EE 705: VLSI Design Lab

Database Management using AES Algorithm

Submitted by

Sumeet Sudarshan Tabhane (213070072)

Shreyansh Neekhre (213079029)

Submitted to

Prof. Sachin Patkar

Electrical Engineering Department

Indian Institute of Technology Bombay

May 2022

Table of contents Pg no

ABSTRACT ---------- 3

TOOL USED ----------4

INTRODUCTION ---------- 5

WORKING OF CIPHER ----------6

DATABASE ----------13

WORK DONE TO ACHEIVE PROBLEM STATEMENT ----------13

COMPILATION REPORT OF PROJECT ----------14

RTL VIEW ---------- 15

RESULT AND DISCUSSION ----------15

STATIC TIMING ANALYSIS ----------17

LAYOUT USING Q-FLOW ----------20

REFERENCES ----------22

ABSTRACT

In today's world, lots of people which are handling accounts on various websites create their accounts, and in order to access that account, they have their user id and password. This information is unique for each user and so must be confidential for everyone. So various website uses some technique to keep them confidential so no one can access them. So the commonly used technique is encryption. Nowadays this technique is almost used for every information-related application. In our project, we created the database with usernames and passwords of 10 users whose password was present in the encrypted form. The encryption method used by us was the AES algorithm in which the cipher key was 128-bit long. All these implementation was done in VHDL and the tool used was Quartus. And further result analysis was done in ModelSim.

Tools Used

Following tools were used while implementing the project:

1. Quartus Prime Lite 18.1
2. ModelSim Altera
3. Qflow

Quartus Prime Lite 18.1

This project was designed in Quartus PrimeLite18.1, a tool for hardware design employing Hardware Description Languages such as VHSIC Hardware Description Language(VHDL), Verilog, and others. This project was totally conceived and coded in VHDL.

ModelSim Altera

The design must be simulated on software so that its functionality may be tested. As a result, ModelSim Altera was employed for that component. Register Transfer Level (RTL) Simulation and GATE Level Simulation are used to test the design.

Qflow

Qflow is a complete toolchain for synthesizing digital circuits, from Verilog source through physical layout for a specific fabrication method.

INTRODUCTION

AES Algorithm

The Advanced Encryption Standard (AES) was created by the US National Institute of Standards and Technology (NIST) in 2001 as a specification for the encryption of electronic data. Despite being more difficult to build, AES is frequently used today because it is substantially stronger than DES and triple DES.

Important points:

- AES is a block cipher.
- The key size can be 128/192/256 bits.
- Encrypt data in blocks of 128 bits each.

That is to say, it accepts 128 bits as input and outputs 128 bits of encrypted ciphertext. AES is based on the substitution-permutation network principle, which entails replacing and shuffling the input data through a series of connected processes.

Working of cipher:-

AES uses bytes rather than bits to conduct operations. The cipher processes 128 bits (or 16 bytes) of input data at a time because the block size is 128 bits.

The number of rounds depends on key length as follows:

- 128-bit key - 10 rounds.
- 192-bit key - 12 rounds.
- 256-bit key - 14 rounds.

