



# UNIVERSITÀ DI PISA

**Department of Computer Engineering**

Master's degree program in Cybersecurity

Hardware and Embedded Security course - Project Report

## **Light Hash Algorithm v2 (DES S-Box based)**

**Professor:**

Prof. Saponara Sergio

**Referent:**

Ing. Crocetti Luca

**Students:**

Costanzo Claudio

Ruta Marco

## Table of contents

1. Specification Analysis.....	1
2. Block Diagram and Design Choices.....	2
<b>2.1 Block Diagram .....</b>	<b>2</b>
<b>2.2 Expected behaviour and usage .....</b>	<b>3</b>
<b>2.3 Design choices .....</b>	<b>4</b>
<b>2.4 FSM: States and transitions .....</b>	<b>5</b>
3. Testbench and Waveform discussion .....	7
4. Implementation of RTL design on FPGA .....	9
5. STA: Static timing Analysis.....	12

## Figure Index

Figure 1 - IV of the digest .....	1
Figure 2 - Block Diagram.....	2
Figure 3 - Expected waveform .....	3
Figure 4 - Block diagram of the design .....	4
Figure 5 - Finite State Machine .....	5
Figure 6 - Waveform empty test.....	7
Figure 7 - Waveform one char test .....	8
Figure 8 - Waveform continuous processing.....	8
Figure 9 - Waveform alternate processing .....	8
Figure 10 - Flow Summary.....	9
Figure 11 - Input ports and FSM .....	10
Figure 12 - HashIteration module.....	10
Figure 13 - Round module.....	10
Figure 14 - Sbox module .....	11
Figure 15 - Output ports .....	11
Figure 16 - Time constraints.....	12
Figure 17 - Frequency measurement 0°C.....	12
Figure 18 - Frequency measurement 85°C.....	12

# 1.Specification Analysis

Light Hash Algorithm v2 (DES S-box Based) is a hash module based on the S-box of DES algorithm, which computes 32-bit digest formed by the concatenation of 8 nibbles (4-bit vectors).

H array represents the digest and for each message the H[i] variables are initialized with the following Initialization Vector (IV):

	<b>H[0]</b>	<b>H[1]</b>	<b>H[2]</b>	<b>H[3]</b>	<b>H[4]</b>	<b>H[5]</b>	<b>H[6]</b>	<b>H[7]</b>
<b>Init. value</b>	4'hF	4'h3	4'hC	4'h2	4'h9	4'hD	4'h4	4'hB

Figure 1 - IV of the digest

The module performs the following operation for each byte M of the input message, to modify the digest vector.

```
for (r = 0; r < 4; r++)
    for(i = 0; i < 8; i++)
        H[i] = (H[(i + 2) mod 8] ⊕ S(M6)) << [i/2]
```

Where:

- **mod n** is the modulo operator by n.
- **⊕** is the XOR binary operator.
- **X << n** is a left circular shift by n bits.
- **[x]** is the floor function applied to x.
- **M<sub>6</sub>** is a 6-bit vector generated from the input byte M by a compression algorithm.
- **S()** is the S-box transformation of DES (S-box 5).

The compression algorithm used to generate the **M<sub>6</sub>** vector is the following and is performed over a single byte input M, assuming M[7:0] where M[7] is the MSB:

$$M_6 = \{M[5], M[7] \oplus M[2], M[3], M[0], M[4] \oplus M[1], M[6]\}$$

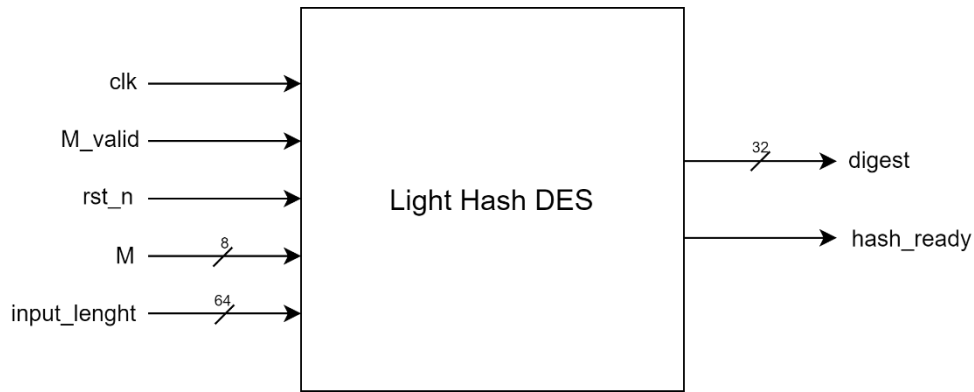
Other specifications of the design are:

