FPGA realization of a public key block cipher

Stig Fjellskaalnes

Master of Science in Computer Science
Submission date: June 2009
Supervisor: Danilo Gligoroski, ITEM
Co-supervisor: Mohamed El-Hadedy, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Recently, a new public key algorithm have been proposed by Gligoroski, Markovski and Knapskog. The algorithm belongs to the class of public keys algorithms realized by multivariate quadratic equations. The authors found out a new class of quasigroups that have special form when expressed as Boolean functions. The quasigroups are multivariate quadratic.

One important characteristic for this new public key algorithm is that it is very fast. Realized in software it can produce digital signatures around 300 times faster than RSA (1024 bit public key length). However, in hardware the algorithm can achieve speeds equivalent to symmetric key primitives both in signature generation and in its verification. That means the algorithm realized in hardware can be 1,000 to 10,000 times faster than corresponding public key algorithms (RSA, Diffie Helman or Elliptic Curve algorithms) realized also in hardware.

The student will have a task to write a VHDL code and to realize the algorithm in FPGA, both encryption and decryption. The realization will use variable public and private keys stored in RAM, not fixed keys stored in ROM blocks.

Assignment given: 15. January 2009
Supervisor: Danilo Gligoroski, ITEM

Abstract

This report will cover the physical realization of a public key algorithm based on multivariate quadratic quasigroups. The intension is that this implementation will use real keys and data. Efforts are also taken in order to reduce area cost as much as possible. The solution will be described and analyzed. This will show wether the measures were successfull or not.

Contents

1 Introduction **9**

2 Theory **11**

2.1 MQQ in general . . . 11

2.2 Logic optimization through minimization . . . 11

2.2.1 Minimization with Karnaugh maps . . . 13

2.2.2 ESPRESSO-II minimization algorithm . . . 13

3 Hardware implementation of MQQ **17**

3.1 Hardware implementations in general . . . 17

3.2 The actual implementation . . . 18

3.2.1 Encryption . . . 18

3.2.2 Decryption . . . 23

3.3 Data and keys . . . 28

3.4 Optimizing the stored fixed values . . . 28

4 Results **31**

4.1 Synthesis . . . 31

4.2 Verification of functionality through simulation . . . 31

4.2.1 Encryption . . . 31

4.2.2 Decryption . . . 33

5 Discussion and conclusion **39**

5.1 Discussion . . . 39

5.1.1 Synthesis . . . 39

5.1.2 Design behaviour simulation and verification . . . 40

5.2 Conclusion . . . 40

5.3 Future work . . . 40

5.4 Contributors . . . 40

6 Abbrevations **41**

A VHDL source code **45**

A.1 Encryption . . . 45

A.2 Decryption . . . 53

B Espresso minimization **67**

C Synthesis report from Decryption **69**

List of Figures

2.1 Karnaugh diagram with reduction 14

3.1 Encryption top level 18
3.2 Encryption state diagram 19
3.3 Expander internal architecture 21
3.4 Expander state diagram 21
3.5 Public Matrix internal architecture 22
3.6 Public Matrix state diagram 23
3.7 Decryption internal architecture 24
3.8 Decryption top level state diagram 24
3.9 Private Matrix internal architecture 25
3.10 Private Matrix state diagram 25
3.11 Dobbertin internal architecture 27
3.12 Sequencer internal architecture 27
3.13 Sequencer state diagram 28
3.14 Master Read Only Memory (ROM) internal architecture 29

4.1 Encryption counter synchronization 32
4.2 Encryption top module missing state 32
4.3 Expander behaviour 33
4.4 Public matrix calculation 34
4.5 Public matrix bit-by-bit XOR 34
4.6 Simulation of startup 35
4.7 Stored private key as a function 35
4.8 XOR procedure 35
4.9 Dobbertin calculation 36
4.10 Sequencer startup 36
4.11 Master ROM calculation 37

List of Tables

2.1 Key generation algorithm 12
2.2 Encryption in Multivariate Quadratic Quasigroups (MQQ) 12
2.3 MQQ public key for 160 bit MQQ 12
2.4 Decryption procedure 13
2.5 Truth table 14
2.6 ESPRESSO-II minimization algorithm 15

3.1 Expansion 20
3.2 Espresso input file format 29

4.1 Synthesis results of the MQQ decryption procedure 31
4.2 Synthesis results of the MQQ encryption procedure 32

Chapter 1

Introduction

This thesis will cover an attempt to realize a fairly new public key algorithm on an Field-progmable Gate Array (FPGA), the MQQ algorithm. Previous attempts of implementing this algorithm has shown that the area consumption of this design is enormous, and must be reduced in order to implement this algorithm in hardware in a practical manner. There are optimization techniques that can, in theory, be used to store the information more efficient than storing it directly, which will be investigated.

In order to compare the results of the optimization, the decryption design from the TTM4530 report [8] written by the same author of this master thesis will be used as a reference.

Chapter 2

Theory

In this chapter the theoretical background for the techniques used in the implementation will be presented.

2.1 MQQ in general

MQQ is, compared to Rivest-Shamir-Adleman (RSA), Diffie-Hellman (DH) and Elliptic curve cryptography (ECC), a new type of public key algorithm. Calculations of encryption and decryption are done with logic operations such as AND and XOR between the actual data and the public and private keys. This is a high contrast to tradidional public key algorithms, where encryption and decryption is done by using more complex mathematical operation. In RSA [1], encryption of a message is done the following way, $c = m^e \pmod n$, where c is the encrypted message, m is the original message, (n, e) is the public key. Encryption is done the same way, $m = c^d \pmod n$, d being the private key exponent. The MQQ public key consists of boolean values arranged in a n×n matrix generated randomly. The MQQ private key is derived from the output of the public key. It follows the following procedure, given in table 2.1.

The MQQ encryption is performed with an expansion of the input data. Then each term is anded with the respective term from the private key. The number of bits determines the number of equations that are to be performed. More bits mean more equations. With 160 bits input data, there will be 160 equations, and each equations have 12881 terms. The result of this operation is joined in a resultant vector, which will be the encrypted data, as shown in table 2.1. This is more thouroughly described in the implementation of the encryption scheme. '+' is here an XOR, and multiplication. Basically, the encryption can be represented as the equation $y = \mathbf{P}(x) \equiv y = \mathbf{A} \cdot X$.

For 160 bit MQQ the public key is defined as a matrix of 160×12881 elements, in the form presented in table 2.1.

Decryption is described in table 2.1. The decryption is done in seven stages total where logical operations (AND, XOR) are done with the input value, and the result is the original data in cleartext.

2.2 Logic optimization through minimization

As described, the public and private keys in MQQ are boolean values stored in matrices with considerable size. In earlier impelmentations of the MQQ the keys have been represented as fixed numbers. Logic optimization and minimization methods can be used to reduce the storage needs for these blocks. This report will explore if this is the case with MQQ.

Algorithm for generating Public and private keys for the MQQ scheme
Input: Integer n, where n = 5k and k $\geq$ 28
Output: public key P: n multivariate quadratic polynomials $P_i(x_1, ..., x_n)$, $i = 1, ..., n$
1. Generate a nonsingular $n \times n$ boolean matrix T (uniformly at random). 2. Call the procedure of definition for $P'(n) : 0, 1^n \rightarrow 0, 1^n$ and from there also obtain the quasigroups $*_1, ..., *_8$ 3. Compute $y = T(P'(T(x)))$ where $x = x_1, ..., x_n$ 4. Output: the public key is y as n multivariate quadratic polynomials $P_i(x_1, ..., x_n), i = 1, ..., n$ and the private key is the tuple $T, *_1, ..., *_8$

Table 2.1: Key generation algorithm

MQQ encryption
$a_0{}^{(1)} + (a_1{}^{(1)} \times x_1) + (a_2{}^{(1)} \times x_2) + ... + (a_{12881}{}^{(1)} \times x_{159} \times x_{160})$ $a_0{}^{(2)} + (a_1{}^{(2)} \times x_1) + (a_2{}^{(2)} \times x_2) + ... + (a_{12881}{}^{(2)} \times x_{159} \times x_{160})$ $a_0{}^{(3)} + (a_1{}^{(3)} \times x_1) + (a_2{}^{(1)} \times x_2) + ... + (a_{12881}{}^{(3)} \times x_{159} \times x_{160})$ . . . $a_0{}^{(160)} + (a_1{}^{(160)} \times x_1) + (a_2{}^{(160)} \times x_2) + ... + (a_{12881}{}^{(160)} \times x_{159} \times x_{160})$

Table 2.2: Encryption in MQQ

160 bit MQQ public key
$a_0{}^{(1)}\ a_1{}^{(1)}\ a_2{}^{(1)}\ a_3{}^{(1)}\ a_4{}^{(1)}\ ...\ a_{12881}{}^{(1)}$ $a_0{}^{(2)}\ a_1{}^{(2)}\ a_2{}^{(2)}\ a_3{}^{(2)}\ a_4{}^{(2)}\ ...\ a_{12881}{}^{(2)}$ $a_0{}^{(3)}\ a_1{}^{(3)}\ a_2{}^{(3)}\ a_3{}^{(3)}\ a_4{}^{(3)}\ ...\ a_{12881}{}^{(3)}$ $a_0{}^{(4)}\ a_1{}^{(4)}\ a_2{}^{(4)}\ a_3{}^{(4)}\ a_4{}^{(4)}\ ...\ a_{12881}{}^{(4)}$ . . . $a_0{}^{(160)}\ a_1{}^{(160)}\ a_2{}^{(160)}\ a_3{}^{(160)}\ a_4{}^{(160)}\ ...\ a_{12881}{}^{(160)}$

Table 2.3: MQQ public key for 160 bit MQQ

Algorithm for decryption/signing with the private key $(T, *_1, ..., *_8)$
Input: A vector $y = (y_1, ..., y_n)$
Output: A vector $x = (x_1, ..., x_n)$ such that $\mathbf{P}(x) = y$
1. Set $y' = T^{-1}(y)$ 2. Set $W = (y'_1, y'_2, y'_3, y'_4, y'_5, y'_6, y'_{11}, y'_{16}, y'_{21}, y'_{26}, y'_{31}, y'_{36}, y'_{41}) = \mathbf{Dob}^{-1}(W)$ 3. Compute $Z = (Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_{11}, Z_{16}, Z_{21}, Z_6, Z_{31}, Z_{36}, Z_{41})$ 4. Set $y_1 \leftarrow Z_1$, $y_2 \leftarrow Z_2$, $y_3 \leftarrow Z_3$, $y_4 \leftarrow Z_4$, $y_5 \leftarrow Z_5$, $y_6 \leftarrow Z_6$, $y_{11} \leftarrow Z_7$, $y_{16} \leftarrow Z_8$, $y_{21} \leftarrow Z_9$, $y_{26} \leftarrow Z_{10}$, $y_{31} \leftarrow Z_{11}$, $y_{36} \leftarrow Z_{12}$, $y_{41} \leftarrow Z_{13}$ 5. Represent y' as $y' = Y_1...Y_k$ where Y_i are vectors of dimension 5 6. By using the left parastrophes $\backslash_i$ of the quasigroups $*_i$, $i = 1, ..., 8$, obtain $x' = X_1...X_k$, such that: $X_1 = Y_1$, $X_2 = X_1 \backslash_1 Y_2$, $X_3 = X_2 \backslash_2 Y_3$ and $X_i = X_{i-1} \backslash_{3+((i+2) \pmod 6)} Y_i$ 7. Compute $x = S^{-1}(x')$

Table 2.4: Decryption procedure

2.2.1 Minimization with Karnaugh maps

Karnaugh maps are widely used when it comes to minimizing and optimizing logical expressions, and to ease the use of boolean algebra. Given the following truth table 2.2.1, this table will be translated to a karnaugh map [5]. The expression can further be reduced, and the result Y from the truth table can be expressed as the boolean function $Y = aB + bD + Cd$ (big capitals = negation).

2.2.2 ESPRESSO-II minimization algorithm

Since the key size of MQQ for 160 bit is big, there is another, more efficient method for minimizing large matrices of boolean values, by using an algorithm called ESPRESSO. The ESPRESSO-II minimization algorithm has been implemented as a lightweight program. The algorithm is listed in table 2.2.2.

Here follows a brief presentation of the different procedures in ESPRESSO-II minimization algorithm [4]. The algorithm starts with an UNWRAP, which is a preprocessor that has to discover any incoming cube sharing whatever may be present in the incoming data. COMPLEMENT computes R or D if F and R are given as the input. EXPAND replaces the cubes of F by prime implicants and makes sure that coverage is minimal as to single-cube containment. The consequence will be that EXPAND reduces the number of cubes in F. The routine ESSENTIAL_PRIMES locates the essential primes which must be present in every cover of F. When detected, they are added to the don't-care set D, which prevents the primes from appearing more than one time. This routine is only executed during the first iteration of LOOP1. IRREDUDANT_COVER sorts the covers of F into totally redundant, relatively essential and partially redundant. All cubes that are totally redundant are discarded, and a minimal subset

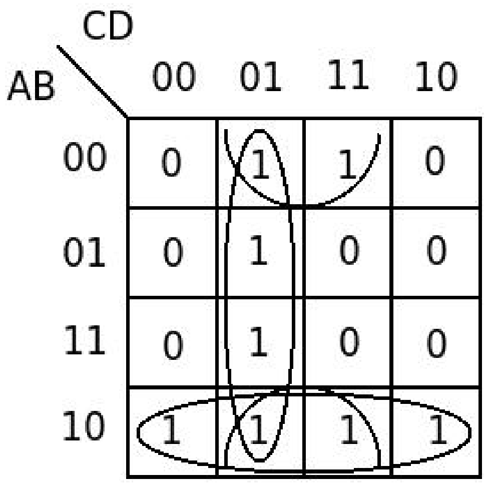


Figure 2.1: Karnaugh diagram with reduction

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Table 2.5: Truth table

```
Begin
$F \leftarrow UNWRAP(F)$
$R \leftarrow COMPLEMENT(F, D)$
$\phi 1^* \leftarrow \phi 2^* \leftarrow \phi 3^* \leftarrow \phi 4^* \leftarrow COST(F)$

LOOP1: $(\phi, F) \leftarrow EXPAND(F, R)$
       if (first - pass)
          $(\phi, F, D, E) \leftarrow ESSENTIAL\_PRIMES(F, D)$
       if $(\phi \equiv \phi 1^*)$ goto OUT
          $\phi 1^* \leftarrow \phi$
          $(\phi, F) \leftarrow IRREDUDANT\_COVER(F, D)$
       if $\phi \equiv \phi 2^*$ goto OUT
          $\phi 2^* \leftarrow \phi$

LOOP2: $(\phi, F) \leftarrow REDUCE(F, D)$
       if $\phi \equiv \phi 3^*$ goto OUT
          $\phi 3^* \leftarrow \phi$
          goto LOOP1

OUT: if $(\phi \equiv \phi 4^*)$ goto QUIT
       $(\phi', F) \leftarrow LAST\_GASP(F, D, R)$
       if $(\phi \equiv \phi')$ goto QUIT
          $\phi 1^* \leftarrow \phi 2^* \leftarrow \phi 3^* \leftarrow \phi 4^* \leftarrow \phi'$
          goto LOOP2

QUIT: $F \leftarrow F \cup E$
       $D \leftarrow D - E$
       $(\phi, F) \leftarrow MAKE\_SPARSE(F, D, R)$
return $(\phi, F)$
End
```

Table 2.6: ESPRESSO-II minimization algorithm

of the two other types plus D will be sufficient to cover all minterms for F. REDUCE improves the result over the local minimums that IRREDUDANT_COVER obtaines. This is done by taking each cube $c \in F$ and then reducing it to the smallest cube $\underline{c}$. LAST_GASP is reminiscent of REDUCE, but uses an order independent reduction process. The ESPRESSO-II algorithm finishes with MAKE_SPARSE. Essential primes are first taken out of the don't-care set D and put back in the cover F. Then the procedure considers the number of cubes in the cover as final. It also attempts to reduce the number of literals by "lowering" the outputs and "raising" the inputs. MAKE_SPARSE also attempts to make the final cover minimal, in a way that no input literal, output literal or product term can be removed while retaining coverage of ff.

Chapter 3

Hardware implementation of MQQ

Since the assignment is to realize the MQQ algorithm in hardware, typically an FPGA, an implementation has been made. This chapter will describe, in detail, a hardware implementation of MQQ written in the hardware description language Very-High-Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL). Both the encryption and decryption has been implemented. As mentioned earlier, since the implementation is intended for realization, optimization and minimization techniques has been tried in order to reduce area cost on the FPGA. Xilinx ISE 10.1 (with all updates and service packs installed) has been used as the Integrated Development Environment (IDE) for this design. ISE also contains the synthesis tool Xilinx Synthesis Tool (XST), which is necessary in order to prepare the design for upload to FPGA. There exists an earlier hardware implementation of MQQ written by Mohamed El-Hadedy. That implementation was developed with emphasis on speed only, no area reduction efforts were made. In this design, a real private key has been used. But, the public key is a randomly generated key with no relation to the private key, because the real public key was not avaliable. This report has the shortened verson of the VHDL source code, the complete source code files are located in the digital attatchment.

3.1 Hardware implementations in general

In order to fully demonstrate the speed of an algorithm, a hardware implementation is needed. In a software environment a program runs on a microprocessor. There, the speed is dependent on the Central Processing Unit (CPU) utilization. This is not the case with hardware implementations. Since a fixed, physical area is being assigned to the design, the run-time of the algorithm is about the same every time the algorithm is running with little or no deviation in run-time. Hardware implementations are useful because the encryption/decryption, and in many cases, key generation, will use dedicated hardware in its operation. This will increase speed, decrease delay and use of resources. In systems where hardware implementations of a cryptographic algorithm is used, the hardware implementations is referred to as a hardware accelerator. Hardware accelereators are especially useful in embedded systems when processing resources are limited. The procedure of encryption/decryption does not change, only the keys and data to be used. Therefore, in an embedded system it will save time, and probably energy to implement the algorithm in hardware.

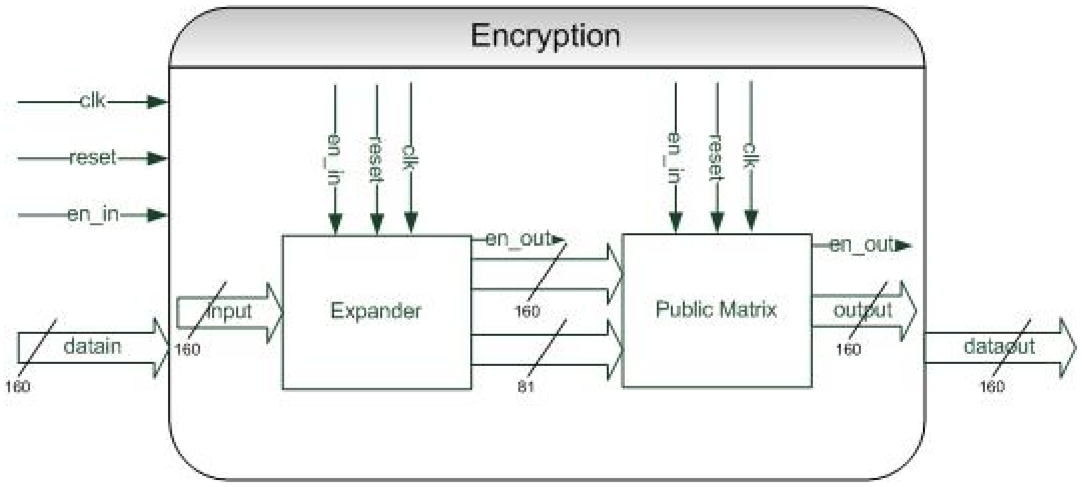


Figure 3.1: Encryption top level

3.2 The actual implementation

As mentioned earlier, the implementation is written in VHDL. All components, both in encryption and decryption have been implemented as a Mealy type Finite State Machine (FSM), with clocked (synchronous) output. The reset signal is active high (has value '1' when active), and synchronous. This is done because an FSM is convinient in hardware realization, which will help to keep the data flow in order. The reason for keeping most of the design clocked is because when a process is clocked, the registers remember the values to the next clock cycle. In this setting it is important, if the output had been made combinatorial (asynchronous), it had been necessary to set the value of each register in every state, and the probability for latches would have been present (a latch is a register with a value that does not change at any time. "Enable" signals are also used, both en_in and en_out. The en_in signal for each module determines wether the module is active or not. It is implemented because it makes it possible to turn off other modules than those that are active. For instance, if the sequencer module is processing the data, the Dobbertin component is not needed, and en_in for Dobbertin can be set to '0'. This means that the Dobbertin module is inactive, which will prevent Dobbertin to send data at the wrong time. And it may also save power. If the Dobbertin module would not have this feature, the module would have been active constantly while the circuit had been working. Enable out (en_out) signals for the different sub components are used as hand-shake signal. When a module signalizes that its data is ready for the next module, that module's en_out is set to '1'.

3.2.1 Encryption

As explained in the theory chapter, the encryption of an input data is calculated as given in table 2.1. Since there are many equations with many terms, the calculation must be split up in more than one stage.

Encryption of MQQ is implemented as the figure 3.1 shows. There are two components, Expander and Public Matrix. Expander expands the input vector from 160 bits to 12881 bits as table 3.2.1 shows, the Public Matrix does the calculation $y = \mathbf{P}(x) \equiv y = \mathbf{A} \cdot X$.

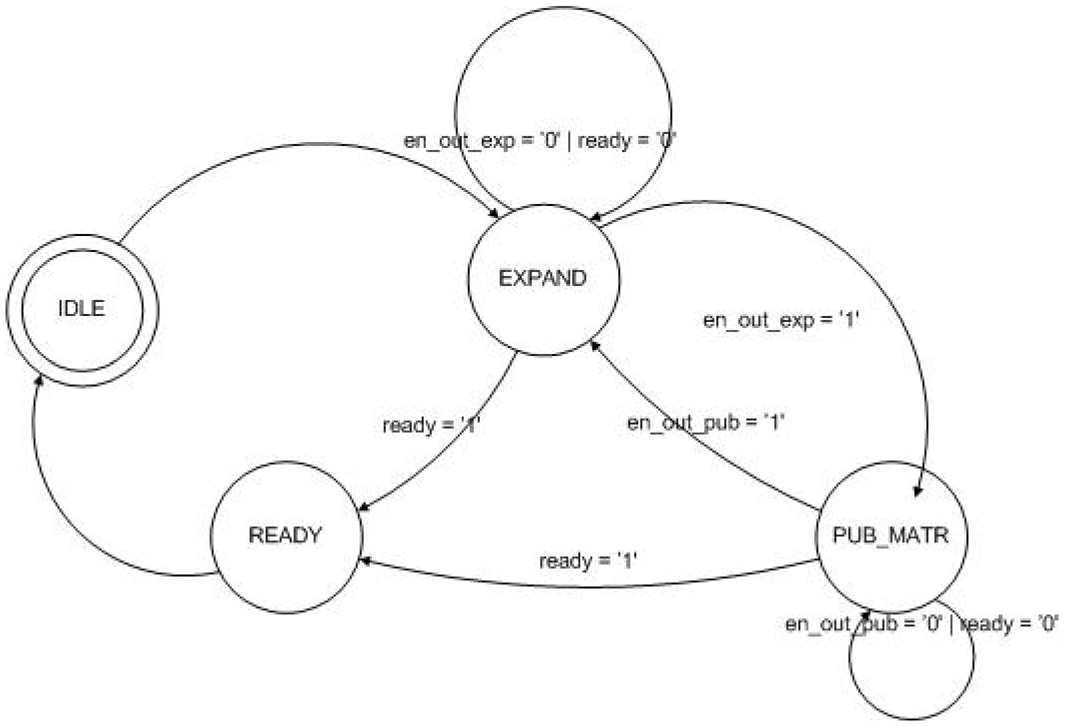


Figure 3.2: Encryption state diagram

Encryption top module

The encryption component is the top level module that controls the data flow, and the state diagram is listed in figure 3.2. The initial state is IDLE, which starts the encryption process by activating the EXPAND module. The top module will stay in EXPAND state until the en_out flag from expander gets the value '1', then the state machine moves to PUB_MATR. As with state EXPAND, the top module will stay in PUB_MATR until the public matrix en_out flag is set to 1, then the state machine moves back to EXPAND. A commom criteria for both PUB_MATR and EXPAND is that if the ready flag, controlled by public matrix is set to '1', the state machine moves to state RES, which indicates that the encryption is completed. In RES, the state machine moves back to IDLE for new data to be encrypted.

Expander

This component does the *AND* expansion of the 160 bit input data into 12881 bits. Since 12881 bits will be too much to send to the public matrix at once, the 12881 bits of data is multiplexed into 80 vectors of length 160 bit, and the last 81 bits is sent as a single vector. This demands two output vectors of 160 and 81 bits respectivly, as the figure 3.3 shows. There are three states in expander, IDLE, COMB and SEL, as shown in figure 3.4. IDLE is the first state, where the input vector is imported into the module. The signal en_out is set to '0', indicating that data is not ready to be sent to Public Matrix. After this has been done, expander moves to the COMB state. Here the expansion takes place, from 160 bit to 12881 bits. This is done by doing *AND* operations between the input and a tmp register, which holds the same value as input. The result is stored in the 12881 bit vector. In theory this should be done within one clock cycle. When this is done, the module reaches the SEL stage. Here the output is calculated. An iterator, inc, is used to push 160 bit of data from the 12881 bit vector to be sent to Public Matrix by writing to output_1. Which 160 bits from the 12881 bit vector to be sent

$$X = \begin{bmatrix} 1 \\ x_1 \\ . \\ . \\ . \\ x_n \\ x_1x_2 \\ x_2x_3 \\ . \\ . \\ . \\ x_{159}x_{160} \\ x_1x_3 \\ x_2x_4 \\ . \\ . \\ . \\ x_{158}x_{160} \\ . \\ . \\ . \\ x_1x_{159} \\ x_2x_{160} \\ x_1x_{160} \end{bmatrix}$$

Table 3.1: Expansion

is determined by the counter, and commented in the source code located in the appendix part of the report (A.2). At the same time the last 81 bit of the 12881 bit vector is also written to the output register output_2. The flag en_out is now set to '1', data is ready to be trasmitted. The state machine now goes back to IDLE, and this process is repeated, until all data has been sent. When this is complete, the iterator stops.

Public Matrix

The Public matrix is the component where the encryption calculation is taking place. It is implemented in the way demonstrated in figure 3.5. The state machine is presented in figure 3.6.

Public Matrix has eight states. Idle is the initial state where temporary registers used in the module is set to '0'. Signals such as en_out, the iterator cnt_2 and done (the ready bit explained in the top level) is also set to 0 here. IDLE initiates the calculation. MATR_AND is the state where the data from Expander is ANDed with the public key. Public Matrix moves to MATR_XOR160 where the result from MATR_AND is XORed with the previous vector, or stored for next iteration with AND. This loop between MATR_AND, MATR_XOR160 and SYNC_160 (the synchronization state for the XORed vector) will run 80 times since there are 80 vectors of length 160 to be ANDed and XORed. The last 81 bit is ANDed with the respective vectors from the public key one time and are awaiting the bit-by-bit XOR.