Encryption

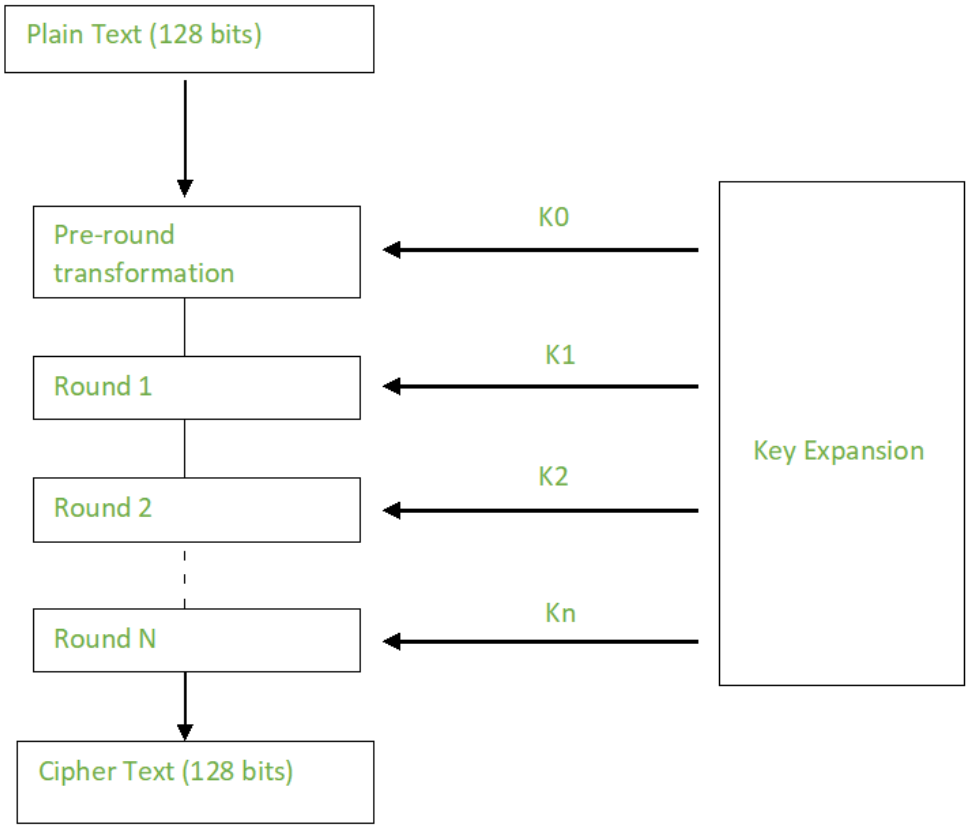


Fig1. Block diagram of AES

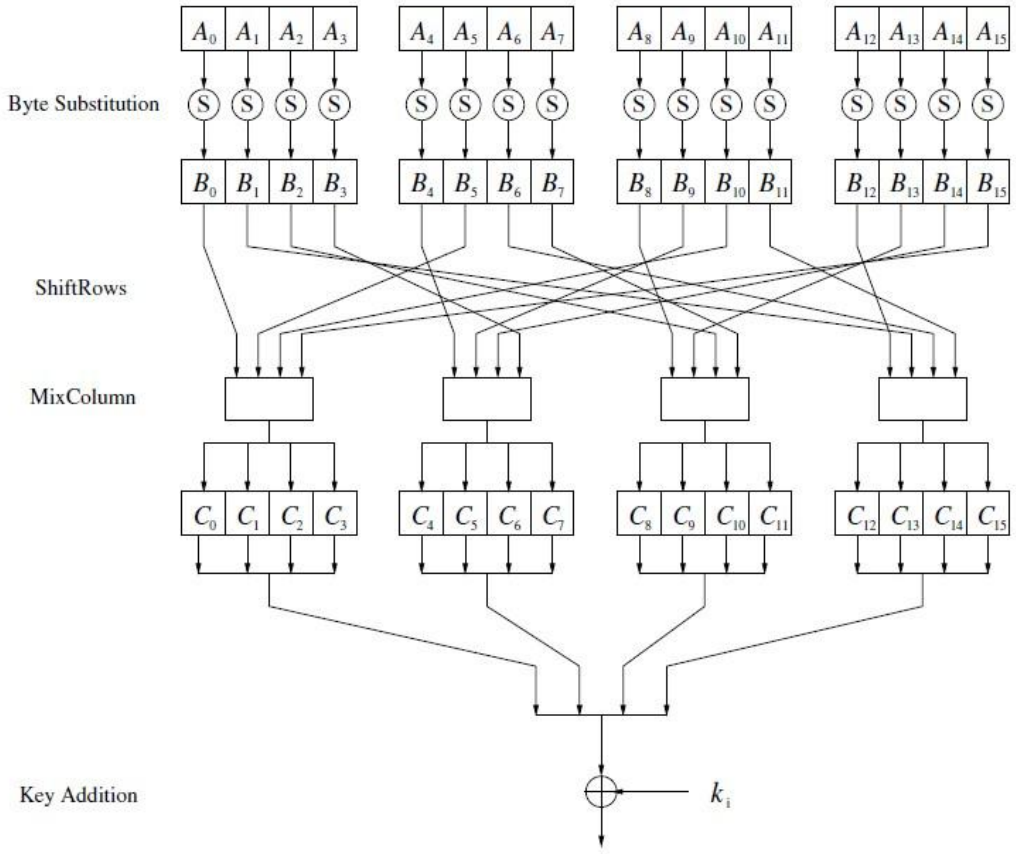


Fig2. Internal Structure Of AES algorithm

In a column-major structure, AES treats each block as a 16 byte (4-bytes x 4 bytes = 128) grid.

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Each round contains 4 steps:

- SubBytes
- ShiftRows
- MixColoumns
- Add Round Key

Last Round doesn't have the MixColoumns.

SubBytes

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Fig3. Substitution Box

The Byte Substitution layer is the first layer in each round, as depicted in Fig3. The Byte Substitution layer is made up of 16 parallel S-Boxes with 8 input and output bits apiece. Unlike DES, where eight different S-Boxes are utilized, all 16 S-Boxes are identical. Each state byte Ai in the layer is replaced or substituted, with another byte Bi: S(Ai) = Bi.

The S-Box is AES's single nonlinear element, implying that For two states A and B, ByteSub(A)+ ByteSub(B) 6= ByteSub(A+B). The S-Box substitution is a bijective mapping, meaning that each of the 28 = 256 possible input elements corresponds to one of the output elements.

One output element is mapped. This allows us to invert the S-Box in a unique way.

For decryption, is required. The S-Box is commonly used in software implementations.

Fig3. shows a 256-by-8 bit lookup table with fixed entries.

ShiftRow

ShiftRows cycles through the state matrix, shifting the second row three bytes to the right, the third row two bytes to the right, and the fourth row one byte to the right. The ShiftRows transformation has no effect on the first row. The ShiftRows transformation is used to improve the diffusion properties of AES.

Input:

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Output:

b_0	b_4	b_8	B_{12}
b_5	b_9	b_{13}	b_1
b_{10}	b_{14}	b_2	b_6
b_{15}	b_3	b_7	b_{11}

MixColoumn

Each column of the state matrix is mixed in the MixColumn step, which is a linear transformation. The MixColumn operation is the most important diffusion element in AES since every input byte influences four output bytes. The ShiftRows and MixColumn layers work together to ensure that after only three rounds, each byte of the state matrix is dependent on all 16 plaintext bytes.

In the following, we denote the 16-byte input state by B and the 16-byte output state by C:

MixColumn(B) =C,

where B is the state after the ShiftRows operation as given in Expression (4.1).

Now, each 4-byte column is considered as a vector and multiplied by a fixed 4×4 matrix. The matrix contains constant entries. Multiplication and addition of the coefficients are done in GF(2^8). As an example, we show how the first four output bytes are computed:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

KeyAddition

The current 16-byte state matrix and a subkey, both of which are 16 bytes, are the two inputs to the Key Addition layer (128 bits). A bitwise XOR operation is used to mix the two inputs. XOR operation is equal to the addition operation in the Galois field GF(2).

