

UNIVERSITY OF PISA



School of Engineering

Master of Science in Computer Engineering

Hardware And Embedded Security

PROJECT DOCUMENTATION

AES-SBOX BASED STREAM CIPHER

WORKGROUP:

Leonardo Cecchelli

Iacopo Pacini

ACADEMIC YEAR 2020/2021

Summary

Specification analysis 3

Block diagram and design choices..... 5

Waveforms 7

Testbench 8

Implementation of RTL design on FPGA and results 9

Specification analysis

The aim of this project is to build a stream cipher based on the AES S-box mechanism. In particular, the plaintext, an 8-bit ASCII coded character, is XORed with the value obtained by the substitution of an 8-bit counter value inside the AES S-Box component. The counter value is initialized with the value of the symmetric key, provided as input, and then incremented whenever a new character is being fed to the circuit. The encryption consists in:

$$C[i] = P[i] \oplus S(CB[i])$$

Where:

$C[i]$ is the 8-bit ASCII code of the i^{th} character of ciphertext.

$P[i]$ is the 8-bit ASCII code of the i^{th} character of plaintext.

$CB[i]$ is the 8-bit value of the i^{th} counter (counter block), for $i = 0, 1, 2, \dots$, and it can be represented by the formula $CB[i] = K + i \bmod 256$, being K the 8-bit symmetric key.

$S()$ is the S-box transformation of AES algorithm, that works over a byte.

\oplus is the XOR operator.

From these requirements we first studied which were the input and output needed for our logic network for the encryption.

Required inputs:

1. symmetric key
2. plaintext
3. din_valid

The symmetric key is an 8-bit numerical value while the plaintext must be interpreted as an 8-bit ASCII coded character. There is also a din_valid pin which must be used to notify cipher when the input values are ready.

Required outputs:

1. dout_ready
2. ciphertext

The ciphertext is the result of the encryption process while the dout_valid pin acts like a notification that the output data is ready, therefore a consumer network needs to check it before consuming the ciphertext data.

From these inputs, that were provided as a requirement, we started thinking how we should treat the arrival of new plaintext in the same encryption session: we decided that our network must be able to manage sequential plaintext arrival. In particular, if the user driving the network keeps the din_valid pin to 1 and changes the plaintext at each clock, the network must be able to sample each new input and provide as output the corresponding encrypted value at the subsequent clock. We also evaluated other possibilities to differentiate the arrival of sequential plaintext, like for example make it explicit by exploiting the din_valid pin: the user piloting the network needs to reset the din_valid pin and raise it up again in the next clock cycle to differentiate different plaintext arrival. None of the solutions looked brilliant because they affect the throughput by wasting clock cycles. The decision of not relying on any explicit input for new plaintext arrival implies that encryption must be performed in just one clock cycle after the corresponding sampling, reducing as much as possible the overhead due to input data variation.

As another design choice we decided to not insert an error pin in case of wrong inputs. So, in case the user provides as plaintext not an ASCII coded character the network will perform the encryption as well and it will not generate an internal error.

Finally, we started reasoning about the decryption module. From our design schemes it basically turned out that the decryption module is exactly the same as the encryption one, where the ciphertext (instead of the plaintext), the symmetric key and din_valid are provided as inputs, while the plaintext and dout_ready are provided as output.

From a mathematical perspective this reasoning is true thanks to those properties of the XOR operator:

$$(1) A \oplus A = 0$$

$$(2) A \oplus 0 = A$$

Suppose we have as input a symmetric key k and a ciphertext c , what will happen is that the substitution inside of the S-Box is the same that the one performed during the encryption (since it still uses just the symmetric key).

Let's call with S the substituted S-Box value from the key.

Encryption is

$$C = P \oplus S.$$

Decryption instead is

$$C \oplus S = (P \oplus S) \oplus S = P \oplus S \oplus S = P \oplus (S \oplus S) = P$$

So, in the end we decided to not create a new module for decryption but instead we will use the same encryption module for that.