When the MATR_AND, MATR_XOR160 and SYNC_160 loop is finished (iterator cnt will have value 79) the next stage in Public Matrix stars, the bit-by-bit XOR, where the result vectors (160 bit and 81 bit) will be XORed down to one bit only. This is done in states MATR_XOR1 with the SYNC_1 as the synchronization state. cnt_2 is the iterator which keeps track of how many times the iteration has been done. When completed (cnt_2 gets vale 159), the top module goes to SYNC state, where a last XOR is performed, between the 160 bit vectors and the 81 bit vector. The result is written to a resultant vector, which will be the encrypted data. In state SEND, the ready bit is set to '1' to indicate that the data is now ready, and the answer is sent on the output port.

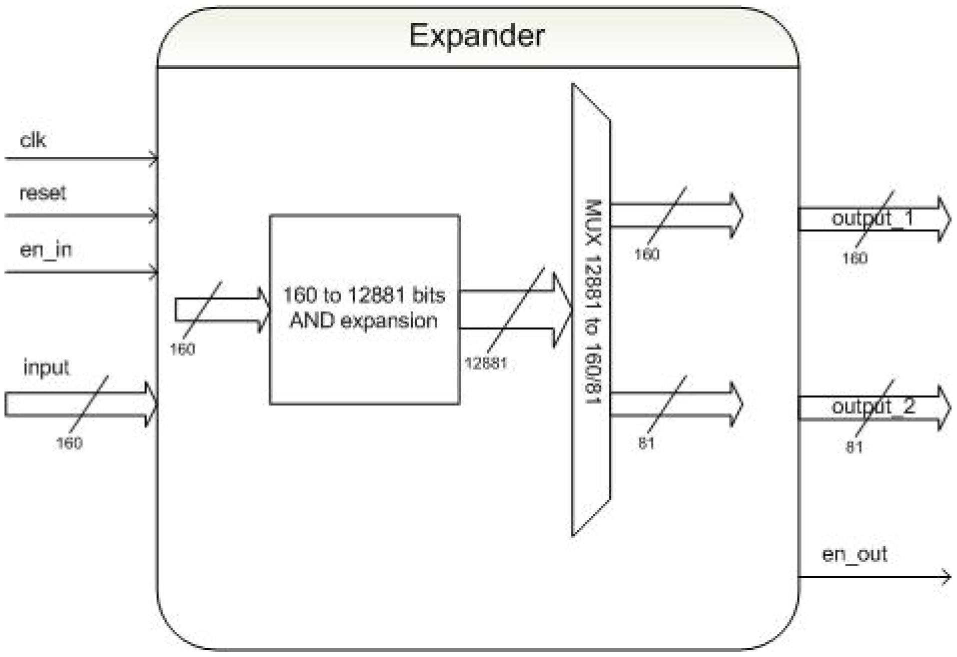


Figure 3.3: Expander internal architecture

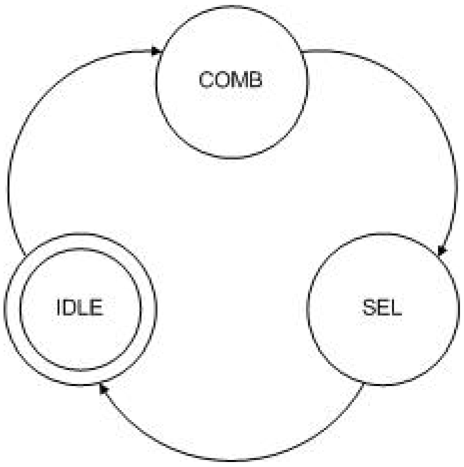


Figure 3.4: Expander state diagram

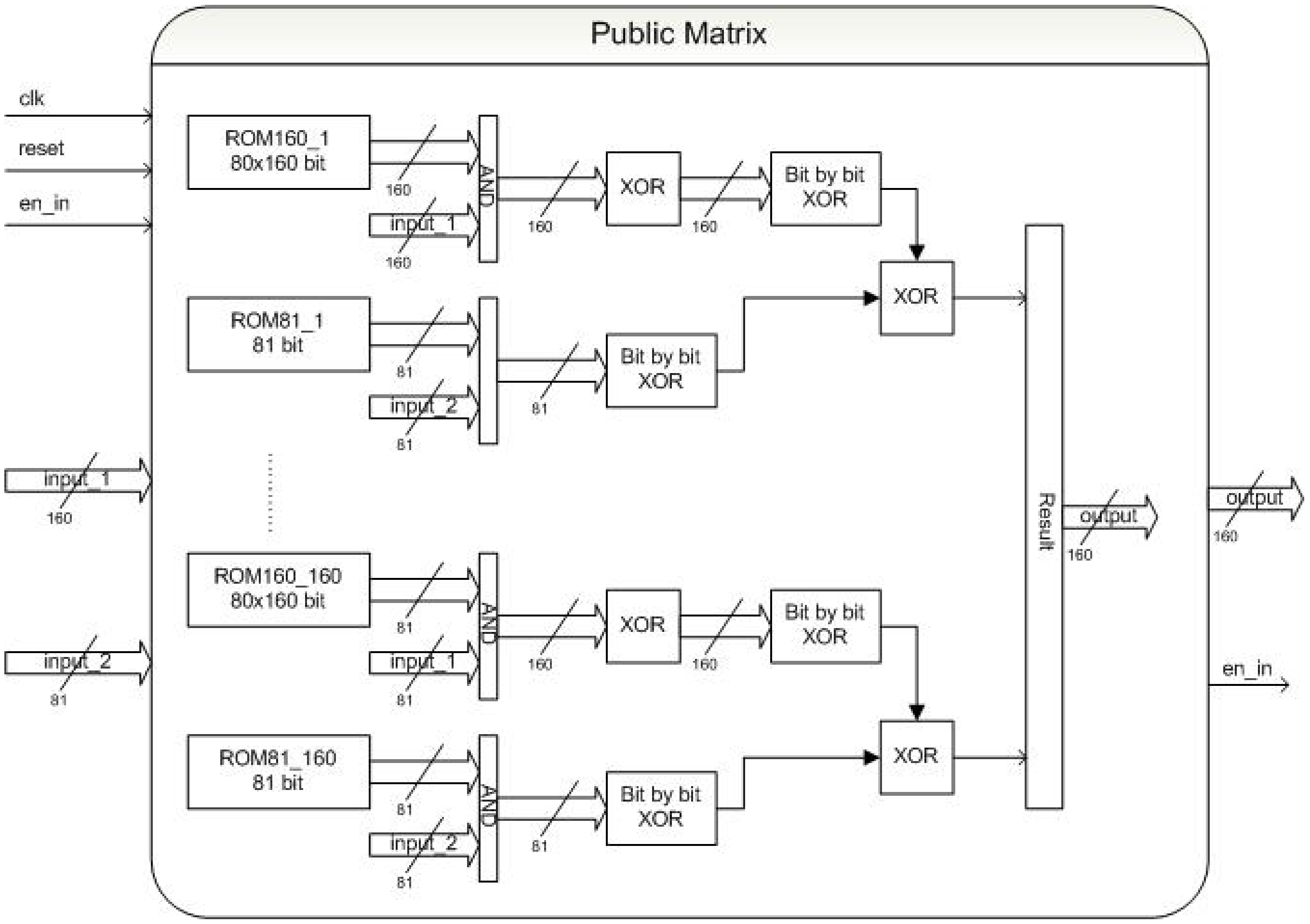


Figure 3.5: Public Matrix internal architecture

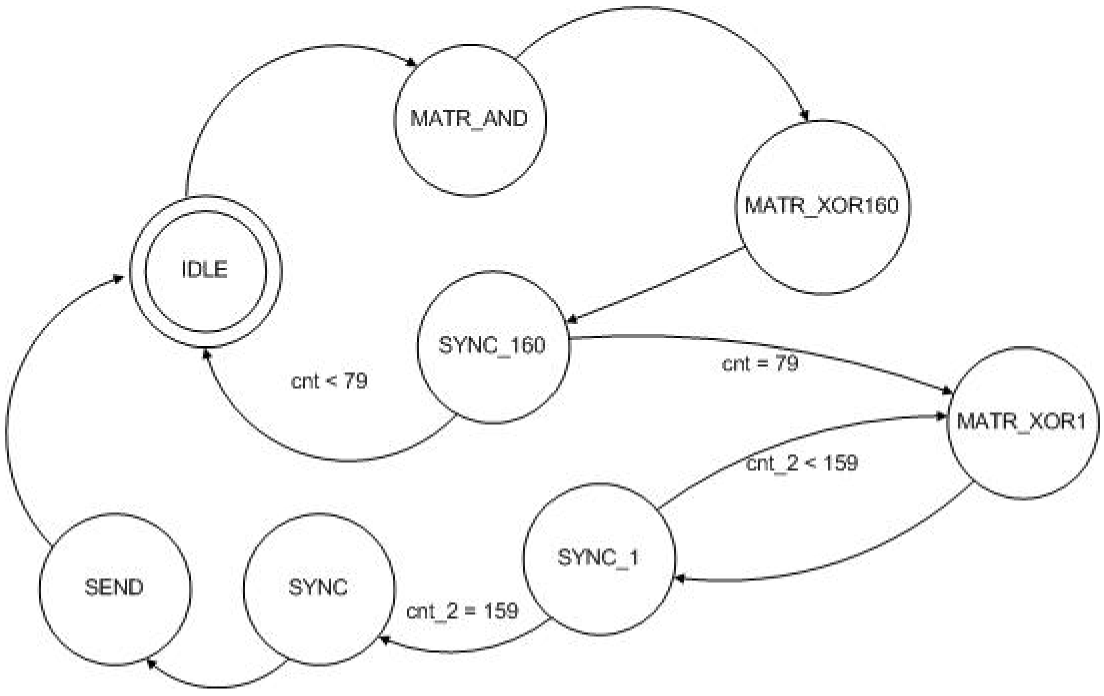


Figure 3.6: Public Matrix state diagram

3.2.2 Decryption

Decryption is implemented in four modules, Private Matrix T, Dobbertin ROM, Sequencer and Private Matrix S, shown in figure 3.7 The component Decryption is the top module that instantiates the four sub-components. It is also in the decryption procedure the logic optimization and minimization are being used. By that way it is possible to observe wether the area cost for Decryption can be reduced compared to storing the public key as fixed values, using a program that is an implementation of the ESPRESSO-II minimization algorithm presented in the theory chapter. To determine the optimization effect, the design in this assignment will be compared with the design from [8], made by the same author as this report.

The decryption top module has ten states, hence figure 3.8. As with encryption top module, the state machine in decryption controls the sub components within decryption.

Private Matrix

There are two instances of Private Matrix, the T and S matrix. They correspond to the first and the seventh step in the decryption algorithm (2.1) of MQQ. Table 3.9 shows the architecture. Private Matrix T and Private Matrix S are identical, but contains different parts of the private key. One important notice is that the private key now is stored as a function of the global iterator cnt, rather than fixed values as done before in [8]. Another modification that has been made is that in the [8] implementation of Public Matrix, there were many 1 bit registers such as tmp_xxx, xor_xxx, sync_xor_xxx (where xxx is a number between 1 and 160). These registers have been replaced with 160 bit registers such as tmp, matr_xor and sync_xor.

Private Matrix has five states, IDLE, ANDOP, XORING, SYNC and PUSH, which figure 3.10 shows.

In IDLE, values from the ROMs are written to corresponding signals and en_out is set to '0' (low). When this is done and the control logic has verified the writing to the signal from_rom, state ANDOP is initiated.

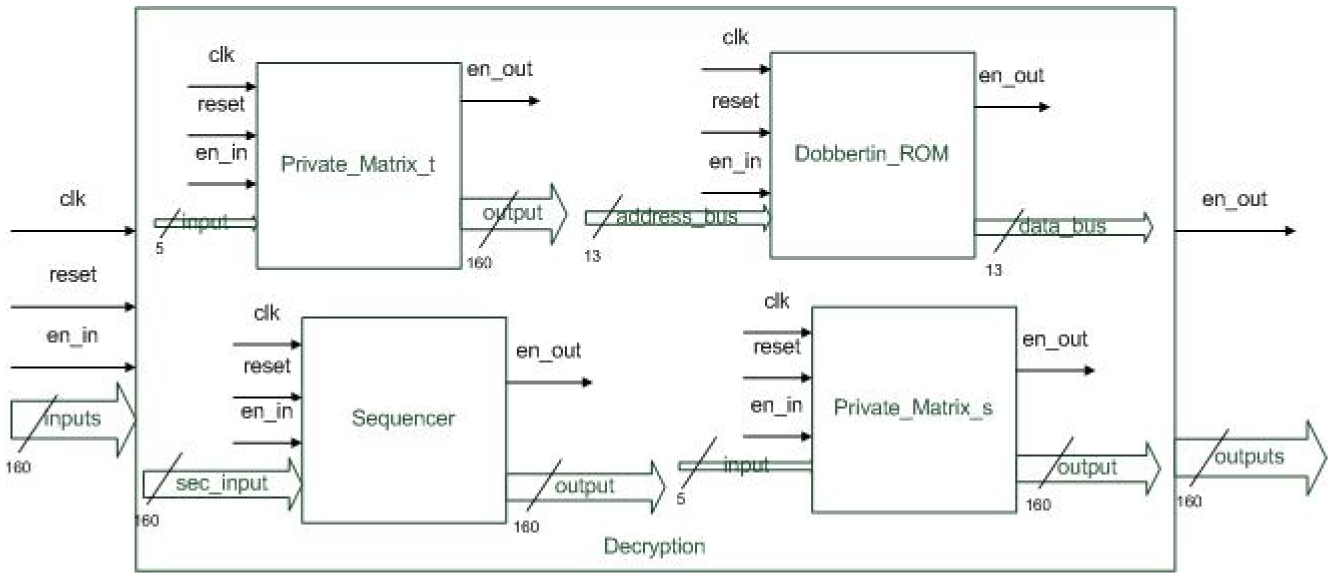


Figure 3.7: Decryption internal architecture

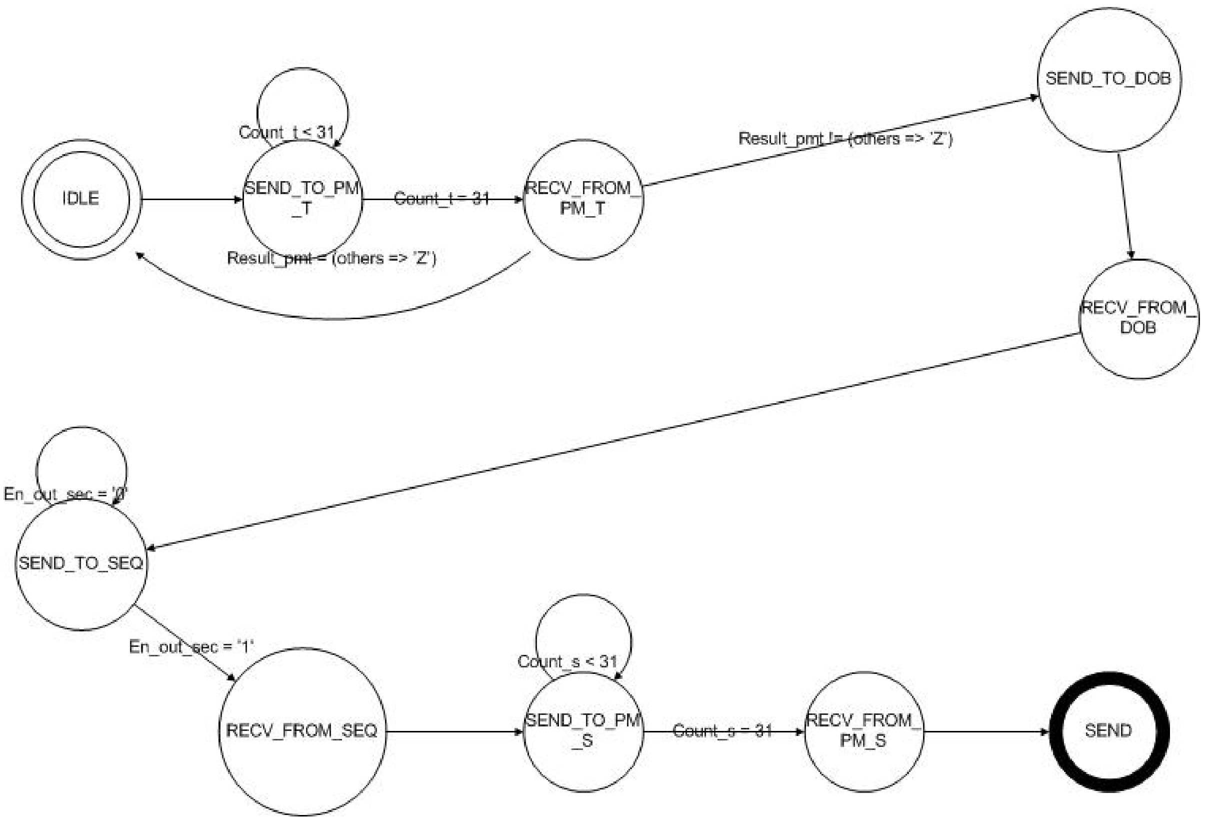


Figure 3.8: Decryption top level state diagram

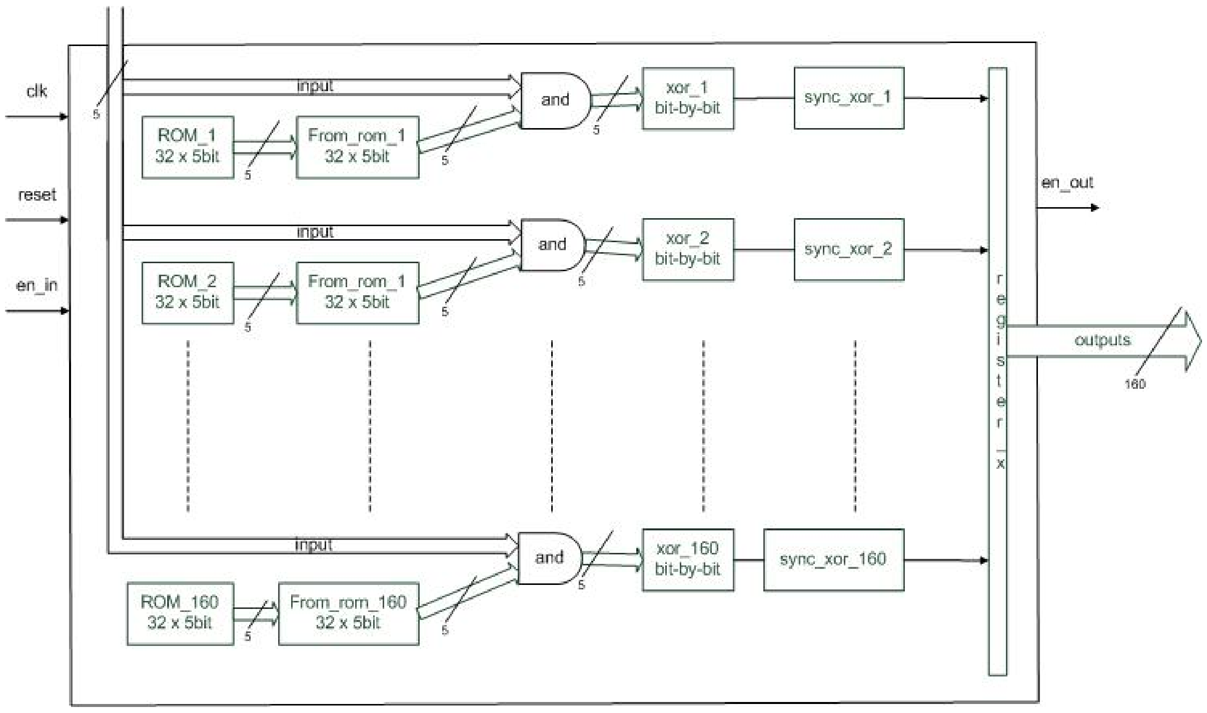


Figure 3.9: Private Matrix internal architecture

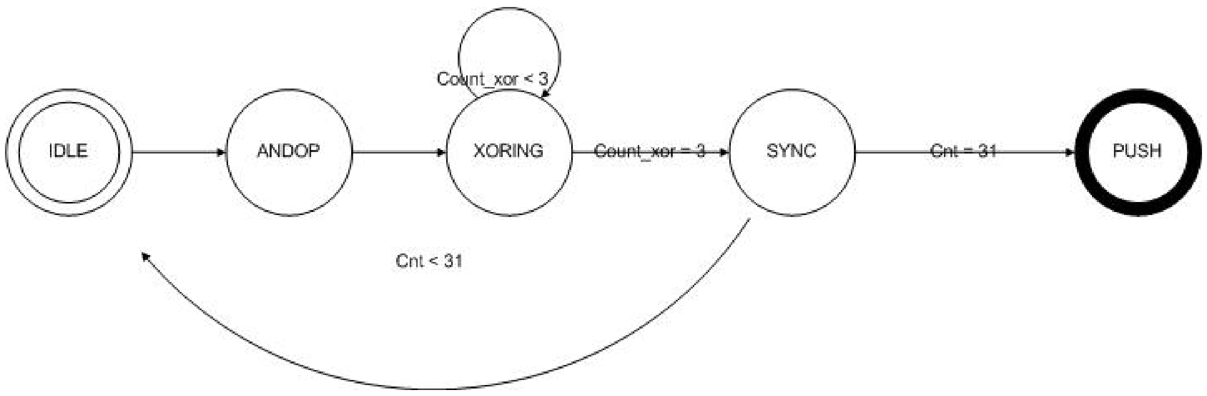


Figure 3.10: Private Matrix state diagram

In ANDOP, the logical AND operation is done between the input vector and the corresponding stored vector from the ROM. To illustrate this better, when the counter (cnt) has value 10, the Private Matrix runs at 11th time. In the previous implementation in [8], the signal from_rom_xxx (xxx is a number between 1 and 160) will have stored the 11th vector from the ROM array. When this is done, ANDOP state is finished, and next state will be XORING. In the current implementation, the signals from_rom_xxx have been removed completely, the ROM blocks that existed in [8] have now been replaced by signals which hold a function of the global counter.

The bit-by-bit XOR operation takes place in the state XORING. This state uses an internal counter, count_xor to keep track of how may times the XOR opeation is done. In [8], a temporary signal, tmp_xxx was used to store the temporary value corresponding to the position of the vector and_rom_in_xxx. The 160 tmp_xxx signals have been replaced by a 160 bit vector tmp. An XOR is then done between the tmp(position between 0 and 159) and and_rom_in_xxx at position counter+1. When the counter reaches 3, the 5 bit result from anding input and the stored ROM vector is bit-by-bit XORed into a single bit, and the state machine shifts state to SYNC.

A modification from the original design is that a new step is being introduced, a SYNC state. This is to keep syncronization, and to complete the XOR step. Here is a description why this step is needed. When the Private Matrix runs for the first time, the sync_xor_xxx get the value from the xor_xxx, the result after the bit-by-bit XOR operation. Again, the sync_xor_xxx and xor_xxx signals have been replaced by 160 bit vectors sync_xor and matr_xor, respectivly. For the second run and so on, a new XOR operation is initiated between the matr_xor and the sync_xor, the latter signal has the value of the previous XOR operation. This is necessary to make sure that the bit-by-bit XOR operation runs as many times as it is supposed to. And when this operation is done, en_out is set to '1' (high). That means that the value can be written to register_x, a synchronization register for the XORed bits.

When the counter reaches value 31, it means that the register_x contains the result, and the Private Matrix is ready to send the data to Dobbertin_ROM component. This is done in state PUSH_OUT.

Dobbertin ROM

The Dobbertin component (3.11) is an implementation of step 2, 3 and 4 in the decryption algorithm (2.1). Also here, the fixed values stored in a ROM structure have been converted to functions of the 13 bit input vector by using the Espresso application.

Sequencer

The sequencer (3.12), that corresponds to the fifth and sixth step of the decryption procedure (2.1) is unchanged from [8].

The component sequencer has seven states. Those are IDLE, MUX2_SEL, SYNC, MUX31_SEL, SEND_TO_MASTER, RECV_MR and PUSH (ref. figure 3.13).