Key Expansion

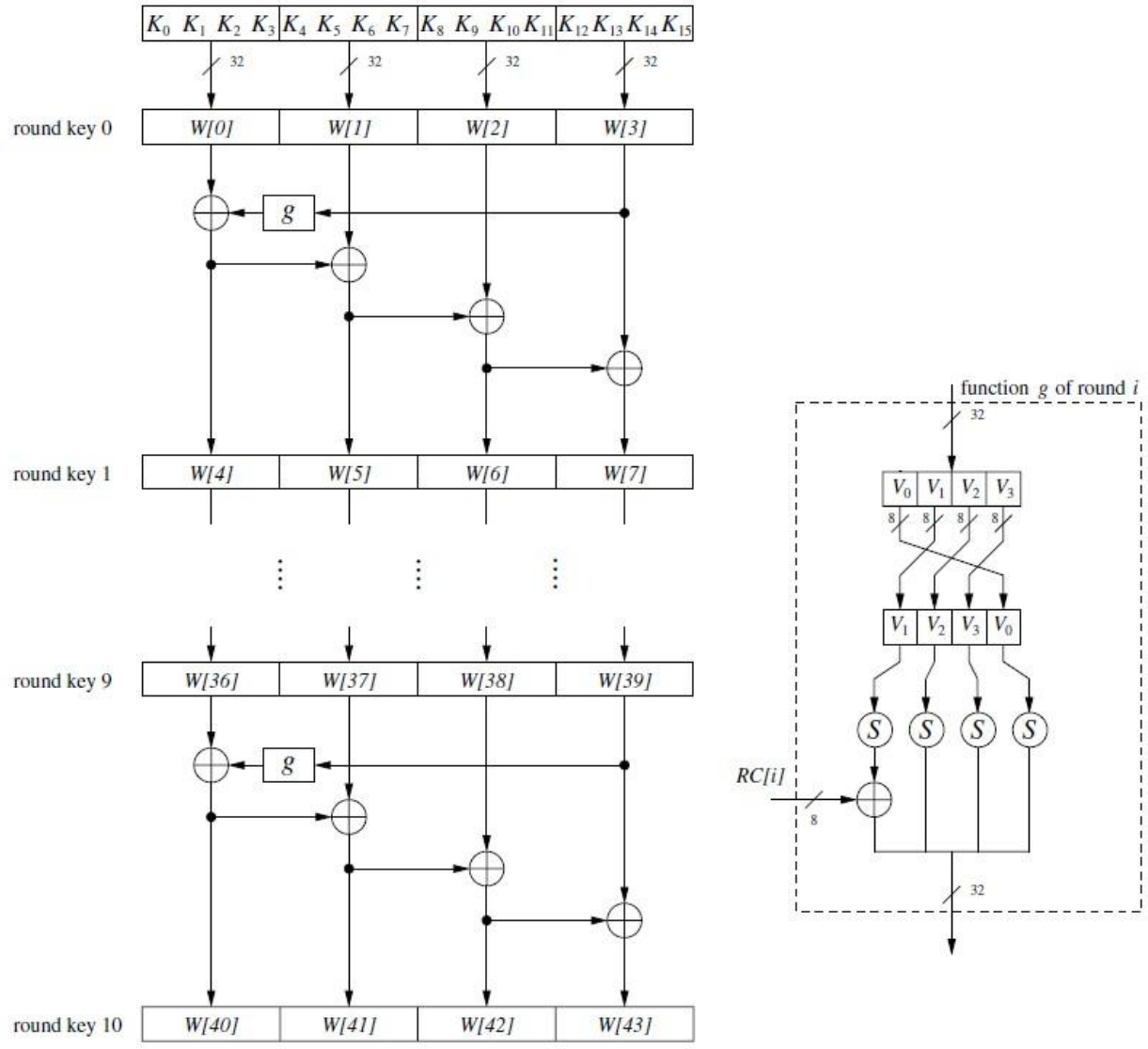


Fig4. Key Expansion

The ll subkeys are stored in a key expansion array with the elementsW[0], . . . ,W[43]. The subkeys are computed as depicted in Fig. 4.5. The elements K0, . . . ,K15 denote the bytes of the original AES key. To begin, we need remember that the first subkey k0 is the original AES key i.e., The first four elements of the key array W were copied. The remaining array components are as follows:

As can be seen in the figure, the leftmost word of a subkey W[4i], where i = 1,.....,10, is computed as

W[4i]=W[4(i-1)]+g(W[4i-1]).

Here g() is a nonlinear function with a four-byte input and output. The remaining three words of a subkey are computed recursively as:

W[4i+j] = W[4i+j-1]+W[4(i-1)+j],

where i = 1, . . . ,10 and j = 1,2, 3. The function g() rotates its four input bytes, performs a byte-wise S-Box substitution, and adds a round coefficient RC to it. The round coefficient is an element of the Galois field GF(28), i.e, an 8-bit value. It is only added to the leftmost byte in the function g(). The round coefficients vary from round to round according to the following rule:

$RC[0] = x^0 = (0000\ 0001)_2$,

$RC[1] = x^1 = (0000\ 0010)_2$,

$RC[2] = x^2 = (0000\ 0100)_2$,

$RC[3] = x^3 = (0000\ 1000)_2$,

⋮

$RC[10] = x^{10} = (00110110)_2$.

Database(hexadecimal)