- The module is expected to process one byte of the input message, assumed as ASCII character, in one clock cycle.
- It has an active low asynchronous reset input port, named *rst\_n*, that can be activated in order to reset the status of the module.
- It has an input port, named *M\_valid*, which has to be set when the character in input has to be considered valid and stable, unset otherwise.
- It has an output port, named *hash\_ready*, which has to be set when the digest in output has to be considered valid and stable, unset otherwise. This flag shall be kept high until a new message digest computation is performed by the module.

## 2. Block Diagram and Design Choices

Based on the specifications described in Chapter 1, this chapter describes the various choices made for the module design and the corresponding architecture.

### 2.1 Block Diagram



*Figure 2 - Block Diagram*

In the *figure 2* it can be seen that there are five incoming connections and three outgoing connections in the module. Regarding these links:

- **Clock** (input, 1 bit): clock signal, all the ports of the module are sampled on the rising edge of the clock.
- **M\_valid** (input, 1 bit): flag used to understand when the input character is valid and stable.
- **rst\_n** (input, 1 bit): low active asynchronous reset signal used to restore the default values of the registers inside the module.
- **M** (input, 8 bit): input byte that has to be processed, represents an 8-bit ASCII character.
- **input\_lenght** (input, 64 bit): specifies the length of the input message.
- **digest** (output, 32 bit): digest result of the processing activity performed over the input message.
- **hash\_ready** (output, 32 bit): flag used to understand when the digest of the input message is ready and stable.

## 2.2 Expected behaviour and usage

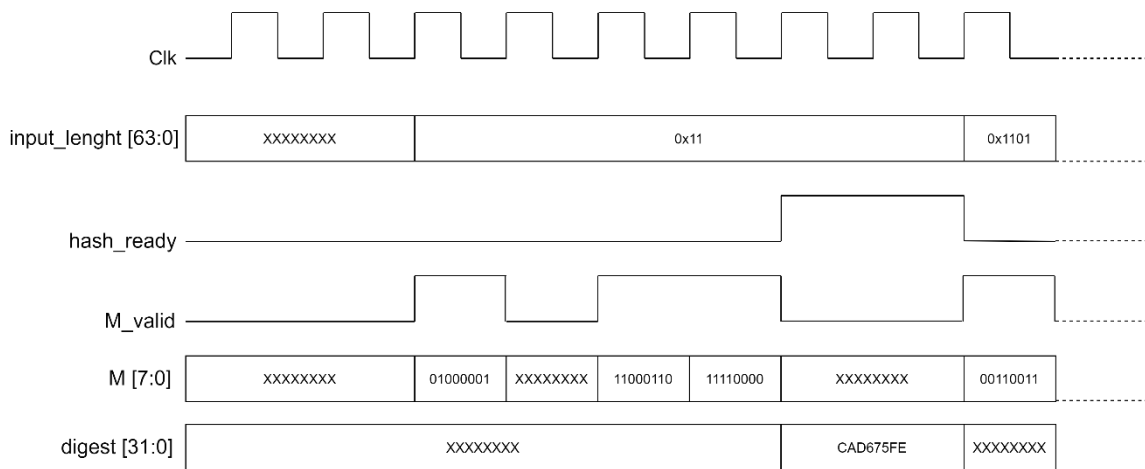


Figure 3 - Expected waveform

In Fig.3 is the expected waveform is reported and below there are some considerations about the usage of the circuit that explain how it should be correctly integrated inside and hardware secure module:

- **input\_lenght**: has to be set to the length of the input message when the first character of the message is provided on the M port. Remains set throughout the processing activity of the message.
- **hash\_ready**: is set only after the last character of the input message is computed and thus, when the output digest is ready and stable; is unset when a new character is provided in input and M\_valid is set because a new processing activity has begun.
- **M\_valid**: has to be set when the input character M has to be considered valid and stable.
- **M**: has to be set to the ASCII value of the character that has to be processed.
- **digest**: it stores the digest value only after the entire message in input is processed, has to be considered valid only if hash\_ready is set.

