VHDL

ENSAM
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

A VHDL Implementation of the Lightweight Cryptographic Algorithm HIGHT

Done by : ABKAOUI Samia

Serrar Rayan Ayoub

Supervised by : ELMOUMNI Soufiane

INTRODUCTION

Cryptography can be described as a set of techniques to encrypt (encode) information. The main purpose of encryption is to protect the information contained in a document, as it usually becomes unreadable after this process. To retrieve the information contained in the document, a reverse process called decrypt is required. Encryption techniques may be described by algorithms, which makes it feasible implementation by computer.

by computers . Over time, several different methods have been developed in order to make it difficult to decode encrypted files. Currently there are sophisticated methods that require a password to retrieve information, such as asymmetric key algorithms, for example. Other methods, such as hash functions, do not allow the decryption of once encrypted content, serving as digital signature of files.

The growing evolution of cryptography can be explained by the development of computing. Strongly widespread in modern world, computing has become ubiquitous and is present in the daily lives of most people. A simple example of the importance of encryption in the world today is the growing use of smartphones for banking transactions. If there were no safe ways to protect the information entered on banking applications, invasions to accounts of thousands of people would be easily seen.

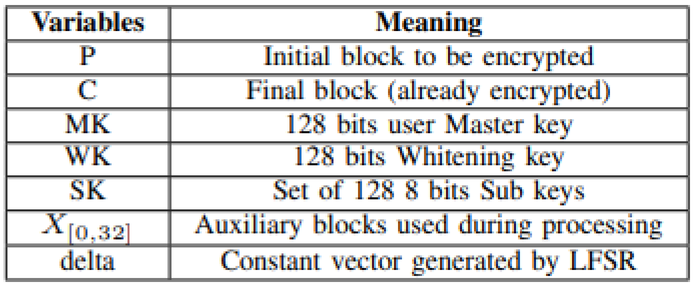
TECHNICAL BACKGROUND

Two major categories of cryptographic algorithms are the block cipher and stream cipher. The former groups the information to be encrypted into blocks from 8 to 16 bytes before the encryption process and then encrypts the whole block. It is possible to use chaining encryption to hinder attacks. The latter encrypts text one bit at time, using the logical operation xor between the bit and the key . A key is a unique string that is used to encrypt and/or decrypt a file. There are two classes of algorithms with respect to the key: symmetric and asymmetric. Symmetric algorithms use only one key to encrypt and decrypt the message, which is a private key that must be distributed among all parties involved in the communication. In the asymmetric cryptography, there are two keys, one public and another private. They function as a plug: while one is used to encrypt, the other is able to restore the information and vice-versa.

RELATED WORK (METHODOLOGY)

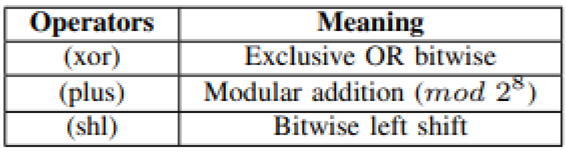
HIGHT algorithm (HIGH security and light weight) was proposed in 2006 by Hong et al. This is a block cipher lightweight cryptographic algorithm. The size of each block is 64 bits and the keys have 128 bits. This algorithm is characterized as a low-cost and low-power, having an ultralight implementation . HIGHT operating principle is based on xor operations, additions to applying mod 2 8 and shifts bitwise to the left. Original paper implementation was performed with 3048 logic elements in a 0.25 μm technology. Tables I and II contain the meaning of variables and operators used in the algorithm, respectively..