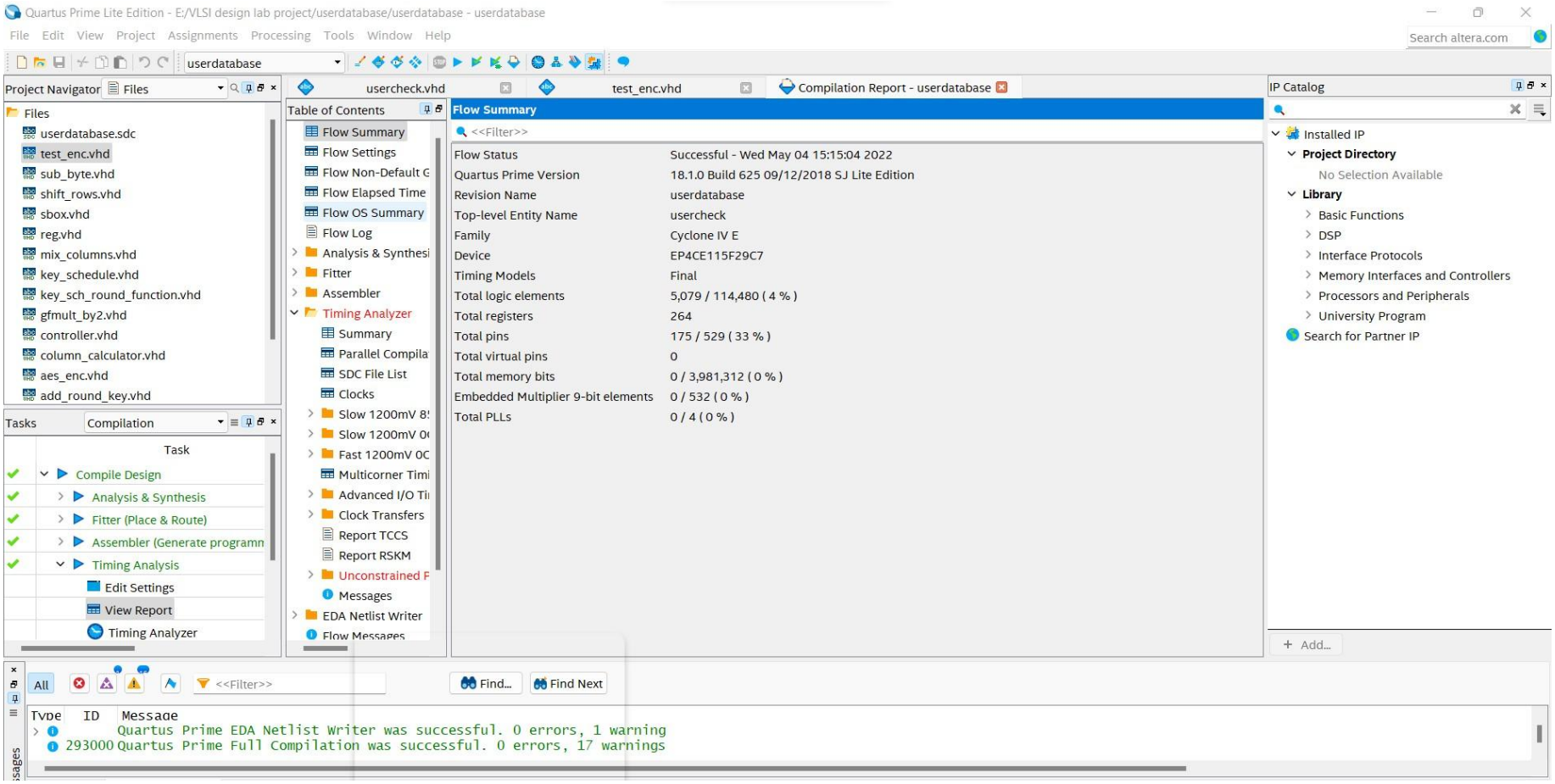
Key used for Encryption: 3c4fcf098815f7aba6d2ae2816157e2b

Userid	Password	Encrypted Password
00001	123654	c97804928533a86b302e315f7e6cc1c8
12345	456987	62f6b1afc1bff6d5914fee5da926fb5b
78459	456321	22faa4ad1f46d005fb533b418d071e98
12456	852369	abfa3ed2fb85909a1cfca4007f988f5b
96587	741258	7f348b4eadfa141e6f73be62a6e6d73e
36148	123879	8c2b799a6d88d0ac82e7ca462a3124a4
85264	231897	4bdcc17f5b92e00477b8fa727e9cbe32
96345	859784	e869b43b8d717a77def52c07bf12bd5a
87936	965874	a05770975a7debc7d18c3ac96ffdf5bc
98756	968521	f03be6a3880a061e02106f3e25be8922

Work Done by us to achieve problem statement

We implemented our problem statement in VHDL and compiled it in Quartus.

- At first we develop the AES algorithm in VHDL.
- Then we created the lookup table for our database which contain **userid** and **encrypted password**.
- Now User will enter his/her **username** and his **password** then according to our code it will find the index in lookup table where it's username is stored and then it will encrypt the entered password and compared it encrypted password at that index and will make **enter** signal high.



Compilation Report for Project

RTL View:

https://drive.google.com/file/d/1ZiXD-UqFmBX9nu68fs_Py8AMFcyjEYf3/view?usp=sharing

Results and Discussion:

Case 1: Userid: 36148 Password:123879 (correct user id and Password)

