Configurable Design and Implementation of the Rijndael Algorithm-AES

Afşin Özpınar
Electrical and Electronics Engineering
Boğaziçi University, Turkey
afsin.ozpinar@superonline.com

Arda Yurdakul
Computer Engineering
Boğaziçi University, Turkey
yurdakul@boun.edu.tr

The purpose of this work is to design configurable and flexible cores of Rijndael-AES algorithm. Rijndael is a cryptographic algorithm, which is accepted as Advanced Encryption Standard by NIST to achieve security and privacy while processing, transferring and storing the data [1-2].

The Rijndael algorithm mainly consists of a symmetric block cipher that can process data blocks of 128, 192 or 256 bits by using key lengths of 128, 196 and 256 bits. The AES accepts only 128 bits data size and 128(AES-128), 192(AES-192) and 256(AES-256) bits key lengths. The Rijndael algorithm is based on round function, and different combinations of the algorithm are structured by repeating this round function different times. Each round function contains uniform and parallel 4 steps, Byte Substitution, Row Shifting, Column Mixing and Key Addition, and each step has its own particular functionality.

In this work, design and implementation of three configurable and flexible cores of Rijndael algorithm are done: an encryptor, a decryptor and a combined encryptor and decryptor. These cores support not only the AES, but also the whole Rijndael algorithm. Another feature of the cores is that they are all designed as using Electronic Code Book (ECB) mode meaning that every single data block is encrypted and decrypted independently from each other. Since ECB is the basic element of all other main modes such as Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB)[3], it is easy to extend the design and implement the other modes.

All the modules in these flexible cores are created by using VHDL language. Some modules are designed by using behavioral style and some are designed fully RTL.

Firstly a common package is created to set the global values those are used by all of the modules and sub modules (Figure 1).