In the INIT state, the 160bit input signal from the Dobbertin ROM is split up in an array consisting of 32 vector with length of 5 bits. When this is done, current state changes to MUX2_OUT. Here the first element of the array is being sent through the multiplexer since the selector is set to '0',and becomes the first element in a similar array which will be the result that the secuencer module generates. The first element is written in the output array in state SYNC. The counters are increased by 1, and the state machine shifts to state MUX31_out. Selector of MUX_2 is set to '1' because the values that are supposed to go through that MUX will not be the first element. Then the output from MUX_31 will be the 5 least significant bits

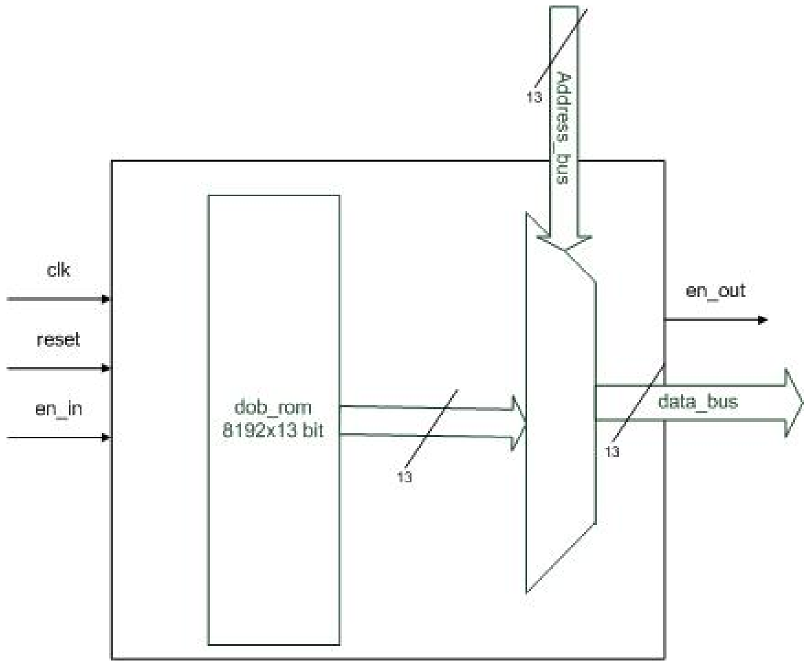


Figure 3.11: Dobbertin internal architecture

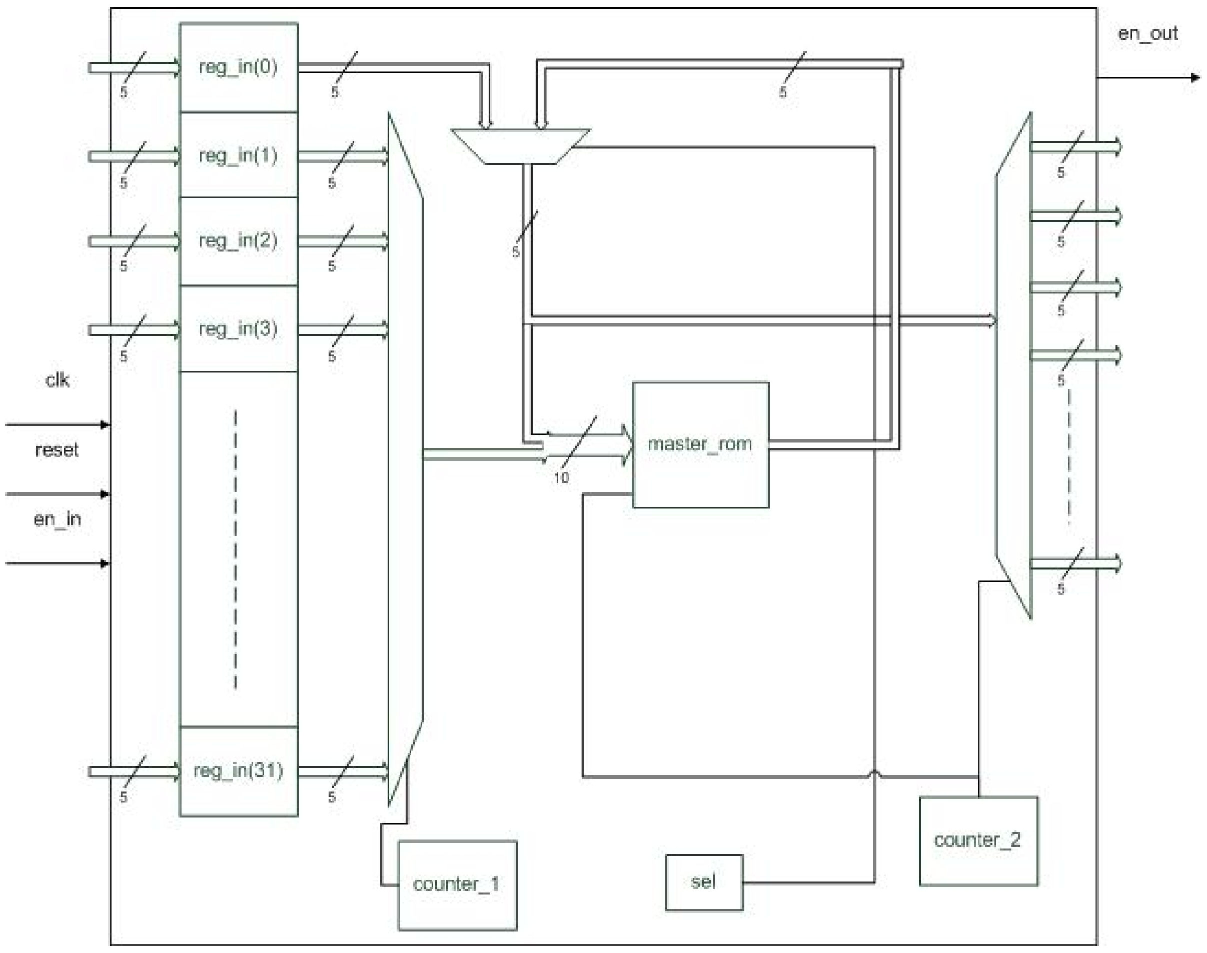


Figure 3.12: Sequencer internal architecture

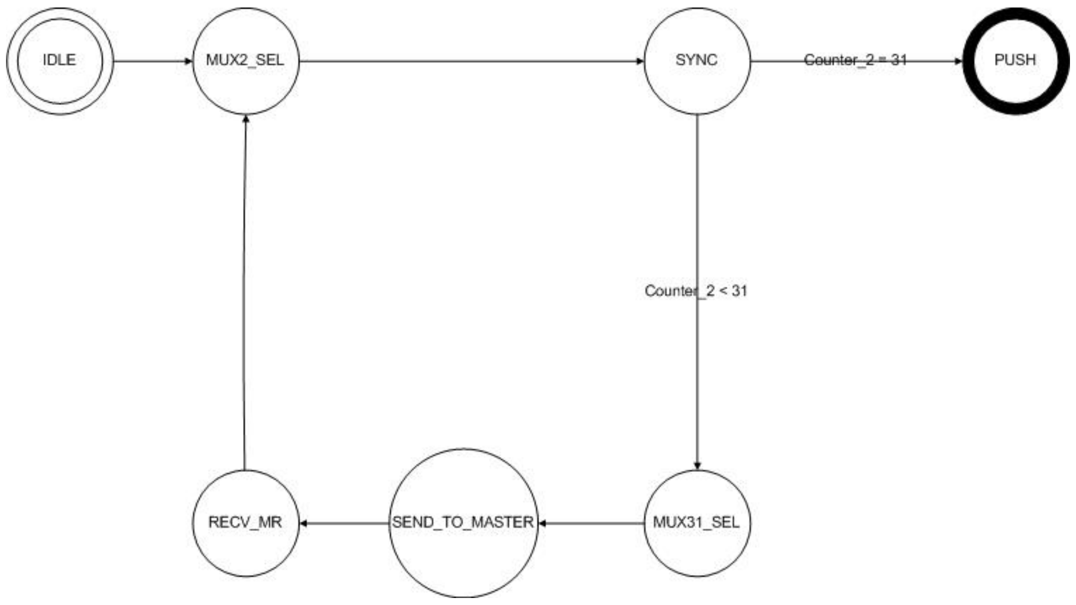


Figure 3.13: Sequencer state diagram

in a 10 bit vector, the most sigificant bits are the 5 bit vector from MUX_2. The 10 bit vector is an address, which is the input to the Master_ROM. After Master_ROM has done its job, the result will go through MUX_2 and written on the output register, and also be used as feedback for the new address to be sent into Master_ROM. This procedure will continue until counter_2 reaches value 31, which means that all 160 bits are processed by the sequencer and are ready to be written to the output register, and the value is used by Private_Matrix_S as input.

Master ROM

The Master ROM is a subcomponent to the sequencer. Also here, the fixed values stored in ROM blocks in the original implementation have been replaced by functions of the 10 bit input vector. There exists a control ROM, which has 2^5 3bit vectors and its function is to be a selector that determines which of the 8 functions to be used. Also the control ROM has been converted to functions of the counter. The control ROM is being controlled by a counter. In this implementation this counter is located in the sequencer(counter_2), and the value from the counter is one of the input ports. This is shown in figure 3.14.

3.3 Data and keys

Though the assignment indicates that real data and keys are to be used, this is not the case. The key generation calculation is complex, so it is uncertain wether the key generation process may practically be implemented in hardware. According to the main supervisor, the keys for the original implementation were generated in Wolfram Mathematica, and the key generation calculation took considerably long time to complete on a modern computer.

3.4 Optimizing the stored fixed values

As mentioned several places in this report, efforts have been made to optimize the stored fixed values, in order to reduce area consumption of the implementation. A program that implements

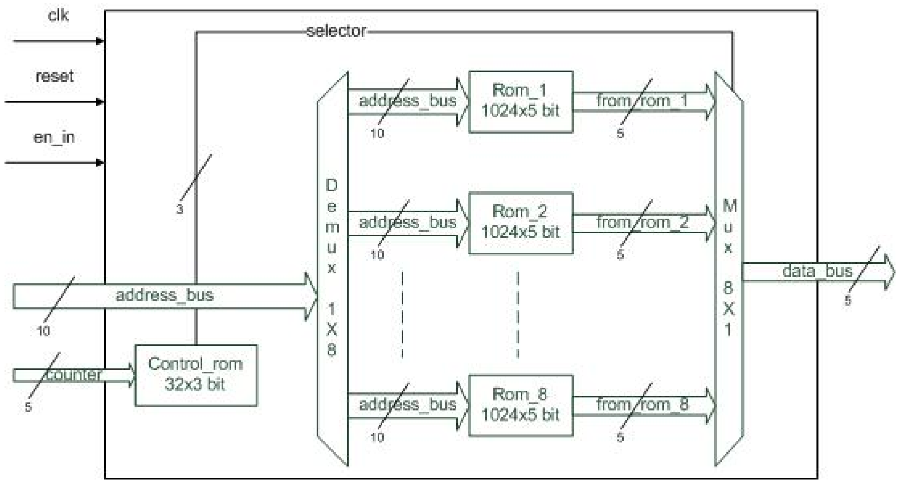


Figure 3.14: Master ROM internal architecture

.i 3	← indicates three input bits		
.o 2	← indicates two output bits		
.ilb A B C	← names the variables in input		
.ob Y Z	← names the variables in output		
.p 8	← number of terms		
000	← input value	output value →	11
001			00
010			10
011			10
100			00
101			01
110		(- means don't care) →	-1
111			-1
.e	← indicator for end of file		

Table 3.2: Espresso input file format

the ESPRESSO-II minimization algorithm [6] has been used to accomplish this task.

To minimize a number of values the following must be done. First, one has to know how many input values to be minimized. For instance, if there are 1024 values to be minimized, each of length 5 bit, then each value must be represented by a unique value between 0 and 1023, 10 bit length and in binary form. There will be ten input functions and five output functions. The input file that the espresso application demands must be arranged in the proper format. An authentic file used in this project is located in the appendix of this report (appendix B). After the program has finished the minimzation, an output file with the reduced expresson is generated. This file has to be modified into valid VHDL syntax. This can easily be done with an advanced text editor, such as TextPad.

Here follows an explanation on how to arrange the data, in table 3.4.

Chapter 4

Results

In this chapter the results of the implementation will be presented. There have been attempts to synthesize both encryption and decryption, which is necessary in order to build a physical realization of this MQQ implementation in VHDL. Simulation of all the modules have also been conducted.

4.1 Synthesis

The encryption and decryption procedures have been synthesized against the FPGA meant for the physical realization, the Xilinx Virtex 5 model xc5vlx110t-1-ff1136 (speed grade -1). Table 4.1 shows a brief summary of the synthesis report for decryption. Synthesis of the decryption with fixed values (from [8]) will also be presented, to show if the optimization through minimization has been successfull or not. Since the synthesis of the encryption procedure failed, the synthesis results for encryption from [3] will be used due to a lack of synthesis results from this implementation, located in table 4.1. This incident will be analyzed and discussed in the Discussion chapter.

It is important to point out that the earlier encryption implementation from [3] was implemented on four FPGAs, so the content in table 4.1 is for one chip out of four.

4.2 Verification of functionality through simulation

Simulation is an important tool to verify that the design acts properly according to the spesification made in the source code. The simulation tool used in this assignment is the ModelSim SE 6.3f by Modeltech.

4.2.1 Encryption

The simulation results from Encryption will be presented in this section.

ESPRESSO-II minimization	Slice Registers	LUTs	Frequency
Yes	5,937	9,950	201.045 MHz
No	5,910	7,703	214.000 MHz

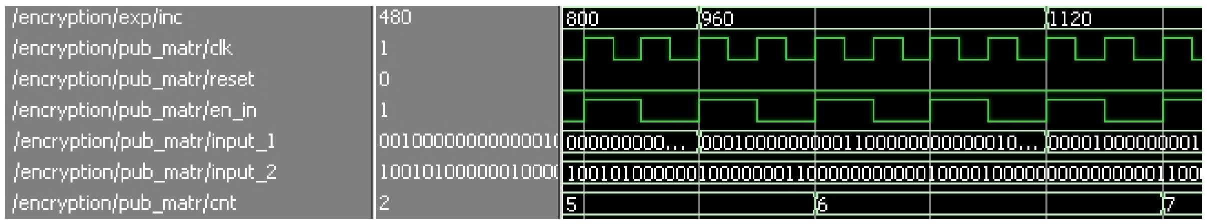
Table 4.1: Synthesis results of the MQQ decryption procedure

Slice Registers	LUTs	Frequency
13,137	25,285	276.7 MHz

Table 4.2: Synthesis results of the MQQ encryption procedure

Encryption top module

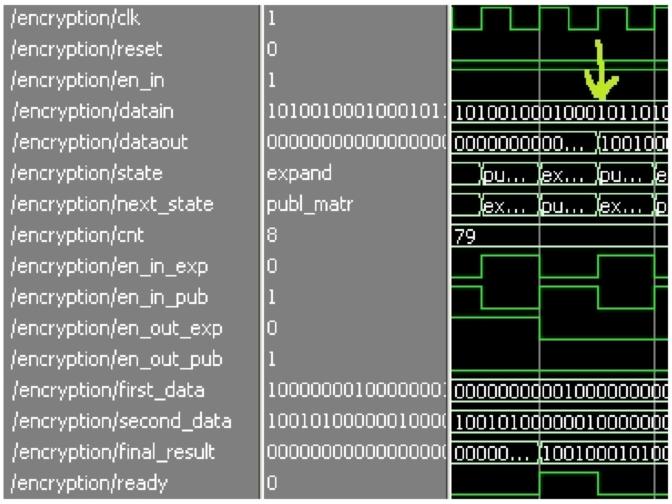
Figure 4.1 shows that the encryption top module makes sure that the global counter cnt and the local counter inside Expander increases with an equal number of clock cycles.



```
/encryption/exp/inc              480
/encryption/pub_matr/clk         1
/encryption/pub_matr/reset       0
/encryption/pub_matr/en_in       1
/encryption/pub_matr/input_1     001000000000000010
/encryption/pub_matr/input_2     10010100000010000
/encryption/pub_matr/cnt         2
```

Figure 4.1: Encryption counter synchronization

When the public matrix module has completed the encryption, the ready bit is set to high, which should make the top module change state to RES (the state where the result from encryption is ready). But figure 4.2 shows that the RES state is never reached, even when the ready bit gets value '1', as pointed out in the figure. The cause for this problem is unclear, debugging has been conducted in order to locate the problem. However, in the process sync_run, the controlling process of the Encrypton top module, dataout is to be equal to the result vector final_result, which is the answer from public matrix. By this way, dataout, the output port from Encrytion which ultimately holds the answer is set to contain the encrypted data at the right time, which means that by this way, the RES state is not needed, and can be removed from the source code.



```
/encryption/clk            1
/encryption/reset          0
/encryption/en_in          1
/encryption/datain         10100100010001011
/encryption/dataout        000000000000000000
/encryption/state          expand
/encryption/next_state     publ_matr
/encryption/cnt            8
/encryption/en_in_exp      0
/encryption/en_in_pub      1
/encryption/en_out_exp     0
/encryption/en_out_pub     1
/encryption/first_data     10000000100000001
/encryption/second_data    10010100000010000
/encryption/final_result   000000000000000000
/encryption/ready          0
```

Figure 4.2: Encryption top module missing state

Expander

The simulation verifies (figure 4.3) that the 160 bit input is expanded into 12881 bits in state COMB, marked in yellow. Also, the 160 bit output from the expanded vector is set in state SEL (red mark). The value of the 160 output changes in the next iteration (green mark), as the counter increases by 160. This is the expected behaviour of the expander module.

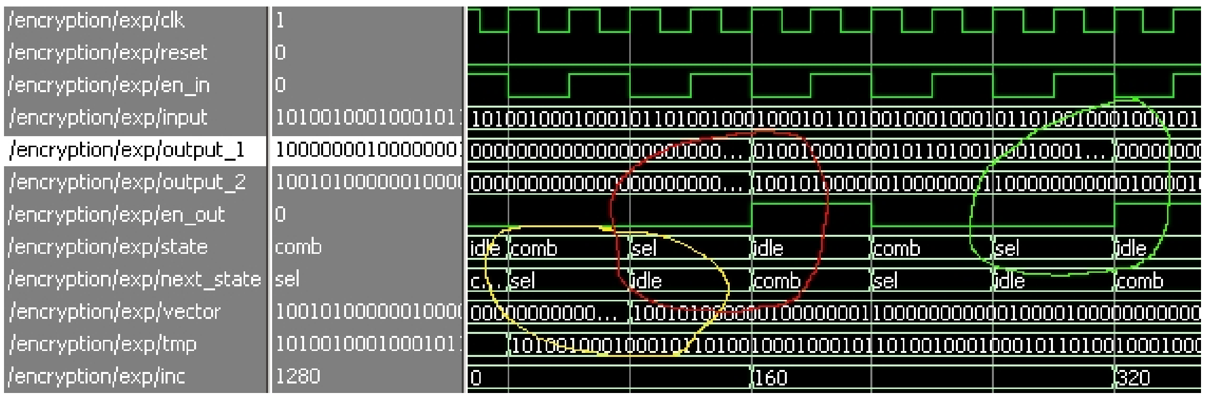


Figure 4.3: Expander behaviour

Public Matrix

Figure 4.4 shows that when indata_1 gets a new vector from Expander in state IDLE, the AND operation takes place in the next state, MATR_AND. One clock cycle later, the 160 bit vector XOR operations are performed. When the global counter increases value by 1, the next vector is written from Expander into indata_1, and the same operation cycle repeats until the global counter reaches 79.

When the last 160 bit vector has been XORed with the previous 79 vectors, the Public matrix enters the bit-by-bit XOR procedure, in order to finish off the encryption calculation. Figure 4.5 shows that this process starts when the requirements for entering this phase are fulfilled.

4.2.2 Decryption

Here, the verification for Decryption through simulation is being presented.

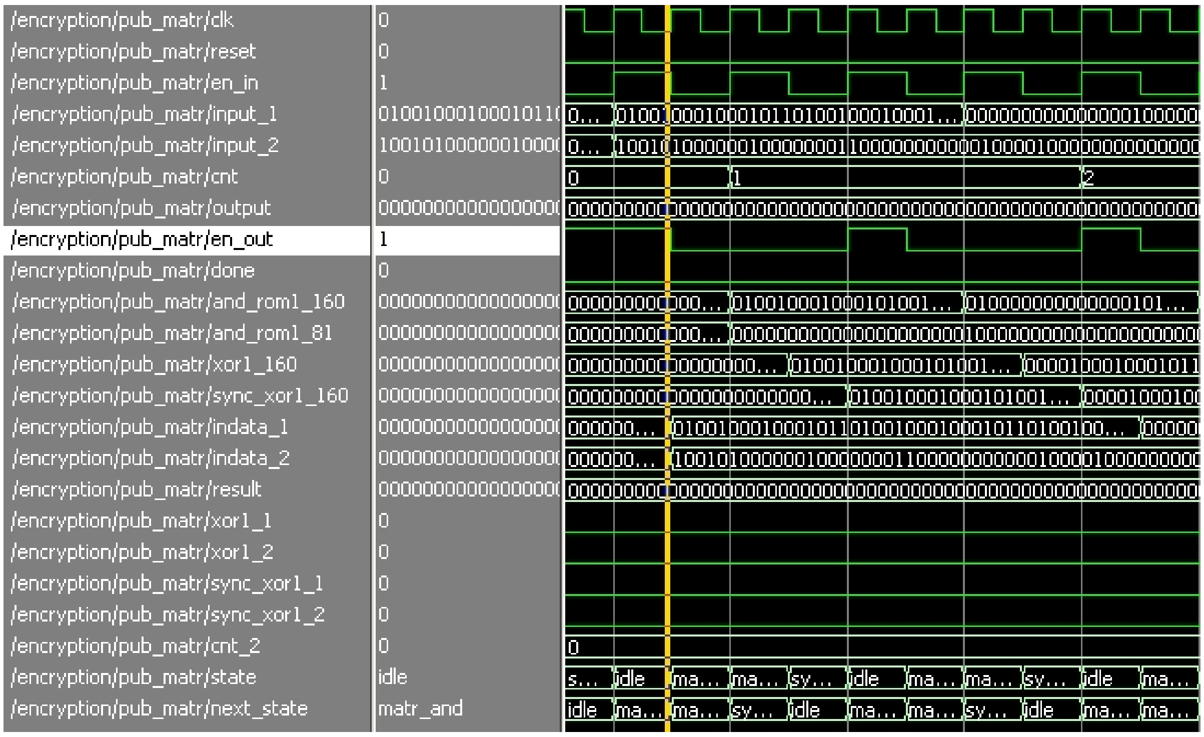
Decryption top module

First, it is necessary to verify that all signals initially are set to '0' when reset is active. As the figure 4.6 indicates, this is the case. There is also possible to see that the decryption circuit goes active when en_in is set to '1'.

Private matrix

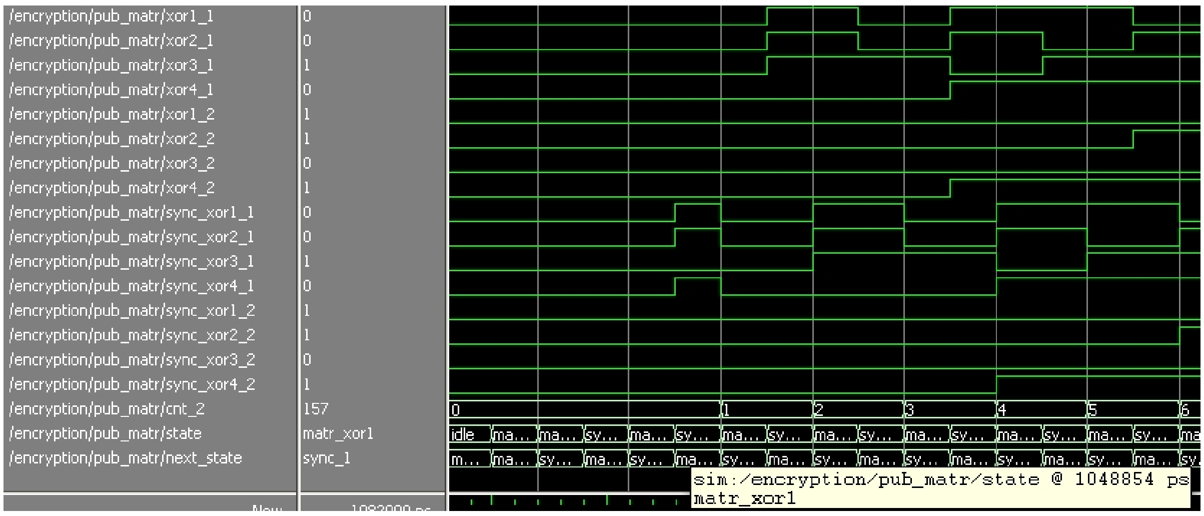
The output from the stored values depend on the counter cnt. Figure 4.7 shows that it takes six clock cycles before the value from storage is calculated from the cnt value.

The XOR operations seem to work as intended (figure 4.8). The current value, that is an AND between input and the calculated private key value are XORed down to one bit.



```
/encryption/pub_matr/clk            0
/encryption/pub_matr/reset          0
/encryption/pub_matr/en_in          1
/encryption/pub_matr/input_1        010010001000101110
/encryption/pub_matr/input_2        100101000000100000
/encryption/pub_matr/cnt            0
/encryption/pub_matr/output         000000000000000000
/encryption/pub_matr/en_out         1
/encryption/pub_matr/done           0
/encryption/pub_matr/and_rom1_160   000000000000000000
/encryption/pub_matr/and_rom1_81    000000000000000000
/encryption/pub_matr/xor1_160       000000000000000000
/encryption/pub_matr/sync_xor1_160  000000000000000000
/encryption/pub_matr/indata_1       000000000000000000
/encryption/pub_matr/indata_2       000000000000000000
/encryption/pub_matr/result         000000000000000000
/encryption/pub_matr/xor1_1         0
/encryption/pub_matr/xor1_2         0
/encryption/pub_matr/sync_xor1_1    0
/encryption/pub_matr/sync_xor1_2    0
/encryption/pub_matr/cnt_2          0
/encryption/pub_matr/state          idle
/encryption/pub_matr/next_state     matr_and
```

Figure 4.4: Public matrix calculation



```
/encryption/pub_matr/xor1_1         0
/encryption/pub_matr/xor2_1         0
/encryption/pub_matr/xor3_1         1
/encryption/pub_matr/xor4_1         0
/encryption/pub_matr/xor1_2         1
/encryption/pub_matr/xor2_2         1
/encryption/pub_matr/xor3_2         0
/encryption/pub_matr/xor4_2         1
/encryption/pub_matr/sync_xor1_1    0
/encryption/pub_matr/sync_xor2_1    0
/encryption/pub_matr/sync_xor3_1    1
/encryption/pub_matr/sync_xor4_1    0
/encryption/pub_matr/sync_xor1_2    1
/encryption/pub_matr/sync_xor2_2    1
/encryption/pub_matr/sync_xor3_2    0
/encryption/pub_matr/sync_xor4_2    1
/encryption/pub_matr/cnt_2          157
/encryption/pub_matr/state          matr_xor1
/encryption/pub_matr/next_state     sync_1

sim:/encryption/pub_matr/state @ 1048854 ps
matr_xor1
```

Figure 4.5: Public matrix bit-by-bit XOR

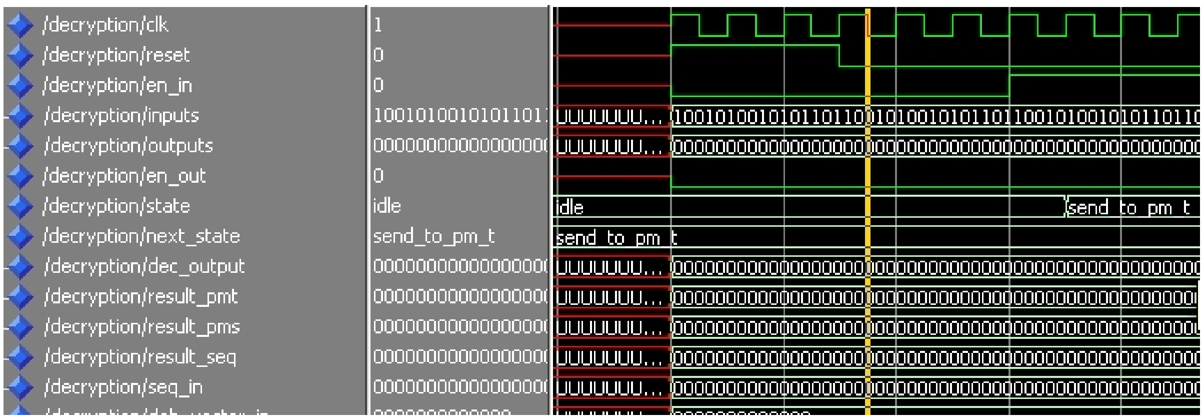

/decryption/clk 1
/decryption/reset 0
/decryption/en_in 0
/decryption/inputs 10010100101011011
/decryption/outputs 0000000000000000
/decryption/en_out 0
/decryption/state idle
/decryption/next_state send_to_pm_t
/decryption/dec_output 0000000000000000
/decryption/result_pmt 0000000000000000
/decryption/result_pms 0000000000000000
/decryption/result_seq 0000000000000000
/decryption/seq_in 0000000000000000
idle
send_to_pm_t

Figure 4.6: Simulation of startup

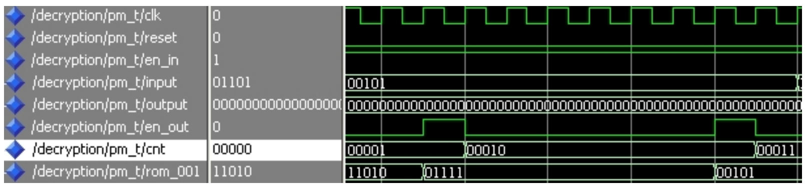

/decryption/pm_t/clk 0
/decryption/pm_t/reset 0
/decryption/pm_t/en_in 1
/decryption/pm_t/input 01101
/decryption/pm_t/output 0000000000000000
/decryption/pm_t/en_out 0
/decryption/pm_t/cnt 00000
/decryption/pm_t/rom_001 11010
00101
00001
00010
00011
11010
01111
00101

Figure 4.7: Stored private key as a function

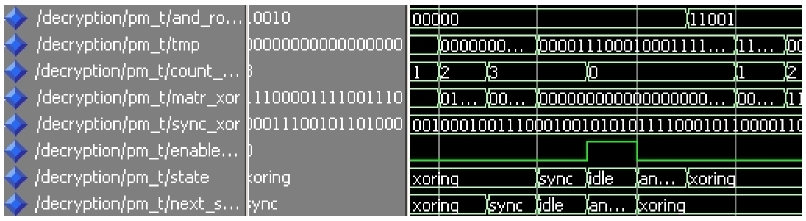
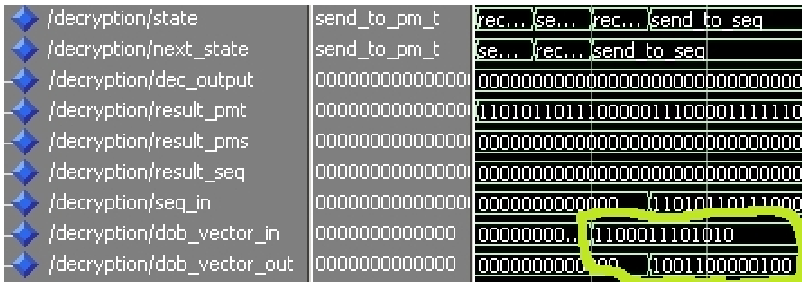

/decryption/pm_t/and_ro... 0010
/decryption/pm_t/tmp 0000000000000000
/decryption/pm_t/count_... 3
/decryption/pm_t/matr_xor 1100001111001110
/decryption/pm_t/sync_xor 0011100101101000
/decryption/pm_t/enable... 0
/decryption/pm_t/state xoring
/decryption/pm_t/next_s... sync
00000
11001
xoring
sync
idle
an...