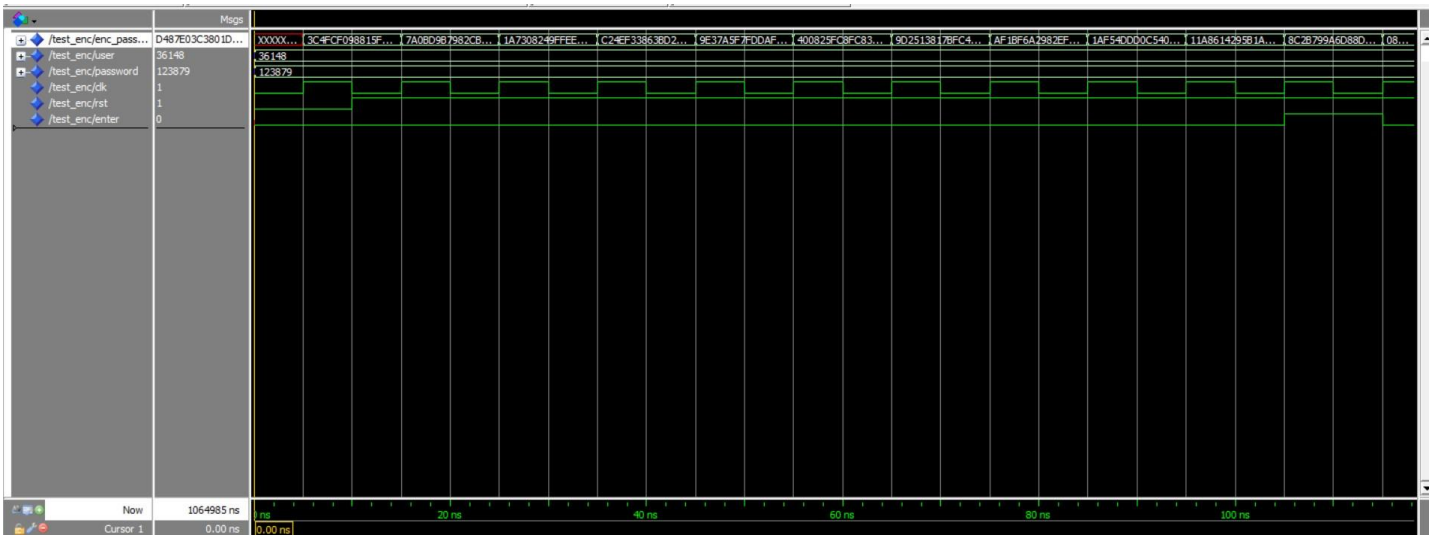


Image link:

https://drive.google.com/file/d/1-74ShCwZZC2U1ACj4GJwXFMTDfkIHnE7/view?usp=sharing

In this case, we can see that if both user id and password are correct then after 10 cycles **enter** signal become **high**. It required 10 cycles because it takes 10 cycles to encrypt the password and then match it with the encrypted password in the database.

Case 2: Userid: 36148 Password:123456 (correct user id and incorrect password)

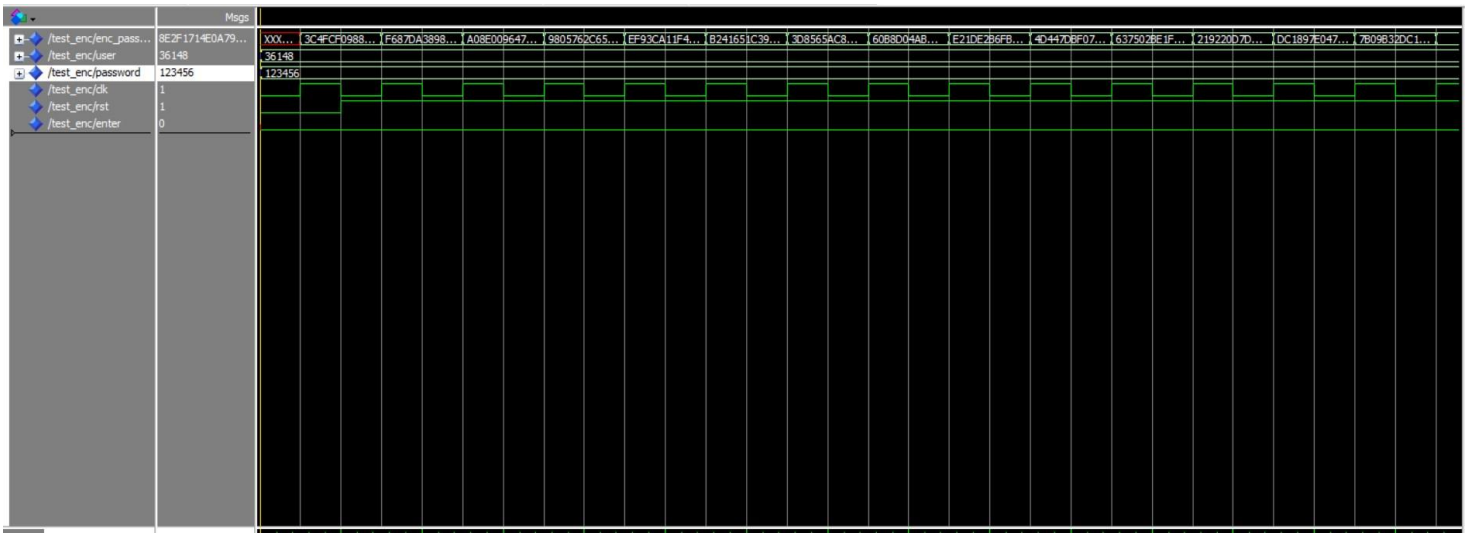
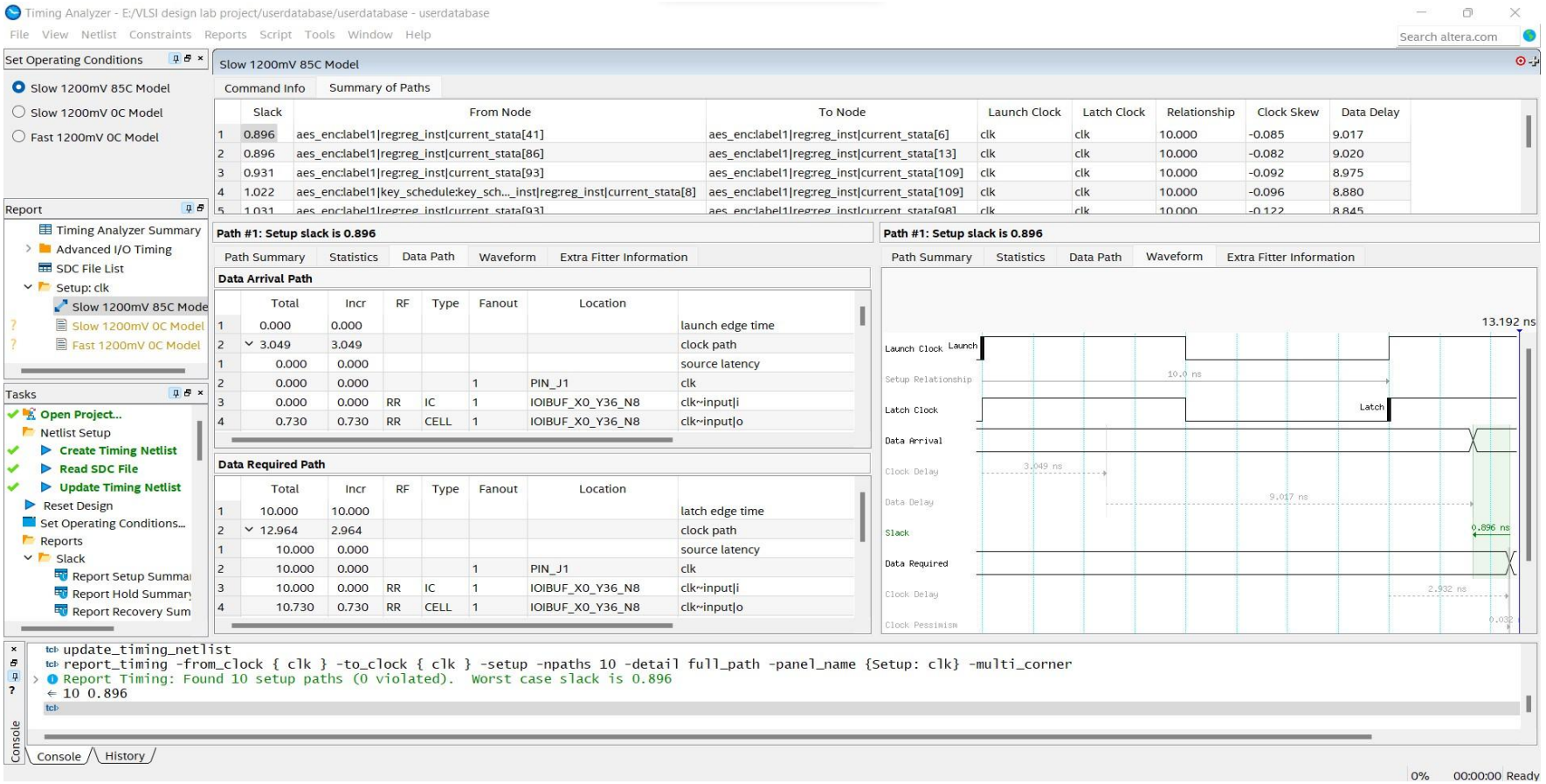