By changing the parameters NBlock, Nkey and Nround in this package, all the modules and sub modules can be reconfigured. After reconfiguration of these parameters, the core should be re-synthesized.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
package RIJNDAEL_TYPES is
  CONSTANT Nblock  : integer :=4; --# of columns for data 4,6,8
  CONSTANT Nkey    : integer  :=4; --# of columns for key(4(128),6(192),8(256))
  CONSTANT Nround : integer  :=10;--# of times to repeat round function 10,12,14
  CONSTANT ExpansionSize :integer := Nblock*(Nround+1); --# of columns for key
expansion table
  TYPE STATE          IS array (0 to ((4*Nblock)-1)) of std_logic_vector(7 downto 0) ;
  TYPE ADDR_STATE   IS array (0 to ((4*Nblock)-1)) of std_logic_vector(8 downto 0) ;
  TYPE WORD             IS array (0 to 3) of std_logic_vector(7 downto 0);
  TYPE KEY               IS array (0 to NKey-1) of WORD ;
  TYPE KEY_BLOCK      IS array (0 to NBlock-1) of WORD ;
  TYPE EXPANDED_KEY IS array (0 to (ExpansionSize-1)) of WORD;
  TYPE T_SBOX           IS array (0 to 255) of std_logic_vector(7 downto 0);
  TYPE T_RCON           IS array (1 to 29) of WORD;
end RIJNDAEL_TYPES;
```

Figure 1. Package Rijndael types

Secondly, sub-modules Byte Substitutor, Row Shifter, Column Mixer, Round Key Adder and Key Expander for the encryptor, inverse of all these modules, except Round Key Adder since it is identical for the both, for the decryptor are designed to implement the round function. All sub modules are designed by using behavioral VHDL.

Thirdly, a controller to control these sub modules and round operations, and a top-level to instantiate all these sub modules are designed for the encryptor and for the decryptor by using RTL style. The top-level of the designs can be seen in Figure 2 and Figure 3.

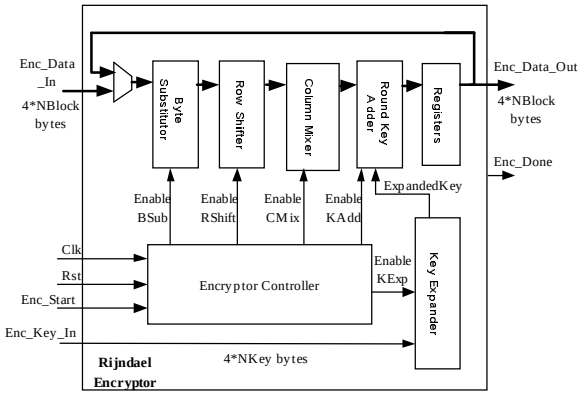

Figure 2. Toplevel of Rijndael encryptor

Simulations for all combinations are run by usingModelsim simulator. After verification of the functionality, all these combinations are synthesized into

a Xilinx FPGA by using the XILINX-ISE, an integrated development tool. Synthesis for some block size and key combinations for different target devices is done without using any constraints. These sample results can be seen in Table 1. Here, f_{max} stands for the maximum operation frequency of the device generated from our core.

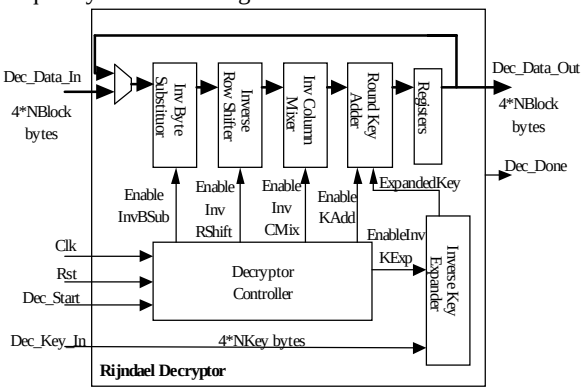

Figure 3. Toplevel of Rijndael decryptor

Table 1. Synthesis results for Rijndael encryptor-decryptor

Design	(Nb,Nk)	Number of Slices	Number of LUTs	f_{max}
Encoder on Xilinx-Virtex-XCV300	**(4,4)**	10921	19724	29.338
	(4,6)	10912	19507	23.646
	(4,8)	13030	23523	23 646
Encoder on Xilinx-Virtex-XCV1000	**(6,6)**	19698	37039	47.077
	(8,8)	32382	58479	42.278
	(8,4)	35320	66587	44.395
Decoder on Xilinx-Virtex-XCV1000	**(4,4)**	13542	25869	39.979
	(4,8)	13625	26058	43.096
	(6,6)	20807	39354	37.096
	(8,4)	33125	63667	41.387
	(8,8)	32743	62928	41.239

Usually encryptor and decryptor units must coexist in the same system. The above designs are quite expensive if we try to put them in one chip. If concurrency in encryption and decryption is not so important, then one can use the combined encryptor-decryptor core in order to have a smaller design. Since encryptor and decryptor modules use some common sub modules, such as most of key expander and round key adder, a combined encryptor-decryptor module is designed and some resources are shared by these modules. Besides, the designs shown in can be targeted to any ASIC or FPGA device. However, FPGAs might also contain ROM and RAM [4]. So for these devices, one can make use of these internal storage units to have more compact designs. Therefore, these built-in storage units of FPGAs are used for some operations in the Rijndael algorithm such as Byte Substitution, which is being done by using look-up tables (Figure 4). Our system generates the required memory space by using the initial key and block size determined by the user, but target device should have at least 8 Kbytes block memory. The simulations for all block and key size combinations validate the functionality of this flexible core as well. Some of the synthesis results are presented in Table 2.

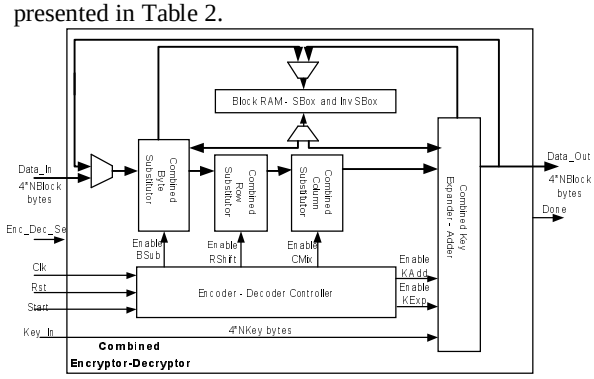

Figure 4. Toplevel of Rijndael combined encryptor-decryptor

These designs are done to give us the initial information about the performance and area metrics of the designs generated by using the flexible core developed in this study. The study still continues for improving the quality of the core in such a way that reconfigurable designs will be generated for the user-defined area, power and performance constraints.

Table 2. Synthesis results for Rijndael combined encryptor-decryptor

Design	(Nb,Nk)	Number of Slices	Number of LUTs	f_{max}
Encoder-Decoder on Xilinx-Virtex-XCV300	**(4,4)**	7046	12163	23.956
	(6,6)	15789	27392	16.689
	(8,8)	23638	45065	20.962
Encoder-Decoder on Xilinx-Virtex-XCV1000	**(6,6)**	19698	37039	47.077
	(8,8)	32382	58479	42.278
	(8,4)	35320	66587	44.395

Acknowledgements

This study is supported through Scientific Research Projects Program of Boğaziçi University and Xilinx University Program.

References

[1] http://csrc.nist.gov/CryptoToolkit/aes/rijndael, NIST's website

[2] http://www.esat.kuleuven.ac.be/~rijmen/rijndael, Rijndael developers' website.

[3] Federal Information Processing Standards Publication 81, DES Modes of Operation, NIST, 1980.

[4] http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp, Xilinx website.