So, in order to use the model these are the right steps to perform:

- 1) The user must set *M\_valid* and send the first byte *M*, along with the input length of the entire message, in order to start a processing activity.
- 2) The user must set *M\_valid* and send each byte to the *M* port, one per clock cycle.
- 3) The module signals when the processing activity ends, and the digest is ready, setting the *hash\_ready* port; then the user can read the computed digest on the *digest* port

## 2.3 Design choices

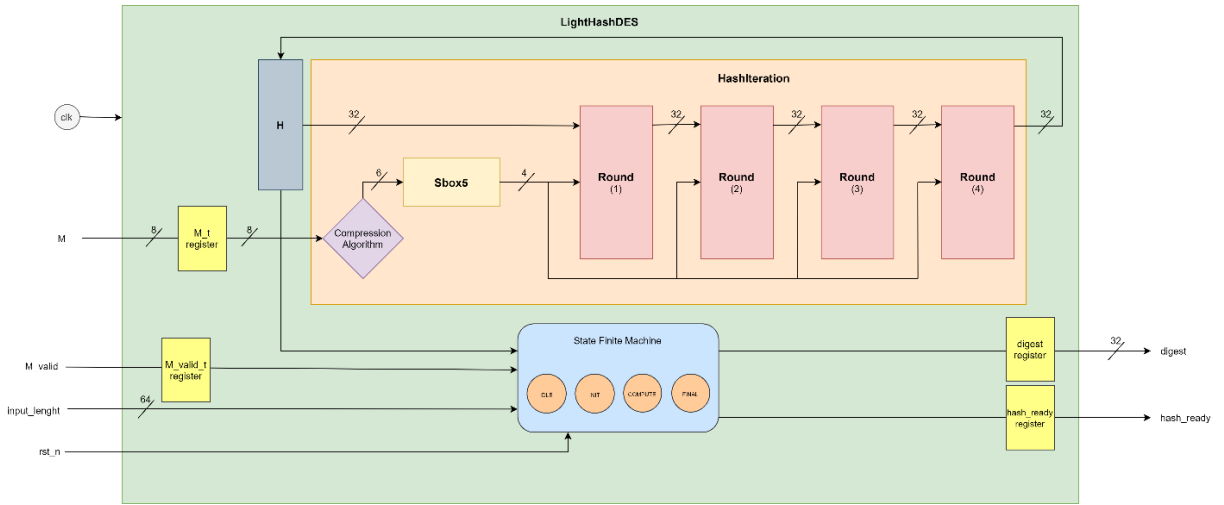


Figure 4 - Block diagram of the design

Here are listed all the design choices that have been made in order to increase the performance and the readability of the implemented design:

- The input byte  $M$  and the flag  $M\_valid$  are stored in two dedicated registers,  $M\_t$  and  $M\_valid\_t$  in order to avoid long combinatory paths.
- The input port  $input\_length$  has been added to let the device know when the last byte is processed, to understand if the digest is ready or if there are more bytes to process.
- The output  $digest$  and the flag  $hash\_ready$  are not directly sent to the pins but are stored in the  $digest$  and  $hash\_ready$  registers in order to avoid long combinatory paths.

Each byte of the message is processed in a single clock cycle by some combinatory sub-modules, the choice of divide the design in more modules (stored in different files) has been made in order to enhance the readability of the code:

- **Sbox5** is a combinatory sub-module which implement a LUT S-Box with 6-bit input and 4-bit output. The interface is composed of:
  - $in$  (input, 6 bit): the 6-bit signal used to access the S-box.
  - $out$  (output, 4 bit): the 4-bit output of the S-box.
- **Round** is a combinatory sub-module that performs the operations: XOR, modulo, floor, and circular shift. The interface is composed of:
  - $s\_box\_out$  (input, 4 bit): 4-bit signal, output of the Sbox5 module.
  - $h\_in$  (input, 32 bit): the temporary 32-bit digest in input.
  - $h\_out$  (output, 32 bit): the temporary 32-bit digest in output.