Image Link:

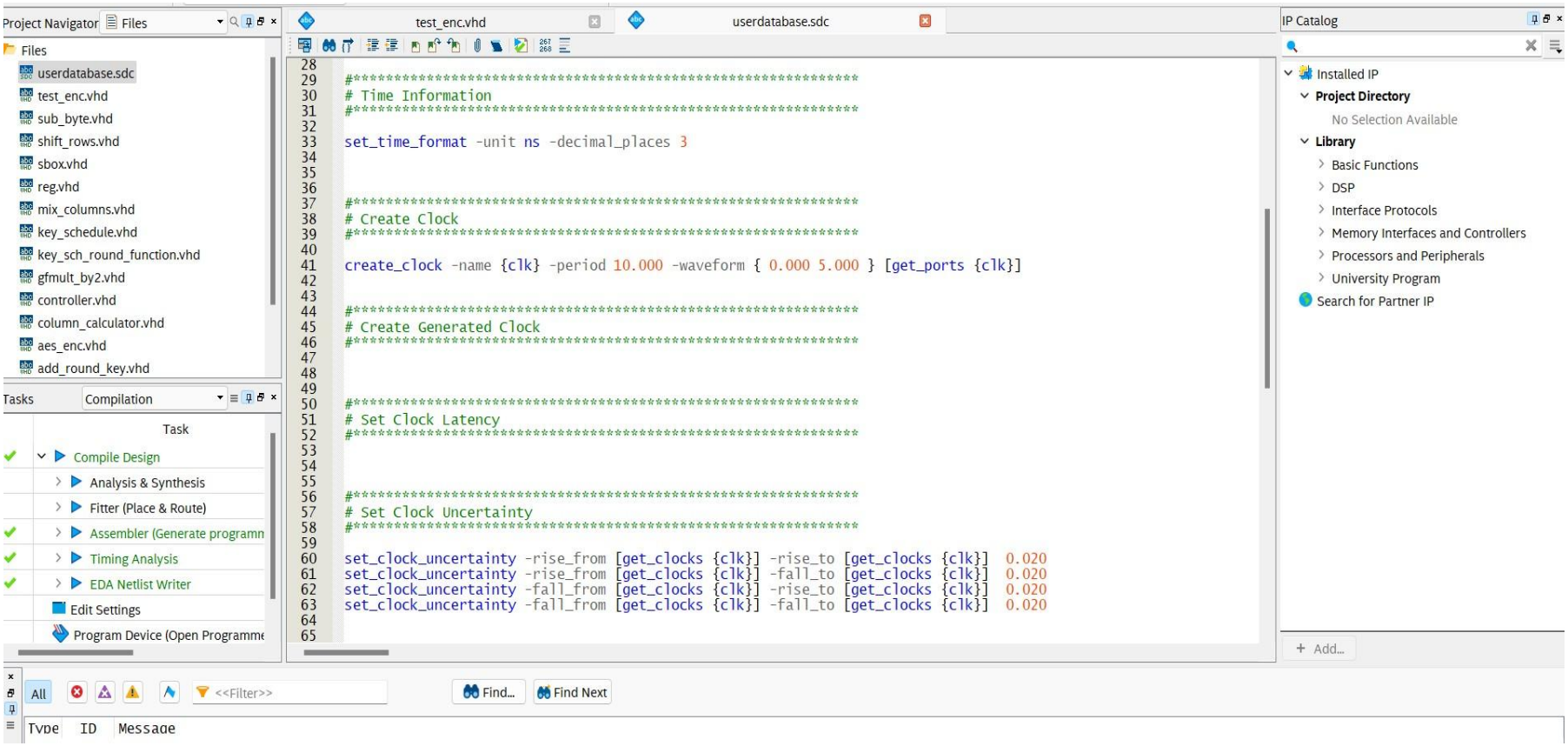
https://drive.google.com/file/d/1bxX-D6d-qTs0a-mzGAh6adPWiPCU1LOU/view?usp=sharing,

In this case, we can see that enter signal is not becoming high due to incorrect password to that in database.