TABLE I. MEANING OF USED VARIABLES



Variables	Meaning
P	Initial block to be encrypted
C	Final block (already encrypted)
MK	128 bits user Master key
WK	128 bits Whitening key
SK	Set of 128 8 bits Sub keys
$X_{[0,32]}$	Auxiliary blocks used during processing
delta	Constant vector generated by LFSR

TABLE II. MEANING OF USED OPERATORS



Operators	Meaning
(xor)	Exclusive OR bitwise
(plus)	Modular addition ($mod\ 2^8$)
(shl)	Bitwise left shift

HIGHT algorithm can be described by the pseudo code in Figure 1, based on . The described algorithm takes into account only the encryption process since, the decryption is simple reverse the functions of processing and rotation, and also the order of application thereof. Each procedure used in the algorithm can be described as follows:

1)HightEncryption: Receives the text P and the user's master key MK. This is the main procedure.

2) KeySchedule: Receives the master key MK and call the methods of key generation 3) and 5).

3) WhiteningKeyGeneration: Use the master key MK to generate the 8 bytes white key WK used in the initial and final transformations.

4) ConstantGeneration: Generates 128 7 bit delta constant values to be used in the generation of sub keys SK by a LFSR process. The initial delta is (1011010)2 – 0x5A in hexadecimal.

5) SubkeyGeneration: Generate, from the delta constant vector, the 128 8 bits sub keys SK used in text rotation function – 4 bytes of SK by RoundFunction.

6) InitialTransfomation: Transforms the input text P into X0 through (xor) and (plus) operations with the first 4 bytes of WK key.

7) RoundFunction: Uses the auxiliary functions F0 and F1 to rotate and generate new intermediaries texts Xi .

8) FinalTransfomation: Transforms the last intermediate text X32 into output text C through (xor) and (plus) operations with the last 4 bytes of key WK.

VHDL Implementation

Our VHDL implementation of HIGHT was based on a software implementation in C programming language by Cazorla et al. We used a finite state machine (FSM) abstraction in our implementation shown in Figure 2..

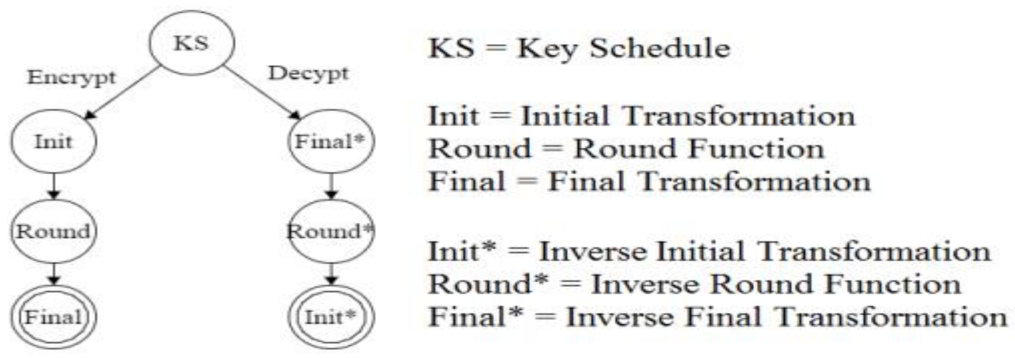
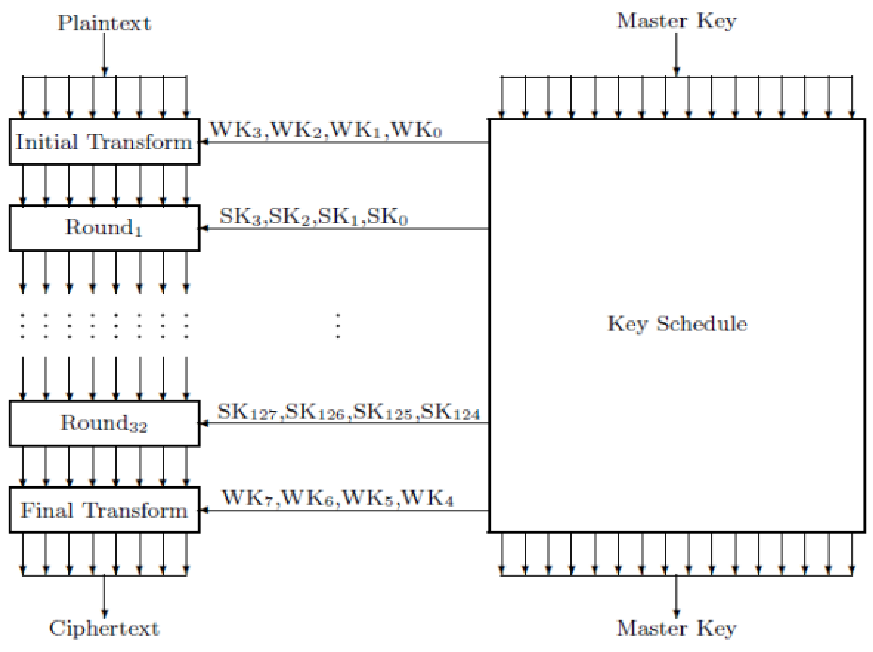