Figure 4.8: XOR procedure

Dobbertin ROM

As described earlier, the Dobbertin component calculates its output based on the input. Figure 4.9 verifies that that is the case.

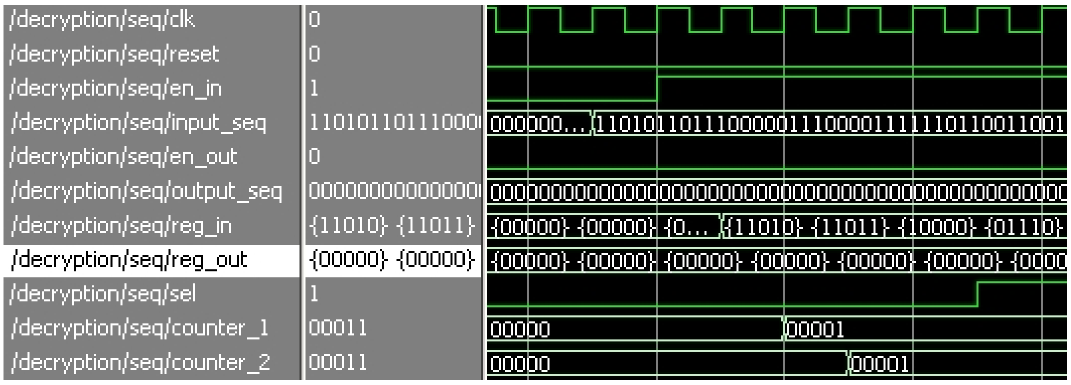


```
/decryption/state           send_to_pm_t       rec... se... rec... send to seq
/decryption/next_state      send_to_pm_t       se... rec... send to seq
/decryption/dec_output      00000000000000     000000000000000000000000000000
/decryption/result_pmt      00000000000000     110101101110000011100001111110
/decryption/result_pms      00000000000000     000000000000000000000000000000
/decryption/result_seq      00000000000000     000000000000000000000000000000
/decryption/seq_in          00000000000000     0000000000000   110101101110000
/decryption/dob_vector_in   0000000000000      00000000...  1100011101010
/decryption/dob_vector_out  0000000000000      0000000000 00   1001100000100
```

Figure 4.9: Dobbertin calculation

Sequencer

When the sequencer goes active (en_in = ’1’), the module starts its work. The temporary reg_in splits the 160 bit input into 32 vectors of length 5 (figure 4.10). Since the sequencer has not been subject to any minimization, this implementation is unchanged from the TTM4530 project.

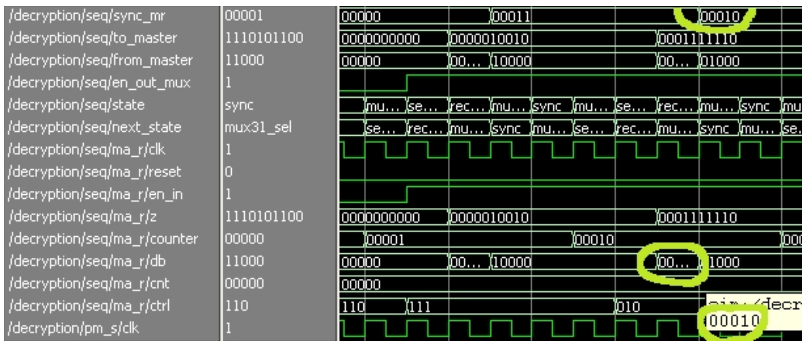


```
/decryption/seq/clk         0
/decryption/seq/reset       0
/decryption/seq/en_in       1
/decryption/seq/input_seq   11010110111000     000000...  1101011011100000111000011111110110011001
/decryption/seq/en_out      0
/decryption/seq/output_seq  00000000000000     000000000000000000000000000000000000000000000
/decryption/seq/reg_in      {11010} {11011}    {00000} {00000} {0... {11010} {11011} {10000} {01110}
/decryption/seq/reg_out     {00000} {00000}    {00000} {00000} {00000} {00000} {00000} {00000} {0000
/decryption/seq/sel         1
/decryption/seq/counter_1   00011              00000   00001
/decryption/seq/counter_2   00011              00000   00001
```

Figure 4.10: Sequencer startup

Master ROM

The output is calculated from the input. There are eight sets of equations, and ctrl determines which set to use. Figure 4.11 shows which value to be sent, and it is marked.



```
/decryption/seq/sync_mr        00001
/decryption/seq/to_master      1110101100
/decryption/seq/from_master    11000
/decryption/seq/en_out_mux     1
/decryption/seq/state          sync
/decryption/seq/next_state     mux31_sel
/decryption/seq/ma_r/clk       1
/decryption/seq/ma_r/reset     0
/decryption/seq/ma_r/en_in     1
/decryption/seq/ma_r/z         1110101100
/decryption/seq/ma_r/counter   00000
/decryption/seq/ma_r/db        11000
/decryption/seq/ma_r/cnt       00000
/decryption/seq/ma_r/ctrl      110
/decryption/pm_s/clk           1
```

Figure 4.11: Master ROM calculation

Chapter 5

Discussion and conclusion

In this chapter the results will be analyzed and discussed, and a conclusion will be formed on basis of the discussion section.

5.1 Discussion

5.1.1 Synthesis

Encryption

The XST was unfortunately unable to synthesize the encryption procedure, due to the complexity of the design. XST ran for almost 72 hours, eventually the computer crashed. The computer used for synthesis of encryption had an Intel Core 2 Duo T7300, 2.0 GHz, 4 MB cache with 4 GB of system Random Access Memory (RAM) and 4 GB of virtual memory, or swap. Ubuntu 9.04 "Jaunty Jackalope" 64 bit version (X86_64 architecture) was the chosen operating system. On the mentioned computer, by synthesizing in a 32 bit operating system caused memory conflict, because a 32 bit operating system can only address 4 GB of memory in total. The sum of the physical and virtual memory (swap) exeeded 4 GB. It appeared that the encryption synthesis occupied all available memory, 6683 MB, causing the computer to run out of available memory. The fact that virtual memory were used, led to that the performance of XST was seriously hampered by this fact. XST eventually caused the computer to crash when it tried to synthesize the module Expansion, and the synthesis of encryption failed. XST never began to synthesize the Public Matrix It is likely that the encryption design needed more memory to work on, since all available memory, also swap were used. It is however unclear if this would have made a difference. [3] inicates that the original implementation also caused problems when attempts were made to implement the encryption on a single FPGA.

If the results from 4.1 should be reffered to, the encryption procedure takes up an enormous amount of area. The numbers in the table are referring to one single chip, while the original implementation of the MQQ encryption used a total of four FPGAs, which means that the numbers related to number of Look-Up Table (LUT)s used, and number of slice registers can be multiplied by four. This means that a realization of encryption cannot be done on a single FPGA, not even the one intended for this thesis, which is one of the larger FPGAs on the market.

Decryption

As the synthesis results (4.1) indicate, the implementation which has been subject to ESPRESSO-II minimization actually has a **higher** area usage than the earlier version without the minimization of the ROM blocks. This is a result that was unexpected in relation to minimization theory. One can also observe that the maximum frequency of the minimized design is lower than the unmodified version, which means that if realized in hardware, the minimized solution will run slower than the unminimized version of the Decryption implementation. In theory, the data should have been compressed, causing the data to use less area in hardware. The stored values are not the same in the minimized implementation as the old implementation from [8], but the difference beween the two versions compared in number of used LUTs are too significant to only be related to the different stored values. There are also drawbacks concerning storing of data as functions rather than raw values. When storing raw data, there is a possibility that the block RAM of the FPGA may be utilized, meaning that number of used LUTs are reduced. By storing data as functions of a vector, the possibility to store it in block RAM is significally lowered, meaning that these functions almost certainely will be stored in LUTs, hence increasing total area consumption.

5.1.2 Design behaviour simulation and verification

To fully determine the effect of minimizing the ROM blocks, the design in this report are quite similar to the design in [8], written by the same author as this report.

5.2 Conclusion

Tact that the Encryption procedure did not synthesize due to XST failure, this MQQ can not be realized in a single FPGA, which was the desired goal of this assignment. Even if the original implemtation of MQQ had been used for realization, it would have been impossible to realize it on one FPGA. Only for encryption, four FPGAs would have been required, which means that a single chip realization for MQQ at this time is not possible.

5.3 Future work

There is a possibility for realization of MQQ that could be investigated, a hardware/software codesign solution. It means that some parts of the design has to be implemented as software that works with the hardware implemented part. There exists a CPU implementation meant for Xilinx FPGAs, the MicroBlaze soft processor core [7]. This may allow software code to run on the soft-core CPU, which could make it possible to implement MQQ on a single FPGA. But this solution will not be a pure hardware implementation, which was the intension in this thesis. But to utilize MicroBlaze, additional software is required, the Embedded Development Kit (EDK), which has to be purchased in addition to ISE.

5.4 Contributors

Mohamed El-Hadedy, PhD student at Q2S, has provided guidance and information which has been crucial in the design process.

Chapter 6

Abbrevations

MQQ Multivariate Quadratic Quasigroups

VHSIC Very-High-Speed Integrated Circuits

VHDL VHSIC Hardware Description Language

FPGA Field-progmable Gate Array

IDE Integrated Development Environment

LUT Look-Up Table

ROM Read Only Memory

DH Diffie-Hellman

RAM Random Access Memory

ECC Elliptic curve cryptography

RSA Rivest-Shamir-Adleman

FSM Finite State Machine

XST Xilinx Synthesis Tool

CPU Central Processing Unit

EDK Embedded Development Kit

Bibliography

[1] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems
`http://people.csail.mit.edu/rivest/Rsapaper.pdf`

[2] A Public Key Block Cipher Based on Multivariate Quadratic Quasigroups
`http://arxiv.org/abs/0808.0247`

[3] High Performance Implementation of a Public Key Block Cipher - MQQ, for FPGA Platforms
`http://eprint.iacr.org/2008/339`

[4] Logic Minimization Algorithms for VLSI Synthesis, ISBN 0-89838-164-9, 1984
`http://portal.acm.org/citation.cfm?id=577427`

[5] Module 4: Logic minimization, Digilent
`www.eecs.wsu.edu/~ee214/Fall2008/M4.pdf`

[6] ESPRESSO: Logic Minimization Software
`http://diamond.gem.valpo.edu/~dhart/ece110/espresso/tutorial.html`

[7] MicroBlaze
`http://en.wikipedia.org/w/index.php?title=MicroBlaze&oldid=295605037`

[8] TTM4530 report, FPGA implemetation of a public key block cipher, report located in the digital attachment

Appendix A

VHDL source code

In this chapter shortened versions of the VHDL source code is listed. Full versions of the source code files are located in the digital attachment. This is done because several source code files have over 10000 lines of code.

A.1 Encryption

Listing A.1: Encryption top module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity encryption is

  port (
    clk    : in std_logic;
    reset  : in std_logic;
    en_in  : in std_logic;
    datain  : in std_logic_vector(159 downto 0);
    dataout : out std_logic_vector(159 downto 0)
  );
end encryption;

architecture rtl of encryption is
  type states is (IDLE,EXPAND,PUBL_MATR,RES);

  component expander
    port (
      clk,reset,en_in  : in std_logic;
      input          : in std_logic_vector(159 downto 0);
      output_1         : out std_logic_vector(159 downto 0);
      output_2         : out std_logic_vector(80 downto 0);
      en_out          : out std_logic

    );
  end component;

  component public_matrix
    port (
      clk,reset,en_in  : in std_logic;
      input_1         : in std_logic_vector(159 downto 0);
      input_2         : in std_logic_vector(80 downto 0);
      cnt           : in integer range 0 to 80;
      output          : out std_logic_vector(159 downto 0);
      en_out          : out std_logic;
      done            : out std_logic
    );
  end component;

  signal state,next_state    : states;
  signal cnt               : integer range 0 to 80;
  signal en_in_exp,en_in_pub  : std_logic;
  signal en_out_exp,en_out_pub  : std_logic;
  signal first_data          : std_logic_vector(159 downto 0);
  signal second_data         : std_logic_vector(80 downto 0);
  signal final_result        : std_logic_vector(159 downto 0);
  signal ready               : std_logic;

begin

  EXP:    expander
  port map(
    clk    => clk,
    reset    => reset,
    en_in    => en_in_exp,
    input    => datain,
    output_1  => first_data,
    output_2  =>  second_data,
    en_out   =>  en_out_exp
  );

  PUB_MATR: public_matrix
  port map (
    clk    => clk,
    reset    => reset,
    en_in    =>  en_in_pub,
    input_1 => first_data,
    input_2 => second_data,
    cnt    => cnt,
    output  =>  final_result,
    en_out  => en_out_pub,
    done     => ready
  );
  sync_run: process(clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        state <= IDLE;
        dataout <= (others => '0');

      elsif (en_in = '1') then
        state <= next_state;
        dataout <= final_result;
      end if;
    end if;
  end process;

  output_decode:  process(clk,state)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        en_in_exp <= '0';
        en_in_pub <= '0';

      elsif (en_in = '1') then
        case (state) is
          when IDLE     =>
            en_in_exp <=  '0';
            en_in_pub <=  '0';

          when EXPAND     =>
            en_in_exp <=  '1';
            en_in_pub <=  '0';

          when PUBL_MATR    =>
            en_in_exp <=  '0';
            en_in_pub <=  '1';
            if (en_out_exp = '1') then
              cnt <= cnt + 1;
              if (cnt = 79) then
                cnt <= cnt + 0;
              end if;
            end if;
          when RES        =>
            --dataout <= final_result;
          when others =>
            null;
        end case;
      end if;
    end if;
  end process;

  next_state_decode: process(state,en_out_exp,ready)
  begin
```

```
    next_state <= state;
    case (state) is
      when IDLE      =>
        next_state <= EXPAND;

      when EXPAND      =>
        if (en_out_exp <= '1') then
          next_state <= PUBL_MATR;
        elsif (ready = '1') then
          next_state <= RES;
        else
          next_state <= EXPAND;
        end if;
      when PUBL_MATR     =>
        if (en_out_pub <= '1') then
          next_state <= EXPAND;
        elsif (ready = '1') then
          next_state <= RES;
        else
          next_state <= PUBL_MATR;
        end if;
      when RES         =>
        next_state <= IDLE;
      when others      =>
        null;
    end case;
  end process;
end rtl;
```

Listing A.2: Expander

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity expander is
  port (
    clk,reset,en_in  : in std_logic;
    input          : in std_logic_vector(159 downto 0);
    output_1         : out std_logic_vector(159 downto 0);
    output_2         : out std_logic_vector(80 downto 0);
    en_out         : out std_logic
  );

end expander;

architecture rtl of expander is




  type states is (IDLE,COMB,SEL);
  signal state,next_state : states;

  signal vector        : std_logic_vector (12880 downto 0);
  signal tmp           : std_logic_vector (159 downto 0);
  signal inc           : integer range 0 to 12880;

begin

  run:  process(clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        state <= IDLE;
      elsif (en_in = '1') then
        state <= next_state;
      end if;
    end if;
  end process;

  output_decode:  process (clk,state)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        vector <= (others => '0');
        en_out <= '0';
```

```
          tmp <= (others => '0');
          inc <= 0;
          output_1 <= (others => '0');
          output_2 <= (others => '0');
        elsif (en_in = '1') then
          case (state) is
            when IDLE =>
              tmp <= input;
              en_out <= '0';
            when COMB =>
              vector(0)                <= '1';
              vector(160 downto 1)       <= tmp;                              --160
              vector(319 downto 161)       <= tmp(158 downto 0) and input(159 downto 1); --159
              vector(477 downto 320)       <= tmp(157 downto 0) and input(159 downto 2); --158
              vector(634 downto 478)       <= tmp(156 downto 0) and input(159 downto 3); --157
              vector(790 downto 635)       <= tmp(155 downto 0) and input(159 downto 4); --156
              vector(945 downto 791)       <= tmp(154 downto 0) and input(159 downto 5); --155
              vector(1099 downto 946)    <= tmp(153 downto 0) and input(159 downto 6); --154
              vector(1252 downto 1100)     <= tmp(152 downto 0) and input(159 downto 7); --153
              vector(1404 downto 1253)     <= tmp(151 downto 0) and input(159 downto 8); --152
              vector(1555 downto 1405)     <= tmp(150 downto 0) and input(159 downto 9); --151
              vector(1705 downto 1556)     <= tmp(149 downto 0) and input(159 downto 10);--150
              vector(1854 downto 1706)     <= tmp(148 downto 0) and input(159 downto 11);--149
              vector(2002 downto 1855)     <= tmp(147 downto 0) and input(159 downto 12);--148
              vector(2149 downto 2003)     <= tmp(146 downto 0) and input(159 downto 13);--147
              vector(2295 downto 2150)     <= tmp(145 downto 0) and input(159 downto 14);--146
              vector(2440 downto 2296)     <= tmp(144 downto 0) and input(159 downto 15);--145
              vector(2584 downto 2441)     <= tmp(143 downto 0) and input(159 downto 16);--144
              vector(2727 downto 2585)     <= tmp(142 downto 0) and input(159 downto 17);--143
              vector(2869 downto 2728)     <= tmp(141 downto 0) and input(159 downto 18);--142
              vector(3010 downto 2870)     <= tmp(140 downto 0) and input(159 downto 19);--141
              vector(3150 downto 3011)     <= tmp(139 downto 0) and input(159 downto 20);--140
              vector(3289 downto 3151)     <= tmp(138 downto 0) and input(159 downto 21);--139
              vector(3427 downto 3290)     <= tmp(137 downto 0) and input(159 downto 22);--138
              vector(3564 downto 3428)     <= tmp(136 downto 0) and input(159 downto 23);--137
              vector(3700 downto 3565)     <= tmp(135 downto 0) and input(159 downto 24);--136
              vector(3835 downto 3701)     <= tmp(134 downto 0) and input(159 downto 25);--135
              vector(3969 downto 3836)     <= tmp(133 downto 0) and input(159 downto 26);--134
              vector(4102 downto 3970)     <= tmp(132 downto 0) and input(159 downto 27);--133
              vector(4234 downto 4103)     <= tmp(131 downto 0) and input(159 downto 28);--132
              vector(4365 downto 4235)     <= tmp(130 downto 0) and input(159 downto 29);--131
              vector(4495 downto 4366)     <= tmp(129 downto 0) and input(159 downto 30);--130
              vector(4624 downto 4496)     <= tmp(128 downto 0) and input(159 downto 31);--129
              vector(4752 downto 4625)     <= tmp(127 downto 0) and input(159 downto 32);--128
              vector(4879 downto 4753)     <= tmp(126 downto 0) and input(159 downto 33);--127
              vector(5005 downto 4880)     <= tmp(125 downto 0) and input(159 downto 34);--126
              vector(5130 downto 5006)     <= tmp(124 downto 0) and input(159 downto 35);--125
              vector(5254 downto 5131)     <= tmp(123 downto 0) and input(159 downto 36);--124
              vector(5377 downto 5255)     <= tmp(122 downto 0) and input(159 downto 37);--123
              vector(5499 downto 5378)     <= tmp(121 downto 0) and input(159 downto 38);--122
              vector(5620 downto 5500)     <= tmp(120 downto 0) and input(159 downto 39);--121
              vector(5740 downto 5621)     <= tmp(119 downto 0) and input(159 downto 40);--120
              vector(5859 downto 5741)     <= tmp(118 downto 0) and input(159 downto 41);--119
              vector(5977 downto 5860)     <= tmp(117 downto 0) and input(159 downto 42);--118
              vector(6094 downto 5978)     <= tmp(116 downto 0) and input(159 downto 43);--117
              vector(6210 downto 6095)     <= tmp(115 downto 0) and input(159 downto 44);--116
              vector(6325 downto 6211)     <= tmp(114 downto 0) and input(159 downto 45);--115
              vector(6439 downto 6326)     <= tmp(113 downto 0) and input(159 downto 46);--114
              vector(6552 downto 6440)     <= tmp(112 downto 0) and input(159 downto 47);--113
              vector(6664 downto 6553)     <= tmp(111 downto 0) and input(159 downto 48);--112
              vector(6775 downto 6665)     <= tmp(110 downto 0) and input(159 downto 49);--111
              vector(6885 downto 6776)     <= tmp(109 downto 0) and input(159 downto 50);--110
              vector(6994 downto 6886)     <= tmp(108 downto 0) and input(159 downto 51);--109
              vector(7102 downto 6995)     <= tmp(107 downto 0) and input(159 downto 52);--108
              vector(7209 downto 7103)     <= tmp(106 downto 0) and input(159 downto 53);--107
              vector(7315 downto 7210)     <= tmp(105 downto 0) and input(159 downto 54);--106
              vector(7420 downto 7316)     <= tmp(104 downto 0) and input(159 downto 55);--105
              vector(7524 downto 7421)     <= tmp(103 downto 0) and input(159 downto 56);--104
              vector(7627 downto 7525)     <= tmp(102 downto 0) and input(159 downto 57);--103
              vector(7729 downto 7628)     <= tmp(101 downto 0) and input(159 downto 58);--102
              vector(7830 downto 7730)     <= tmp(100 downto 0) and input(159 downto 59);--101
              vector(7930 downto 7831)     <= tmp(99 downto 0) and input(159 downto 60);-- 100
              vector(8029 downto 7931)     <= tmp(98 downto 0) and input(159 downto 61);--   99
              vector(8127 downto 8030)     <= tmp(97 downto 0) and input(159 downto 62);--   98
              vector(8224 downto 8128)     <= tmp(96 downto 0) and input(159 downto 63);--   97
              vector(8320 downto 8225)     <= tmp(95 downto 0) and input(159 downto 64);--   96
              vector(8415 downto 8321)     <= tmp(94 downto 0) and input(159 downto 65);--   95
              vector(8509 downto 8416)     <= tmp(93 downto 0) and input(159 downto 66);--   94
              vector(8602 downto 8510)     <= tmp(92 downto 0) and input(159 downto 67);--   93
              vector(8694 downto 8603)     <= tmp(91 downto 0) and input(159 downto 68);--   92
              vector(8785 downto 8695)     <= tmp(90 downto 0) and input(159 downto 69);--   91
              vector(8875 downto 8786)     <= tmp(89 downto 0) and input(159 downto 70);--   90
              vector(8964 downto 8876)     <= tmp(88 downto 0) and input(159 downto 71);--   89
              vector(9052 downto 8965)     <= tmp(87 downto 0) and input(159 downto 72);--   88
              vector(9139 downto 9053)    <= tmp(86 downto 0) and input(159 downto 73);--  87
              vector(9225 downto 9140)    <= tmp(85 downto 0) and input(159 downto 74);--  86
              vector(9310 downto 9226)    <= tmp(84 downto 0) and input(159 downto 75);--  85
              vector(9394 downto 9311)    <= tmp(83 downto 0) and input(159 downto 76);--  84
              vector(9477 downto 9395)    <= tmp(82 downto 0) and input(159 downto 77);--  83
              vector(9559 downto 9478)    <= tmp(81 downto 0) and input(159 downto 78);--  82
              vector(9640 downto 9560)    <= tmp(80 downto 0) and input(159 downto 79);--  81
              vector(9720 downto 9641)    <= tmp(79 downto 0) and input(159 downto 80);--  80
              vector(9799 downto 9721)    <= tmp(78 downto 0) and input(159 downto 81);--  79
              vector(9877 downto 9800)    <= tmp(77 downto 0) and input(159 downto 82);--  78
              vector(9954 downto 9878)    <= tmp(76 downto 0) and input(159 downto 83);--  77
              vector(10030 downto 9955)   <= tmp(75 downto 0) and input(159 downto 84);--  76
              vector(10105 downto 10031)  <= tmp(74 downto 0) and input(159 downto 85);--  75
              vector(10179 downto 10106)  <= tmp(73 downto 0) and input(159 downto 86);--  74
              vector(10252 downto 10180)  <= tmp(72 downto 0) and input(159 downto 87);--  73
              vector(10324 downto 10253)  <= tmp(71 downto 0) and input(159 downto 88);--  72
              vector(10395 downto 10325)  <= tmp(70 downto 0) and input(159 downto 89);--  71
              vector(10465 downto 10396)  <= tmp(69 downto 0) and input(159 downto 90);--  70
              vector(10534 downto 10466)  <= tmp(68 downto 0) and input(159 downto 91);--  69
              vector(10602 downto 10535)  <= tmp(67 downto 0) and input(159 downto 92);--  68
              vector(10669 downto 10603)  <= tmp(66 downto 0) and input(159 downto 93);--  67
              vector(10735 downto 10670)  <= tmp(65 downto 0) and input(159 downto 94);--  66
              vector(10800 downto 10736)  <= tmp(64 downto 0) and input(159 downto 95);--  65
              vector(10864 downto 10801)  <= tmp(63 downto 0) and input(159 downto 96);--  64
              vector(10927 downto 10865)  <= tmp(62 downto 0) and input(159 downto 97);--  63
              vector(10989 downto 10928)  <= tmp(61 downto 0) and input(159 downto 98);--  62
              vector(11050 downto 10990)  <= tmp(60 downto 0) and input(159 downto 99);--  61
              vector(11110 downto 11051)  <= tmp(59 downto 0) and input(159 downto 100);-- 60
              vector(11169 downto 11111)  <= tmp(58 downto 0) and input(159 downto 101);-- 59
              vector(11227 downto 11170)  <= tmp(57 downto 0) and input(159 downto 102);-- 58
              vector(11284 downto 11228)  <= tmp(56 downto 0) and input(159 downto 103);-- 57
              vector(11340 downto 11285)  <= tmp(55 downto 0) and input(159 downto 104);-- 56
              vector(11395 downto 11341)  <= tmp(54 downto 0) and input(159 downto 105);-- 55
              vector(11449 downto 11396)  <= tmp(53 downto 0) and input(159 downto 106);-- 54
              vector(11502 downto 11450)  <= tmp(52 downto 0) and input(159 downto 107);-- 53
              vector(11554 downto 11503)  <= tmp(51 downto 0) and input(159 downto 108);-- 52
              vector(11605 downto 11555)  <= tmp(50 downto 0) and input(159 downto 109);-- 51
              vector(11655 downto 11606)  <= tmp(49 downto 0) and input(159 downto 110);-- 50
              vector(11704 downto 11656)  <= tmp(48 downto 0) and input(159 downto 111);-- 49
              vector(11752 downto 11705)  <= tmp(47 downto 0) and input(159 downto 112);-- 48
              vector(11799 downto 11753)  <= tmp(46 downto 0) and input(159 downto 113);-- 47
              vector(11845 downto 11800)  <= tmp(45 downto 0) and input(159 downto 114);-- 46
              vector(11890 downto 11846)  <= tmp(44 downto 0) and input(159 downto 115);-- 45
              vector(11934 downto 11891)  <= tmp(43 downto 0) and input(159 downto 116);-- 44
              vector(11977 downto 11935)  <= tmp(42 downto 0) and input(159 downto 117);-- 43
              vector(12019 downto 11978)  <= tmp(41 downto 0) and input(159 downto 118);-- 42
              vector(12060 downto 12020)  <= tmp(40 downto 0) and input(159 downto 119);-- 41
              vector(12100 downto 12061)  <= tmp(39 downto 0) and input(159 downto 120);-- 40
              vector(12139 downto 12101)  <= tmp(38 downto 0) and input(159 downto 121);-- 39
              vector(12177 downto 12140)  <= tmp(37 downto 0) and input(159 downto 122);-- 38
              vector(12214 downto 12178)  <= tmp(36 downto 0) and input(159 downto 123);-- 37
              vector(12250 downto 12215)  <= tmp(35 downto 0) and input(159 downto 124);-- 36
              vector(12285 downto 12251)  <= tmp(34 downto 0) and input(159 downto 125);-- 35
              vector(12319 downto 12286)  <= tmp(33 downto 0) and input(159 downto 126);-- 34
              vector(12352 downto 12320)  <= tmp(32 downto 0) and input(159 downto 127);-- 33
              vector(12384 downto 12353)  <= tmp(31 downto 0) and input(159 downto 128);-- 32
              vector(12415 downto 12385)  <= tmp(30 downto 0) and input(159 downto 129);-- 31
              vector(12445 downto 12416)  <= tmp(29 downto 0) and input(159 downto 130);-- 30
              vector(12474 downto 12446)  <= tmp(28 downto 0) and input(159 downto 131);-- 29
              vector(12502 downto 12475)  <= tmp(27 downto 0) and input(159 downto 132);-- 28
              vector(12529 downto 12503)  <= tmp(26 downto 0) and input(159 downto 133);-- 27
              vector(12555 downto 12530)  <= tmp(25 downto 0) and input(159 downto 134);-- 26
              vector(12580 downto 12556)  <= tmp(24 downto 0) and input(159 downto 135);-- 25
              vector(12604 downto 12581)  <= tmp(23 downto 0) and input(159 downto 136);-- 24
              vector(12627 downto 12605)  <= tmp(22 downto 0) and input(159 downto 137);-- 23
              vector(12649 downto 12628)  <= tmp(21 downto 0) and input(159 downto 138);-- 22
              vector(12670 downto 12650)  <= tmp(20 downto 0) and input(159 downto 139);-- 21
              vector(12690 downto 12671)  <= tmp(19 downto 0) and input(159 downto 140);-- 20
              vector(12709 downto 12691)  <= tmp(18 downto 0) and input(159 downto 141);-- 19
              vector(12727 downto 12710)  <= tmp(17 downto 0) and input(159 downto 142);-- 18
              vector(12744 downto 12728)  <= tmp(16 downto 0) and input(159 downto 143);-- 17
              vector(12760 downto 12745)  <= tmp(15 downto 0) and input(159 downto 144);-- 16
              vector(12775 downto 12761)  <= tmp(14 downto 0) and input(159 downto 145);-- 15
              vector(12789 downto 12776)  <= tmp(13 downto 0) and input(159 downto 146);-- 14
              vector(12802 downto 12790)  <= tmp(12 downto 0) and input(159 downto 147);-- 13
              vector(12814 downto 12803)  <= tmp(11 downto 0) and input(159 downto 148);-- 12
              vector(12825 downto 12815)  <= tmp(10 downto 0) and input(159 downto 149);-- 11
              vector(12835 downto 12826)  <= tmp(9 downto 0) and input(159 downto 150); -- 10
              vector(12844 downto 12836)  <= tmp(8 downto 0) and input(159 downto 151); --  9
              vector(12852 downto 12845)  <= tmp(7 downto 0) and input(159 downto 152); --  8
              vector(12859 downto 12853)  <= tmp(6 downto 0) and input(159 downto 153); --  7
              vector(12865 downto 12860)  <= tmp(5 downto 0) and input(159 downto 154); --  6
              vector(12870 downto 12866)  <= tmp(4 downto 0) and input(159 downto 155); --  5
              vector(12874 downto 12871)  <= tmp(3 downto 0) and input(159 downto 156); --  4
              vector(12877 downto 12875)  <= tmp(2 downto 0) and input(159 downto 157); --  3
              vector(12879 downto 12878)  <= tmp(1 downto 0) and input(159 downto 158); --  2
              vector(12880)           <= tmp(0) and input(159);               --  1


            when SEL  =>
              -- the output_1 register value is determined by
              -- the value of the counter, which increases by 160
              -- each time this state is active
              output_1 <= vector((159 + inc) downto (0 + inc));
              output_2 <= vector(12880 downto 12800);
              if (inc >= 12640) then
                inc <= inc + 0;
              else
                inc <= inc + 160;
              end if;
              en_out <= '1';
          end case;
        end if;
      end if;
    end process;

    next_decode: process(state,inc)
    begin
      next_state <= state;
      case (state) is
        when IDLE =>
          next_state <= COMB;
        when COMB =>
          next_state <= SEL;
        when SEL    =>
          next_state <= IDLE;
          --end if;
      end case;
    end process;
  end rtl;
```