Block diagram and design choices

The device is implemented as a Mealy machine operating in four possible states, stored into STAR register:

- **S0**, reset state triggered by a negative edge of the *_reset* signal meant for keeping the state untouched regardless of how inputs are driven. State kept until the very next positive edge of the *_reset* signal.
- **S1_INIT**, reached right after S0. In this state both *plain_text* and *symmetric_key* input signals are stored in the correspondent registers when *din_valid* pin is high.
- **S1**, like the above one, except for the fact that the key stored in S1_INIT state is kept updated as new input are supplied to the circuit.
- **S2**, finally, in this state the output is ready and stored in TXT_OUT_CHAR register and *dout_ready* signal is set to "1". As long the input signal *din_valid* is kept high this state will not be exited providing the outer world a new valid output every cycle, otherwise the circuit enters S1 state waiting for a new valid input.

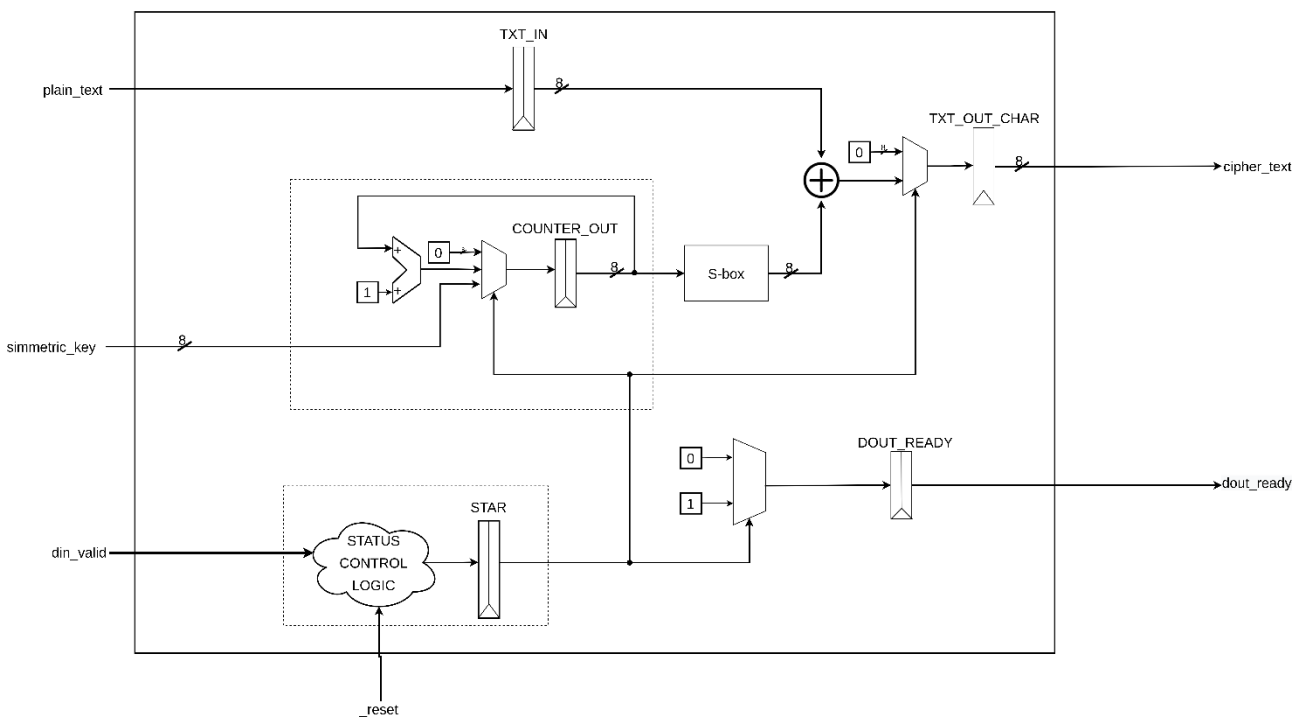


Figure 1 logic representation of the encryption/decryption circuit.

Such architecture allows the user to continuously supply input (both plain text and cipher text) and consequently retrieving valid output every clock cycle. The presence of state S1 can be useful in the circumstance in which a train of input data cannot be furnished. In this state, in fact, the circuits maintain the key for the next byte and waits for a valid input.

Next image shows a theoretical complete round of execution.

