VHDL Implementation of Fully Pipelined Advanced Encryption Standard

Abdullah Al Mamun
Student Id- 201403680

Outlines

Abstraction

- Before sending the information through an electronic medium, we should be aware of data ***CIA***.
- According to the NIST, AES is the latest secure encryption algorithm that is ***invulnerable***.
- However, data size is growing up dramatically, thus we need ***high throughput*** using pipelining.
- AES can be implemented in both software and hardware. As compared to software implementation hardware implementation of AES has an advantage of increased throughput and more security.
- In this research, VHDL based implementation of 128 AES using fully pipelined architecture is proposed.

Introduction

- A cryptographic algorithm proposed by ***Joan Daemen*** of Proton World International and ***Vincent Fijmen*** of Katholieke University at Leuven in Oct 2000 in the security contest and NIST announced that the Rijndael Algorithm was the winner of the contest.
- The Rijndael algorithm is a ***symmetric block cipher*** that can process data blocks of 128 bits through the use of cipher keys with lengths of 128, 192, and 256 bits.
- Early implementation was software based; researchers were started hardware implementation few years ago.

AES Algorithm

- Live animation http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf

Problem Statement

- In general, a FPGA based device is fully dedicated to run/process a single operation.
- Plaintexts are very large in amount MB,GB,TB
- Unfortunately, AES can encrypt 128 bit block at a time.
- But, we need high throughput (output data volume)
- How?

Proposed Solution

- AES used simple mathematics but repetitive algorithm.
- ***Pipeline circuit = High throughput***
- Also, Partially parallel to reduce the latency.

AES Background

- AES is a combination of Galois Field GF (2^8) polynomial addition and multiplication.
- Addition is ***XOR***

e.g.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \quad \text{(Polynomial notation)}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \quad \text{(Binary notation)}$$

$$\{57\} \oplus \{83\} = \{d4\} \quad \text{(Hexadecimal notation)}$$

Multiplication

For example, $\{57\} \bullet \{13\} = \{fe\}$ because

$$\{57\} \bullet \{02\} = \text{xtime}(\{57\}) = \{ae\}$$

$$\{57\} \bullet \{04\} = \text{xtime}(\{ae\}) = \{47\}$$

$$\{57\} \bullet \{08\} = \text{xtime}(\{47\}) = \{8e\}$$

$$\{57\} \bullet \{10\} = \text{xtime}(\{8e\}) = \{07\},$$

Thus,

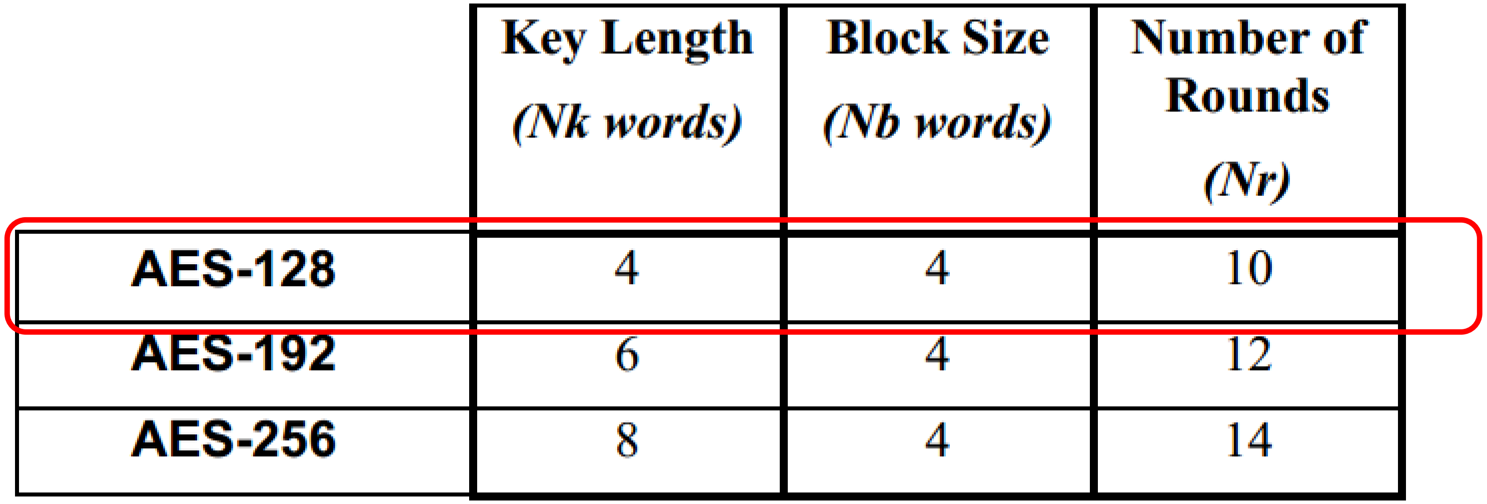
$$\begin{aligned} \{57\} \bullet \{13\} &= \{57\} \bullet (\{01\} \bullet \{02\} \bullet \{10\}) \\ &= \{57\} \bullet \{ae\} \bullet \{07\} \\ &= \{fe\}. \end{aligned}$$