KS
Encrypt
Decypt
Init
Round
Final
Final*
Round*
Init*
KS = Key Schedule
Init = Initial Transformation
Round = Round Function
Final = Final Transformation
Init* = Inverse Initial Transformation
Round* = Inverse Round Function
Final* = Inverse Final Transformation

Fig. 2. HIGHT's Finite State Machine



- *Key Schedule*: Receives the master key MK and call the methods of key generation.
- *Whitening Key Generation*: Uses the master key MK to generate a 8 bytes white key WK.
- *Constant Generation*: Generates 128 7 bit delta constant values to be used in the generation of subkeys SK by a LFSR process. The initial delta is (1011010)2 – 0x5A in hexadecimal.
- *Subkey Generation*: Generates, from the delta constant vector, 128 sub keys SK of 8 bits each used in text rotation function – 4 bytes of SK by Round Function.
- *Initial Transformation*: Transforms text P into X_0 through xor operations and addition mod 2^8 with the first 4 bytes of WK.
- *Round Function* (Figure 3) : Uses two auxiliary functions (F_0 and F_1) to rotate and generate new intermediaries texts X_i.
- *Final Transformation*: Transforms the last intermediate text X_{32} into cipher text C through xor operations and addition mod 2^8 with the last 4 bytes of WK.

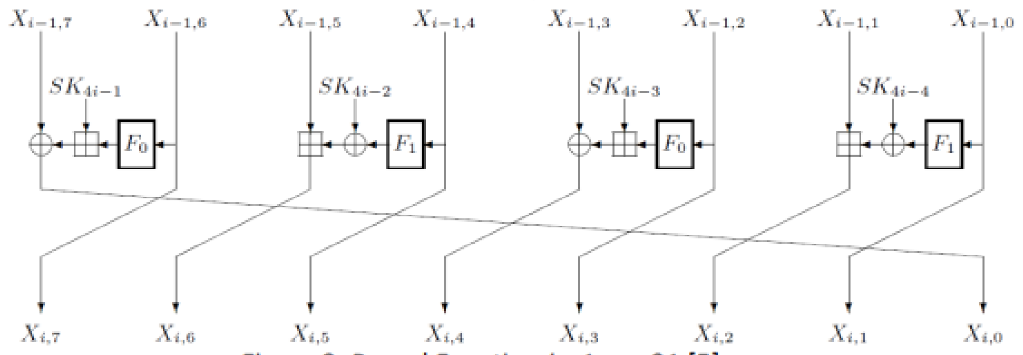
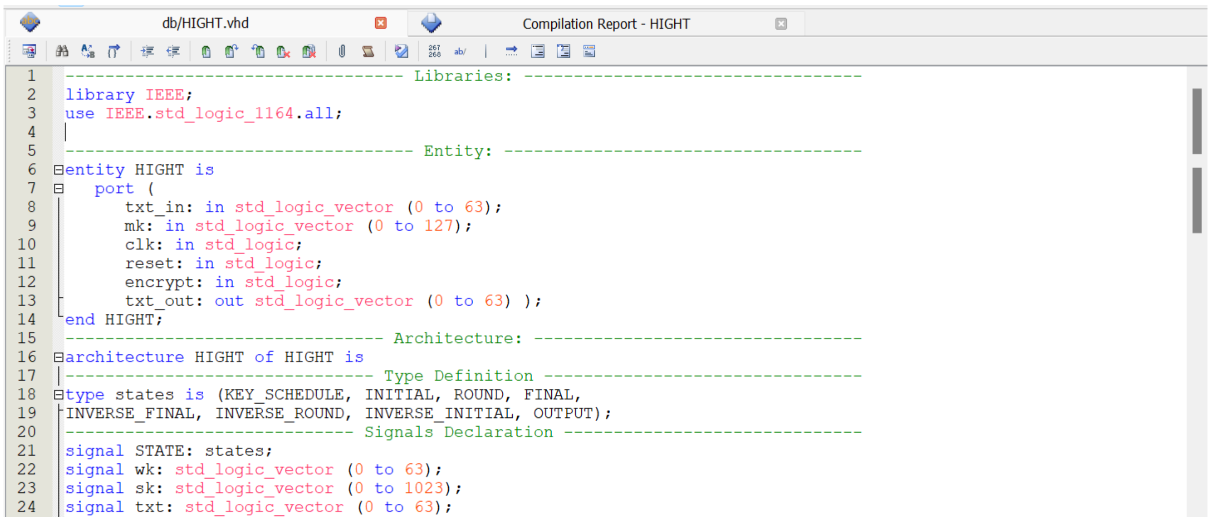