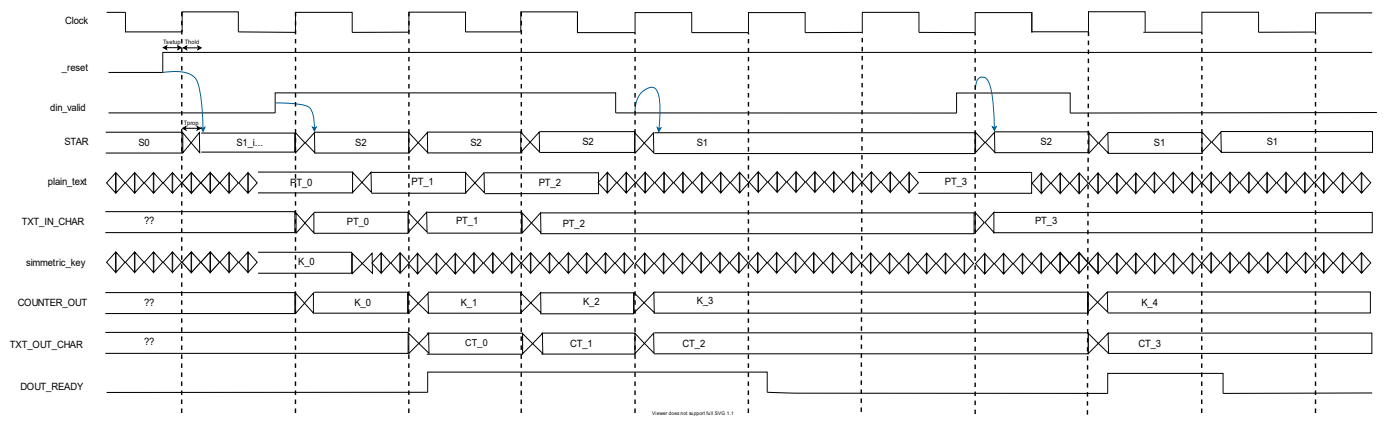


Figure 2 Theoretical round of execution

Waveforms

The simulation shown in figure 3 is a practical demonstration of the architecture explained above. It is possible to denote the positive edge of the *rst_n* signal that is followed by the assignment of the symmetric key and the first byte to be encrypted respectively on the *symmetric_key* and *txt_in_char* pins. The effective process of encryption, once sampled the plaintext (and the key as well), takes only the time of traversing the LUT and a barrier of XORs. Both are implemented as asynchronous logic, whose propagation time is well contained in a clock cycle.

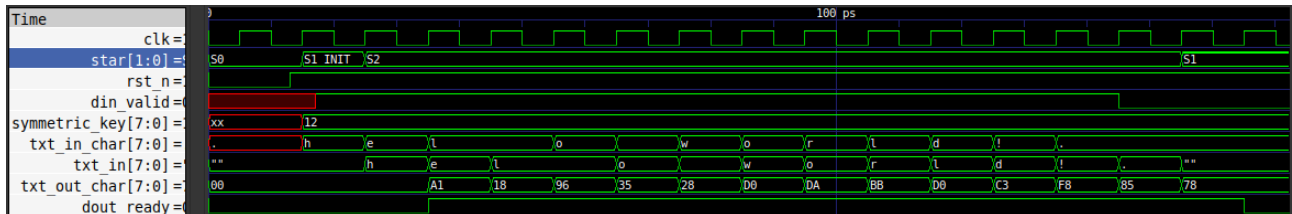


Figure 3 simulation of file encryption containing "hello world!" with key 8'h12

The same can be told for the decryption process:

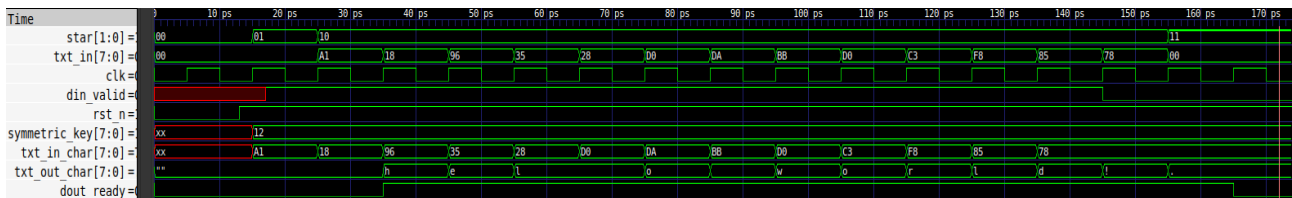


Figure 4 simulation of file decryption

Testbench

The test phase is composed by a testbench written in Verilog and a script written in Python simulating the circuit, capable of producing valid output for a given plaintext.