- **HashIteration** is a combinatory sub-module that performs: the compression algorithm over the input byte, the S-box look-up operation, and instances four *Round* sub-module, in order to perform the round operation four time. The interface is composed of:
  - $M$  (input, 8 bit): 8-bit signal, representing the message byte to process.
  - $h$  (input, 32 bit): the 32-bit temporary digest in input.
  - $h\_out$  (output, 32 bit): the temporary 32-bit digest in output.

All these sub-modules are integrated in the top-level module **lightHashDES** that includes the control network and the FSM which regulates the dataflow of the hash function.

## 2.4 FSM: States and transitions

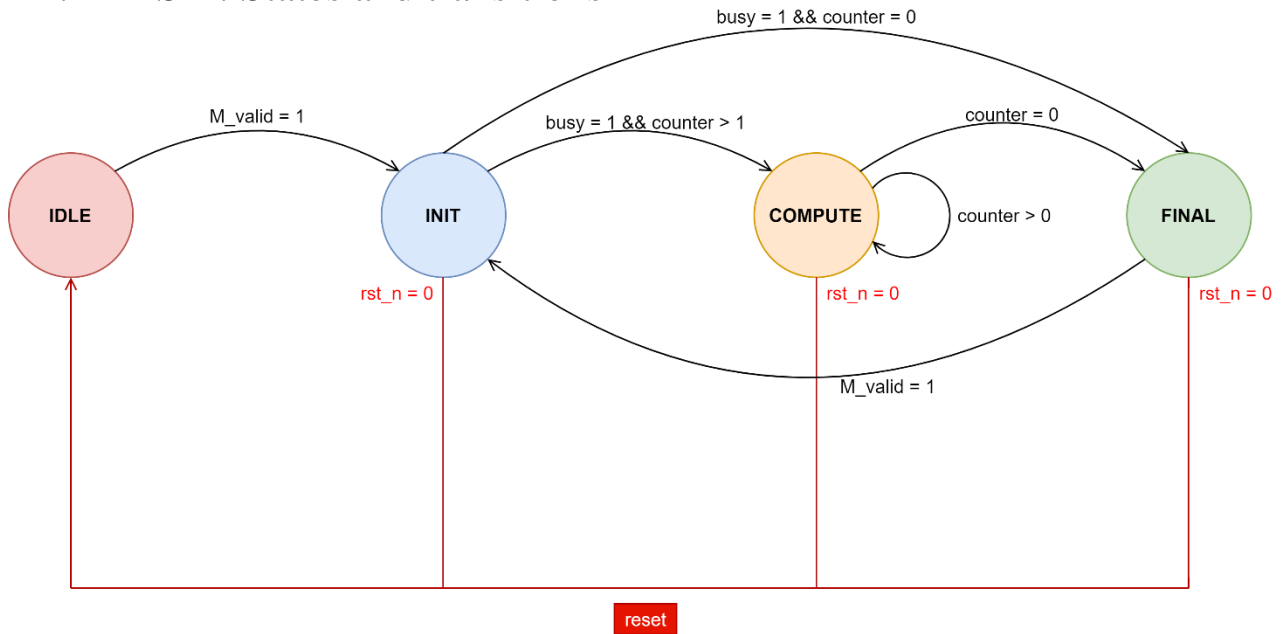


Figure 5 - Finite State Machine

The **lightHashDES** module integrates a Finite State Machine that has been implemented in order to deal in a clearer way with the data flow of the design. The FSM has four states, and the transitions are executed checking conditions over the values of support variables and input/output ports of the module:

The support variables used by the FSM are:

- **Busy:** is a binary variable that is set when a computation of a digest is performed. Otherwise, is unset when the module is not computing a digest.
- **Counter:** is a 64-bit variable used to keep trace of the remaining number of bytes to process. When the first byte of the input message is given in input, the length of message is stored in counter, and after each byte computation the counter variable is decreased by one.