Figure 3: Round Function *i* = 1, ... , 31 [5].

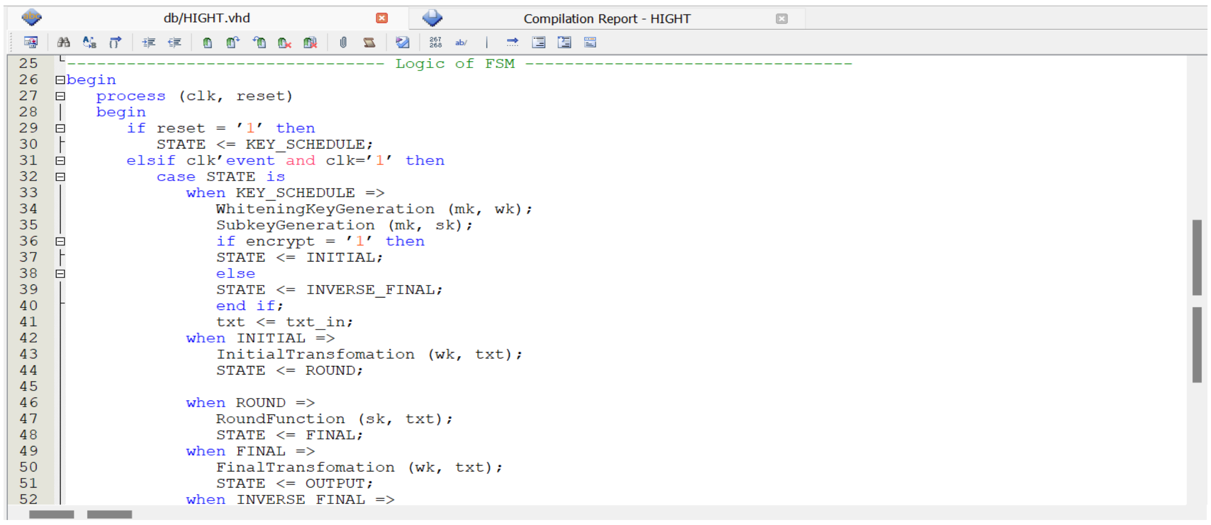
HOW WE DEVIDED THE CODE :

The algorithm input is a text P and a master key MK. There is also a control signal, designating which operation should be performed: encrypt or decrypt the text. Moreover, as the implementation was based on FSM, it was necessary to use clock and reset, since, to transition to a next state, it must be ensured that all operations of current state finished.