Listing A.3: Public Matrix

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity public_matrix is
  port (
    clk,reset,en_in   : in std_logic;
    input_1         : in std_logic_vector(159 downto 0);
    input_2         : in std_logic_vector(80 downto 0);
    cnt           : in integer range 0 to 80;
    output          : out std_logic_vector(159 downto 0);
    en_out          : out std_logic;
    done            : out std_logic
  );
end public_matrix;

architecture rtl of public_matrix is
  type states is (IDLE,MATR_AND,MATR_XOR160,MATR_XOR1,SYNC_160,SYNC_1,SYNC,SEND);
  type pr_rom is array (79 downto 0) of std_logic_vector(159 downto 0);
  type pr2_rom is array (0 downto 0) of std_logic_vector(80 downto 0);

constant rom160_1 : pr_rom  := (
"
    10101000111001100011111010101010000000000000000111110010011001100111011011010101011000001101000111010101101111101110110
    ",
"
    11110100010000111101010101101010101111110011000110101111011110100010101001111101111010111010000011000000011100011101
    ",
"
    01100001111000110011111111000000110011110010001100010101011011110111000110110101110101010111000001000011010101010 01
    ",
.
.
.

constant rom81_1   : std_logic_vector(80 downto 0) := (
"011010011011001110100111110000100011110001100001110101000001011010000000010000110");

constant rom81_2   : std_logic_vector(80 downto 0) := (
"100100100001011011010011111110100010010111010101100110001010011110111001110100011");
.
.
.


  signal and_rom1_160     : std_logic_vector(159 downto 0);
.
.
.
  signal and_rom1_81      : std_logic_vector(80 downto 0);
.
.
.
  signal xor1_160     : std_logic_vector(159 downto 0);
.
.
.
  signal sync_xor1_160      : std_logic_vector(159 downto 0);
.
.
.
  signal indata_1       : std_logic_vector(159 downto 0);
  signal indata_2       : std_logic_vector(80 downto 0);
  signal result         : std_logic_vector(159 downto 0);

  signal xor1_1     : std_logic;
.
.
.
  signal xor1_2     : std_logic;
.
.
.
  signal sync_xor1_1      : std_logic;
.
.
.
  signal sync_xor1_2      : std_logic;
.
.
.
--  signal cnt             : integer range 0 to 80;
  signal cnt_2          : integer range 0 to 160;
  signal state,next_state : states;

begin
  sync_proc: process(clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        state <= IDLE;
      else
        state <= next_state;
      end if;
    end if;
  end process;

  out_decode: process(state,clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        indata_1 <= (others => '0');
        indata_2 <= (others => '0');

        and_rom1_160 <= (others => '0');
.
.
.
        and_rom1_81 <= (others => '0');
.
.
.
        xor1_160 <= (others => '0');
.
.
.
        sync_xor1_160 <= (others => '0');
.
.
.
        xor1_1 <= '0';
.
.
```

```
.
        xor1_2 <= '0';
.
.
.
        sync_xor1_1 <= '0';
.
.
.
        sync_xor1_2 <= '0';
.
.
.
        cnt_2 <= 0;
        en_out <= '0';
        done <= '0';
        output <= (others => '0');
        result <= (others => '0');

      else
        case (state) is
          when IDLE =>
            indata_1 <= input_1;
            indata_2  <= input_2;
            en_out <= '0';
            done <= '0';
          when MATR_AND =>
            and_rom1_160 <= rom160_1(cnt) and indata_1;
.
.
,
            if (cnt <= 79) then
              and_rom1_81 <= rom81_1 and indata_2;
.
.
.           end if;

          when MATR_XOR160 =>
            if (cnt = 0) then
              sync_xor1_160 <= and_rom1_160;
.
.
.             en_out <= '1';
            else
              xor1_160 <= and_rom1_160 xor sync_xor1_160;
.
.
.           end if;

          when MATR_XOR1 =>
            en_out <= '0';
            if (cnt_2 = 0) then
              sync_xor1_1 <= sync_xor1_160(cnt_2);
.
.
.
              sync_xor1_2 <= and_rom1_81(cnt_2);
.
.
.
            else
              xor1_1 <= sync_xor1_160(cnt_2) xor sync_xor1_1;
.
.
.
              if (cnt_2 <= 80) then
                xor1_2 <= sync_xor1_2 xor and_rom1_81(cnt_2);
.
.
.

              end if;
            end if;
          when SYNC_160 =>
            sync_xor1_160 <= xor1_160;
.
.
.           en_out <= '1';
          when SYNC_1 =>
            sync_xor1_1 <= xor1_1;
.
.
.
            sync_xor1_2 <= xor1_2;
```

```
.
.
.
            if (cnt_2 = 159) then
              cnt_2 <= cnt_2 + 0;
            else
              cnt_2 <= cnt_2 + 1;
            end if;

          when SYNC =>
            result(0) <= sync_xor1_1 xor sync_xor1_2;
.
.
.
          when SEND =>
            output <= result;
            done <= '1';
        end case;
      end if;
    end if;
  end process;

  next_decode: process(state,cnt,cnt_2)
  begin
    next_state <= state;
    case (state) is
      when IDLE => next_state <= MATR_AND;

      when MATR_AND => next_state <= MATR_XOR160;

      when MATR_XOR160 => next_state <= SYNC_160;

      when MATR_XOR1 => next_state <= SYNC_1;

      when SYNC_160 =>
        if (cnt = 79) then
          next_state <= MATR_XOR1;
        else
          next_state <= IDLE;
        end if;

      when SYNC_1 =>
        if (cnt_2 = 159) then
          next_state <= SYNC;
        else
          next_state <=MATR_XOR1;
        end if;
      when SYNC => next_state <= SEND;

      when SEND =>
        next_state <= IDLE;
    end case;
  end process;
end rtl;
```

A.2 Decryption

Listing A.4: Decryption top module

```
--------------------------------------------------------------------------------
-- Company: NTNU
-- Engineer: Stig Fjellskaalnes
--
-- Create Date:    13:14:48 09/26/2008
-- Design Name:
-- Module Name:    decryption - rtl
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--------------------------------------------------------------------------------
library IEEE;
```

```vhdl
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity decryption is
  port (
    clk     : in std_logic;
    reset   : in std_logic;
    en_in   : in std_logic;
    inputs  : in std_logic_vector (159 downto 0);
    outputs : out std_logic_vector (159 downto 0);
    en_out  : out std_logic
    );

end decryption;

architecture rtl of decryption is

  type states is (IDLE,SEND_TO_PM_T,RECV_FROM_PM_T,SEND_TO_DOB,
          RECV_FROM_DOB,SEND_TO_SEQ,RECV_FROM_SEQ,
          SEND_TO_PM_S,RECV_FROM_PM_S,SEND);

  component private_matrix_s
    port (
      clk      : in std_logic;
      reset     : in std_logic;
      en_in     : in std_logic;
      input     : in std_logic_vector (4 downto 0);
      cnt      : in std_logic_vector (4 downto 0);
      output    : out std_logic_vector (159 downto 0);
      en_out    : out std_logic
    );
  end component;

  component dobbertin_rom      -- defining the Dobbertin component
                       -- in the decryption top level
    port (
      z             : in std_logic_vector (12 downto 0);
      clk,reset,en_in : in std_logic;
      db             : out std_logic_vector(12 downto 0);
      en_out         : out std_logic
    );
  end component;

  component sequencer
    port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        en_in     : in  std_logic;
        input_seq : in   std_logic_vector (159 downto 0);
        en_out    : out std_logic;
        output_seq  : out std_logic_vector (159 downto 0)
    );
  end component;

  signal state,next_state : states;
  --signal dec_input      : std_logic_vector (159 downto 0);
  signal dec_output      : std_logic_vector (159 downto 0);
  signal result_pmt      : std_logic_vector (159 downto 0);
  signal result_pms      : std_logic_vector (159 downto 0);
  signal result_seq      : std_logic_vector (159 downto 0);
  signal seq_in        : std_logic_vector (159 downto 0);
  signal dob_vector_in     : std_logic_vector (12 downto 0);
  signal dob_vector_out : std_logic_vector (12 downto 0);
  signal count_t         : std_logic_vector (4 downto 0);           -- counter private_matrix t
  signal count_s         : std_logic_vector (4 downto 0);           -- counter private_matrix s
  signal shift_pmt       : std_logic_vector (4 downto 0);
  signal shift_pms       : std_logic_vector (4 downto 0);
  signal en_in_pm_t      : std_logic;
  signal en_in_pm_s      : std_logic;
  signal en_in_dob       : std_logic;
  signal en_in_seq       : std_logic;
  signal en_out_seq      : std_logic;
  signal en_out_pm_t     : std_logic;
  signal en_out_pm_s     : std_logic;
  signal en_out_dob      : std_logic;

begin
  DR: DOBBERTIN_ROM         -- initializing the Dobbertin
```

```
                      -- ROM component in the decryption
                      --  circuit
  port map(
    clk      =>  clk,
    reset      =>  reset,
    en_in      =>  en_in_dob,
    en_out     =>  en_out_dob,
    z        =>  dob_vector_in,
    db         => dob_vector_out
  );

  PM_T: PRIVATE_MATRIX_S      -- initializing the Private Matrix
                      -- (T) component in the decryption
                      --  circuit
  port map(
    clk      =>  clk,
    reset      =>  reset,
    en_in      =>  en_in_pm_t,
    input      =>  shift_pmt,
    cnt      => count_t,
    output     =>  result_pmt,
    en_out     =>  en_out_pm_t
  );

  SEQ: SEQUENCER            -- initilizing the sequencer
                      -- component in the decryption
                      -- circuit

  port map(
    clk      => clk,
    reset      =>  reset,
    en_in      => en_in_seq,
    input_seq => seq_in,
    en_out     => en_out_seq,
    output_seq  => result_seq
    );

  PM_S: PRIVATE_MATRIX_S      -- initializing the Private Matrix
                      -- (T) component in the decryption
                      --  circuit
  port map(
    clk      =>  clk,
    reset      =>  reset,
    en_in      =>  en_in_pm_s,
    input      =>  shift_pms,
    cnt      => count_s,
    output     =>  result_pms,
    en_out     =>  en_out_pm_s
    );

  running: process (clk, en_in)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        state <= IDLE;

      elsif (en_in = '1') then
        state <= next_state;

      end if;
    end if;
  end process;

  output_dec: process (clk,en_in,state)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        dob_vector_in <= (others => '0');
        --dec_input <= (others => '0');
        dec_output <= (others => '0');
--          result_pmt <= (others => '0');
--          result_pms <= (others => '0');
--          result_seq <= (others => '0');
        outputs <= (others => '0');
        shift_pmt <= (others => '0');
        shift_pms <= (others => '0');
        en_out <= '0';
        en_in_dob <= '0';
        en_in_pm_t <= '0';
        en_in_pm_s <= '0';
--          en_out_dob <= '0';
--          en_out_seq <= '0';
        en_in_seq <= '0';
        count_t <= (others => '0');
        count_s <= (others => '0');

        seq_in <= (others => '0');


      elsif (en_in = '1') then
        case (state) is
          when IDLE =>

            --dec_input <=
                "100110110100111100110010110011011001101101001111001100101100110110011011010011110011001011001101100110

            en_in_pm_t <= '0';
            if (count_t <= "11111") then
              count_t <= (others => '0');
            end if;


          when SEND_TO_PM_T =>
            en_in_pm_t <= '1';
            if (count_t = "11111" and en_out_pm_t = '1') then
              null;
            else
              shift_pmt <= inputs ((4+(conv_integer(count_t)*5)) downto (0+(conv_integer(count_t)*5)))
                  ;
              if (en_out_pm_t = '1') then
                count_t <= count_t + 1;
              end if;
            end if;

          when RECV_FROM_PM_T =>

            case (result_pmt) is
              when ("
                  ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
                  ") =>
                null;
              when others =>
                en_in_dob <= '1';
            end case;

          when SEND_TO_DOB =>
            en_in_pm_t <= '0';
            en_in_dob <= '1';
            dob_vector_in(5 downto 0) <= result_pmt(5 downto 0);
            dob_vector_in(6) <= result_pmt(10);
            dob_vector_in(7) <= result_pmt(15);
            dob_vector_in(8) <= result_pmt(20);
            dob_vector_in(9) <= result_pmt(25);
            dob_vector_in(10) <= result_pmt(30);
            dob_vector_in(11) <= result_pmt(35);
            dob_vector_in(12) <= result_pmt(40);

          when RECV_FROM_DOB =>
            if (en_out_dob = '1') then
              seq_in <= result_pmt;
              case (dob_vector_out) is
                when "ZZZZZZZZZZZZZ" =>
                  null;

                when others =>
                  seq_in(5 downto 0) <= dob_vector_out(5 downto 0);
                  seq_in(10) <= dob_vector_out(6);
                  seq_in(15) <= dob_vector_out(7);
                  seq_in(20) <= dob_vector_out(8);
                  seq_in(25) <= dob_vector_out(9);
                  seq_in(30) <= dob_vector_out(10);
                  seq_in(35) <= dob_vector_out(11);
                  seq_in(40) <= dob_vector_out(12);
                  --en_out_dob <= '0';
              end case;
            end if;

          when SEND_TO_SEQ =>
            en_in_seq <= '1';
            en_in_dob <= '0';
          when RECV_FROM_SEQ =>
            en_in_seq <= '0';
          when SEND_TO_PM_S =>
            en_in_pm_s <= '1';
            if (count_s = "11111" and en_out_pm_s = '1') then
              null;
            else
              shift_pms <= result_seq(((4+(conv_integer(count_s)*5))) downto (0+((conv_integer(count_s
                  )*5))));
              if (en_out_pm_s = '1') then
                count_s <= count_s + 1;
              end if;
            end if;

          when RECV_FROM_PM_S =>
            en_in_pm_s <= '0';
            dec_output <= result_pms;

          when SEND =>
            outputs <= dec_output;
          when others =>
            null;
        end case;
      end if;
    end if;
  end process;

  -- calculates next state

  next_state_dec: process(state,count_t,en_out_pm_t,
              en_out_seq,count_s,en_out_pm_s)
  begin
    next_state <= state;

    case (state) is
      --
      --
      when IDLE =>
        next_state <= SEND_TO_PM_T;

      -- receives data from private matrix only when
      -- counter reaches 31, when the private matrix is done
      -- processing the 160bit data
      when SEND_TO_PM_T =>
        if (count_t = 31 and en_out_pm_t = '1') then
            next_state <= RECV_FROM_PM_T;
          else
            next_state <= SEND_TO_PM_T;

        end if;

      -- if the data is not valid, new data is being imported
      -- and the state machine moves back to IDLE.
      -- else, send data to dobbertin ROM
      when RECV_FROM_PM_T =>
        case result_pmt is
          when ("
              ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
              ") =>
            next_state <= IDLE;
          when others =>
            next_state <= SEND_TO_DOB;
        end case;

      when SEND_TO_DOB =>
          next_state <= RECV_FROM_DOB;

      when RECV_FROM_DOB =>
        next_state <= SEND_TO_SEQ;

      when SEND_TO_SEQ =>
        case (en_out_seq) is
          when '0' =>
            next_state <= SEND_TO_SEQ;
          when '1' =>
            next_state <= RECV_FROM_SEQ;
          when others =>
            next_state <= IDLE;
        end case;

      when RECV_FROM_SEQ =>
        next_state <= SEND_TO_PM_S;
      when SEND_TO_PM_S =>
        if (count_s = 31 and en_out_pm_s = '1') then
          next_state <= RECV_FROM_PM_S;
        else
          next_state <= SEND_TO_PM_S;
        end if;

      when RECV_FROM_PM_S =>
        case result_pmt is
          when ("
              ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

```
                  ") =>
              next_state <= IDLE;
            when others =>
              next_state <= SEND;
          end case;

        when SEND =>
          next_state <= IDLE;

        when others =>
          next_state <= IDLE;

    end case;
  end process;


end rtl;
```

Listing A.5: Private Matrix

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity private_matrix_s is
  port (
    clk    : in std_logic;
    reset    : in std_logic;
    en_in    : in std_logic;
    cnt    : in std_logic_vector (4 downto 0);
    input    : in std_logic_vector (4 downto 0);
    output   : out std_logic_vector (159 downto 0);
    en_out   : out std_logic
    );

end private_matrix_s;

architecture rtl of private_matrix_s is

  type priv_state is (IDLE,ANDOP,XORING,SYNC,PUSH_OUT);

  signal rom_001        :    std_logic_vector (4 downto 0);
  signal rom_002        :    std_logic_vector (4 downto 0);
.
.
.

  signal and_rom_in_1        :    std_logic_vector (4 downto 0);
  signal and_rom_in_2        :    std_logic_vector (4 downto 0);
.
.
.
  signal tmp          : std_logic_vector(159 downto 0);

  signal count_xor       : integer range 0 to 4 := 0;
  signal matr_xor      : std_logic_vector(159 downto 0);
  signal sync_xor      : std_logic_vector(159 downto 0);

  signal enable_out      : std_logic;
  signal state, next_state : priv_state;
  signal register_x      : std_logic_vector (159 downto 0);

begin
  sync_run: process(clk, en_in, enable_out, register_x)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        output <= (others => '0');
        en_out <= '0';
        state <= IDLE;

        --  cnt <= (others => '0');
        --count_xor <= 0;
      elsif (en_in = '1') then
        state <= next_state;
        en_out <= enable_out;
        if (cnt = "11111" and enable_out = '1') then
          output <= register_x;
        end if;
      end if;
    end if;
  end process;

  -- output_dec is made synchronous, so that all signals remember their
  -- value at all times when not set again
  output_dec: process (clk,state,cnt)
  begin
    if (clk'event and clk = '1') then

      if (reset = '1') then
--          output <= (others => '0');
        enable_out <= '0';
        count_xor <= 0;
        tmp <= (others => '0');
        matr_xor <= (others => '0');
        sync_xor <= (others => '0');
        register_x <= (others => '0');

        and_rom_in_1 <= "00000";
.
.
.
        rom_001 <= "00000";
.
.
.

      elsif (en_in = '1') then

        case (state) is

          when IDLE =>

            -- makes sure that en_out port is set to '0'
            enable_out <= '0';
            --output <= (others => '0');

rom_001(4) <= ( not cnt(2) and cnt(1) and cnt(0)) or (cnt(4) and not cnt(3) and not cnt(2) and cnt(1)) or (
     not cnt(4) and not cnt(3) and cnt(1) and cnt(0)) or (cnt(4) and not cnt(1) and not cnt(0)) or (cnt(4)
     and cnt(3) and cnt(1) and cnt(0)) or ( not cnt(4) and cnt(3) and cnt(1) and not cnt(0)) or ( not cnt(2)
     and not cnt(1) and not cnt(0));

rom_001(3) <= ( not cnt(4) and not cnt(3) and not cnt(2) and cnt(0)) or ( not cnt(4) and cnt(3) and cnt(1)
     and not cnt(0)) or ( not cnt(4) and not cnt(2) and cnt(1) and cnt(0)) or ( not cnt(2) and not cnt(1) and not cnt(0)) or (
    cnt(4) and cnt(3) and not cnt(1)) or ( not cnt(3) and cnt(2) and not cnt(0)) or (cnt(4) and cnt(2)
     and not cnt(1));

rom_001(2) <= (cnt(3) and cnt(2) and not cnt(0)) or (cnt(4) and cnt(2) and not cnt(1) and cnt(0)) or (
     not cnt(3) and cnt(2) and cnt(1)) or (cnt(4) and not cnt(2) and cnt(1) and cnt(0)) or ( not cnt(3)
     and not cnt(2) and cnt(0)) or (cnt(4) and not cnt(3) and not cnt(2) and not cnt(1)) or ( not cnt(4) and not cnt(3)
     and cnt(1)) or ( not cnt(4) and cnt(2) and not cnt(0));

rom_001(1) <= (cnt(3) and cnt(2) and not cnt(1) and cnt(0)) or (cnt(3) and not cnt(2) and cnt(1)
     and cnt(0)) or ( not cnt(4) and cnt(2) and cnt(1) and cnt(0)) or (cnt(4) and not cnt(3) and not cnt(1)
     and not cnt(0)) or (cnt(4) and not cnt(3) and not cnt(2) and not cnt(0)) or (cnt(4) and cnt(3) and cnt(2)
     and cnt(1) and not cnt(0)) or ( not cnt(3) and not cnt(2) and not cnt(1)) or ( not cnt(4) and cnt(3) and cnt(2)
     and not cnt(1)) or ( not cnt(4) and cnt(3) and not cnt(2) and cnt(1)) or (cnt(4) and not cnt(2) and not cnt(1));

rom_001(0) <= ( not cnt(3) and cnt(1) and not cnt(0)) or ( not cnt(3) and not cnt(2) and not cnt(1) and cnt(0)) or (
    cnt(4) and not cnt(2) and not cnt(1) and cnt(0)) or (cnt(4) and not cnt(3) and not cnt(1)) or ( not cnt(4)
     and cnt(3) and cnt(2)) or (cnt(3) and cnt(2) and cnt(1) and cnt(0)) or ( not cnt(4) and cnt(3)
     and not cnt(0)) or ( not cnt(2) and cnt(1) and not cnt(0));
.
.
.


          when ANDOP =>
            -- makes the logical and operation between the input
            -- value and the stored value inside the respective
            -- ROM
            and_rom_in_1 <= rom_001 and input;
.
```