Algorithm Specification

Plaintext: Raw data is going to be encrypted (128 bits/block)

Cipher: Encrypted data (128 bits/block)

Key = 128 bits



	Key Length *(Nk words)*	**Block Size** *(Nb words)*	**Number of Rounds** *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Target Device & Tools

- Target device: Xilinx Spartan6 XC6LX16-CS324

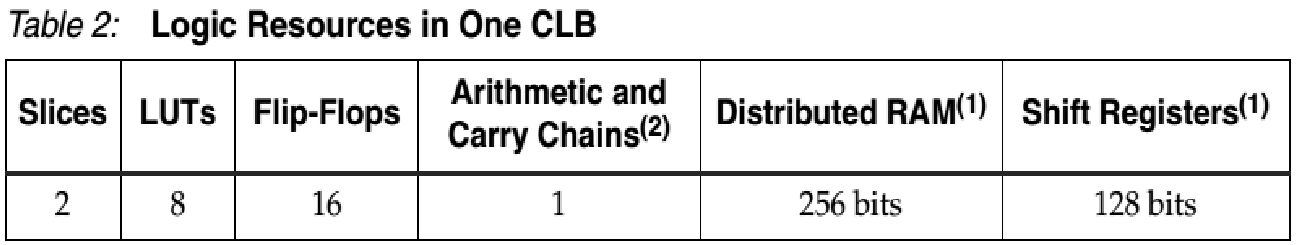


Table 2: **Logic Resources in One CLB**

Slices	LUTs	Flip-Flops	Arithmetic and Carry Chains[2]	Distributed RAM[1]	Shift Registers[1]
2	8	16	1	256 bits	128 bits

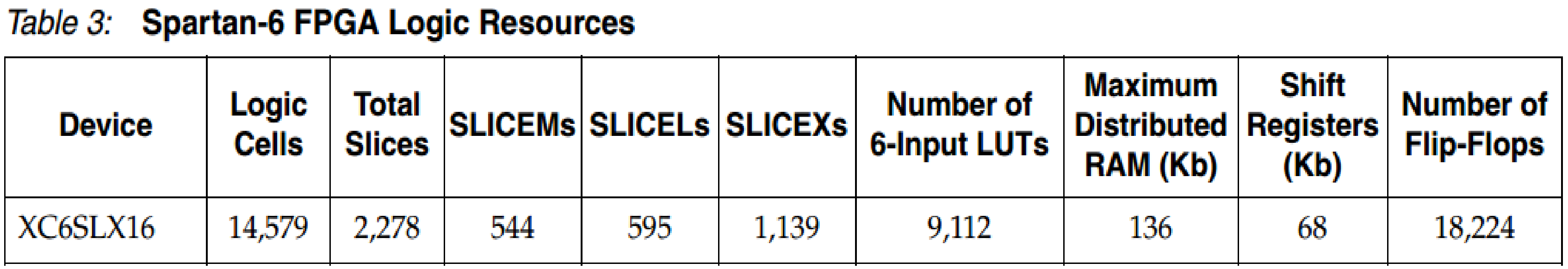
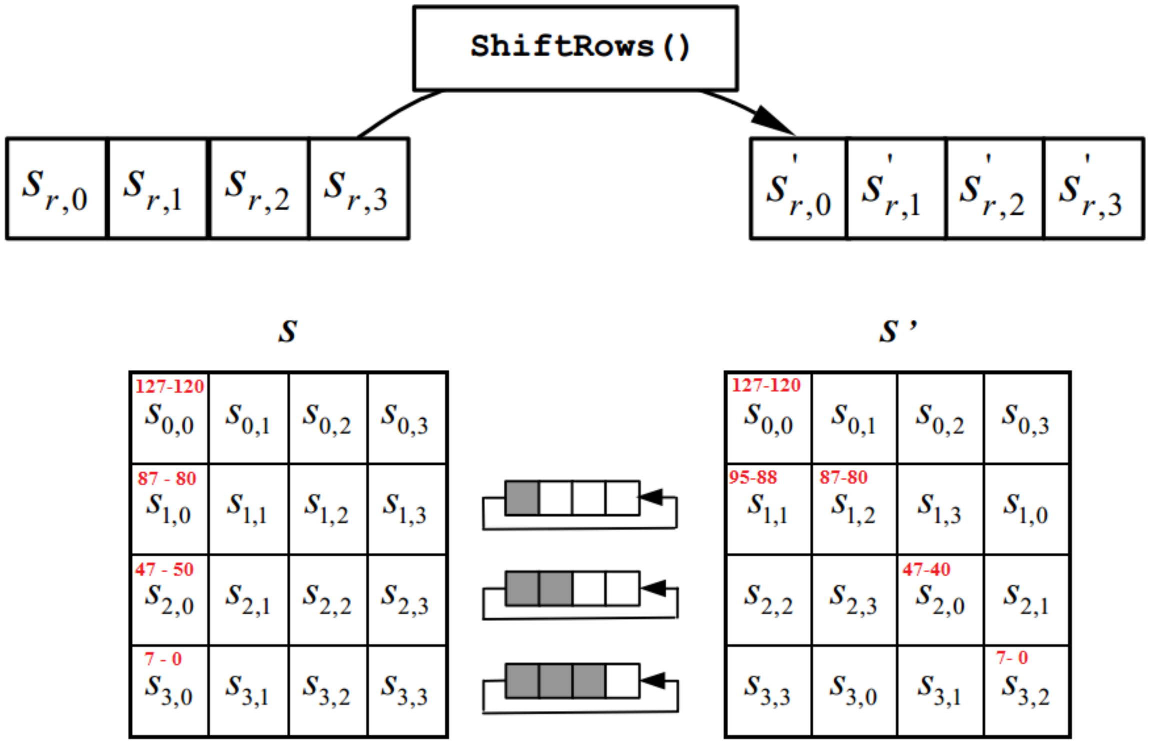