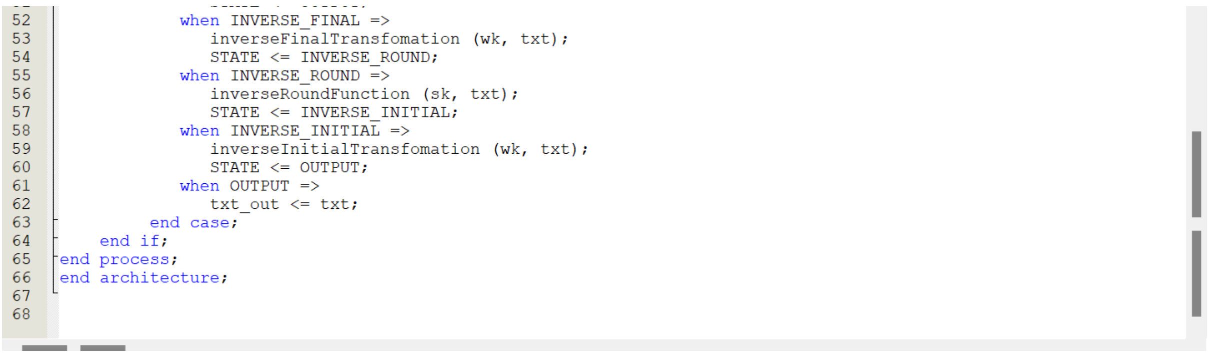
For a better code organization, it was divided into two files: one containing the logic of the FSM and the other with the implementations of the functions and procedures. The main code contains the entity and architecture, where inputs, outputs, operations and FSM transitions are defined. Auxiliary code consists of a package included by main code.



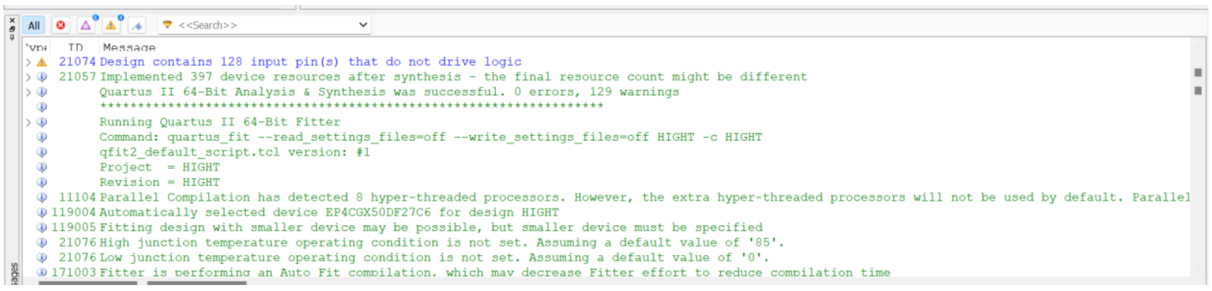
```vhdl
--------------------------------- Libraries: ---------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

---------------------------------- Entity: -----------------------------------
entity HIGHT is
   port (
      txt_in: in std_logic_vector (0 to 63);
      mk: in std_logic_vector (0 to 127);
      clk: in std_logic;
      reset: in std_logic;
      encrypt: in std_logic;
      txt_out: out std_logic_vector (0 to 63) );
end HIGHT;
------------------------------- Architecture: --------------------------------
architecture HIGHT of HIGHT is
------------------------------- Type Definition --------------------------------
type states is (KEY_SCHEDULE, INITIAL, ROUND, FINAL,
INVERSE_FINAL, INVERSE_ROUND, INVERSE_INITIAL, OUTPUT);
----------------------------- Signals Declaration -----------------------------
signal STATE: states;
signal wk: std_logic_vector (0 to 63);
signal sk: std_logic_vector (0 to 1023);
signal txt: std_logic_vector (0 to 63);
```