The states of the FSA are:

- 1) **IDLE:** the module is waiting for a new message to process, sets hash\_ready = 0 and busy = 0.
  - If M\_valid = 1 a transition from IDLE to INIT state is triggered.
- 2) **INIT:** the module sets the IV values of the digest, stores the length of the input message in the *counter* register, sets busy = 1, hash ready = 0, stores M in the register *M\_t*, and M\_valid in *M\_valid\_t*.
  - If counter > 0 and busy = 1 a transition to COMPUTE state is triggered.
  - If counter = 0 and busy = 1 a transition to FINAL state is triggered.
  - If rst\_n = 0 a transition to the IDLE state is triggered.
- 3) **COMPUTE:** the module processes the byte that is stored in the register *M\_t*, updates the temporary digest value, and decrements the counter by one. If *M\_valid* is set, the module stores the next byte to process in the *M\_t* register.
  - If counter > 0 a self- referencing transition to COMPUTE state is triggered.
  - If counter = 0 a transition to FINAL state is triggered.
  - If rst\_n = 0 a transition to the IDLE state is triggered.
- 4) **FINAL:** the module sets hash\_ready = 1, busy = 0, and stores the temporary digest value in the output port digest.
  - If M\_valid = 1 a transition to INIT state is triggered.
  - If rst\_n = 0 a transition to the IDLE state is triggered.



### 3. Testbench and Waveform discussion

In this section are shown the tests which have been performed over the implemented design and a brief discussion is given over the resulting waveforms. The testbench was performed using Modelsim and below are listed the tests conducted and the signals present in each waveform.

Three different tests have been conducted, each having its own particular purpose:

- **TestbenchEmptyAndOneChar:** it aims to test two different corner cases: in the first one an empty string is given in input, and the result is expected to be the IV of H; In the second one a single character is given in input and the result is expected to be a known digest that has been computed by hand before.
- **TestbenchStrings:** it targets three strings that have been declared in the test set-up. Two of the strings are equal: *string\_1a* = *string\_1b* = "Welcome\_to\_testing", and the third one differs from the previous ones by only one byte : *string\_2a* = "Welcoma\_to\_testing". The digests of *string\_1a* and *string\_2a* have been computed in two different ways: for *string\_1a* one byte was given in input per clock cycle while for *string\_1b* one byte was given in input every two clock cycles. This choice was made in order to demonstrate how the behaviour of the model is the same both under continuous and alternating computation, in fact, the result is the same digest. The digest of *string\_2a* has been computed in a continuous way, only to show that the result digest differs from *string\_1a* and *string\_1b* digest.
- **TestbenchLongString:** it aims to stress the module with a long message as input (2000 characters). In this case was not possible to easily obtain by hand an expected output like the previous test cases, hence, this test verifies the correct behaviour of the model in terms of amount of clock cycles needed to produce the digest.

Below are reported four significant waveforms, in order to better understand the behaviour of the model.



Figure 6 - Waveform empty test

In Fig. 6 is shown the waveform related to the processing of an empty message. This process lasts two clock cycles, this is because only INIT state and FINAL state are crossed. *M\_valid* starts the computation and at the end of it both *digest* and *hash\_ready* are set at the same time. The result digest is the IV uploaded in the INIT state.

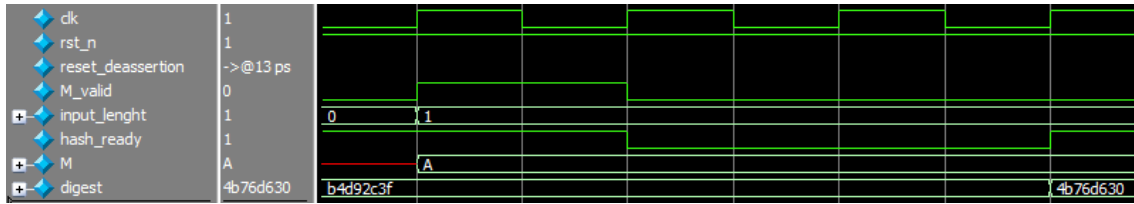


Figure 7 - Waveform one char test

In Fig. 7 is shown the waveform related to the processing of a single-character message. This process lasts three clock cycles. *M\_valid* starts the computation and the previous digest (computed over the empty message) is not valid anymore because *hash\_ready* is set to zero. At the end both *digest* and *hash\_ready* are set at the same time and the output digest is the expected one.