```
.
.
          when XORING =>
            -- the actual bit-by-bit xor operation
            tmp(0) <= and_rom_in_1(count_xor);
            matr_xor(0) <= and_rom_in_1(count_xor+1) xor tmp(0);
.
.
.
            if not (count_xor = 3) then
              count_xor <= count_xor + 1;
            end if;

          when SYNC =>
            -- sets the xor counter to 0
            count_xor <= 0;
            -- keep synchronization of the xor'ed bits
            if (cnt = "00000") then
              sync_xor <= matr_xor;

              enable_out <= '1';
              -- does the last xor step between the previous
              -- and the current 5 bit input vector
            else

              sync_xor <= matr_xor xor sync_xor;
              enable_out <= '1';
            end if;

          when PUSH_OUT =>
            -- pushes out the processed vector on output
            enable_out <= '1';
            register_x <= sync_xor;

          when others =>

            enable_out <= '0';
            rom_001 <= (others => '0');
.
.
.
        end case;
      end if;
    end if;
  end process;

  next_state_dec: process (state,input,cnt,count_xor)
  begin
    next_state <= state;
    case (state) is
      when IDLE =>
        -- makes sure that rom has the correct value
        -- as to the running time given by signal cnt
        next_state <= ANDOP;
      when ANDOP =>
        next_state <= XORING;
      when XORING =>
        if (count_xor < 3) then
          next_state <= XORING;
        else
          next_state <= SYNC;
        end if;
      when SYNC =>
        if (cnt = "11111") then
          next_state <= PUSH_OUT;
        else
          next_state <= IDLE;
        end if;
      when PUSH_OUT =>
        next_state <= IDLE;
      when others =>
       next_state <= IDLE;
    end case;
  end process;

end rtl;
```

Listing A.6: Dobbertin ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
```

```


entity Dobbertin_ROM is

  port (

    z             : in std_logic_vector (12 downto 0);
    clk, reset, en_in  : in std_logic;
    db              : out std_logic_vector(12 downto 0);
    en_out          : out std_logic
  );
end Dobbertin_ROM;

architecture rtl of Dobbertin_ROM is
begin
  process(clk, reset, en_in)
  begin

    if(clk'event and clk = '1') then
      if (reset = '1') then
        db <= (others => '0');
        en_out <= '0';
      elsif (en_in = '1') then
        en_out <= '1';
          db(12) <= ( not z(12) and  not z(11) and  not z(10) and  not z(9) and z(8) and  not z(7) and
                z(6) and z(4) and  not z(3) and  not z(0))  or  ( .... and z(10) and  not z(9) and  not
                z(7)
          and z(6) and z(5) and  not z(4) and z(3) and z(1) and z(0));
        else
          en_out <= '0';
      end if;
    end if;
  end process;
end rtl;
```

Listing A.7: Sequencer

```

-- Company: NTNU
-- Engineer: Stig Fjellskaalnes
--
-- Create Date:     11:26:53 10/22/2008
-- Design Name:
-- Module Name:     sequencer - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sequencer is
    port (
      clk        : in  std_logic;
      reset      : in  std_logic;
      en_in      : in  std_logic;
      input_seq  : in   std_logic_vector (159 downto 0);
      en_out     : out std_logic;
      output_seq  : out std_logic_vector (159 downto 0)
    );
end sequencer;

architecture Behavioral of sequencer is

  component master_rom
    port (
      clk,  reset, en_in   : in std_logic;
      z            : in std_logic_vector (9 downto 0); -- address bus
      counter        : in std_logic_vector (4 downto 0);
      db             : out std_logic_vector (4 downto 0) -- data bus
    );
  end component;

  type reg_input is array ((2**5)-1 downto 0) of std_logic_vector (4 downto 0);
  type states is (IDLE,MUX2_SEL,SEND_TO_MASTER,RECV_MR,MUX31_SEL,SYNC,PUSH);
  signal reg_in,reg_out : reg_input;
  signal sel          : std_logic;
  signal counter_1      : std_logic_vector (4 downto 0);
  signal counter_2      : std_logic_vector (4 downto 0);
  signal mux31_out      : std_logic_vector (4 downto 0);
  signal mux2_out,sync_mr : std_logic_vector (4 downto 0);
  signal to_master      : std_logic_vector (9 downto 0);
  signal from_master    : std_logic_vector (4 downto 0);
  signal en_out_mux     : std_logic;
  signal state,next_state : states;


begin

  MA_R: master_rom         -- initializing the master rom
                      -- component in the sequencer
                      --  circuit
  port map(
    clk      =>  clk,
    reset      =>  reset,
    counter    => counter_2,
    en_in      =>  en_out_mux,
--     en_out     =>  en_out_mar,
    z        =>  to_master,
    db         => from_master
    );
  sec_run : process (clk, en_in, counter_1, counter_2)

  begin

  if (clk'event and clk = '1') then
    if (reset = '1') then
      state <= IDLE;


    elsif (en_in = '1') then
      state <= next_state;

    end if;
  end if;

  end process;

  output_dec: process(clk,en_in,state)
  begin

    if (clk'event and clk = '1') then

      if (reset = '1') then
        reg_in(0) <= (others => '0');
        reg_in(1) <= (others => '0');
        reg_in(2) <= (others => '0');
        reg_in(3) <= (others => '0');
        reg_in(4) <= (others => '0');
        reg_in(5) <= (others => '0');
        reg_in(6) <= (others => '0');
        reg_in(7) <= (others => '0');
        reg_in(8) <= (others => '0');
        reg_in(9) <= (others => '0');
        reg_in(10) <= (others => '0');
        reg_in(11) <= (others => '0');
        reg_in(12) <= (others => '0');
        reg_in(13) <= (others => '0');
        reg_in(14) <= (others => '0');
        reg_in(15) <= (others => '0');
        reg_in(16) <= (others => '0');
        reg_in(17) <= (others => '0');
        reg_in(18) <= (others => '0');
        reg_in(19) <= (others => '0');
        reg_in(20) <= (others => '0');
        reg_in(21) <= (others => '0');
        reg_in(22) <= (others => '0');
        reg_in(23) <= (others => '0');
        reg_in(24) <= (others => '0');
        reg_in(25) <= (others => '0');
        reg_in(26) <= (others => '0');
        reg_in(27) <= (others => '0');
        reg_in(28) <= (others => '0');
        reg_in(29) <= (others => '0');
        reg_in(30) <= (others => '0');
        reg_in(31) <= (others => '0');
        to_master <= (others => '0');
        en_out <= '0';
        sel <= '0';
        en_out_mux <= '0';
        mux31_out <= (others => '0');
        sync_mr <= (others => '0');
        mux2_out <= (others => '0');
        --from_master <= (others => '0');
        output_seq <= (others => '0');
        reg_out <= reg_in;
        counter_1 <= "00000";
        counter_2 <= "00000";

      elsif (en_in = '1') then
        case (state) is
          when IDLE =>
            reg_in(0) <= input_seq(4 downto 0);
            reg_in(1) <= input_seq(9 downto 5);
            reg_in(2) <= input_seq(14 downto 10);
            reg_in(3) <= input_seq(19 downto 15);
            reg_in(4) <= input_seq(24 downto 20);
            reg_in(5) <= input_seq(29 downto 25);
            reg_in(6) <= input_seq(34 downto 30);
            reg_in(7) <= input_seq(39 downto 35);
            reg_in(8) <= input_seq(44 downto 40);
            reg_in(9) <= input_seq(49 downto 45);
            reg_in(10) <= input_seq(54 downto 50);
            reg_in(11) <= input_seq(59 downto 55);
            reg_in(12) <= input_seq(64 downto 60);
            reg_in(13) <= input_seq(69 downto 65);
            reg_in(14) <= input_seq(74 downto 70);
            reg_in(15) <= input_seq(79 downto 75);
            reg_in(16) <= input_seq(84 downto 80);
            reg_in(17) <= input_seq(89 downto 85);
            reg_in(18) <= input_seq(94 downto 90);
            reg_in(19) <= input_seq(99 downto 95);
            reg_in(20) <= input_seq(104 downto 100);
            reg_in(21) <= input_seq(109 downto 105);
            reg_in(22) <= input_seq(114 downto 110);
            reg_in(23) <= input_seq(119 downto 115);
            reg_in(24) <= input_seq(124 downto 120);
            reg_in(25) <= input_seq(129 downto 125);
            reg_in(26) <= input_seq(134 downto 130);
            reg_in(27) <= input_seq(139 downto 135);
            reg_in(28) <= input_seq(144 downto 140);
            reg_in(29) <= input_seq(149 downto 145);
            reg_in(30) <= input_seq(154 downto 150);
            reg_in(31) <= input_seq(159 downto 155);
            sel <= '0';
            en_out <= '0';

          when MUX2_SEL =>
            counter_1 <= counter_1 + 1;
            case (sel) is
              when '1' => mux2_out <= sync_mr;
              when '0' => mux2_out <= reg_in(0);
              when others => mux2_out <= (others => 'Z');
            end case;

          when SEND_TO_MASTER =>
            sel <= '1';
            to_master (9 downto 5) <= mux2_out;
            to_master (4 downto 0) <= mux31_out;

          when RECV_MR =>
            sync_mr <= from_master;

          when MUX31_SEL =>

            case (counter_1) is
              when "00001" =>
                mux31_out <= reg_in(1);
                en_out_mux <= '1';
              when "00010" =>
                mux31_out <= reg_in(2);
                en_out_mux <= '1';
              when "00011" =>
                mux31_out <= reg_in(3);
                en_out_mux <= '1';
              when "00100" =>
          mux31_out <= reg_in(4);
          en_out_mux <= '1';
        when "00101" =>
          mux31_out <= reg_in(5);
          en_out_mux <= '1';
        when "00110" =>
          mux31_out <= reg_in(6);
          en_out_mux <= '1';
        when "00111" =>
          mux31_out <= reg_in(7);
          en_out_mux <= '1';
        when "01000" =>
          mux31_out <= reg_in(8);
          en_out_mux <= '1';
        when "01001" =>
          mux31_out <= reg_in(9);
          en_out_mux <= '1';
        when "01010" =>
          mux31_out <= reg_in(10);
          en_out_mux <= '1';
        when "01011" =>
          mux31_out <= reg_in(11);
          en_out_mux <= '1';
        when "01100" =>
          mux31_out <= reg_in(12);
          en_out_mux <= '1';
        when "01101" =>
          mux31_out <= reg_in(13);
          en_out_mux <= '1';
        when "01110" =>
          mux31_out <= reg_in(14);
          en_out_mux <= '1';
        when "01111" =>
          mux31_out <= reg_in(15);
          en_out_mux <= '1';
        when "10000" =>
          mux31_out <= reg_in(16);
          en_out_mux <= '1';
        when "10001" =>
          mux31_out <= reg_in(17);
          en_out_mux <= '1';
        when "10010" =>
          mux31_out <= reg_in(18);
          en_out_mux <= '1';
        when "10011" =>
          mux31_out <= reg_in(19);
          en_out_mux <= '1';
        when "10100" =>
          mux31_out <= reg_in(20);
          en_out_mux <= '1';
        when "10101" =>
          mux31_out <= reg_in(21);
          en_out_mux <= '1';
        when "10110" =>
          mux31_out <= reg_in(22);
          en_out_mux <= '1';
        when "10111" =>
          mux31_out <= reg_in(23);
          en_out_mux <= '1';
        when "11000" =>
          mux31_out <= reg_in(24);
          en_out_mux <= '1';
        when "11001" =>
          mux31_out <= reg_in(25);
          en_out_mux <= '1';
        when "11010" =>
          mux31_out <= reg_in(26);
          en_out_mux <= '1';
        when "11011" =>
          mux31_out <= reg_in(27);
          en_out_mux <= '1';
        when "11100" =>
          mux31_out <= reg_in(28);
          en_out_mux <= '1';
        when "11101" =>
          mux31_out <= reg_in(29);
          en_out_mux <= '1';
        when "11110" =>
          mux31_out <= reg_in(30);
          en_out_mux <= '1';
        when "11111" =>
          mux31_out <= reg_in(31);
          en_out_mux <= '1';

```

```
              when others =>
                en_out_mux <= '0';
                mux31_out <= (others => 'Z');
            end case;

          when SYNC =>
            reg_out(conv_integer(counter_2)) <= mux2_out;
            if (counter_2 < "11111") then
              counter_2 <= counter_2 + 1;
            end if;

          when PUSH =>
            output_seq(4 downto 0) <= reg_out(0);
            output_seq(9 downto 5) <= reg_out(1);
            output_seq(14 downto 10) <= reg_out(2);
            output_seq(19 downto 15) <= reg_out(3);
            output_seq(24 downto 20) <= reg_out(4);
            output_seq(29 downto 25) <= reg_out(5);
            output_seq(34 downto 30) <= reg_out(6);
            output_seq(39 downto 35) <= reg_out(7);
            output_seq(44 downto 40) <= reg_out(8);
            output_seq(49 downto 45) <= reg_out(9);
            output_seq(54 downto 50) <= reg_out(10);
            output_seq(59 downto 55) <= reg_out(11);
            output_seq(64 downto 60) <= reg_out(12);
            output_seq(69 downto 65) <= reg_out(13);
            output_seq(74 downto 70) <= reg_out(14);
            output_seq(79 downto 75) <= reg_out(15);
            output_seq(84 downto 80) <= reg_out(16);
            output_seq(89 downto 85) <= reg_out(17);
            output_seq(94 downto 90) <= reg_out(18);
            output_seq(99 downto 95) <= reg_out(19);
            output_seq(104 downto 100) <= reg_out(20);
            output_seq(109 downto 105) <= reg_out(21);
            output_seq(114 downto 110) <= reg_out(22);
            output_seq(119 downto 115) <= reg_out(23);
            output_seq(124 downto 120) <= reg_out(24);
            output_seq(129 downto 125) <= reg_out(25);
            output_seq(134 downto 130) <= reg_out(26);
            output_seq(139 downto 135) <= reg_out(27);
            output_seq(144 downto 140) <= reg_out(28);
            output_seq(149 downto 145) <= reg_out(29);
            output_seq(154 downto 150) <= reg_out(30);
            output_seq(159 downto 155) <= reg_out(31);
            counter_2 <= "00000";
            en_out <= '1';
        end case;
      end if;
    end if;

  end process;

  next_state_dec: process(state,counter_1,counter_2)
  begin
    next_state <= state;
    case (state) is
      when IDLE =>
        case (counter_1) is
          when "00000" =>
            next_state <= MUX2_SEL;

          when others =>
            next_state <= MUX31_SEL;
          end case;
      when MUX2_SEL =>
        next_state <= SYNC;
      when SEND_TO_MASTER =>
        next_state <= RECV_MR;
      when RECV_MR =>
        next_state <= MUX2_SEL;
      when MUX31_SEL =>
        next_state <= SEND_TO_MASTER;

      when SYNC =>
        if (counter_2 = "11111") then
          next_state <= PUSH;
        else
          next_state <= MUX31_SEL;
        end if;

      when PUSH =>
        next_state <= IDLE;
    end case;
  end process;
```

```


end Behavioral;
```

Listing A.8: Master ROM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity master_rom is
  port (
    clk, reset,en_in   : in std_logic;
    z            : in std_logic_vector (9 downto 0); -- address bus
    counter        : in std_logic_vector (4 downto 0);
    db             : out std_logic_vector (4 downto 0) -- data bus
  );
end master_rom;

architecture rtl of master_rom is
  signal cnt    : std_logic_vector(4 downto 0);
  signal ctrl   : std_logic_vector (2 downto 0);
begin
  sel_rom: process (clk,ctrl)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        db <= (others => '0');
      elsif (en_in = '1') then
        case (ctrl) is

          when "000" =>
db(4) <= ( not z(8) and  not z(7) and  not z(5) and  not z(4) and  not z(2) and  not z(1) and  not z(0))  or  (z(8) and z(7) and  not z(5)
     and  not z(4) and  not z(2) and  not z(1) and  not z(0))  or  (z(8) and  not z(7) and z(5) and  not z(4) and  not z(2) and  not z(1) and  not z(0))  or  ( ... ) and  not z(5) and z(4) and z(3) and z(2) and z(1) and z(0));

          when others =>
            db <= (others => 'Z');
        end case;
      end if;
    end if;
  end process;

  control: process(clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        ctrl <= (others => '0');
        cnt <= (others => '0');
      elsif (counter < "11111") then
        --cnt <= cnt + 1;
        ctrl(2) <= ( not counter(1) and  not counter(0))  or  ( not counter(2) and  not counter(1))  or  ( not counter(3) and  not counter(1))  or  ( not counter(4) and  not counter(1));
        ctrl(1) <= ( not counter(2));
        ctrl(0) <= (counter(4) and counter(3) and counter(2) and counter(1) and  not counter(0))  or  ( not counter(2) and counter(0))  or  ( not counter(3) and counter(0))  or  (not counter(4) and counter(0));
      else
        cnt <= (others => '0');
      end if;
    end if;
  end process;
end rtl;
```

Appendix B

Espresso minimization

Here is an example of an input file for minimization through the espresso application.

Listing B.1: Control ROM espresso minimization input file

```
# ROM_Control
.i 5
.o 3
.ilb  z(4) z(3) z(2) z(1) z(0)
.ob  Ctrl(2) Ctrl(1) Ctrl(0)
.p 31
00000                                110
00001                                111
00010                                010
00011                                011
00100                                100
00101                                101
00110                                000
00111                                001
01000                                110
01001                                111
01010                                010
01011                                011
01100                                100
01101                                101
01110                                000
01111                                001
10000                                110
10001                                111
10010                                010
10011                                011
10100                                100
10101                                101
10110                                000
10111                                001
11000                                110
11001                                111
11010                                010
11011                                011
11100                                100
11101                                000
11110                                001
.e
```

Listing B.2: Control ROM espresso minimization result file

```
# ROM_Control
Ctrl(2) = (!z(1)&!z(0)) | (!z(2)&!z(1)) | (!z(3)&!z(1)) | (!z(4)&!z(1));

Ctrl(1) = (!z(2));

Ctrl(0) = (z(4)&z(3)&z(2)&z(1)&!z(0)) | (!z(2)&z(0)) | (!z(3)&z(0)) | (
    !z(4)&z(0));
```

Appendix C

Synthesis report from Decryption

Here is the complete synthesis report of Decryption.

Listing C.1: Decryption synthesis report

```
Release 10.1 - xst K.39 (lin)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.
-->
Parameter TMPDIR set to /home/stig/Documents/ttm4900/mqq_final/xst/projnav.tmp


Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.06 secs

-->
Parameter xsthdpdir set to /home/stig/Documents/ttm4900/mqq_final/xst


Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.06 secs

-->
Reading design: decryption.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Report
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report
     9.1) Device utilization summary
     9.2) Partition Resource Summary
     9.3) TIMING REPORT


=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "decryption.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "decryption"
Output Format                      : NGC
Target Device                      : xc5vlx110t-1-ff1136

---- Source Options
Top Module Name                    : decryption
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
```

```
ROM Extraction                    : Yes
Mux Style                         : Auto
Decoder Extraction                : YES
Priority Encoder Extraction       : YES
Shift Register Extraction         : YES
Logical Shifter Extraction        : YES
XOR Collapsing                    : YES
ROM Style                         : Auto
Mux Extraction                    : YES
Resource Sharing                  : YES
Asynchronous To Synchronous       : NO
Use DSP Block                     : auto
Automatic Register Balancing      : No

---- Target Options
LUT Combining                     : off
Reduce Control Sets               : off
Add IO Buffers                    : YES
Global Maximum Fanout             : 100000
Add Generic Clock Buffer(BUFG)    : 32
Register Duplication              : YES
Slice Packing                     : YES
Optimize Instantiated Primitives  : NO
Use Clock Enable                  : Auto
Use Synchronous Set               : Auto
Use Synchronous Reset             : Auto
Pack IO Registers into IOBs       : auto
Equivalent register Removal       : YES

---- General Options
Optimization Goal                 : Speed
Optimization Effort               : 1
Power Reduction                   : NO
Library Search Order              : decryption.lso
Keep Hierarchy                    : NO
Netlist Hierarchy                 : as_optimized
RTL Output                        : Yes
Global Optimization               : AllClockNets
Read Cores                        : YES
Write Timing Constraints          : NO
Cross Clock Analysis              : NO
Hierarchy Separator               : /
Bus Delimiter                     : <>
Case Specifier                    : maintain
Slice Utilization Ratio           : 100
BRAM Utilization Ratio            : 100
DSP48 Utilization Ratio           : 100
Verilog 2001                      : YES
Auto BRAM Packing                 : NO
Slice Utilization Ratio Delta     : 5

=========================================================================


=========================================================================
*                          HDL Compilation                              *
=========================================================================
Compiling vhdl file "/home/stig/Documents/ttm4900/mqq_final/VHDL/master_rom.vhd" in Library work.
Entity <master_rom> compiled.
Entity <master_rom> (Architecture <rtl>) compiled.
Compiling vhdl file "/home/stig/Documents/ttm4900/mqq_final/VHDL/dobbertin_rom.vhd" in Library work.
Architecture rtl of Entity dobbertin_rom is up to date.
Compiling vhdl file "/home/stig/Documents/ttm4900/mqq_final/VHDL/private_matrix_t.vhd" in Library work
    .
Architecture rtl of Entity private_matrix_s is up to date.
Compiling vhdl file "/home/stig/Documents/ttm4900/mqq_final/VHDL/sequencer.vhd" in Library work.
Architecture behavioral of Entity sequencer is up to date.
Compiling vhdl file "/home/stig/Documents/ttm4900/mqq_final/VHDL/decryption.vhd" in Library work.
Architecture rtl of Entity decryption is up to date.

=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for entity <decryption> in library <work> (architecture <rtl>).

Analyzing hierarchy for entity <dobbertin_rom> in library <work> (architecture <rtl>).

Analyzing hierarchy for entity <private_matrix_s> in library <work> (architecture <rtl>).

Analyzing hierarchy for entity <sequencer> in library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <master_rom> in library <work> (architecture <rtl>).


```

```
=========================================================================
*                          HDL Analysis                                 *
=========================================================================
Analyzing Entity <decryption> in library <work> (Architecture <rtl>).
INFO:Xst:1561 - "/home/stig/Documents/ttm4900/mqq_final/VHDL/decryption.vhd" line 332: Mux is complete
      : default of case is discarded
INFO:Xst:2679 - Register <en_out> in unit <decryption> has a constant value of 0 during circuit
      operation. The register is replaced by logic.
Entity <decryption> analyzed. Unit <decryption> generated.

Analyzing Entity <dobbertin_rom> in library <work> (Architecture <rtl>).
Entity <dobbertin_rom> analyzed. Unit <dobbertin_rom> generated.

Analyzing Entity <private_matrix_s> in library <work> (Architecture <rtl>).
Entity <private_matrix_s> analyzed. Unit <private_matrix_s> generated.

Analyzing Entity <sequencer> in library <work> (Architecture <behavioral>).
INFO:Xst:1561 - "/home/stig/Documents/ttm4900/mqq_final/VHDL/sequencer.vhd" line 190: Mux is complete
      : default of case is discarded
Entity <sequencer> analyzed. Unit <sequencer> generated.

Analyzing Entity <master_rom> in library <work> (Architecture <rtl>).
INFO:Xst:1561 - "/home/stig/Documents/ttm4900/mqq_final/VHDL/master_rom.vhd" line 3099: Mux is
      complete : default of case is discarded
INFO:Xst:2679 - Register <cnt> in unit <master_rom> has a constant value of 00000 during circuit
      operation. The register is replaced by logic.
Entity <master_rom> analyzed. Unit <master_rom> generated.


=========================================================================
*                         HDL Synthesis                                 *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <dobbertin_rom>.
    Related source file is "/home/stig/Documents/ttm4900/mqq_final/VHDL/dobbertin_rom.vhd".
    Found 13-bit register for signal <db>.
    Found 1-bit register for signal <en_out>.
    Summary:
  inferred  14 D-type flip-flop(s).
Unit <dobbertin_rom> synthesized.