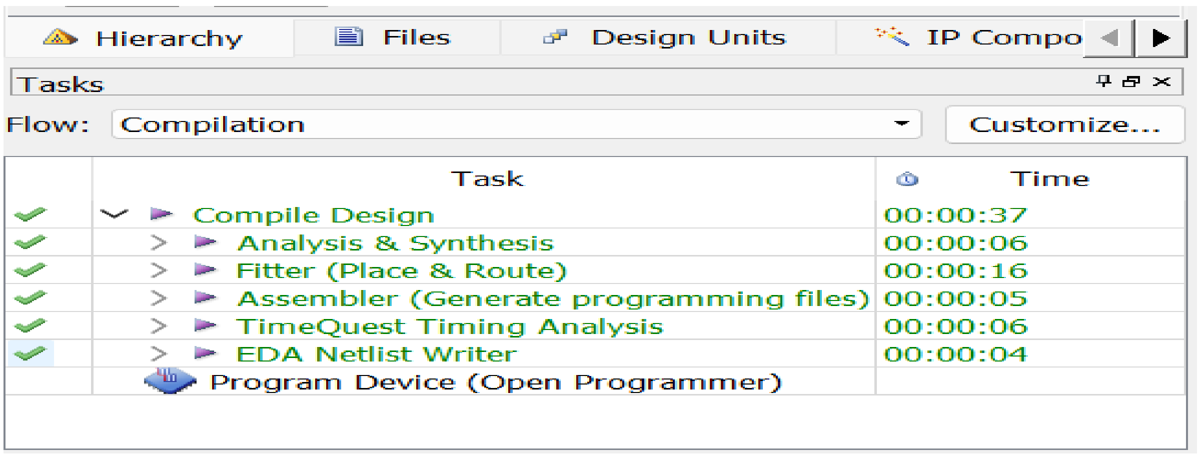
```vhdl
-------------------------------- Logic of FSM --------------------------------
begin
   process (clk, reset)
   begin
      if reset = '1' then
         STATE <= KEY_SCHEDULE;
      elsif clk'event and clk='1' then
         case STATE is
            when KEY_SCHEDULE =>
               WhiteningKeyGeneration (mk, wk);
               SubkeyGeneration (mk, sk);
               if encrypt = '1' then
               STATE <= INITIAL;
               else
               STATE <= INVERSE_FINAL;
               end if;
               txt <= txt_in;
            when INITIAL =>
               InitialTransfomation (wk, txt);
               STATE <= ROUND;

            when ROUND =>
               RoundFunction (sk, txt);
               STATE <= FINAL;
            when FINAL =>
               FinalTransfomation (wk, txt);
               STATE <= OUTPUT;
            when INVERSE_FINAL =>
```



```vhdl
            when INVERSE_FINAL =>
               inverseFinalTransfomation (wk, txt);
               STATE <= INVERSE_ROUND;
            when INVERSE_ROUND =>
               inverseRoundFunction (sk, txt);
               STATE <= INVERSE_INITIAL;
            when INVERSE_INITIAL =>
               inverseInitialTransfomation (wk, txt);
               STATE <= OUTPUT;
            when OUTPUT =>
               txt_out <= txt;
         end case;
      end if;
end process;
end architecture;
```



```
Type  ID     Message
      21074  Design contains 128 input pin(s) that do not drive logic
      21057  Implemented 397 device resources after synthesis - the final resource count might be different
             Quartus II 64-Bit Analysis & Synthesis was successful. 0 errors, 129 warnings
             *******************************************************************
             Running Quartus II 64-Bit Fitter
             Command: quartus_fit --read_settings_files=off --write_settings_files=off HIGHT -c HIGHT
             qfit2_default_script.tcl version: #1
             Project  = HIGHT
             Revision = HIGHT
      11104  Parallel Compilation has detected 8 hyper-threaded processors. However, the extra hyper-threaded processors will not be used by default. Parallel
     119004  Automatically selected device EP4CGX50DF27C6 for design HIGHT
     119005  Fitting design with smaller device may be possible, but smaller device must be specified
      21076  High junction temperature operating condition is not set. Assuming a default value of '85'.
      21076  Low junction temperature operating condition is not set. Assuming a default value of '0'.
     171003  Fitter is performing an Auto Fit compilation, which may decrease Fitter effort to reduce compilation time
```