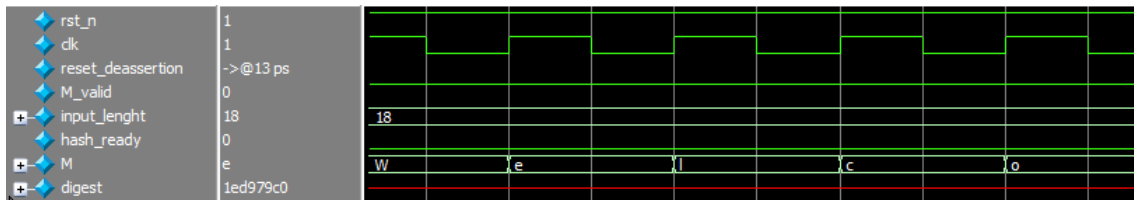


Figure 8 - Waveform continuous processing

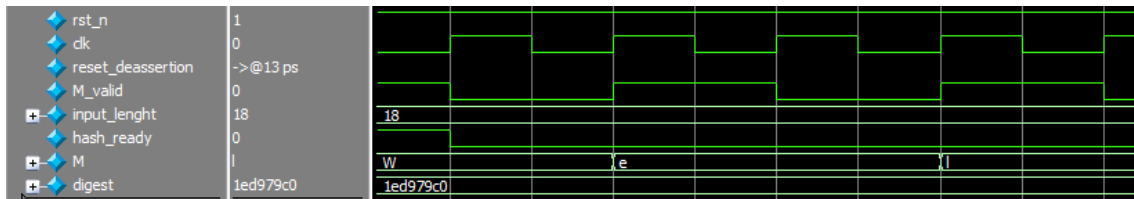
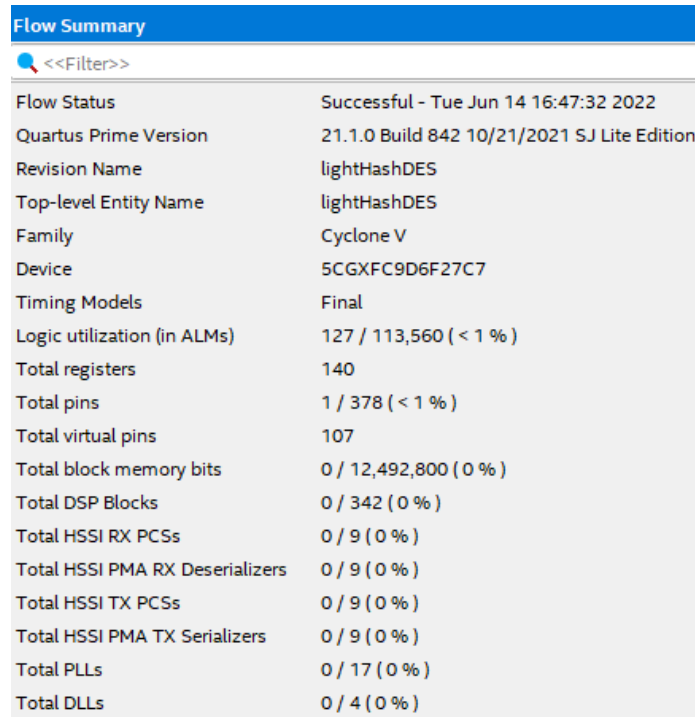


Figure 9 - Waveform alternate processing

In Fig.8 and Fig.9 are shown the waveforms related to a continuous and a non-continuous byte processing. In both cases *M\_valid* starts the computation, in the case of continuous computation (Fig.8) *M\_valid* is kept high for the whole execution time, meanwhile, in the case of non-continuous computation (Fig. 9), *M\_valid* is periodically (every two clock cycles) set and unset for the whole execution time.

## 4.Implementation of RTL design on FPGA

The Circuit Design, Physical Design, and Static Time Analysis steps were performed using the tool Quartus Prime by Intel. The Static Time Analysis is described in the following chapter. The technology on which the model was chosen to be synthesized is the FPGA 5CGXFC9D6F27C7 of the Cyclone V family by Intel.



Flow Status	Successful - Tue Jun 14 16:47:32 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	lightHashDES
Top-level Entity Name	lightHashDES
Family	Cyclone V
Device	5CGXFC9D6F27C7
Timing Models	Final
Logic utilization (in ALMs)	127 / 113,560 ( < 1 % )
Total registers	140
Total pins	1 / 378 ( < 1 % )
Total virtual pins	107
Total block memory bits	0 / 12,492,800 ( 0 % )
Total DSP Blocks	0 / 342 ( 0 % )
Total HSSI RX PCSs	0 / 9 ( 0 % )
Total HSSI PMA RX Deserializers	0 / 9 ( 0 % )
Total HSSI TX PCSs	0 / 9 ( 0 % )
Total HSSI PMA TX Serializers	0 / 9 ( 0 % )
Total PLLs	0 / 17 ( 0 % )
Total DLLs	0 / 4 ( 0 % )

*Figure 10 - Flow Summary*

In Fig.10 is reported the flow summary of Quartus synthesis, it can be seen that the resource usage of target technology is low (< 1%) so the design can be easily integrated on the FPGA board along with other components. Only one physical pin is used, and it is the Clock pin, input and output pins are assigned as virtual pins, in order to increase the overall frequency of the circuit.

The number of virtual pins used is coherent with the number of inputs and outputs of the RTL design generated with Modelsim (107):

- input\_lenght: 64-bit
- digest: 32-bit
- M: 8-bit
- M\_valid: 1-bit
- rst\_n: 1-bit
- hash\_ready: 1 bit

In the following section are reported several significant screenshots of Quartus RTL viewer, in order to show how the tool synthesized the developed design.

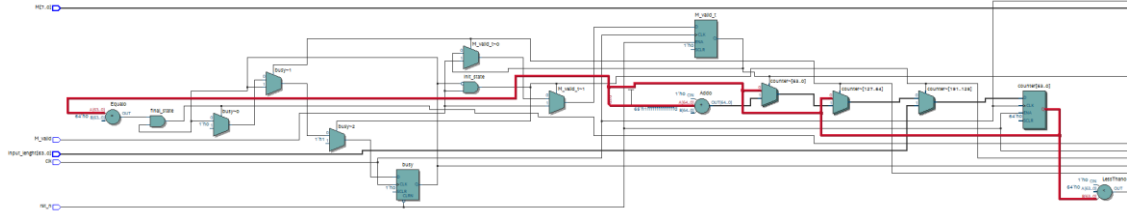


Figure 11 - Input ports and FSM

In Fig.11 are shown the input ports of the model and part of the FSM network. The highlighted signal is the length of the message, which is stored inside the counter register and decreased by one for each byte computation. The counter register output signal is compared two times, in order to see if the last byte was processed or if there are other bytes to process. In the figure are also shown the busy and M\_valid\_t registers, which are used in order to determine the next state of the FSM.

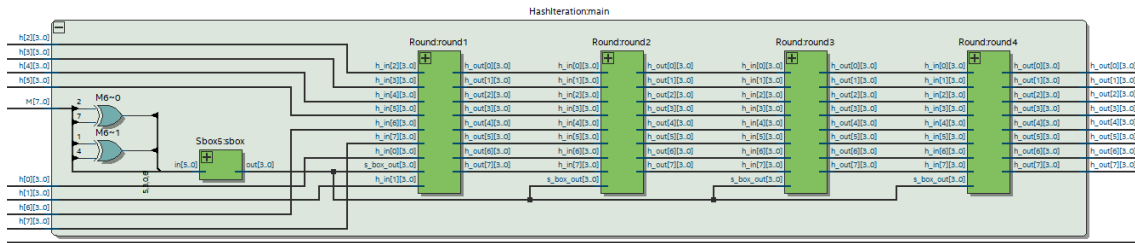


Figure 12 - HashIteration module

In Fig. 12 is shown the HashIteration module, which integrates 4 Round module, 1 Sbox5 module, and the compression algorithm which computes the signal  $M_6$ .

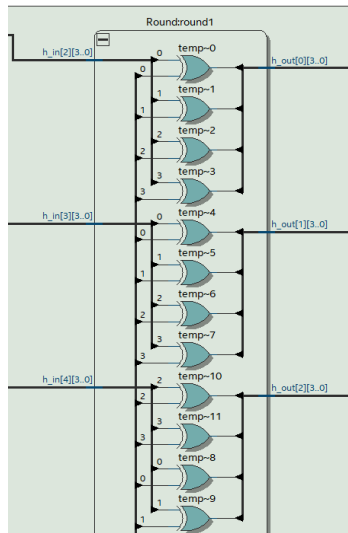


Figure 13 - Round module

In Fig.13 is shown a part of the Round module, which executes the XOR and circular shift operations over the temporary digest and the Sbox5 output signal.

