.

3.2 Implementation

The VHDL code was that describes the system was inputed to the Vivado webpack 2018.13 using the FPGA 7z007s-clg400 as a base. The results of synthesis in terms of material are shown in table 3.1.

LUTs	**flip-flops**	**f7-muxes**	**f8-muxes**
2700	815	67	24

Table 3.1: Results of synthesis in terms of material

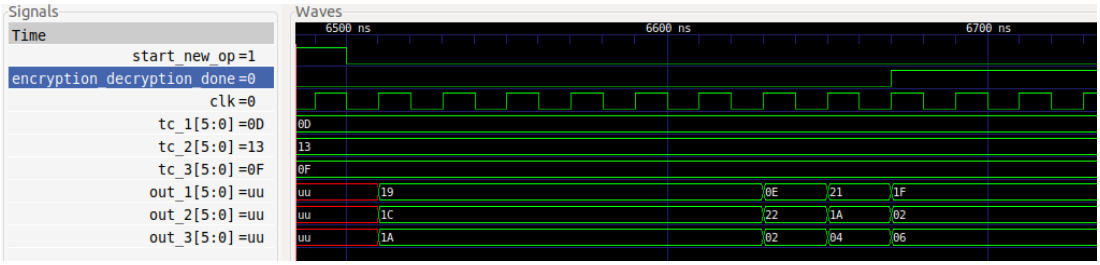

Figure 3.8: Encryption and decryption require 8 clock cycles

To calculate the duration of a clock cycle (T) the system is implemented with gradualy decreasing period on the constraints file. The optimal period for the circuit is that for which Total Negative Slack (TNS) is zero while Worst Negative Slack (WNS), Worst Hold Slack (WHS)and Worst Pulse Width Slack (WPWS) have positive values. This was found to be 29 ns.

Period	**30 ns**	**29 ns**
Frequency	33.333 MHz	34.483 MHz
TNS	0 ns	0 ns
WNS	1.317 ns	0.328 ns
WHS	0.138 ns	0.146 ns
WPWS	1309 ns	1309 ns

Table 3.2: Finding the clock period

Performance

As figure 3.8 shows encryption and decryptioon take eight clock cycles each. One extra clock cycle is needed for the encryption and decryption process to be initated. In each iteration of the process three 6 bit characters are encrypted or decrypted. Considering what was mentioned above, as well as the fact that each clock cycle lasts for 29 ns it is concluded that in 261 ns. Therefore the system has a throughput of 68,965 Mbps.

Proposed design compared to AES candidates

In [6], implementations of the five final candidates of the competition by NIST(National Institute of Standards and Technology) to define the AES (Advanced Encryption Standard) are compared regarding to material usage, encryption-decryption time and key generation time. The 128 key version of each algorithm were considerd

As can be told from table 3.4 the proposed system is inferior To the AES

The clock cycles needed to complete key generation for diferent lengths of symmetric key are shown in table 3.2. The total time is shown intable 3.3.

Key Length (bit)	**Writing symmetric key**	**Inversion of symmetric key**	**writing matrix key**	**Matrix key inversion**	**Total**
32	2	5	6	61	74
64	4	8	6	61	79
128	8	12	6	61	87
256	16	19	6	61	102

Table 3.3: Clock cycles needed for key generation

Key length (bits)	**Key generation time (ns)**
32	0,002146
64	0,002291
128	0,002523
256	0,002958

Table 3.4: Key generation time

candidates when it comes to ecryption-decryption speed. It uses less material than Rijndael, which won the competition, but as can be seen in table 3.5 the Rinjdael implementation achieves better throughput. The proposed implementation achieves better throughput than the MARS and TWOFISH implementations. In table 3.4 it is shown that the proposed design achieves better key generation times than the AES candidates, even when a larger key is used.

System	Key length (bits)	Throughput (Mbps)	Slices	Key generation (sec)
Rijndael	128	353,00	5673	0,07
MARS	128	101,88	6896	1,96
RC6	128	112,87	2650	0,17
Serpent	128	148,95	2550	0,9
Twofish	128	173,06	9363	0,18
Proposed	32	68,965	3606	$0,002146 * 10^{-3}$
Proposed	64	68,965	3606	$0,002291 * 10^{-3}$
Proposed	128	68,965	3606	$0,002523 * 10^{-3}$
Proposed	256	68,965	3606	$0,002958 * 10^{-3}$

Table 3.5: Proposed design compared to AES candidates

Implementation	$\frac{\text{throughput}}{\text{slices}}$
Rijndael	0,0622
MARS	0,0147
RC6	0,0425
Serpent	0,0584
Twofish	0,0184
Proposed	0,0191

Table 3.6: Throughput/slice comparison

Bibliography

[1] Muhammad Asif Habib et al. "Speeding up the internet of things: Leaiot: A lightweight encryption algorithm toward low-latency communication for the internet of things". In: *IEEE Consumer Electronics Magazine* 7.6 (2018), pp. 31–37.

[2] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory (2nd Edition)*. USA: Prentice-Hall, Inc., 2005. ISBN: 0131862391.

[3] Βασίλειος Αν Κάτος και Γεώργιος Χρ Στεφανίδης. *Τεχνικές Κρυπτογραφίας και Κρυπτανάλυσης*. Θεσσαλονίκη: Ζύγός, 2003.

[4] Paul Barrett. "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 311–323. ISBN: 978-3-540-47721-1.

[5] Chris S. Wallace. "A Suggestion for a Fast Multiplier". In: *IEEE Transactions on Electronic Computers* EC-13.1 (1964), pp. 14–17. DOI: 10.1109/PGEC.1964.263830.

[6] Andreas Dandalis, Viktor K. Prasanna, and Jose D.P. Rolim. "A Comparative Study of Performance of AES Final Candidates Using FPGAs". In: *Cryptographic Hardware and Embedded Systems — CHES 2000*. Ed. by Çetin K. Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 125–140. ISBN: 978-3-540-44499-2.