Static Timing Analysis



Report of Timing Analyzer

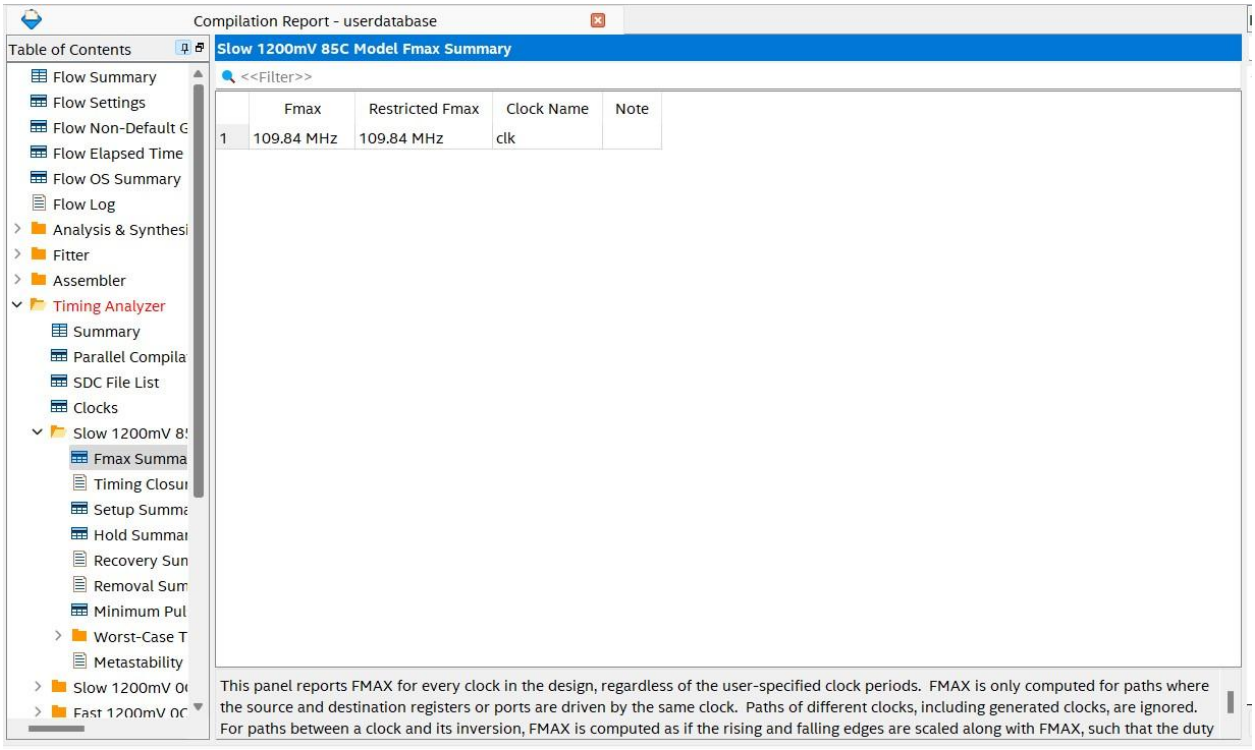


SDC file generated after removing slack

Image Link for a report of timing analyzer:

https://drive.google.com/file/d/1tyD-4XE11WPXui9CZcMu17ok-6pqhUeu/view?usp=sharing

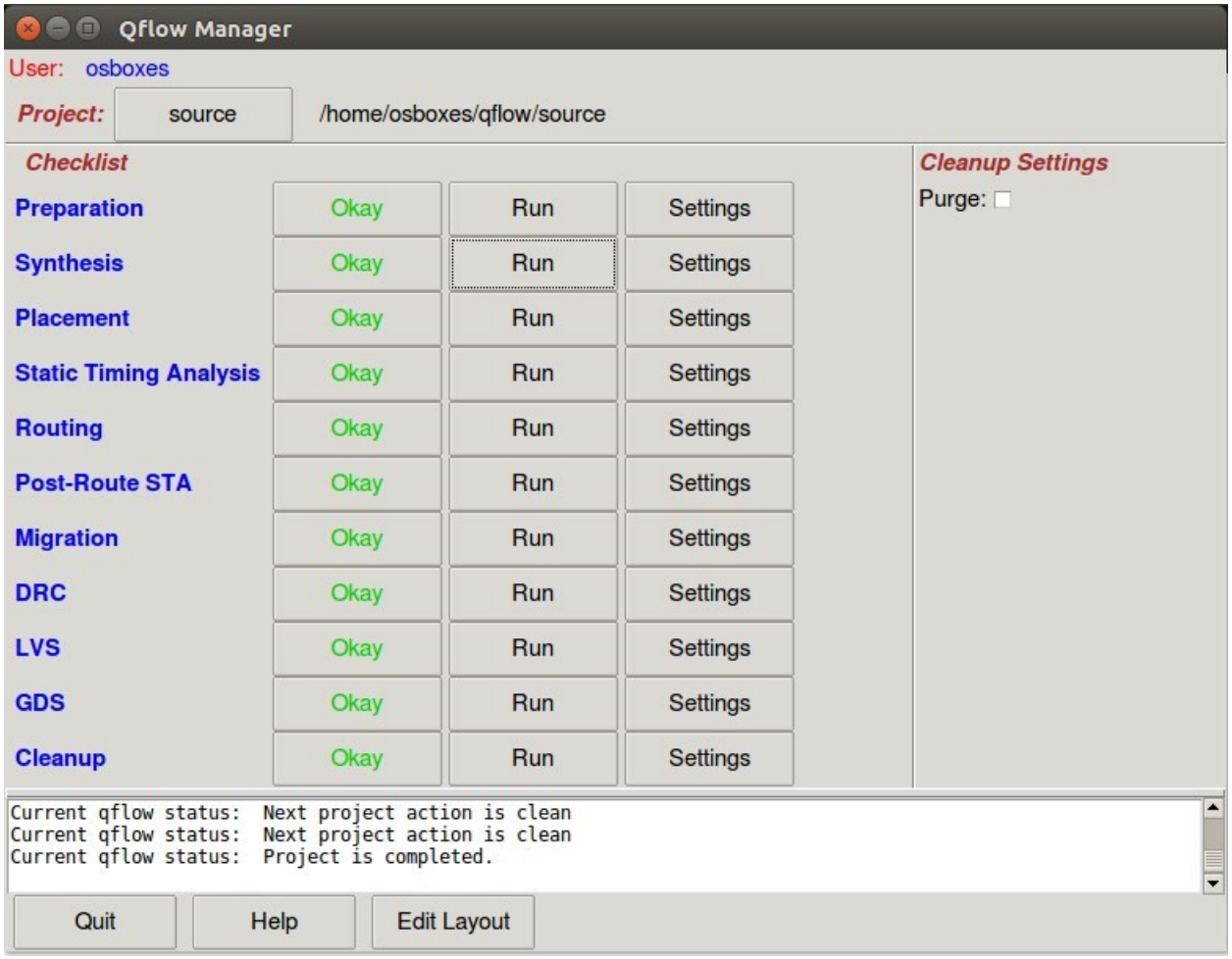
From the above image of the report of the timing analyzer we can see that all the negative slack have been removed.