Table 3: **Spartan-6 FPGA Logic Resources**

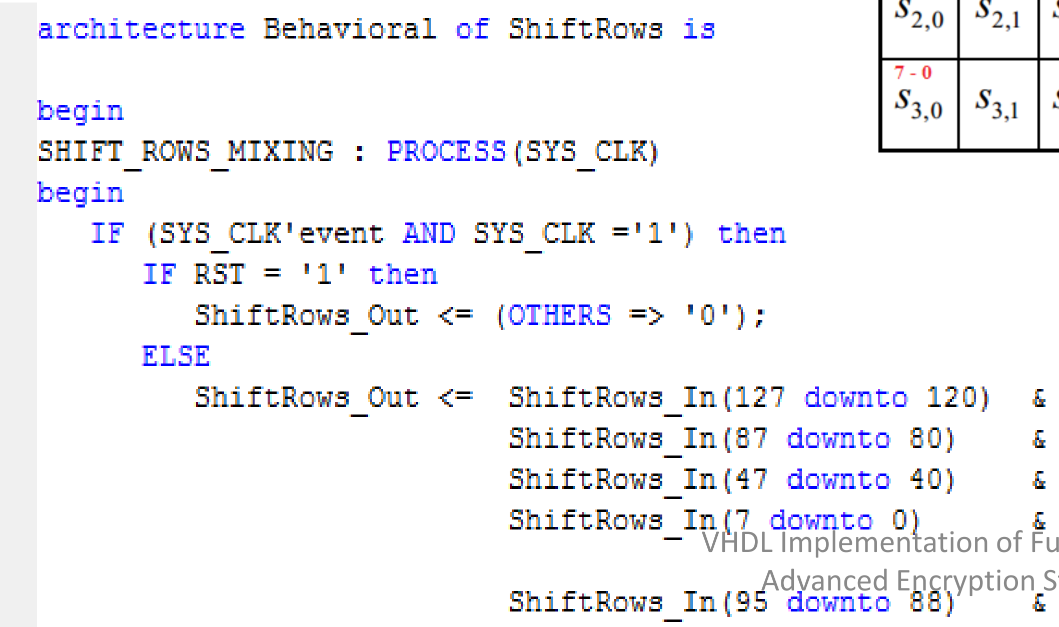
Device	Logic Cells	Total Slices	SLICEMs	SLICELs	SLICEXs	Number of 6-Input LUTs	Maximum Distributed RAM (Kb)	Shift Registers (Kb)	Number of Flip-Flops
XC6SLX16	14,579	2,278	544	595	1,139	9,112	136	68	18,224

ShifRow Operation


ShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
s
s'
127-120
87 - 80
47 - 50
7 - 0
127-120
95-88
87-80
47-40
7- 0