Synthesizing Unit <private_matrix_s>.
    Related source file is "/home/stig/Documents/ttm4900/mqq_final/VHDL/private_matrix_t.vhd".
    Found finite state machine <FSM_0> for signal <state>.
    -----------------------------------------------------------------------
    | States             | 5                                              |
    | Transitions        | 7                                              |
    | Inputs             | 2                                              |
    | Outputs            | 9                                              |
    | Clock              | clk (rising_edge)                              |
    | Clock enable       | en_in (positive)                               |
    | Reset              | reset (positive)                               |
    | Reset type         | synchronous                                    |
    | Reset State        | idle                                           |
    | Power Up State     | idle                                           |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <en_out>.
    Found 160-bit register for signal <output>.
    Found 5-bit register for signal <and_rom_in_1>.
    Found 5-bit register for signal <and_rom_in_10>.
    Found 5-bit register for signal <and_rom_in_100>.
    Found 5-bit register for signal <and_rom_in_101>.
    Found 5-bit register for signal <and_rom_in_102>.
    Found 5-bit register for signal <and_rom_in_103>.
    Found 5-bit register for signal <and_rom_in_104>.
    Found 5-bit register for signal <and_rom_in_105>.
    Found 5-bit register for signal <and_rom_in_106>.
    Found 5-bit register for signal <and_rom_in_107>.
    Found 5-bit register for signal <and_rom_in_108>.
    Found 5-bit register for signal <and_rom_in_109>.
    Found 5-bit register for signal <and_rom_in_11>.
    Found 5-bit register for signal <and_rom_in_110>.
    Found 5-bit register for signal <and_rom_in_111>.
    Found 5-bit register for signal <and_rom_in_112>.
    Found 5-bit register for signal <and_rom_in_113>.
    Found 5-bit register for signal <and_rom_in_114>.
    Found 5-bit register for signal <and_rom_in_115>.
    Found 5-bit register for signal <and_rom_in_116>.
    Found 5-bit register for signal <and_rom_in_117>.
    Found 5-bit register for signal <and_rom_in_118>.
```

```
|       Found 5-bit register for signal <and_rom_in_119>.                                   |
|       Found 5-bit register for signal <and_rom_in_12>.                                    |
|       Found 5-bit register for signal <and_rom_in_120>.                                   |
|       Found 5-bit register for signal <and_rom_in_121>.                                   |
|       Found 5-bit register for signal <and_rom_in_122>.                                   |
|       Found 5-bit register for signal <and_rom_in_123>.                                   |
|       Found 5-bit register for signal <and_rom_in_124>.                                   |
|       Found 5-bit register for signal <and_rom_in_125>.                                   |
|       Found 5-bit register for signal <and_rom_in_126>.                                   |
|       Found 5-bit register for signal <and_rom_in_127>.                                   |
|       Found 5-bit register for signal <and_rom_in_128>.                                   |
|       Found 5-bit register for signal <and_rom_in_129>.                                   |
|       Found 5-bit register for signal <and_rom_in_13>.                                    |
|       Found 5-bit register for signal <and_rom_in_130>.                                   |
|       Found 5-bit register for signal <and_rom_in_131>.                                   |
|       Found 5-bit register for signal <and_rom_in_132>.                                   |
|       Found 5-bit register for signal <and_rom_in_133>.                                   |
|       Found 5-bit register for signal <and_rom_in_134>.                                   |
|       Found 5-bit register for signal <and_rom_in_135>.                                   |
|       Found 5-bit register for signal <and_rom_in_136>.                                   |
|       Found 5-bit register for signal <and_rom_in_137>.                                   |
|       Found 5-bit register for signal <and_rom_in_138>.                                   |
|       Found 5-bit register for signal <and_rom_in_139>.                                   |
|       Found 5-bit register for signal <and_rom_in_14>.                                    |
|       Found 5-bit register for signal <and_rom_in_140>.                                   |
|       Found 5-bit register for signal <and_rom_in_141>.                                   |
|       Found 5-bit register for signal <and_rom_in_142>.                                   |
|       Found 5-bit register for signal <and_rom_in_143>.                                   |
|       Found 5-bit register for signal <and_rom_in_144>.                                   |
|       Found 5-bit register for signal <and_rom_in_145>.                                   |
|       Found 5-bit register for signal <and_rom_in_146>.                                   |
|       Found 5-bit register for signal <and_rom_in_147>.                                   |
|       Found 5-bit register for signal <and_rom_in_148>.                                   |
|       Found 5-bit register for signal <and_rom_in_149>.                                   |
|       Found 5-bit register for signal <and_rom_in_15>.                                    |
|       Found 5-bit register for signal <and_rom_in_150>.                                   |
|       Found 5-bit register for signal <and_rom_in_151>.                                   |
|       Found 5-bit register for signal <and_rom_in_152>.                                   |
|       Found 5-bit register for signal <and_rom_in_153>.                                   |
|       Found 5-bit register for signal <and_rom_in_154>.                                   |
|       Found 5-bit register for signal <and_rom_in_155>.                                   |
|       Found 5-bit register for signal <and_rom_in_156>.                                   |
|       Found 5-bit register for signal <and_rom_in_157>.                                   |
|       Found 5-bit register for signal <and_rom_in_158>.                                   |
|       Found 5-bit register for signal <and_rom_in_159>.                                   |
|       Found 5-bit register for signal <and_rom_in_16>.                                    |
|       Found 5-bit register for signal <and_rom_in_160>.                                   |
|       Found 5-bit register for signal <and_rom_in_17>.                                    |
|       Found 5-bit register for signal <and_rom_in_18>.                                    |
|       Found 5-bit register for signal <and_rom_in_19>.                                    |
|       Found 5-bit register for signal <and_rom_in_2>.                                     |
|       Found 5-bit register for signal <and_rom_in_20>.                                    |
|       Found 5-bit register for signal <and_rom_in_21>.                                    |
|       Found 5-bit register for signal <and_rom_in_22>.                                    |
|       Found 5-bit register for signal <and_rom_in_23>.                                    |
|       Found 5-bit register for signal <and_rom_in_24>.                                    |
|       Found 5-bit register for signal <and_rom_in_25>.                                    |
|       Found 5-bit register for signal <and_rom_in_26>.                                    |
|       Found 5-bit register for signal <and_rom_in_27>.                                    |
|       Found 5-bit register for signal <and_rom_in_28>.                                    |
|       Found 5-bit register for signal <and_rom_in_29>.                                    |
|       Found 5-bit register for signal <and_rom_in_3>.                                     |
|       Found 5-bit register for signal <and_rom_in_30>.                                    |
|       Found 5-bit register for signal <and_rom_in_31>.                                    |
|       Found 5-bit register for signal <and_rom_in_32>.                                    |
|       Found 5-bit register for signal <and_rom_in_33>.                                    |
|       Found 5-bit register for signal <and_rom_in_34>.                                    |
|       Found 5-bit register for signal <and_rom_in_35>.                                    |
|       Found 5-bit register for signal <and_rom_in_36>.                                    |
|       Found 5-bit register for signal <and_rom_in_37>.                                    |
|       Found 5-bit register for signal <and_rom_in_38>.                                    |
|       Found 5-bit register for signal <and_rom_in_39>.                                    |
|       Found 5-bit register for signal <and_rom_in_4>.                                     |
|       Found 5-bit register for signal <and_rom_in_40>.                                    |
|       Found 5-bit register for signal <and_rom_in_41>.                                    |
|       Found 5-bit register for signal <and_rom_in_42>.                                    |
|       Found 5-bit register for signal <and_rom_in_43>.                                    |
|       Found 5-bit register for signal <and_rom_in_44>.                                    |
|       Found 5-bit register for signal <and_rom_in_45>.                                    |
|       Found 5-bit register for signal <and_rom_in_46>.                                    |
|       Found 5-bit register for signal <and_rom_in_47>.                                    |
|       Found 5-bit register for signal <and_rom_in_48>.                                    |
|       Found 5-bit register for signal <and_rom_in_49>.                                    |
|       Found 5-bit register for signal <and_rom_in_5>.                                     |
|     Found 5-bit register for signal <and_rom_in_50>.                                          |
|     Found 5-bit register for signal <and_rom_in_51>.                                          |
|     Found 5-bit register for signal <and_rom_in_52>.                                          |
|     Found 5-bit register for signal <and_rom_in_53>.                                          |
|     Found 5-bit register for signal <and_rom_in_54>.                                          |
|     Found 5-bit register for signal <and_rom_in_55>.                                          |
|     Found 5-bit register for signal <and_rom_in_56>.                                          |
|     Found 5-bit register for signal <and_rom_in_57>.                                          |
|     Found 5-bit register for signal <and_rom_in_58>.                                          |
|     Found 5-bit register for signal <and_rom_in_59>.                                          |
|     Found 5-bit register for signal <and_rom_in_6>.                                           |
|     Found 5-bit register for signal <and_rom_in_60>.                                          |
|     Found 5-bit register for signal <and_rom_in_61>.                                          |
|     Found 5-bit register for signal <and_rom_in_62>.                                          |
|     Found 5-bit register for signal <and_rom_in_63>.                                          |
|     Found 5-bit register for signal <and_rom_in_64>.                                          |
|     Found 5-bit register for signal <and_rom_in_65>.                                          |
|     Found 5-bit register for signal <and_rom_in_66>.                                          |
|     Found 5-bit register for signal <and_rom_in_67>.                                          |
|     Found 5-bit register for signal <and_rom_in_68>.                                          |
|     Found 5-bit register for signal <and_rom_in_69>.                                          |
|     Found 5-bit register for signal <and_rom_in_7>.                                           |
|     Found 5-bit register for signal <and_rom_in_70>.                                          |
|     Found 5-bit register for signal <and_rom_in_71>.                                          |
|     Found 5-bit register for signal <and_rom_in_72>.                                          |
|     Found 5-bit register for signal <and_rom_in_73>.                                          |
|     Found 5-bit register for signal <and_rom_in_74>.                                          |
|     Found 5-bit register for signal <and_rom_in_75>.                                          |
|     Found 5-bit register for signal <and_rom_in_76>.                                          |
|     Found 5-bit register for signal <and_rom_in_77>.                                          |
|     Found 5-bit register for signal <and_rom_in_78>.                                          |
|     Found 5-bit register for signal <and_rom_in_79>.                                          |
|     Found 5-bit register for signal <and_rom_in_8>.                                           |
|     Found 5-bit register for signal <and_rom_in_80>.                                          |
|     Found 5-bit register for signal <and_rom_in_81>.                                          |
|     Found 5-bit register for signal <and_rom_in_82>.                                          |
|     Found 5-bit register for signal <and_rom_in_83>.                                          |
|     Found 5-bit register for signal <and_rom_in_84>.                                          |
|     Found 5-bit register for signal <and_rom_in_85>.                                          |
|     Found 5-bit register for signal <and_rom_in_86>.                                          |
|     Found 5-bit register for signal <and_rom_in_87>.                                          |
|     Found 5-bit register for signal <and_rom_in_88>.                                          |
|     Found 5-bit register for signal <and_rom_in_89>.                                          |
|     Found 5-bit register for signal <and_rom_in_9>.                                           |
|     Found 5-bit register for signal <and_rom_in_90>.                                          |
|     Found 5-bit register for signal <and_rom_in_91>.                                          |
|     Found 5-bit register for signal <and_rom_in_92>.                                          |
|     Found 5-bit register for signal <and_rom_in_93>.                                          |
|     Found 5-bit register for signal <and_rom_in_94>.                                          |
|     Found 5-bit register for signal <and_rom_in_95>.                                          |
|     Found 5-bit register for signal <and_rom_in_96>.                                          |
|     Found 5-bit register for signal <and_rom_in_97>.                                          |
|     Found 5-bit register for signal <and_rom_in_98>.                                          |
|     Found 5-bit register for signal <and_rom_in_99>.                                          |
|     Found 3-bit register for signal <count_xor>.                                              |
|     Found 3-bit adder for signal <count_xor$addsub0000> created at line 5237.                 |
|     Found 1-bit register for signal <enable_out>.                                             |
|     Found 160-bit register for signal <matr_xor>.                                             |
|     Found 1-bit xor2 for signal <matr_xor_0$xor0000> created at line 4916.                    |
|     Found 1-bit xor2 for signal <matr_xor_1$xor0000> created at line 4918.                    |
|     Found 1-bit xor2 for signal <matr_xor_10$xor0000> created at line 4936.                   |
|     Found 1-bit xor2 for signal <matr_xor_100$xor0000> created at line 5116.                  |
|     Found 1-bit xor2 for signal <matr_xor_101$xor0000> created at line 5118.                  |
|     Found 1-bit xor2 for signal <matr_xor_102$xor0000> created at line 5120.                  |
|     Found 1-bit xor2 for signal <matr_xor_103$xor0000> created at line 5122.                  |
|     Found 1-bit xor2 for signal <matr_xor_104$xor0000> created at line 5124.                  |
|     Found 1-bit xor2 for signal <matr_xor_105$xor0000> created at line 5126.                  |
|     Found 1-bit xor2 for signal <matr_xor_106$xor0000> created at line 5128.                  |
|     Found 1-bit xor2 for signal <matr_xor_107$xor0000> created at line 5130.                  |
|     Found 1-bit xor2 for signal <matr_xor_108$xor0000> created at line 5132.                  |
|     Found 1-bit xor2 for signal <matr_xor_109$xor0000> created at line 5134.                  |
|     Found 1-bit xor2 for signal <matr_xor_11$xor0000> created at line 4938.                   |
|     Found 1-bit xor2 for signal <matr_xor_110$xor0000> created at line 5136.                  |
|     Found 1-bit xor2 for signal <matr_xor_111$xor0000> created at line 5138.                  |
|     Found 1-bit xor2 for signal <matr_xor_112$xor0000> created at line 5140.                  |
|     Found 1-bit xor2 for signal <matr_xor_113$xor0000> created at line 5142.                  |
|     Found 1-bit xor2 for signal <matr_xor_114$xor0000> created at line 5144.                  |
|     Found 1-bit xor2 for signal <matr_xor_115$xor0000> created at line 5146.                  |
|     Found 1-bit xor2 for signal <matr_xor_116$xor0000> created at line 5148.                  |
|     Found 1-bit xor2 for signal <matr_xor_117$xor0000> created at line 5150.                  |
|     Found 1-bit xor2 for signal <matr_xor_118$xor0000> created at line 5152.                  |
|     Found 1-bit xor2 for signal <matr_xor_119$xor0000> created at line 5154.                  |
|     Found 1-bit xor2 for signal <matr_xor_12$xor0000> created at line 4940.                   |
|     Found 1-bit xor2 for signal <matr_xor_120$xor0000> created at line 5156.                  |
Found 1-bit xor2 for signal <matr_xor_121$xor0000> created at line 5158.
Found 1-bit xor2 for signal <matr_xor_122$xor0000> created at line 5160.
Found 1-bit xor2 for signal <matr_xor_123$xor0000> created at line 5162.
Found 1-bit xor2 for signal <matr_xor_124$xor0000> created at line 5164.
Found 1-bit xor2 for signal <matr_xor_125$xor0000> created at line 5166.
Found 1-bit xor2 for signal <matr_xor_126$xor0000> created at line 5168.
Found 1-bit xor2 for signal <matr_xor_127$xor0000> created at line 5170.
Found 1-bit xor2 for signal <matr_xor_128$xor0000> created at line 5172.
Found 1-bit xor2 for signal <matr_xor_129$xor0000> created at line 5174.
Found 1-bit xor2 for signal <matr_xor_13$xor0000> created at line 4942.
Found 1-bit xor2 for signal <matr_xor_130$xor0000> created at line 5176.
Found 1-bit xor2 for signal <matr_xor_131$xor0000> created at line 5178.
Found 1-bit xor2 for signal <matr_xor_132$xor0000> created at line 5180.
Found 1-bit xor2 for signal <matr_xor_133$xor0000> created at line 5182.
Found 1-bit xor2 for signal <matr_xor_134$xor0000> created at line 5184.
Found 1-bit xor2 for signal <matr_xor_135$xor0000> created at line 5186.
Found 1-bit xor2 for signal <matr_xor_136$xor0000> created at line 5188.
Found 1-bit xor2 for signal <matr_xor_137$xor0000> created at line 5190.
Found 1-bit xor2 for signal <matr_xor_138$xor0000> created at line 5192.
Found 1-bit xor2 for signal <matr_xor_139$xor0000> created at line 5194.
Found 1-bit xor2 for signal <matr_xor_14$xor0000> created at line 4944.
Found 1-bit xor2 for signal <matr_xor_140$xor0000> created at line 5196.
Found 1-bit xor2 for signal <matr_xor_141$xor0000> created at line 5198.
Found 1-bit xor2 for signal <matr_xor_142$xor0000> created at line 5200.
Found 1-bit xor2 for signal <matr_xor_143$xor0000> created at line 5202.
Found 1-bit xor2 for signal <matr_xor_144$xor0000> created at line 5204.
Found 1-bit xor2 for signal <matr_xor_145$xor0000> created at line 5206.
Found 1-bit xor2 for signal <matr_xor_146$xor0000> created at line 5208.
Found 1-bit xor2 for signal <matr_xor_147$xor0000> created at line 5210.
Found 1-bit xor2 for signal <matr_xor_148$xor0000> created at line 5212.
Found 1-bit xor2 for signal <matr_xor_149$xor0000> created at line 5214.
Found 1-bit xor2 for signal <matr_xor_15$xor0000> created at line 4946.
Found 1-bit xor2 for signal <matr_xor_150$xor0000> created at line 5216.
Found 1-bit xor2 for signal <matr_xor_151$xor0000> created at line 5218.
Found 1-bit xor2 for signal <matr_xor_152$xor0000> created at line 5220.
Found 1-bit xor2 for signal <matr_xor_153$xor0000> created at line 5222.
Found 1-bit xor2 for signal <matr_xor_154$xor0000> created at line 5224.
Found 1-bit xor2 for signal <matr_xor_155$xor0000> created at line 5226.
Found 1-bit xor2 for signal <matr_xor_156$xor0000> created at line 5228.
Found 1-bit xor2 for signal <matr_xor_157$xor0000> created at line 5230.
Found 1-bit xor2 for signal <matr_xor_158$xor0000> created at line 5232.
Found 1-bit xor2 for signal <matr_xor_159$xor0000> created at line 5234.
Found 1-bit xor2 for signal <matr_xor_16$xor0000> created at line 4948.
Found 1-bit xor2 for signal <matr_xor_17$xor0000> created at line 4950.
Found 1-bit xor2 for signal <matr_xor_18$xor0000> created at line 4952.
Found 1-bit xor2 for signal <matr_xor_19$xor0000> created at line 4954.
Found 1-bit xor2 for signal <matr_xor_2$xor0000> created at line 4920.
Found 1-bit xor2 for signal <matr_xor_20$xor0000> created at line 4956.
Found 1-bit xor2 for signal <matr_xor_21$xor0000> created at line 4958.
Found 1-bit xor2 for signal <matr_xor_22$xor0000> created at line 4960.
Found 1-bit xor2 for signal <matr_xor_23$xor0000> created at line 4962.
Found 1-bit xor2 for signal <matr_xor_24$xor0000> created at line 4964.
Found 1-bit xor2 for signal <matr_xor_25$xor0000> created at line 4966.
Found 1-bit xor2 for signal <matr_xor_26$xor0000> created at line 4968.
Found 1-bit xor2 for signal <matr_xor_27$xor0000> created at line 4970.
Found 1-bit xor2 for signal <matr_xor_28$xor0000> created at line 4972.
Found 1-bit xor2 for signal <matr_xor_29$xor0000> created at line 4974.
Found 1-bit xor2 for signal <matr_xor_3$xor0000> created at line 4922.
Found 1-bit xor2 for signal <matr_xor_30$xor0000> created at line 4976.
Found 1-bit xor2 for signal <matr_xor_31$xor0000> created at line 4978.
Found 1-bit xor2 for signal <matr_xor_32$xor0000> created at line 4980.
Found 1-bit xor2 for signal <matr_xor_33$xor0000> created at line 4982.
Found 1-bit xor2 for signal <matr_xor_34$xor0000> created at line 4984.
Found 1-bit xor2 for signal <matr_xor_35$xor0000> created at line 4986.
Found 1-bit xor2 for signal <matr_xor_36$xor0000> created at line 4988.
Found 1-bit xor2 for signal <matr_xor_37$xor0000> created at line 4990.
Found 1-bit xor2 for signal <matr_xor_38$xor0000> created at line 4992.
Found 1-bit xor2 for signal <matr_xor_39$xor0000> created at line 4994.
Found 1-bit xor2 for signal <matr_xor_4$xor0000> created at line 4924.
Found 1-bit xor2 for signal <matr_xor_40$xor0000> created at line 4996.
Found 1-bit xor2 for signal <matr_xor_41$xor0000> created at line 4998.
Found 1-bit xor2 for signal <matr_xor_42$xor0000> created at line 5000.
Found 1-bit xor2 for signal <matr_xor_43$xor0000> created at line 5002.
Found 1-bit xor2 for signal <matr_xor_44$xor0000> created at line 5004.
Found 1-bit xor2 for signal <matr_xor_45$xor0000> created at line 5006.
Found 1-bit xor2 for signal <matr_xor_46$xor0000> created at line 5008.
Found 1-bit xor2 for signal <matr_xor_47$xor0000> created at line 5010.
Found 1-bit xor2 for signal <matr_xor_48$xor0000> created at line 5012.
Found 1-bit xor2 for signal <matr_xor_49$xor0000> created at line 5014.
Found 1-bit xor2 for signal <matr_xor_5$xor0000> created at line 4926.
Found 1-bit xor2 for signal <matr_xor_50$xor0000> created at line 5016.
Found 1-bit xor2 for signal <matr_xor_51$xor0000> created at line 5018.
Found 1-bit xor2 for signal <matr_xor_52$xor0000> created at line 5020.
Found 1-bit xor2 for signal <matr_xor_53$xor0000> created at line 5022.
Found 1-bit xor2 for signal <matr_xor_54$xor0000> created at line 5024.
Found 1-bit xor2 for signal <matr_xor_55$xor0000> created at line 5026.
Found 1-bit xor2 for signal <matr_xor_56$xor0000> created at line 5028.
Found 1-bit xor2 for signal <matr_xor_57$xor0000> created at line 5030.
Found 1-bit xor2 for signal <matr_xor_58$xor0000> created at line 5032.
Found 1-bit xor2 for signal <matr_xor_59$xor0000> created at line 5034.
Found 1-bit xor2 for signal <matr_xor_6$xor0000> created at line 4928.
Found 1-bit xor2 for signal <matr_xor_60$xor0000> created at line 5036.
Found 1-bit xor2 for signal <matr_xor_61$xor0000> created at line 5038.
Found 1-bit xor2 for signal <matr_xor_62$xor0000> created at line 5040.
Found 1-bit xor2 for signal <matr_xor_63$xor0000> created at line 5042.
Found 1-bit xor2 for signal <matr_xor_64$xor0000> created at line 5044.
Found 1-bit xor2 for signal <matr_xor_65$xor0000> created at line 5046.
Found 1-bit xor2 for signal <matr_xor_66$xor0000> created at line 5048.
Found 1-bit xor2 for signal <matr_xor_67$xor0000> created at line 5050.
Found 1-bit xor2 for signal <matr_xor_68$xor0000> created at line 5052.
Found 1-bit xor2 for signal <matr_xor_69$xor0000> created at line 5054.
Found 1-bit xor2 for signal <matr_xor_7$xor0000> created at line 4930.
Found 1-bit xor2 for signal <matr_xor_70$xor0000> created at line 5056.
Found 1-bit xor2 for signal <matr_xor_71$xor0000> created at line 5058.
Found 1-bit xor2 for signal <matr_xor_72$xor0000> created at line 5060.
Found 1-bit xor2 for signal <matr_xor_73$xor0000> created at line 5062.
Found 1-bit xor2 for signal <matr_xor_74$xor0000> created at line 5064.
Found 1-bit xor2 for signal <matr_xor_75$xor0000> created at line 5066.
Found 1-bit xor2 for signal <matr_xor_76$xor0000> created at line 5068.
Found 1-bit xor2 for signal <matr_xor_77$xor0000> created at line 5070.
Found 1-bit xor2 for signal <matr_xor_78$xor0000> created at line 5072.
Found 1-bit xor2 for signal <matr_xor_79$xor0000> created at line 5074.
Found 1-bit xor2 for signal <matr_xor_8$xor0000> created at line 4932.
Found 1-bit xor2 for signal <matr_xor_80$xor0000> created at line 5076.
Found 1-bit xor2 for signal <matr_xor_81$xor0000> created at line 5078.
Found 1-bit xor2 for signal <matr_xor_82$xor0000> created at line 5080.
Found 1-bit xor2 for signal <matr_xor_83$xor0000> created at line 5082.
Found 1-bit xor2 for signal <matr_xor_84$xor0000> created at line 5084.
Found 1-bit xor2 for signal <matr_xor_85$xor0000> created at line 5086.
Found 1-bit xor2 for signal <matr_xor_86$xor0000> created at line 5088.
Found 1-bit xor2 for signal <matr_xor_87$xor0000> created at line 5090.
Found 1-bit xor2 for signal <matr_xor_88$xor0000> created at line 5092.
Found 1-bit xor2 for signal <matr_xor_89$xor0000> created at line 5094.
Found 1-bit xor2 for signal <matr_xor_9$xor0000> created at line 4934.
Found 1-bit xor2 for signal <matr_xor_90$xor0000> created at line 5096.
Found 1-bit xor2 for signal <matr_xor_91$xor0000> created at line 5098.
Found 1-bit xor2 for signal <matr_xor_92$xor0000> created at line 5100.
Found 1-bit xor2 for signal <matr_xor_93$xor0000> created at line 5102.
Found 1-bit xor2 for signal <matr_xor_94$xor0000> created at line 5104.
Found 1-bit xor2 for signal <matr_xor_95$xor0000> created at line 5106.
Found 1-bit xor2 for signal <matr_xor_96$xor0000> created at line 5108.
Found 1-bit xor2 for signal <matr_xor_97$xor0000> created at line 5110.
Found 1-bit xor2 for signal <matr_xor_98$xor0000> created at line 5112.
Found 1-bit xor2 for signal <matr_xor_99$xor0000> created at line 5114.
Found 160-bit register for signal <register_x>.
Found 5-bit register for signal <rom_001>.
Found 5-bit register for signal <rom_002>.
Found 5-bit register for signal <rom_003>.
Found 5-bit register for signal <rom_004>.
Found 5-bit register for signal <rom_005>.
Found 5-bit register for signal <rom_006>.
Found 5-bit register for signal <rom_007>.
Found 5-bit register for signal <rom_008>.
Found 5-bit register for signal <rom_009>.
Found 5-bit register for signal <rom_010>.
Found 5-bit register for signal <rom_011>.
Found 5-bit register for signal <rom_012>.
Found 5-bit register for signal <rom_013>.
Found 5-bit register for signal <rom_014>.
Found 5-bit register for signal <rom_015>.
Found 5-bit register for signal <rom_016>.
Found 5-bit register for signal <rom_017>.
Found 5-bit register for signal <rom_018>.
Found 5-bit register for signal <rom_019>.
Found 5-bit register for signal <rom_020>.
Found 5-bit register for signal <rom_021>.
Found 5-bit register for signal <rom_022>.
Found 5-bit register for signal <rom_023>.
Found 5-bit register for signal <rom_024>.
Found 5-bit register for signal <rom_025>.
Found 5-bit register for signal <rom_026>.
Found 5-bit register for signal <rom_027>.
Found 5-bit register for signal <rom_028>.
Found 5-bit register for signal <rom_029>.
Found 5-bit register for signal <rom_030>.
Found 5-bit register for signal <rom_031>.
Found 5-bit register for signal <rom_032>.
Found 5-bit register for signal <rom_033>.
```