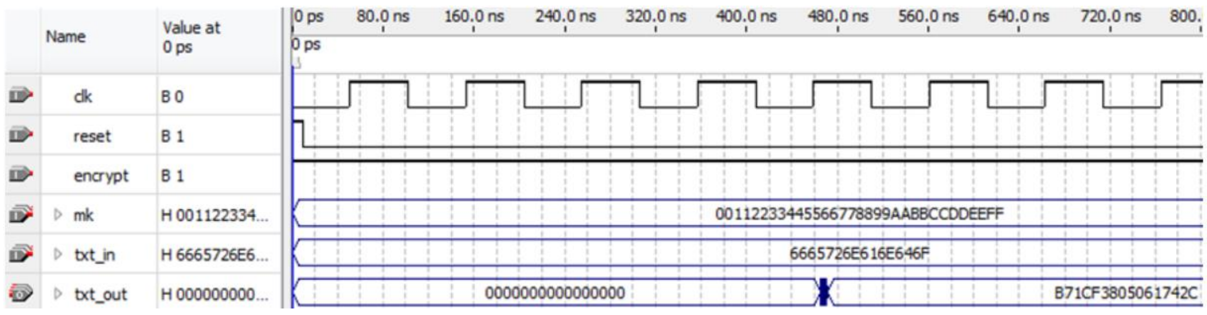


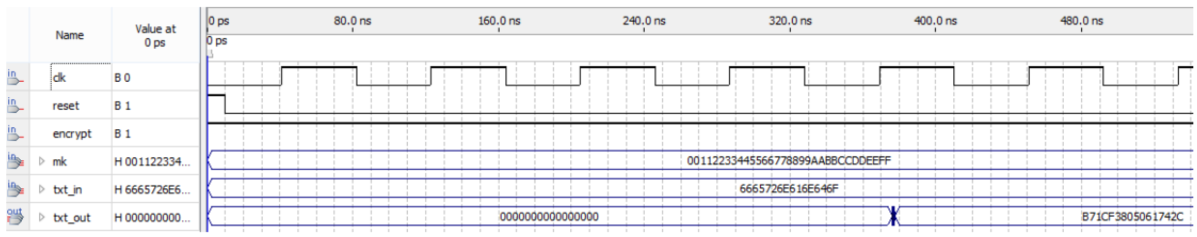
Hierarchy | Files | Design Units | IP Compo

Tasks

Flow: Compilation | Customize...

	Task	Time
✓	Compile Design	00:00:37
✓	Analysis & Synthesis	00:00:06
✓	Fitter (Place & Route)	00:00:16
✓	Assembler (Generate programming files)	00:00:05
✓	TimeQuest Timing Analysis	00:00:06
✓	EDA Netlist Writer	00:00:04
	Program Device (Open Programmer)	





Conclusion

Conclusion : After a review of lightweight cryptographic algorithms, the HIGHT (HIGH security and light weight) algorithm was chosen for a FSM VHDL implementation. our implementation outperformed in frequency and throughput. However, a high number of logical elements was used, once both **encryption** and **decryption** were implemented in the same code. The next step could be the improvement of the current code and the implementation of other lightweight cryptographic algorithms.

Cryptography is a constantly advancing area, being driven by ubiquitous computing. Due to the large amount of important devices with limited space and power consumption, the lightweight cryptography is a viable solution for security on such devices. In particular, algorithms implemented in hardware are usually more efficient than those designed for software.

FIN