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

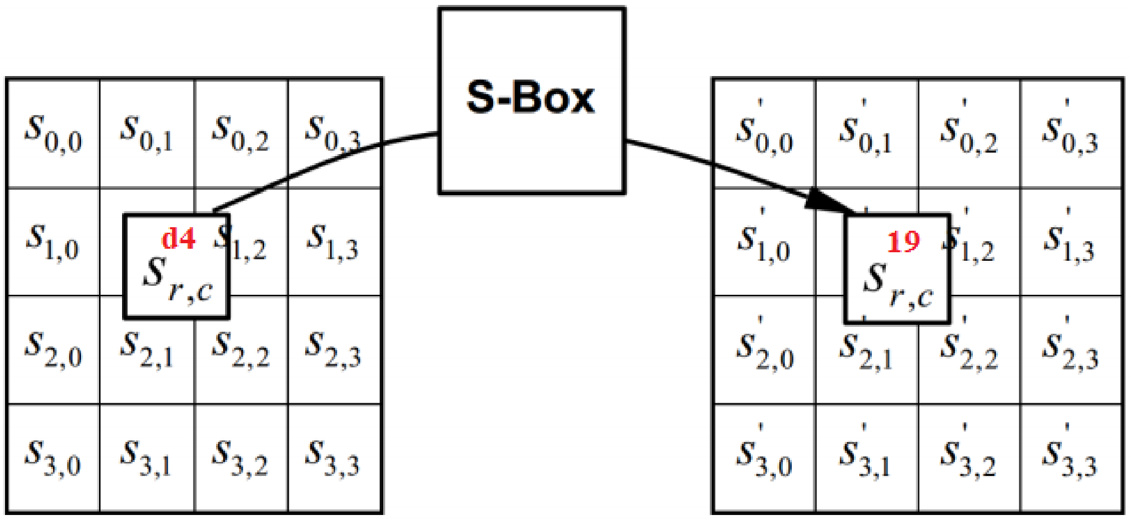


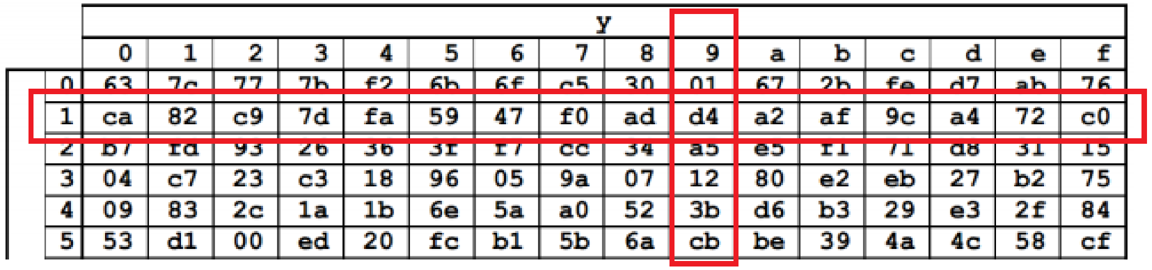
```
architecture Behavioral of ShiftRows is

begin
SHIFT_ROWS_MIXING : PROCESS(SYS_CLK)
begin
   IF (SYS_CLK'event AND SYS_CLK ='1') then
      IF RST = '1' then
         ShiftRows_Out <= (OTHERS => '0');
      ELSE
         ShiftRows_Out <=  ShiftRows_In(127 downto 120)  &
                           ShiftRows_In(87 downto 80)    &
                           ShiftRows_In(47 downto 40)    &
                           ShiftRows_In(7 downto 0)      &

                           ShiftRows_In(95 downto 88)    &
```