```
|     Found 5-bit register for signal <rom_034>.                                    |
|     Found 5-bit register for signal <rom_035>.                                    |
|     Found 5-bit register for signal <rom_036>.                                    |
|     Found 5-bit register for signal <rom_037>.                                    |
|     Found 5-bit register for signal <rom_038>.                                    |
|     Found 5-bit register for signal <rom_039>.                                    |
|     Found 5-bit register for signal <rom_040>.                                    |
|     Found 5-bit register for signal <rom_041>.                                    |
|     Found 5-bit register for signal <rom_042>.                                    |
|     Found 5-bit register for signal <rom_043>.                                    |
|     Found 5-bit register for signal <rom_044>.                                    |
|     Found 5-bit register for signal <rom_045>.                                    |
|     Found 5-bit register for signal <rom_046>.                                    |
|     Found 5-bit register for signal <rom_047>.                                    |
|     Found 5-bit register for signal <rom_048>.                                    |
|     Found 5-bit register for signal <rom_049>.                                    |
|     Found 5-bit register for signal <rom_050>.                                    |
|     Found 5-bit register for signal <rom_051>.                                    |
|     Found 5-bit register for signal <rom_052>.                                    |
|     Found 5-bit register for signal <rom_053>.                                    |
|     Found 5-bit register for signal <rom_054>.                                    |
|     Found 5-bit register for signal <rom_055>.                                    |
|     Found 5-bit register for signal <rom_056>.                                    |
|     Found 5-bit register for signal <rom_057>.                                    |
|     Found 5-bit register for signal <rom_058>.                                    |
|     Found 5-bit register for signal <rom_059>.                                    |
|     Found 5-bit register for signal <rom_060>.                                    |
|     Found 5-bit register for signal <rom_061>.                                    |
|     Found 5-bit register for signal <rom_062>.                                    |
|     Found 5-bit register for signal <rom_063>.                                    |
|     Found 5-bit register for signal <rom_064>.                                    |
|     Found 5-bit register for signal <rom_065>.                                    |
|     Found 5-bit register for signal <rom_066>.                                    |
|     Found 5-bit register for signal <rom_067>.                                    |
|     Found 5-bit register for signal <rom_068>.                                    |
|     Found 5-bit register for signal <rom_069>.                                    |
|     Found 5-bit register for signal <rom_070>.                                    |
|     Found 5-bit register for signal <rom_071>.                                    |
|     Found 5-bit register for signal <rom_072>.                                    |
|     Found 5-bit register for signal <rom_073>.                                    |
|     Found 5-bit register for signal <rom_074>.                                    |
|     Found 5-bit register for signal <rom_075>.                                    |
|     Found 5-bit register for signal <rom_076>.                                    |
|     Found 5-bit register for signal <rom_077>.                                    |
|     Found 5-bit register for signal <rom_078>.                                    |
|     Found 5-bit register for signal <rom_079>.                                    |
|     Found 5-bit register for signal <rom_080>.                                    |
|     Found 5-bit register for signal <rom_081>.                                    |
|     Found 5-bit register for signal <rom_082>.                                    |
|     Found 5-bit register for signal <rom_083>.                                    |
|     Found 5-bit register for signal <rom_084>.                                    |
|     Found 5-bit register for signal <rom_085>.                                    |
|     Found 5-bit register for signal <rom_086>.                                    |
|     Found 5-bit register for signal <rom_087>.                                    |
|     Found 5-bit register for signal <rom_088>.                                    |
|     Found 5-bit register for signal <rom_089>.                                    |
|     Found 5-bit register for signal <rom_090>.                                    |
|     Found 5-bit register for signal <rom_091>.                                    |
|     Found 5-bit register for signal <rom_092>.                                    |
|     Found 5-bit register for signal <rom_093>.                                    |
|     Found 5-bit register for signal <rom_094>.                                    |
|     Found 5-bit register for signal <rom_095>.                                    |
|     Found 5-bit register for signal <rom_096>.                                    |
|     Found 5-bit register for signal <rom_097>.                                    |
|     Found 5-bit register for signal <rom_098>.                                    |
|     Found 5-bit register for signal <rom_099>.                                    |
|     Found 5-bit register for signal <rom_100>.                                    |
|     Found 5-bit register for signal <rom_101>.                                    |
|     Found 5-bit register for signal <rom_102>.                                    |
|     Found 5-bit register for signal <rom_103>.                                    |
|     Found 5-bit register for signal <rom_104>.                                    |
|     Found 5-bit register for signal <rom_105>.                                    |
|     Found 5-bit register for signal <rom_106>.                                    |
|     Found 5-bit register for signal <rom_107>.                                    |
|     Found 5-bit register for signal <rom_108>.                                    |
|     Found 5-bit register for signal <rom_109>.                                    |
|     Found 5-bit register for signal <rom_110>.                                    |
|     Found 5-bit register for signal <rom_111>.                                    |
|     Found 5-bit register for signal <rom_112>.                                    |
|     Found 5-bit register for signal <rom_113>.                                    |
|     Found 5-bit register for signal <rom_114>.                                    |
|     Found 5-bit register for signal <rom_115>.                                    |
|     Found 5-bit register for signal <rom_116>.                                    |
|     Found 5-bit register for signal <rom_117>.                                    |
    Found 5-bit register for signal <rom_118>.
    Found 5-bit register for signal <rom_119>.
    Found 5-bit register for signal <rom_120>.
    Found 5-bit register for signal <rom_121>.
    Found 5-bit register for signal <rom_122>.
    Found 5-bit register for signal <rom_123>.
    Found 5-bit register for signal <rom_124>.
    Found 5-bit register for signal <rom_125>.
    Found 5-bit register for signal <rom_126>.
    Found 5-bit register for signal <rom_127>.
    Found 5-bit register for signal <rom_128>.
    Found 5-bit register for signal <rom_129>.
    Found 5-bit register for signal <rom_130>.
    Found 5-bit register for signal <rom_131>.
    Found 5-bit register for signal <rom_132>.
    Found 5-bit register for signal <rom_133>.
    Found 5-bit register for signal <rom_134>.
    Found 5-bit register for signal <rom_135>.
    Found 5-bit register for signal <rom_136>.
    Found 5-bit register for signal <rom_137>.
    Found 5-bit register for signal <rom_138>.
    Found 5-bit register for signal <rom_139>.
    Found 5-bit register for signal <rom_140>.
    Found 5-bit register for signal <rom_141>.
    Found 5-bit register for signal <rom_142>.
    Found 5-bit register for signal <rom_143>.
    Found 5-bit register for signal <rom_144>.
    Found 5-bit register for signal <rom_145>.
    Found 5-bit register for signal <rom_146>.
    Found 5-bit register for signal <rom_147>.
    Found 5-bit register for signal <rom_148>.
    Found 5-bit register for signal <rom_149>.
    Found 5-bit register for signal <rom_150>.
    Found 5-bit register for signal <rom_151>.
    Found 5-bit register for signal <rom_152>.
    Found 5-bit register for signal <rom_153>.
    Found 5-bit register for signal <rom_154>.
    Found 5-bit register for signal <rom_155>.
    Found 5-bit register for signal <rom_156>.
    Found 5-bit register for signal <rom_157>.
    Found 5-bit register for signal <rom_158>.
    Found 5-bit register for signal <rom_159>.
    Found 5-bit register for signal <rom_160>.
    Found 3-bit comparator less for signal <state$cmp_lt0000> created at line 5441.
    Found 160-bit register for signal <sync_xor>.
    Found 160-bit xor2 for signal <sync_xor$xor0000> created at line 5252.
    Found 160-bit register for signal <tmp>.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_0$mux0000> created at line 4915.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_1$mux0000> created at line 4917.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_10$mux0000> created at line 4935.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_100$mux0000> created at line 5115.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_101$mux0000> created at line 5117.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_102$mux0000> created at line 5119.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_103$mux0000> created at line 5121.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_104$mux0000> created at line 5123.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_105$mux0000> created at line 5125.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_106$mux0000> created at line 5127.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_107$mux0000> created at line 5129.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_108$mux0000> created at line 5131.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_109$mux0000> created at line 5133.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_11$mux0000> created at line 4937.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_110$mux0000> created at line 5135.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_111$mux0000> created at line 5137.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_112$mux0000> created at line 5139.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_113$mux0000> created at line 5141.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_114$mux0000> created at line 5143.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_115$mux0000> created at line 5145.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_116$mux0000> created at line 5147.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_117$mux0000> created at line 5149.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_118$mux0000> created at line 5151.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_119$mux0000> created at line 5153.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_12$mux0000> created at line 4939.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_120$mux0000> created at line 5155.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_121$mux0000> created at line 5157.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_122$mux0000> created at line 5159.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_123$mux0000> created at line 5161.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_124$mux0000> created at line 5163.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_125$mux0000> created at line 5165.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_126$mux0000> created at line 5167.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_127$mux0000> created at line 5169.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_128$mux0000> created at line 5171.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_129$mux0000> created at line 5173.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_13$mux0000> created at line 4941.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_130$mux0000> created at line 5175.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_131$mux0000> created at line 5177.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_132$mux0000> created at line 5179.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_133$mux0000> created at line 5181.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_134$mux0000> created at line 5183.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_135$mux0000> created at line 5185.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_136$mux0000> created at line 5187.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_137$mux0000> created at line 5189.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_138$mux0000> created at line 5191.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_139$mux0000> created at line 5193.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_14$mux0000> created at line 4943.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_140$mux0000> created at line 5195.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_141$mux0000> created at line 5197.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_142$mux0000> created at line 5199.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_143$mux0000> created at line 5201.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_144$mux0000> created at line 5203.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_145$mux0000> created at line 5205.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_146$mux0000> created at line 5207.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_147$mux0000> created at line 5209.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_148$mux0000> created at line 5211.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_149$mux0000> created at line 5213.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_15$mux0000> created at line 4945.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_150$mux0000> created at line 5215.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_151$mux0000> created at line 5217.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_152$mux0000> created at line 5219.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_153$mux0000> created at line 5221.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_154$mux0000> created at line 5223.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_155$mux0000> created at line 5225.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_156$mux0000> created at line 5227.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_157$mux0000> created at line 5229.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_158$mux0000> created at line 5231.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_159$mux0000> created at line 5233.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_16$mux0000> created at line 4947.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_17$mux0000> created at line 4949.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_18$mux0000> created at line 4951.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_19$mux0000> created at line 4953.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_2$mux0000> created at line 4919.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_20$mux0000> created at line 4955.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_21$mux0000> created at line 4957.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_22$mux0000> created at line 4959.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_23$mux0000> created at line 4961.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_24$mux0000> created at line 4963.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_25$mux0000> created at line 4965.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_26$mux0000> created at line 4967.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_27$mux0000> created at line 4969.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_28$mux0000> created at line 4971.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_29$mux0000> created at line 4973.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_3$mux0000> created at line 4921.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_30$mux0000> created at line 4975.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_31$mux0000> created at line 4977.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_32$mux0000> created at line 4979.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_33$mux0000> created at line 4981.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_34$mux0000> created at line 4983.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_35$mux0000> created at line 4985.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_36$mux0000> created at line 4987.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_37$mux0000> created at line 4989.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_38$mux0000> created at line 4991.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_39$mux0000> created at line 4993.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_4$mux0000> created at line 4923.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_40$mux0000> created at line 4995.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_41$mux0000> created at line 4997.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_42$mux0000> created at line 4999.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_43$mux0000> created at line 5001.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_44$mux0000> created at line 5003.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_45$mux0000> created at line 5005.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_46$mux0000> created at line 5007.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_47$mux0000> created at line 5009.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_48$mux0000> created at line 5011.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_49$mux0000> created at line 5013.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_5$mux0000> created at line 4925.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_50$mux0000> created at line 5015.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_51$mux0000> created at line 5017.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_52$mux0000> created at line 5019.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_53$mux0000> created at line 5021.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_54$mux0000> created at line 5023.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_55$mux0000> created at line 5025.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_56$mux0000> created at line 5027.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_57$mux0000> created at line 5029.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_58$mux0000> created at line 5031.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_59$mux0000> created at line 5033.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_6$mux0000> created at line 4927.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_60$mux0000> created at line 5035.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_61$mux0000> created at line 5037.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_62$mux0000> created at line 5039.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_63$mux0000> created at line 5041.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_64$mux0000> created at line 5043.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_65$mux0000> created at line 5045.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_66$mux0000> created at line 5047.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_67$mux0000> created at line 5049.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_68$mux0000> created at line 5051.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_69$mux0000> created at line 5053.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_7$mux0000> created at line 4929.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_70$mux0000> created at line 5055.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_71$mux0000> created at line 5057.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_72$mux0000> created at line 5059.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_73$mux0000> created at line 5061.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_74$mux0000> created at line 5063.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_75$mux0000> created at line 5065.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_76$mux0000> created at line 5067.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_77$mux0000> created at line 5069.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_78$mux0000> created at line 5071.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_79$mux0000> created at line 5073.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_8$mux0000> created at line 4931.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_80$mux0000> created at line 5075.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_81$mux0000> created at line 5077.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_82$mux0000> created at line 5079.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_83$mux0000> created at line 5081.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_84$mux0000> created at line 5083.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_85$mux0000> created at line 5085.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_86$mux0000> created at line 5087.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_87$mux0000> created at line 5089.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_88$mux0000> created at line 5091.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_89$mux0000> created at line 5093.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_9$mux0000> created at line 4933.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_90$mux0000> created at line 5095.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_91$mux0000> created at line 5097.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_92$mux0000> created at line 5099.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_93$mux0000> created at line 5101.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_94$mux0000> created at line 5103.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_95$mux0000> created at line 5105.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_96$mux0000> created at line 5107.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_97$mux0000> created at line 5109.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_98$mux0000> created at line 5111.
    Found 1-bit 5-to-1 multiplexer for signal <tmp_99$mux0000> created at line 5113.
    Summary:
  inferred   1 Finite State Machine(s).
  inferred 2405 D-type flip-flop(s).
  inferred   1 Adder/Subtractor(s).
  inferred   1 Comparator(s).
  inferred 160 Multiplexer(s).
Unit <private_matrix_s> synthesized.


Synthesizing Unit <master_rom>.
    Related source file is "/home/stig/Documents/ttm4900/mqq_final/VHDL/master_rom.vhd".
WARNING:Xst:646 - Signal <cnt> is assigned but never used. This unconnected signal will be trimmed
     during the optimization process.
    Found 5-bit register for signal <db>.
    Found 3-bit register for signal <ctrl>.
    Found 5-bit comparator less for signal <ctrl_0$cmp_lt0000> created at line 3239.
    Found 1-bit 8-to-1 multiplexer for signal <db_0$mux0001> created at line 49.
    Found 1-bit 8-to-1 multiplexer for signal <db_1$mux0001> created at line 49.
    Found 1-bit 8-to-1 multiplexer for signal <db_2$mux0001> created at line 49.
    Found 1-bit 8-to-1 multiplexer for signal <db_3$mux0001> created at line 49.
    Found 1-bit 8-to-1 multiplexer for signal <db_4$mux0001> created at line 49.
    Summary:
  inferred   8 D-type flip-flop(s).
  inferred   1 Comparator(s).
  inferred   5 Multiplexer(s).
Unit <master_rom> synthesized.


Synthesizing Unit <sequencer>.
    Related source file is "/home/stig/Documents/ttm4900/mqq_final/VHDL/sequencer.vhd".
    Found finite state machine <FSM_1> for signal <state>.
    -----------------------------------------------------------------------
    | States             | 7                                              |
    | Transitions        | 9                                              |
    | Inputs             | 2                                              |
    | Outputs            | 14                                             |
    | Clock              | clk (rising_edge)                              |
    | Clock enable       | en_in (positive)                               |
    | Reset              | reset (positive)                               |
    | Reset type         | synchronous                                    |
    | Reset State        | idle                                           |
    | Power Up State     | idle                                           |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 32x1-bit ROM for signal <en_out_mux$mux0000> created at line 204.
    Found 1-bit register for signal <en_out>.
    Found 160-bit register for signal <output_seq>.
    Found 5-bit up counter for signal <counter_1>.
    Found 5-bit register for signal <counter_2>.
    Found 5-bit adder for signal <counter_2$addsub0000> created at line 307.
    Found 5-bit comparator less for signal <counter_2$cmp_lt0000> created at line 306.
    Found 1-bit register for signal <en_out_mux>.
    Found 5-bit register for signal <Mtridata_mux31_out> created at line 139.
    Found 1-bit register for signal <Mtrien_mux31_out> created at line 139.
    Found 5-bit register for signal <mux2_out>.
    Found 5-bit tristate buffer for signal <mux31_out>.
    Found 160-bit register for signal <reg_in>.
    Found 160-bit register for signal <reg_out>.
    Found 1-bit register for signal <sel>.
    Found 5-bit register for signal <sync_mr>.
    Found 10-bit register for signal <to_master>.
    Summary:
  inferred   1 Finite State Machine(s).
  inferred   1 ROM(s).
  inferred   1 Counter(s).
  inferred 514 D-type flip-flop(s).
  inferred   1 Adder/Subtractor(s).
  inferred   1 Comparator(s).
  inferred   5 Tristate(s).
Unit <sequencer> synthesized.


Synthesizing Unit <decryption>.
    Related source file is "/home/stig/Documents/ttm4900/mqq_final/VHDL/decryption.vhd".
    Found finite state machine <FSM_2> for signal <state>.
    -----------------------------------------------------------------------
    | States             | 10                                             |
    | Transitions        | 15                                             |
    | Inputs             | 5                                              |
    | Outputs            | 14                                             |
    | Clock              | clk (rising_edge)                              |
    | Clock enable       | en_in (positive)                               |
    | Reset              | reset (positive)                               |
    | Reset type         | synchronous                                    |
    | Reset State        | idle                                           |
    | Power Up State     | idle                                           |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 160-bit register for signal <outputs>.
    Found 5-bit up counter for signal <count_s>.
    Found 5-bit register for signal <count_t>.
    Found 5-bit adder for signal <count_t$addsub0000> created at line 212.
    Found 160-bit register for signal <dec_output>.
    Found 13-bit register for signal <dob_vector_in>.
    Found 1-bit register for signal <en_in_dob>.
    Found 1-bit register for signal <en_in_pm_s>.
    Found 1-bit register for signal <en_in_pm_t>.
    Found 1-bit register for signal <en_in_seq>.
    Found 160-bit register for signal <seq_in>.
    Found 5-bit register for signal <shift_pms>.
    Found 5x3-bit multiplier for signal <shift_pms$mult0000> created at line 267.
    Found 5-bit register for signal <shift_pmt>.
    Found 5x3-bit multiplier for signal <shift_pmt$mult0000> created at line 210.
    Summary:
  inferred   1 Finite State Machine(s).
  inferred   1 Counter(s).
  inferred 512 D-type flip-flop(s).
  inferred   1 Adder/Subtractor(s).
  inferred   2 Multiplier(s).
Unit <decryption> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# ROMs                                                 : 1
 32x1-bit ROM                                          : 1
# Multipliers                                          : 2
 5x3-bit multiplier                                    : 2
# Adders/Subtractors                                   : 4
 3-bit adder                                           : 2
 5-bit adder                                           : 2
# Counters                                             : 2
 5-bit up counter                                      : 2
# Registers                                            : 3018
 1-bit register                                        : 2617
 160-bit register                                    : 8
 3-bit register                                      : 2
 5-bit register                                      : 391
# Comparators                                        : 4
 3-bit comparator less                               : 2
 5-bit comparator less                               : 2
# Multiplexers                                       : 325
 1-bit 5-to-1 multiplexer                            : 320
 1-bit 8-to-1 multiplexer                            : 5
# Tristates                                          : 1
 5-bit tristate buffer                               : 1
# Xors                                               : 322
 1-bit xor2                                          : 320
 160-bit xor2                                        : 2

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================

Analyzing FSM <FSM_2> for best encoding.
Optimizing FSM <state/FSM> on signal <state[1:4]> with sequential encoding.
--------------------------------
 State            | Encoding
--------------------------------
 idle             | 0000
 send_to_pm_t     | 0001
 recv_from_pm_t   | 0010
 send_to_dob      | 0011
 recv_from_dob    | 0100
 send_to_seq      | 0101
 recv_from_seq    | 0110
 send_to_pm_s     | 0111
 recv_from_pm_s   | 1000
 send             | 1001
--------------------------------
Analyzing FSM <FSM_1> for best encoding.
Optimizing FSM <SEQ/state/FSM> on signal <state[1:3]> with gray encoding.
--------------------------------
 State            | Encoding
--------------------------------
 idle             | 000
 mux2_sel         | 001
 send_to_master   | 111
 recv_mr          | 110
 mux31_sel        | 011
 sync             | 010
 push             | 101
--------------------------------
Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <PM_T/state/FSM> on signal <state[1:3]> with gray encoding.
Optimizing FSM <PM_S/state/FSM> on signal <state[1:3]> with gray encoding.
-----------------------
 State      | Encoding
-----------------------
 idle       | 000
 andop      | 001
 xoring     | 011
 sync       | 010
 push_out   | 110
-----------------------
Loading device for application Rf_Device from file '5vlx110t.nph' in environment /opt/Xilinx/10.1/ISE.

Synthesizing (advanced) Unit <decryption>.
  Found pipelined multiplier on signal <shift_pmt_mult0000>:
    - 1 pipeline level(s) found in a register on signal <count_t>.
    Pushing register(s) into the multiplier macro.
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance of the multiplier
     Mmult_shift_pms_mult0000 by adding 2 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance of the multiplier
     Mmult_shift_pmt_mult0000 by adding 2 register level(s).
INFO:Xst:2385 - HDL ADVISOR - You can improve the performance of the multiplier
     Mmult_shift_pms_mult0000 by adding 2 register level(s).
Unit <decryption> synthesized (advanced).

Synthesizing (advanced) Unit <sequencer>.
INFO:Xst - In order to maximize performance and save block RAM resources, the small ROM <
     Mrom_en_out_mux_mux0000> will be implemented on LUT. If you want to force its implementation on
     block, use option/constraint rom_style.
Unit <sequencer> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

```

```
Macro Statistics
# ROMs                                                 : 1
 32x1-bit ROM                                          : 1
# Multipliers                                          : 2
 5x3-bit multiplier                                    : 1
 5x3-bit registered multiplier                         : 1
# Adders/Subtractors                                   : 4
 3-bit adder                                           : 2
 5-bit adder                                           : 2
# Counters                                             : 2
 5-bit up counter                                      : 2
# Registers                                            : 5871
 Flip-Flops                                            : 5871
# Comparators                                          : 4
 3-bit comparator less                                 : 2
 5-bit comparator less                                 : 2
# Multiplexers                                         : 325
 1-bit 5-to-1 multiplexer                              : 320
 1-bit 8-to-1 multiplexer                              : 5
# Xors                                                 : 322
 1-bit xor2                                            : 320
 160-bit xor2                                          : 2

=========================================================================

=========================================================================
*                          Low Level Synthesis                          *
=========================================================================
INFO:Xst:2261 - The FF/Latch <count_t_3> in Unit <decryption> is equivalent to the following FF/Latch, which will be removed : <Mmult_shift_pmt_mult0000_1>
INFO:Xst:2261 - The FF/Latch <count_t_4> in Unit <decryption> is equivalent to the following FF/Latch, which will be removed : <Mmult_shift_pmt_mult0000_0>
INFO:Xst:2261 - The FF/Latch <count_t_0> in Unit <decryption> is equivalent to the following FF/Latch, which will be removed : <Mmult_shift_pmt_mult0000_4>
INFO:Xst:2261 - The FF/Latch <count_t_1> in Unit <decryption> is equivalent to the following FF/Latch, which will be removed : <Mmult_shift_pmt_mult0000_3>
INFO:Xst:2261 - The FF/Latch <count_t_2> in Unit <decryption> is equivalent to the following FF/Latch, which will be removed : <Mmult_shift_pmt_mult0000_2>
WARNING:Xst:2042 - Unit sequencer: 5 internal tristates are replaced by logic (pull-up yes): mux31_out<0>, mux31_out<1>, mux31_out<2>, mux31_out<3>, mux31_out<4>.

Optimizing unit <decryption> ...

Optimizing unit <dobbertin_rom> ...

Optimizing unit <private_matrix_s> ...
WARNING:Xst:1293 - FF/Latch <count_xor_2> has a constant value of 0 in block <private_matrix_s>. This FF/Latch will be trimmed during the optimization process.
WARNING:Xst:1293 - FF/Latch <count_xor_2> has a constant value of 0 in block <private_matrix_s>. This FF/Latch will be trimmed during the optimization process.

Optimizing unit <master_rom> ...

Optimizing unit <sequencer> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block decryption, actual ratio is 18.
FlipFlop dob_vector_in_12 has been replicated 1 time(s)
FlipFlop dob_vector_in_3 has been replicated 1 time(s)
FlipFlop dob_vector_in_4 has been replicated 10 time(s)
FlipFlop dob_vector_in_5 has been replicated 12 time(s)
FlipFlop dob_vector_in_6 has been replicated 9 time(s)
FlipFlop dob_vector_in_7 has been replicated 10 time(s)
FlipFlop dob_vector_in_8 has been replicated 9 time(s)
FlipFlop dob_vector_in_9 has been replicated 6 time(s)

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                                            : 5937
 Flip-Flops                                            : 5937

=========================================================================

=========================================================================
*                           Partition Report                            *
=========================================================================

Partition Implementation Status
-------------------------------

```

```
  No Partitions were found in this design.

---------------------------------

=========================================================================
*                          Final Report                                 *
=========================================================================
Final Results
RTL Top Level Output File Name     : decryption.ngr
Top Level Output File Name         : decryption
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 324

Cell Usage :
# BELS                             : 10493
#      GND                         : 1
#      INV                         : 3
#      LUT1                        : 4
#      LUT2                        : 1848
#      LUT3                        : 501
#      LUT4                        : 412
#      LUT5                        : 1443
#      LUT6                        : 5739
#      MUXCY                       : 10
#      MUXF7                       : 518
#      MUXF8                       : 3
#      VCC                         : 1
#      XORCY                       : 10
# FlipFlops/Latches                : 5937
#      FDE                         : 161
#      FDR                         : 1
#      FDRE                        : 5775
# Clock Buffers                    : 2
#      BUFG                        : 1
#      BUFGP                       : 1
# IO Buffers                       : 323
#      IBUF                        : 162
#      OBUF                        : 161
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vlx110tff1136-1


Slice Logic Utilization:
 Number of Slice Registers:             5937  out of  69120     8%
 Number of Slice LUTs:                  9950  out of  69120    14%
    Number used as Logic:               9950  out of  69120    14%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  11463
   Number with an unused Flip Flop:     5526  out of  11463    48%
   Number with an unused LUT:           1513  out of  11463    13%
   Number of fully used LUT-FF pairs:   4424  out of  11463    38%
   Number of unique control sets:         64

IO Utilization:
 Number of IOs:                          324
 Number of bonded IOBs:                  324  out of    640    50%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                 2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.
```

```

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 5937  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -1

   Minimum period: 4.974ns (Maximum Frequency: 201.045MHz)
   Minimum input arrival time before clock: 4.204ns
   Maximum output required time after clock: 3.259ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 4.974ns (frequency: 201.045MHz)
  Total number of paths / destination ports: 102627 / 11860
-------------------------------------------------------------------------
Delay:               4.974ns (Levels of Logic = 6)
  Source:            dob_vector_in_8_5 (FF)
  Destination:       DR/db_2 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: dob_vector_in_8_5 to DR/db_2
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            12   0.471   1.033  dob_vector_in_8_5 (dob_vector_in_8_5)
     LUT5:I0->O            2   0.094   0.581  DR/db_2_or0000891 (DR/db_2_or0000_bdd170)
     LUT5:I3->O            1   0.094   0.789  DR/db_2_or0000113148 (DR/db_2_or0000113148)
     LUT5:I1->O            1   0.094   0.480  DR/db_2_or0000113328 (DR/db_2_or0000113328)
     LUT6:I5->O            1   0.094   0.480  DR/db_2_or0000113389 (DR/db_2_or0000113389)
     LUT6:I5->O            1   0.094   0.576  DR/db_2_or0000113402 (DR/db_2_or0000113402)
     LUT6:I4->O            1   0.094   0.000  DR/db_2_or0000126880 (DR/db_2_or0000)
     FDRE:D                   -0.018          DR/db_2
    ----------------------------------------
    Total                      4.974ns (1.035ns logic, 3.939ns route)
                                       (20.8% logic, 79.2% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 1062 / 907
-------------------------------------------------------------------------
Offset:              4.204ns (Levels of Logic = 5)
  Source:            inputs<100> (PAD)
  Destination:       shift_pmt_0 (FF)
  Destination Clock: clk rising

  Data Path: inputs<100> to shift_pmt_0
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             1   0.818   0.576  inputs_100_IBUF (inputs_100_IBUF)
     LUT2:I0->O            1   0.094   1.069  shift_pmt_mux0001<0>1414_SW1 (N1367)
     LUT6:I0->O            1   0.094   0.789  shift_pmt_mux0001<0>1414 (shift_pmt_mux0001<0>1414)
     LUT6:I2->O            1   0.094   0.576  shift_pmt_mux0001<0>1620 (shift_pmt_mux0001<0>1620)
     LUT6:I4->O            1   0.094   0.000  shift_pmt_mux0001<0>11355 (shift_pmt_mux0001<0>)
     FDRE:D                   -0.018          shift_pmt_0
    ----------------------------------------
    Total                      4.204ns (1.194ns logic, 3.010ns route)
                                       (28.4% logic, 71.6% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 160 / 160
-------------------------------------------------------------------------
Offset:              3.259ns (Levels of Logic = 1)
  Source:            outputs_159 (FF)
  Destination:       outputs<159> (PAD)
  Source Clock:      clk rising
  Data Path: outputs_159 to outputs<159>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
      FDRE:C->Q            1   0.471   0.336  outputs_159 (outputs_159)
      OBUF:I->O                2.452          outputs_159_OBUF (outputs<159>)
    ----------------------------------------
    Total                      3.259ns (2.923ns logic, 0.336ns route)
                                       (89.7% logic, 10.3% route)

=========================================================================


Total REAL time to Xst completion: 19145.00 secs
Total CPU time to Xst completion: 19061.31 secs

-->


Total memory usage is 1005084 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    4 (   0 filtered)
Number of infos    :   14 (   0 filtered)
```