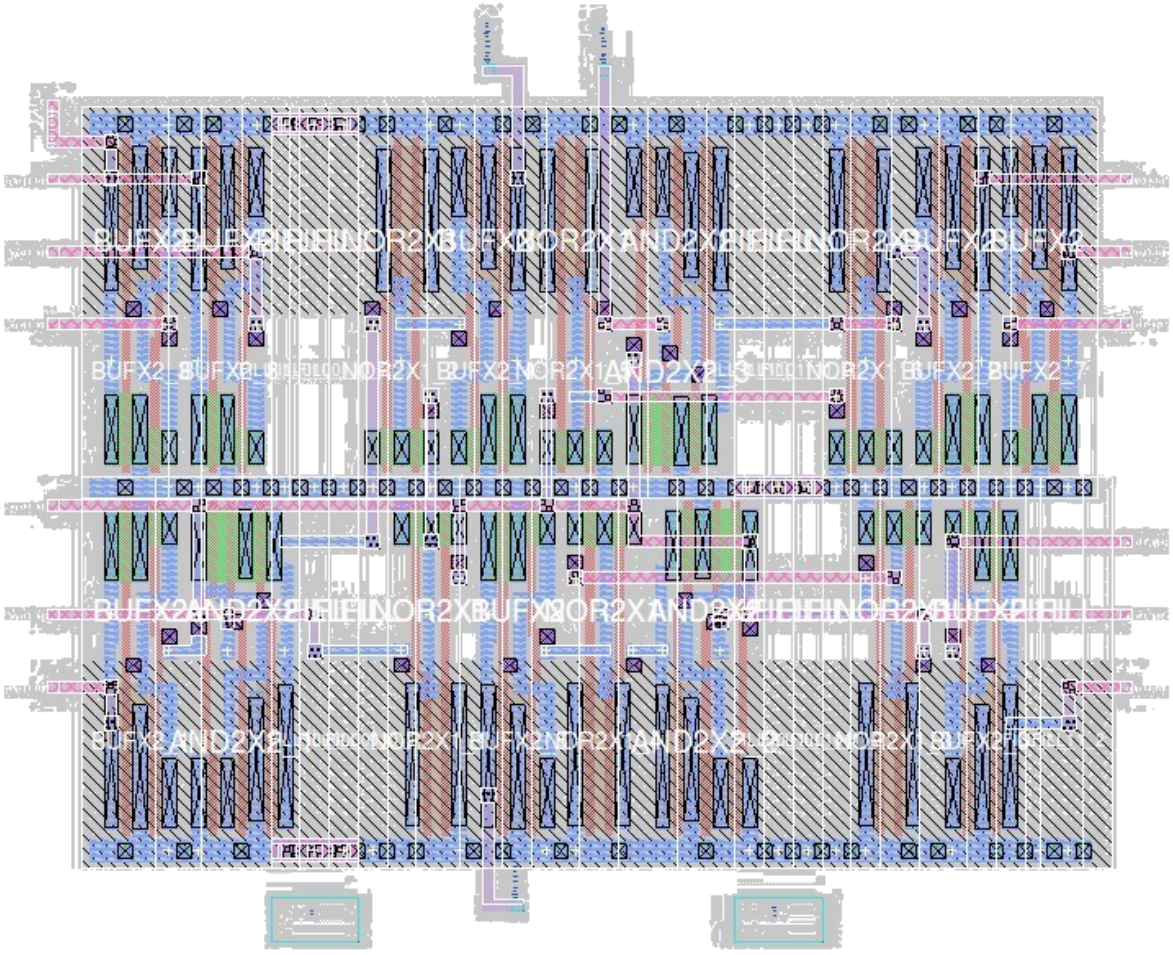
Fmax after static timing analysis has been reported as 109.84 MHz.

Layout using Q-Flow

We have created a layout of one entity named **gfmult_by2** in this we have 7-bit input and the output is two multiplied by input in Galos Field Mathematics. As our whole work was in VHDL so we have converted it into Verilog because Q-Flow accepts Verilog File.



Qflow window showing Okay status for all design flow



Layout Reported by Qflow viewed in Magic

References:

[1] Understanding Cryptography A Textbook for Students and Practitioners by Christof Paar. Jan Pelzl

[2] https://www.geeksforgeeks.org/advanced-encryption-standard-aes/