The testbench consists simply in feeding the cipher the conte of a cipher-text one byte at a time and then storing the output into a brand-new file. The latter is then matched with the correspondent target output obtained leveraging the Python script.

Instead, to check the validity of the file containing deciphered text it is enough to confront them with the original text-file previously encrypted and the decrypted.

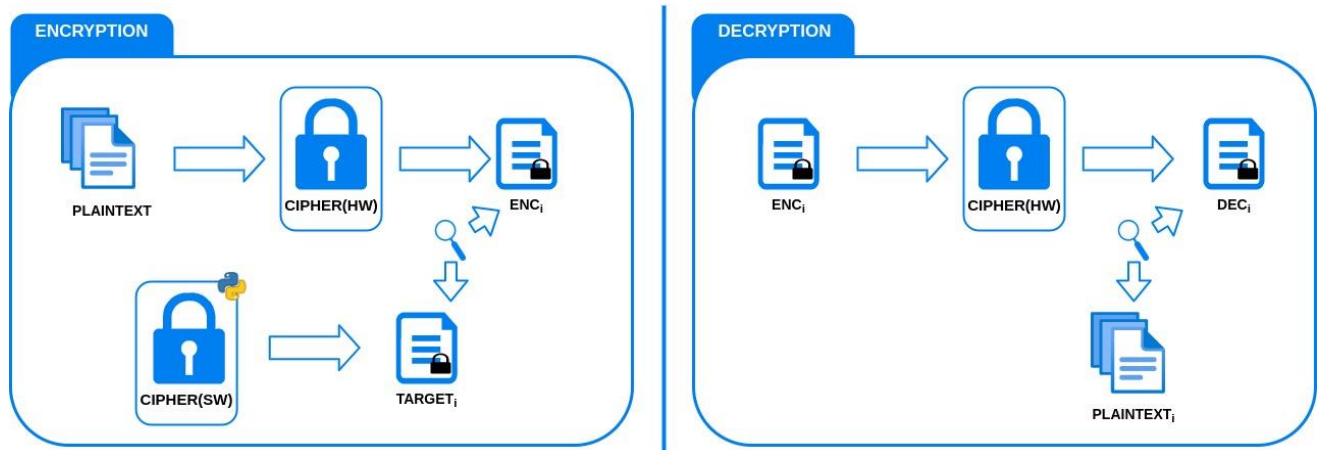


Figure 5 High-Level representation of the testing flow

Implementation of RTL design on FPGA and results

We implemented the RTL design on the device 5CGXFC9D6F27C7 of the Cyclone V family.

Compilation Report - aes_sbox_stream_cipher

Flow Summary

Flow Status: Successful - Tue Jun 15 20:27:25 2021

Quartus Prime Version: 20.1.1 Build 720 11/11/2020 SJ Lite Edition

Revision Name: aes_sbox_stream_cipher

Top-level Entity Name: aes_sbox_stream_cipher

Family: Cyclone V

Device: 5CGXFC9D6F27C7

Timing Models: Final

Logic utilization (in ALMs): 72 / 113,560 (< 1 %)

Total registers: 32

Total pins: 1 / 378 (< 1 %)

Total virtual pins: 27

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 %)

Tasks

Task	Time
Compile Design	
Analysis & Synthesis	00:00:16
Fitter (Place & Route)	00:01:10
Assembler (Generate programming files)	
Timing Analysis	
EDA Netlist Writer	

Static Timing Analysis (STA)

We added a Synopsis Design Constraints named `timing_constraints.sdc` where added a timing constraint for the clock but also the input and output delays.

```
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] [get_ports {rst_n din_valid simmetric_key[*] txt_in_char[*]}]
```

```
set_input_delay -max 2 -clock [get_clocks clk] [get_ports {rst_n din_valid simmetric_key[*] txt_in_char[*]}]
```

```
set_output_delay -min 1 -clock [get_clocks clk] [get_ports {dout_ready[*] txt_out_char[*]}]
```

```
set_output_delay -max 2 -clock [get_clocks clk] [get_ports {dout_ready[*] txt_out_char[*]}]
```

After we ran the STA without the virtual pin assignment, we obtained these results in terms of maximum frequency:

Slow 1100mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	94.19 MHz	94.19 MHz	clk	

Slow 1100mV 0C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	93.94 MHz	93.94 MHz	clk	

Although when checking the Timing Closure Recommendations tab, we found that there were some failing setup paths with the output pins of the ciphertext data. This was due to the fact that we had not performed yet the virtual pin assignment.