SubBytes





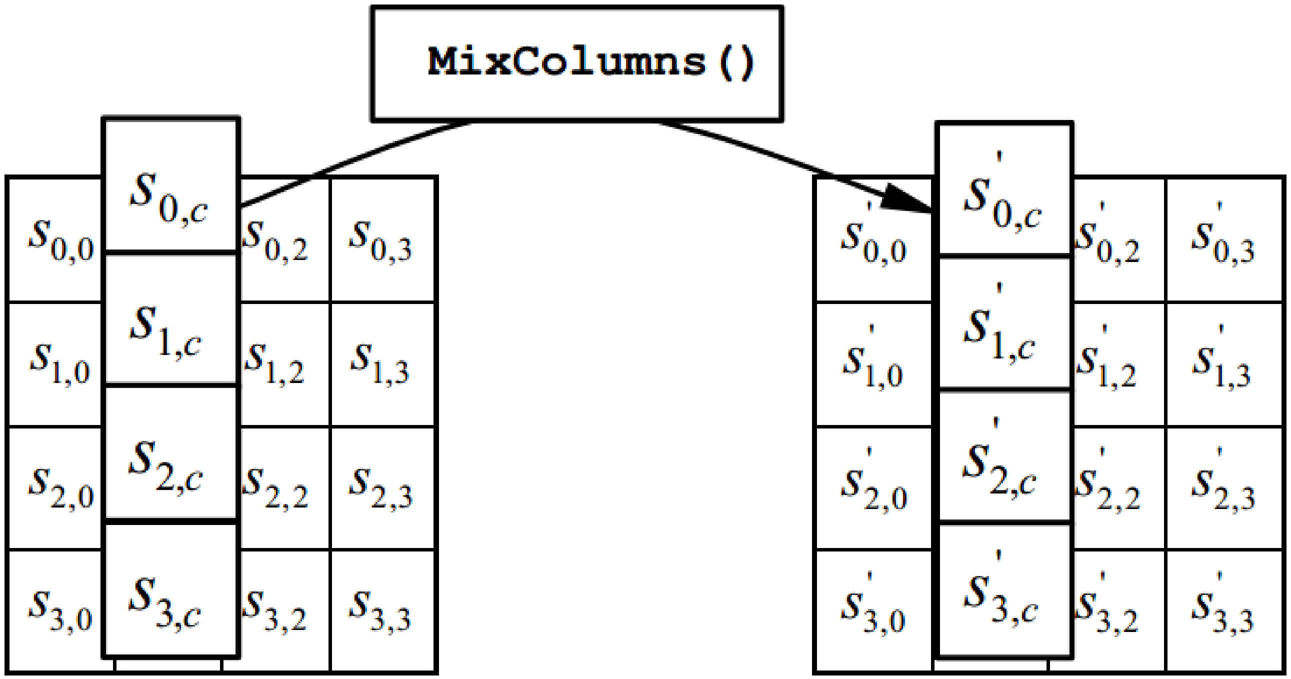
		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf

```
SUB_ARRAY :  For i in 0 to 15 generate
    begin
       SUB_BYTES_BUF((((i+1)*8)-1) downto(i*8)) <= SBOX(conv_integer(SubBytes_IN((((i+1)*8)-1) downto(i*8))));
    end generate;
```

Why ShiftRows before SubBytes?

- Since, goal is high throughput that can achieve by using instruction pipelining technology.
- There are ***no dependencies*** found between SubByte and ShiftRows operations that indicates one can be done before another.
- All operations of AES algorithm are ***column based except the ShiftRows***. If the ShiftRows is done before SubByte, the position of plaintext will be fixed column wise and remain operations can be pipelined easily.

MixColumn Operation


MixColumns()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}.$$

MixColumn Example

For example,

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} d4 \\ bF \\ 5d \\ 30 \end{bmatrix} = d4 \bullet 02 \oplus bf \bullet 03 \oplus 5d \oplus 30$$

$$d4 \bullet 02 = \mathrm{xtime(d4)} = \mathrm{b3}$$

$$\mathrm{bf} \bullet 02 = \mathrm{xtime(bf)} = 65$$

$$\mathrm{bf} \bullet 03 = \mathrm{bf}(01 \oplus 02) = \mathrm{bf} \oplus 65 = \mathrm{ba}$$

Thus, $d4 \bullet 02 \oplus bf \bullet 03 \oplus 5d \oplus 30 = 04$

Parallelism in MiXColumn

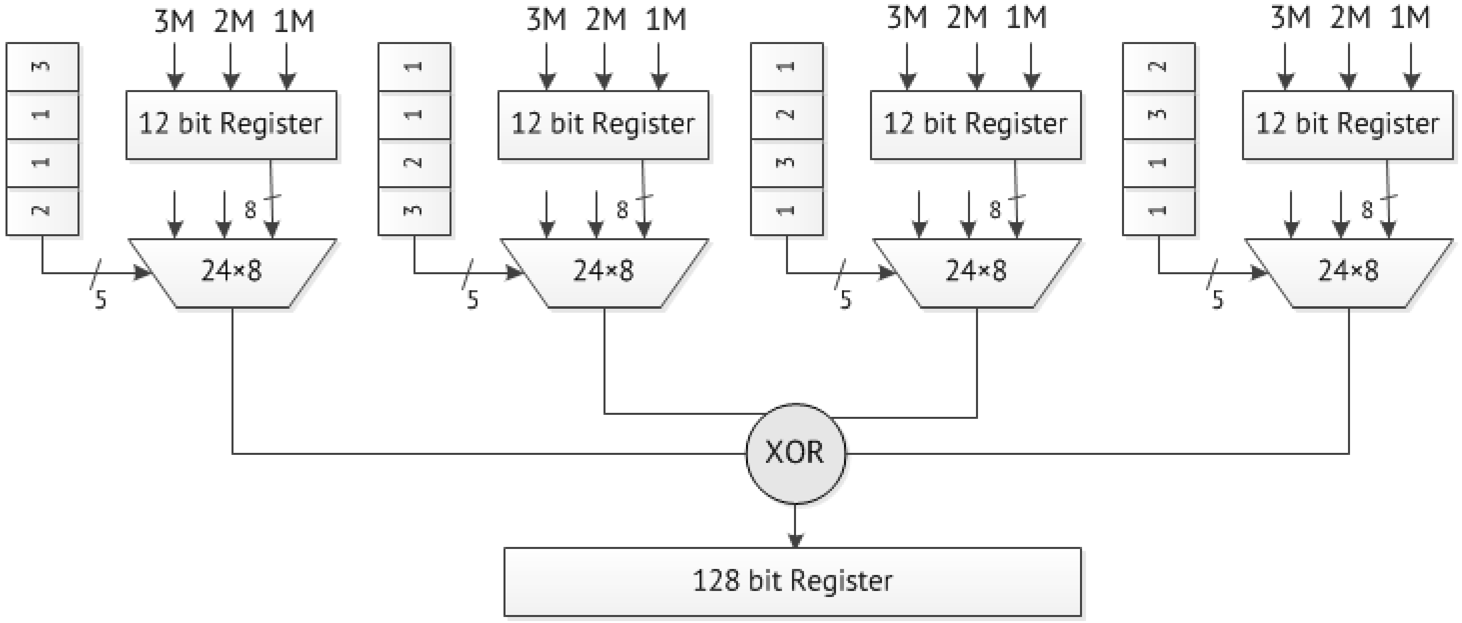


Fig: High latency low throughput

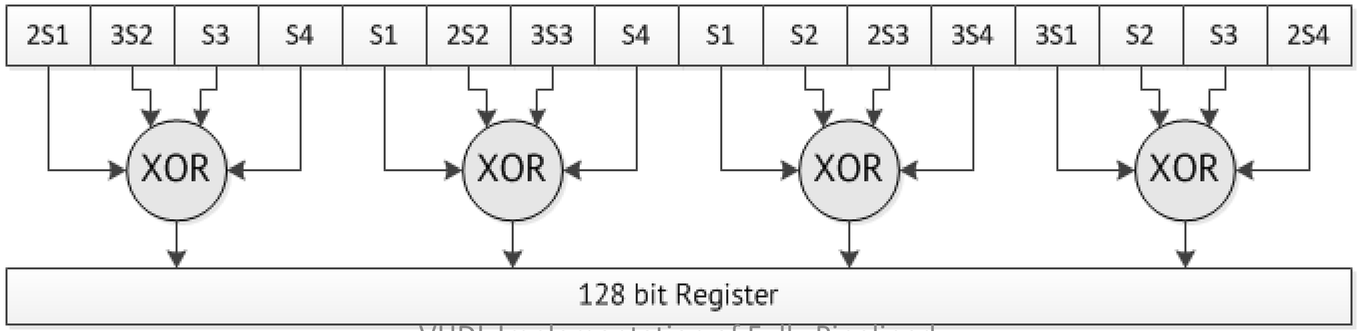
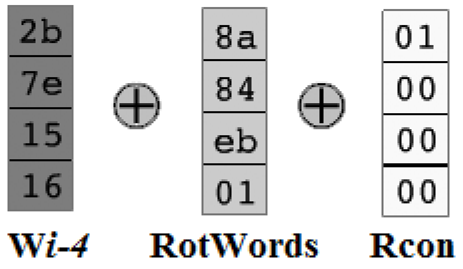
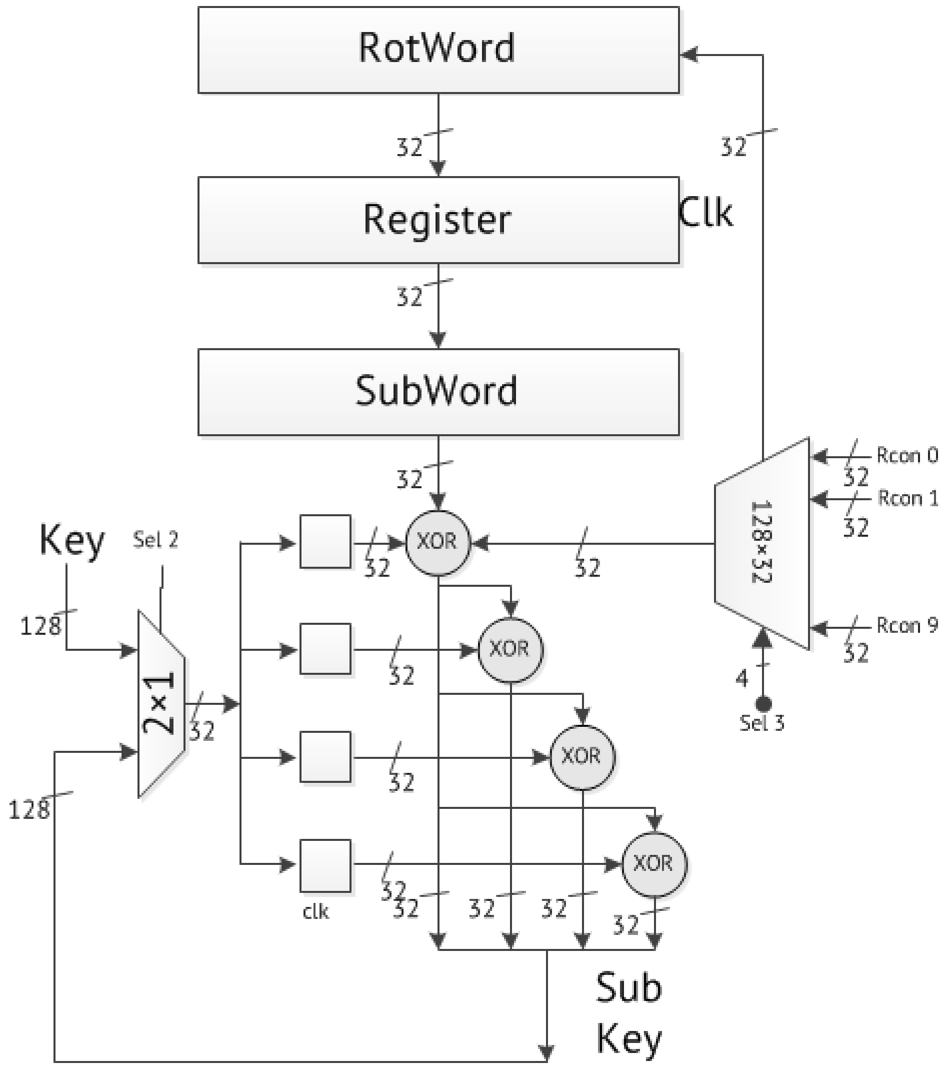


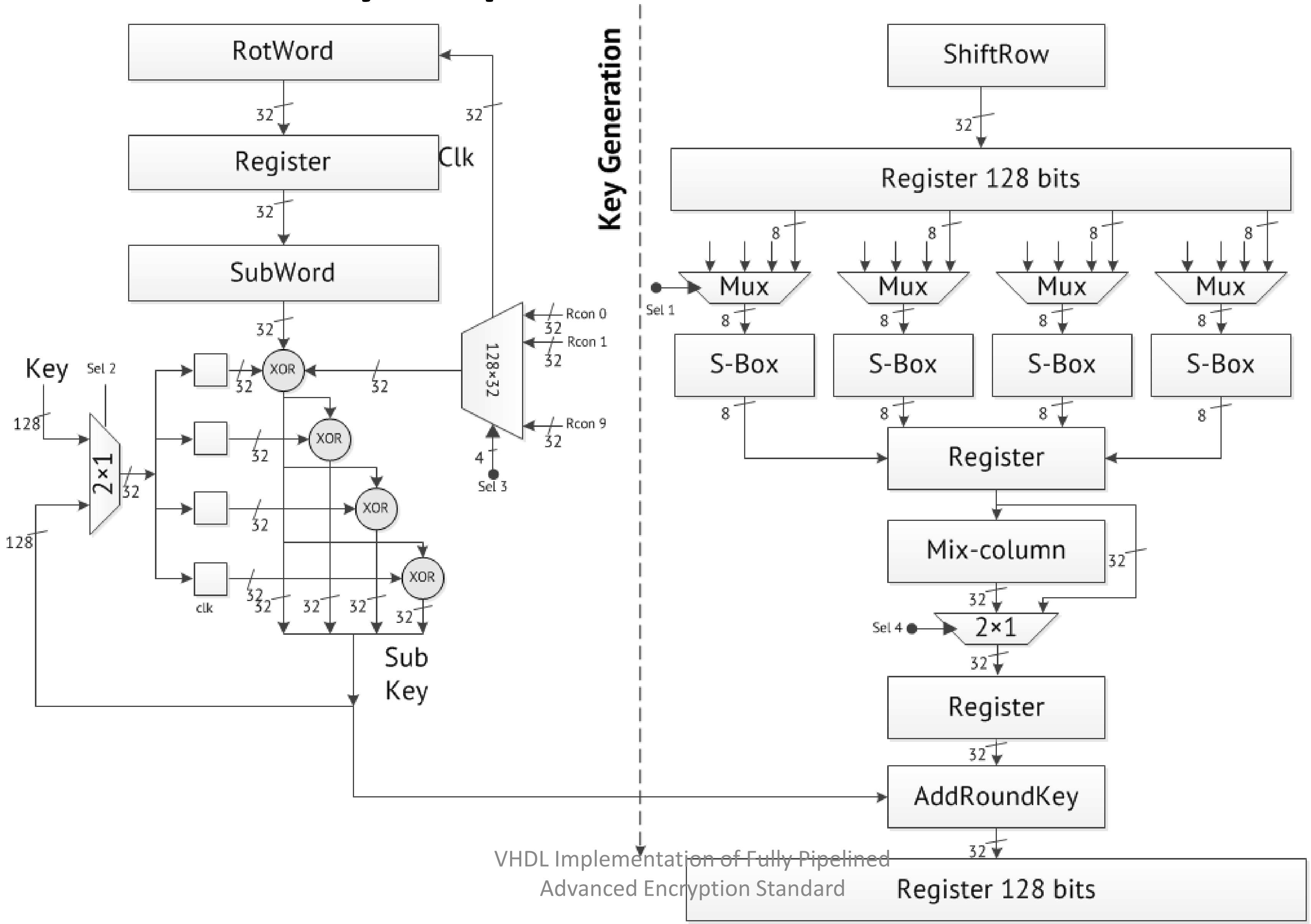
Fig: Low latency High throughput

KeyScheduler

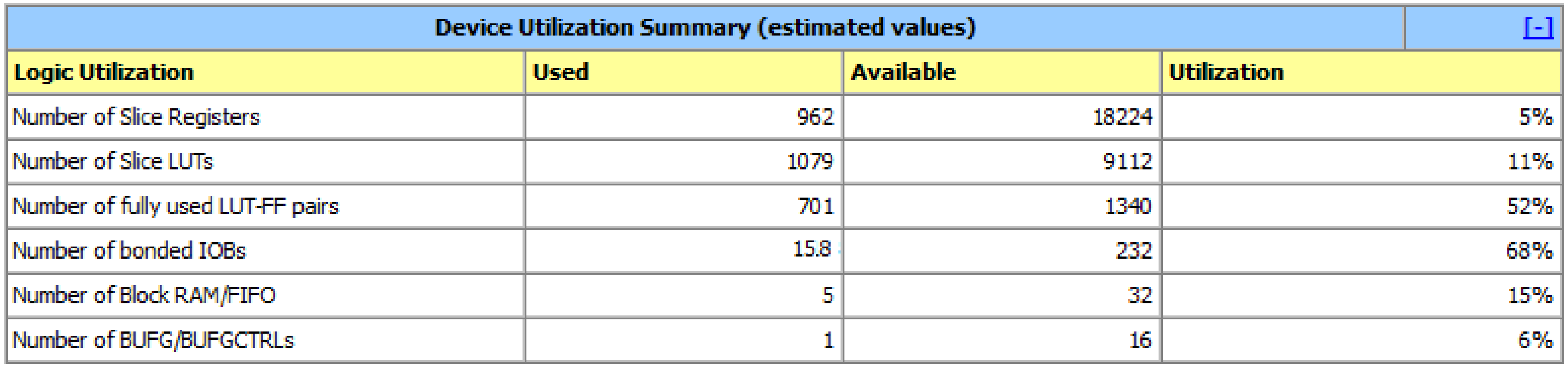




Fully Pipelined AES Circuit



Result: Resource Utalization



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	962	18224	5%
Number of Slice LUTs	1079	9112	11%
Number of fully used LUT-FF pairs	701	1340	52%
Number of bonded IOBs	158	232	68%
Number of Block RAM/FIFO	5	32	15%
Number of BUFG/BUFGCTRLs	1	16	6%

Result: Comparison

Design	FPGA	Frequency (MHz)	Throughput (Mbits)	Latency (ns)
Hodjat et al [3]	Virtex-II Pro XC2VP20	169.1	21,640	420
Zambreno et al [8]	Virtex-II XC2V4000	184.1	23,570	163
Zhang et al [2]	Virtex-E XCV1000E-8	168.4	21,556	416
Benaissa et al []	Virtex-E XCV2000E-8	184.8	23,654	379
This work	Spartan6 XC6LX16-CS324	185.2	22,832	407

Related Work

Conclusion and Future Plan

References

Thank you
?