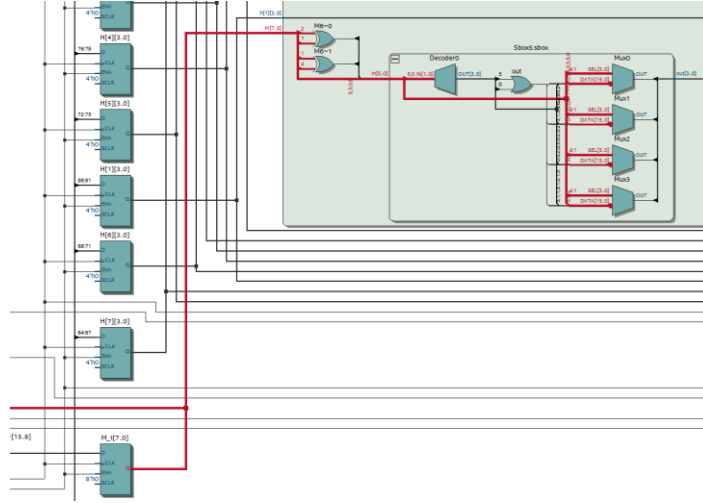


Figure 14 - Sbox module

In Fig.14 is shown the implementation of the Sbox that is integrated inside the HashIteration module. The input signal of the Sbox module is  $M_6$ , which drives the four MUX that constitutes the Sbox design.

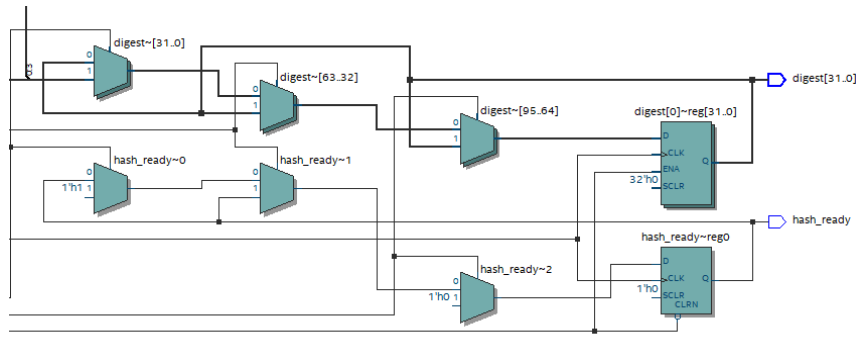


Figure 15 - Output ports

In Fig.15 is shown the output network of the model. The two output ports, digest, and hash\_ready, are driven by the output of two dedicated registers.

## 5.STA: Static timing Analysis

As last step of the project a Static Timing Analysis has been performed over the developed model. In the file “SDC1.sdc” are defined the following timing constraints:

```
create_clock -name clk -period 10 [get_ports clk]
set_false_path -from [get_ports rst_n] -to [get_clocks clk]

set_input_delay -min 1 -clock [get_clocks clk] [all_inputs]
set_input_delay -max 2 -clock [get_clocks clk] [all_inputs]
set_output_delay -min 1 -clock [get_clocks clk] [all_outputs]
set_output_delay -max 2 -clock [get_clocks clk] [all_outputs]
```

Figure 16 - Time constraints

- A target clock frequency of 100 MHz was specified, defining a 10 nanoseconds clock period.
- The rst\_n signal was unpaired from the clock signal because rst\_n is the only signal that has to be asynchronous from the clock signal.
- Minimum, and maximum delay have been defined for each input and output port, the minimum accepted delay is 10% of the Clock period, the maximum accepted delay is 20% of it.

Below are reported the frequency measurements, one obtained during the Slow Test at 0°C and the other during the Slow Test at 85°C.

Slow 1100mV 0C Model Fmax Summary			
<<Filter>>			
	Fmax	Restricted Fmax	Clock Name
1	129.48 MHz	129.48 MHz	clk

Figure 17 - Frequency measurement 0°C

Slow 1100mV 85C Model Fmax Summary			
<<Filter>>			
	Fmax	Restricted Fmax	Clock Name
1	129.62 MHz	129.62 MHz	clk

Figure 18 - Frequency measurement 85°C

As it can be seen, the timing requirements are met in both cases. The worst case is at 0°C with a frequency equal to 129.48 MHz, and this is the one that has to be taken in account in order to deal with the worst possible environmental conditions.