Timing Closure Recommendations			
Summary			
This design contains failing setup paths with a worst-case slack of -0.617 ns. Run Report Timing Closure Recommendations for recommendations on how to close setup timing. For recommendations for any particular path, click the appropriate link in the table below.			
Top Failing Paths			
Slack	From	To	Recommendations
1 -0.617	txt_out_char[4]~reg0	txt_out_char[4]	Report recommendations for this path
2 -0.582	txt_out_char[7]~reg0	txt_out_char[7]	Report recommendations for this path
3 -0.580	txt_out_char[1]~reg0	txt_out_char[1]	Report recommendations for this path
4 -0.545	txt_out_char[3]~reg0	txt_out_char[3]	Report recommendations for this path
5 -0.461	txt_out_char[0]~reg0	txt_out_char[0]	Report recommendations for this path


We then performed the virtual pin assignment with the following settings:

Assignment Editor									
Filter on node names: * Category: All									
tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag	
1 ✓		txt...har	Virtual Pin	On	Yes	aes_...pher			
2 ✓		sim...key	Virtual Pin	On	Yes	aes_...pher			
3 ✓		do...dy	Virtual Pin	On	Yes	aes_...pher			
4 ✓		rst_n	Virtual Pin	On	Yes	aes_...pher			
5 ✓		din...lid	Virtual Pin	On	Yes	aes_...pher			
6 ✓		txt...har	Virtual Pin	On	Yes	aes_...pher			
7	<<new>>	<<new>>	<<new>>						

By re-running analysis & synthesis task, the fitter task and the STA analysis we then ended up with those final results:


Slow model

Slow 1100mV 85C Model Fmax Summary

 <<Filter>>


	Fmax	Restricted Fmax	Clock Name	Note
1	131.18 MHz	131.18 MHz	clk	

Slow 1100mV 85C Model Setup Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	2.377	0.000

Slow 1100mV 85C Model Hold Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.462	0.000

Slow 1100mV 0C Model Fmax Summary

 <<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	130.29 MHz	130.29 MHz	clk	

Slow 1100mV 0C Model Setup Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	2.325	0.000

Slow 1100mV 0C Model Hold Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.139	0.000

Fast model

Fast 1100mV 85C Model Setup Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	5.132	0.000

Fast 1100mV 85C Model Hold Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.181	0.000

Fast 1100mV 0C Model Setup Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	5.304	0.000

Fast 1100mV 0C Model Hold Summary

 <<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.172	0.000

Finally, no failing setup paths were found, and the worst-case slack is 2.377ns. The slack basically indicates the margin by which a timing requirement is met or not met. In the specific the slack is the difference between the required time and the arrival time in each path. In this case since it is positive it means that there is not a path which is too slow, instead the worst one is fast enough to meet all timing requirements.

Timing Closure Recommendations

Summary

This design does not contain any failing setup paths. The worst-case slack is 2.377 ns.

Top Failing Paths

No paths fail setup timing.

Last thing we did was to generate the code to program the FPGA device. This operation was done just to make sure that there were no errors during this task, since we did not physically have a Cyclone V FPGA device at our disposal.

The screenshot shows the Quartus Programmer window with the 'Assembler Summary' tab selected. The window title is 'Programmer - C:/Users/leona/Desktop/AES_Sbox_based_Stream_Cipher/quartus/aes_sbox_stream_cipher - a...'. The 'Hardware Setup...' button is set to 'No Hardware' and the 'Mode' is 'JTAG'. The 'Enable real-time ISP to allow background programming when available' checkbox is unchecked. The 'Tasks' pane on the left shows the following tasks: 'Compile Design', 'Analysis & Synthesis', 'Fitter (Place & Route)', 'Assembler (Generate programming file)', 'Timing Analysis', and 'EDA Netlist Writer'. The 'Assembler (Generate programming file)' task is selected and highlighted with a red box. The 'Assembler Summary' table shows the following data:

File	Device	Checksum	Usercode	Program	Verify	Blank	Examine	Security
output_files/aes_sbox_stream_cipher.sof	5CGXFC9D...	01D7C...	01D7C...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Below the table, there is a diagram of the FPGA device '5CGXFC9D6F27' with 'TDI' and 'TDO' pins. The 'TDI' pin is connected to the 'TDO' pin.

18236 Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment