International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

Implementation and Validation of Skien Cryptographic Hash Function Using High Speed Reversible Adders in Verilog Hdl

Sravya Degala & Dandamudi.Vijendra Kumar
[1] (M.Tech) [2] (M.tech), assistant.professor
[1] Email-id: degala.sravya@gmail.com , GIET an autonomous institute, Rajamundry, East godavari
[2] Email-id: Vijendra.dandamudi@gmail.com , GIET an autonomous institute, Rajamundry, East godavari

Abstract: *Security has become a curial aspect in the design and use of computer system and network. Hash functions are used for many applications in cryptography mainly in digital signatures and message authentication code and in network security. Hash functions play a significant role in today's cryptographic applications. SHA (Secure Hash Algorithm) is a famous message compress standard used in computer cryptography, it can compress a long message to become a short message abstract. In this paper, SHA is implemented using Verilog HDL. In this paper encryption and decryption of the of the cipher text is implemented using SHA algorithm, where different types of adders are used in the process of encryption and compared. The proposed work deals with the construction of high speed adder circuits. Design and modeling of various adders like Ripple Carry Adder, Kogge Stone Adder, and Brent Kung Adder is done by using CMOS and GDI logic and comparative analysis is coated.*

Keywords—*Ripple Carry Adder (RCA), Kogge Stone Adder (KSA), Brunt Kung Adder (BKA), Multiplier using adder, Gate count, number of transistors, Power, and Delay.*

I. INTRODUCTION

ADDERS are a key building block in arithmetic and logic units (ALUs) and hence increasing their speed and reducing their power/energy consumption strongly affect the speed and power consumption of processors. The variations increase uncertainties in the aforesaid performance parameters. In addition, the small sub threshold current causes a large delay for the circuits operating in the sub threshold region. Recently, the near-threshold region has been considered as a region that provides a more desirable tradeoff point between delay and power dissipation compared with that of the sub threshold one, because it results in lower delay compared with the sub threshold region and significantly lowers switching and leakage powers compared with the super threshold region. There are many adder families with different delays, power consumptions, and area usages. Examples include ripple carry adder (RCA), carry increment adder (CIA), carry skip adder (CSKA), carry select adder (CSLA), and parallel prefix adders (PPAs). In electronic, an adder is a digital circuit that performs addition of number. In many computers and other kind of processors, adders are used not only in the Arithmetic Logic unit (ALU), but also in other parts of the processors. In this paper a different set of adders like RCA, KSA, BKA are of the adders are tabulated. Addition is the most fundamental operation in any digital system. A simple adder performs the addition of given two numbers and the result is sum of those two numbers. Multiplication operation greatly depends on adder operation as it is one of the key hardware block in most digital signal processing system. Its main block is Arithmetic unit. The number of multiplication operation is performed by a series or parallel addition concept.

With the advances in Very Large Scale Integration (VLSI) technology, arithmetic operations are penetrating into more and more applications. The basic operation found in most arithmetic components is the binary addition and Multiplication. Computations needs to be performed using low-power, area-efficient circuits operating at greater speed. Addition is the most basic arithmetic operation; and adder is the most fundamental arithmetic component of the processor. In addition, each of the resulting output bits are depending on its corresponding inputs. It is very important operation because it involves a carry ripple step i.e the carry from the previous bits addition should propagates to next bits of addition.

Multiplication is an operation that occurs frequently in digital signal processing and many other applications. The present development in processor designs aim is design of low power multiplier. So, the need for low power multipliers has increased.

Generally the computational performance of DSP processors is affected by its multipliers performance.. The architecture of Adders like Ripple Carry Adder, Carry Look Ahead Adder, Kogge Stone Adder and Brent-Kung Adders have advantages with respect to power, area and complexity.

II. RELATED WORK

Cryptography is one of the most useful fields in the wireless communication area and personal communications yet e ms, where information security has become more and more important area of interest .Cryptographic algorithms take care of specific information on security requirements such as data integrity, confidentiality a nd data origin authentication. To assure that a communication is authentic, the authentication service is of much concern. The function of authentication services is to assure recipient that the message is from the source it claims .In computer security, the process of attempting to verify the digital identity of the sender of a piece of information is known as authentication.

In order to make a very secure cryptographic portable electronic device, the selected well-known algorithm must be trusted, time-tested and widely peer-reviewed in the global cryptographic community. A one-way hash function is an algorithm that takes input data and irreversibly creates a digest of that data. One of the trusted one-way hash function are SHA-1 (Secure Hash Algorithm),SHA-256, SHA-384 and SHA-512. SHA algorithms are called secure because, for a given algorithm, it is computationally infeasible 1) to find a message that corresponds to a given message digest or 2) to find two different messages that produce the same message digest. Any change to a message will, with a very high probability results in a different message digest. This will result in a verification failure when the secure hash algorithm is used with a digital signature algorithm or a keyed-hash message authentication algorithm. Overview of SHA- 1 is a complex algorithm that involves multiple 32-bit, 5-way additions, complex logical functions, data shifting and a great deal of repetition. Generally implementations of the SHA-1 algorithm have required large die areas and so made fairly expensive portable device. A proposed method has been applied to be relatively inexpensive one. The architecture is presented for SHA-1 hash function. The implementation is conducted using Verilog HDL on Xilinx FPGA device. The synthesis results are presented and compared with other SHA-1 implementations. Here, the hardware terms of system performance (throughput), operating frequency and covered area are compared. Hash algorithms, also called as message digest algorithms, are generating a unique fixed length bit vector for an arbitrary-length message M. The bit vector is called the hash of the message and it is denoted as H. This hash value should be the same each time the same input is hashed. A hash function used in cryptography should be one way and collision resistant. The purpose of a hash function is to produce a fingerprint of a file, message or other block of data.

III. PROPOSED SYSTEM

Skein

Any implementer of Skein has to choose which options to enable. The simplest implementations only implement straight hashing with a fixed output size. After that, the most useful options to support are probably:

• Variable output sizes (in byte increments) up to one block Longer outputs
• Key input for a MAC
• PRNG
• Personalization

We expect that the public-key field, key derivation, and tree hashing will be used less frequently. Skein defines output sizes of arbitrary bit length, but we recommend that implementations restrict themselves to whole bytes. There are specific uses for odd bit lengths (e.g., elliptic curves) and the odd bit length provides a symmetry with the arbitrary bit length of the inputs, but in practice, we rarely see arbitrary bit length values being used.

Hardware Implementations

Threefish:

Threefish is the tweakable block cipher at the core of skein, denied with a 256,52 and 1024-bit block size. The threefish block takes three inputs,

plaintext, userkey(256 bits each) and a tweak of 128 bits. The output of the function is ciphertext of 256 bits.The core of Threefish is the MIX function. In hardware, this is straightforward to implement. To achieve high performance it is important to use a fast-carry adder and not a ripple-carry adder. Ripple-carry adders are very slow in the worst case; the carry ripples from the least significant bit to the most significant bit, which limits the maximum clock frequency. There are well-known techniques for fast carry propagation in adders, and these should be used for speed-sensitive implementations. The rotations and word permutations do not require any gates, but they do take up routing space. The most natural way to implement Threefish is to either implement 8 rounds, or the full 72 or 80 rounds. An implementation that tries to implement only 1 or 4 rounds needs to accommodate different rotation constants in each MIX, leading to a number of multiplexers. The key schedule can be implemented in several ways. The simplest one is to store the extended key and extended tweak in two shift registers and clock the shift registers once for each subkey. Note that the final state of the shift registers can be directly computed, so implementations that want to perform decryption can efficiently generate the subkeys in reverse order.

UBI :

In hardware, UBI is implemented like any other block chaining mode. There are no special considerations, other than the need to buffer the last input block until it is known whether this is the last block of the message or not. The main components of threefish are the mix and permute function shown in fig 3.1.

Encryption

MIX functions:

The MIX function is an integral part of threefish ciphers. It takes two inputs of 64-bit each and performs three basic operations on the data words. Addition modulo 2^{64}, Airthmetic shift and XOR operation. The block diagram shown in fig 3.2. Threefish's MIX function is derived from Helix and Phelix. Initially, we had a more complex MIX function, with 2 adds, 2 XORs, and 4 rotations. The advantage of a more complex mixing function is that x86 CPUs, which have only 7 usable 32-bit registers, can load all of the function's inputs into registers and execute the entire MIX function without loads or stores. However, our cryptographic analysis showed that more rounds of a simpler mixing function are more secure, for a given number of CPU clock cycles. Another candidate design included a MIX function with 3 add/XOR operations and 2 rotations, but our performance measurements also showed that—contrary to what the chip's documentation suggests—the current generation of Intel CPUs can only perform one rotate operation per clock cycle. This limitation causes a significant speed penalty on x64 CPUs, so we abandoned it, in keeping with the principle that additional rounds more than make up for the simpler MIX function.

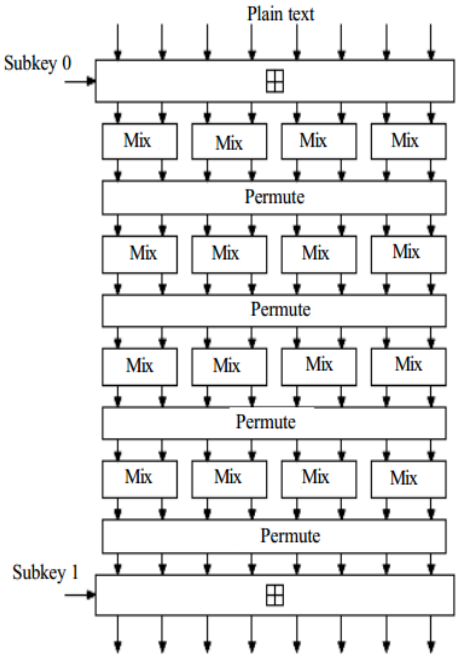


Fig 3.1:Threefish block for encryption

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

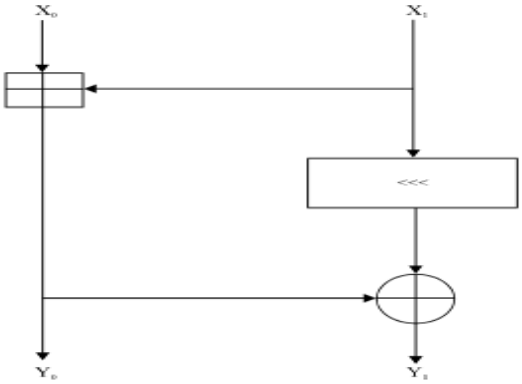


Fig 3.2:MIX Function

Permutation operations:

The permutation function used to diffuse the outputs obtained from the previous rounds of MIX function and scramble them to create new inputs for the following MIX round equation 2 show key scheduling algorithm.

$$sk_0 = k_0;\ sk_1 = k_1 + t_0$$
$$sk_2 = k_2 + t_1;\ sk_3 = k_3 >> \{2\}$$

Key scheduling Algorithm:

After every four round of MIX and permute, a subkey is added to current threefish state. For the first round, user key and plaintext are XORed and then after every four rounds, Key Scheduling algorithm uses tweak and the key of previous rounds to generate a subkey.

The key from the previous round is divided into four 64 bit words and the tweak is divided into two 64-bit words. Key scheduling algorithm makes use of modular addition 2^{64} and logical right shift of constant arbitrary value decided by the designer. The 464 bit words from these operations concatenate to form a new subkey shown in fig 3.3.

Thus, key schedule algorithm generates a new subkey every time. It can be thus concluded that due to the addition of a 256 bit subkey after every four rounds, a high level of security can be achieved by making the cryptosystem immune to cryptanalysis.

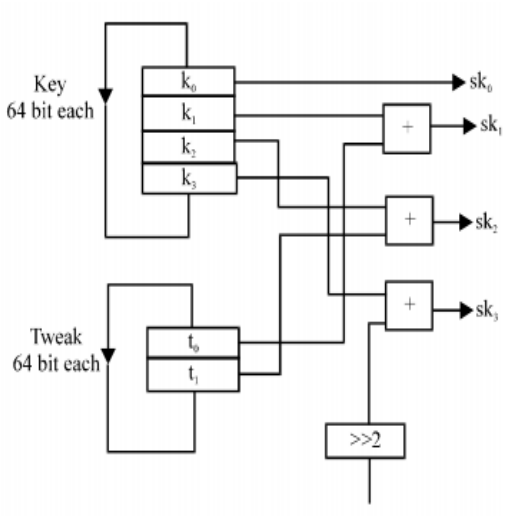


Fig 3.3:Key scheduling algorithm

Decryption:

Decryption is the exact inverse of the encryption algorithm. To decrypt the ciphertext generated from the threefish block cipher, the text is passed through the inverse threefish cipher which consists of the inverse mix function and the inverse permutation table. Also, the keys the supplied in reverse order, i.e, the 18^{th} subkey of encryption system is the 1^{st} subkey of the decryption algorithm and vice versa. The keys used for encryption are used for decryption as well, in reverse order.

Inverse mix function:

The inverse MIX function is an integral part of the inverse threefish cipher. It is created by using the inverse of the operators of the MIX function. It consists of three digital systems, XOR operator, left shifter and subtractor. The subtraction in this module is performed with the help of a carry look ahead adder. Figure 3.4 shows the block diagram of the inverse MIX function.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

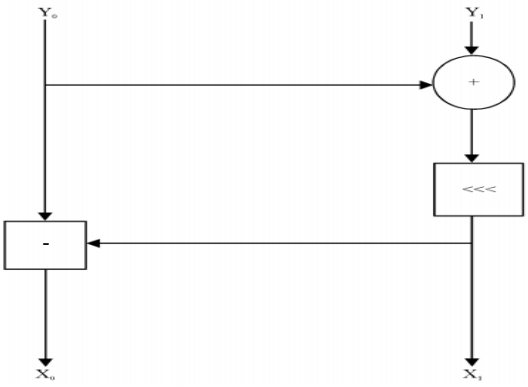


Fig 3.4: inverse MIX function

Inverse permutation operation:

The inverse permutation function is used to diffuse the outputs obtained from the previous rounds of inverse MIX function and scramble them to create new inputs for the following inverse MIX round.

IV. DESIGN OF ADDERS

RIPPLE CARRY ADDER:

The RCA block can be constructed by cascading full adder blocks in series. It is possible to create a logical circuit using multiple full adder to add N-bit numbers. Each full adder inputs a Cin, which is the Cout of the previous adder. This kind of adder is RCA, since each carry bit "ripples" to the next full adder. For an n bit adder it requires n 1 bit full adder. Drawbacks of Ripple Carry Adder: The RCA is relatively slow since each full adder must wait for the carry bit to be calculated from the previous full adder. The 8-bit ripple carry adder shown in fig 4.1.

Equation of Ripple Carry Adder .

The corresponding boolean expressions are given here to construct a ripple carry adder. In the half adder circuit the sum and carry bits are defined as

$$sum = A \oplus B$$
$$carry = AB$$

In the full adder circuit the Sum and Carry output is defined by inputs A, B and Carryin as

$$Sum=ABC + ABC + ABC + ABC$$
$$Carry=ABC + ABC + ABC + ABC$$

Having these we could design the circuit. But, we first check to see if there are any logically equivalent statements that would lead to a more structured equivalent circuit. With a little algebraic manipulation, one can see that

$$Sum= ABC + ABC + ABC + ABC$$
$$= (AB + AB) C + (AB + AB)$$
$$C = (A \oplus B) C + (A \oplus B) C$$
$$=A \oplus B \oplus C$$
$$Carry= ABC + ABC + ABC + ABC$$
$$= AB + (AB + AB) C = AB + (A \oplus B) C$$

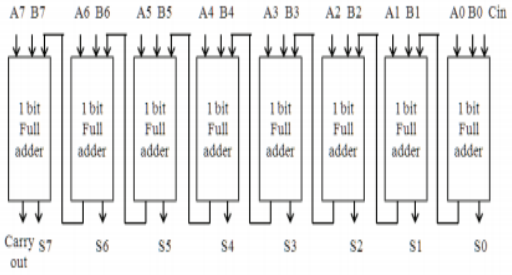


Figure 4.1: Block diagram of 8-bit Ripple Carry Adder

CARRY SAVE ADDER:

A carry-save adder is a type of digital adder, used in computer micro architecture to compute the sum of three or more *n*-bit numbers in binary. It differs from other digital adders in that it outputs two numbers of the same dimensions as the inputs, one which is a sequence of partial sum bits and another which is a sequence of carry bits. Consider the sum: 12345678+87654322=100000000.

Using basic arithmetic, we calculate right to left, "8+2=0, carry 1", "7+2+1=0, carry 1", "6+3+1=0, carry 1", and so on to the end of the sum. Although we know the last digit of the result at once, we cannot know the first digit until we have gone through every digit in the calculation, passing the carry from each digit to the one on its left. Thus adding two *n*-digit numbers has to take a time proportional to *n*, even if the machinery we are using would otherwise be capable of performing many calculations simultaneously.

In electronic terms, using bits (binary digits), this means that even if we have *n* one-bit adders at our disposal, we still have to allow a time proportional to *n* to allow a possible carry to propagate from one end of the number to the other. Until we have done this,

1. We do not know the result of the addition.
2. We do not know whether the result of the addition is larger or smaller than a given number (for instance, we do not know whether it is positive or negative).

A carry look-ahead adder can reduce the delay. In principle the delay can be reduced so that it is proportional to log*n*, but for large numbers this is no longer the case, because even when carry look-ahead is implemented, the distances that signals have to travel on the chip increase in proportion to *n*, and propagation delays increase at the same rate.

BRENT KUNG ADDER:

The type of structure of any adder greatly affects the speed of the circuit. The logarithmic structure is considered to be one of the fastest structures. The logarithmic concept is used to combine its operands in a tree- like fashion. The logarithmic delay is obtained by restructuring the look-ahead adder. The restructuring is dependent on the associative property, and the delay is obtained to be equal to (log2N) t, where 'N' is the number of input bits to the adder and t is the propagation delay time. Hence it is seen that this structure greatly reduces the delay, and would be especially beneficial for a structure with large number of inputs. In the following section, a structure known as the Brent Kung Structure, which was first proposed by Brent and Kung in 1982 and which uses the logarithmic concept, is discussed. This structure used an operator known as the dot (·) operator, which is explained in the architecture, for its basic blocks.

Brent Kung Architecture

In order to approach the structure known as the Brent Kung Structure, which uses the logarithmic concept, the entire architecture is easily understood by dividing the system into three separate stages:

1. Generate/Propagate Generation
2. The Dot (·) Operation
3. Sum generation

Generate/Propagate Generation

If the inputs to the adder are given by the signals A and B, then the generate and propagate signals are obtained according to the following equations.

G = A.B (4.1) P = AÅ. B (4.2)

The Dot (·) Operation = The most important building block in the Brent Kung Structure is the dot (·) operator. The basic inputs to this structure are the generate and propagate signals generated in the previous stage. The · operator is a function that takes in two sets of inputs-- (g, p) and (g', p')-- and generates a set of outputs-- (g + pg', pp').

These building blocks are used for the generation of the carry signals in the structure. For the generation of the carry signals, the carry for the kth bit from the carry look-ahead concept is given by

Co,k = Gk +Pk(Gk-1 +Pk-1 +P k-1 (…+P1(G0 +P0 Ci,0))) (4.3)

Using the dot operator explained above the Equation 4.3 can be written for the different carry signals as

Co,0 = G0 +P Ci,0 = a (G0,P0) Co,1 = G1 + G0 P1 = a ((G1 , P1)·(G0, P0)) …… C0,k = a ((Gk,Pk)·(Gk-1,Pk-1)·…·(G0,P0)) (4.4)

where a is a function defined in order to access all the tuples. All the carry signals generated at different stages in the structure. In the structure, two binary tree structure are represented -- the forward and the reverse trees. The forward binary tree alone is not sufficient for the generation of all the carry signals. It can only generate the signals shown as Co,0,Co,1, Co,3 and Co,7. The remaining carry signals are generated by the reverse binary tree.

Sum Generation.

The final stage in this architecture is the sum generation stage. The sum is given by

S= AÅ BÅ C (4.5)

where A and B are the input signals, and C is the carry signal. The carry is obtained from the dot

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

operator stage discussed earlier, and the exclusive of A and B is actually the propagate signal itself. Hence the sum 'S' can finally be represented and realized as

S = P Å C (4.6)

Brent Kung Parallel Prefix Adder has a low fan-out from each prefix cell but has a long critical path and is not capable of extremely high speed addition. In spite of that, this adder proposed as an optimized and regular design of a parallel adder that addresses the problems of connecting gates in a way to minimize chip area. Accordingly, it considered as one of the better tree adders for minimizing wiring tracks, fan out and gate count and used as a basis for many other networks.

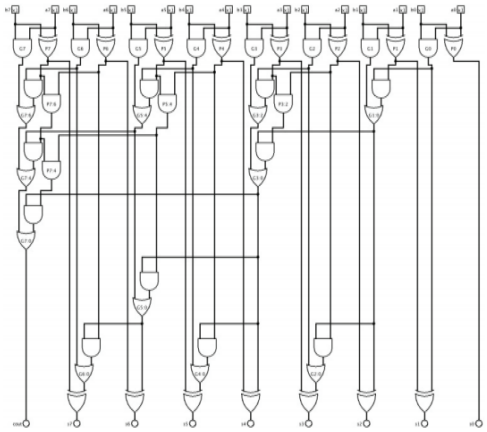
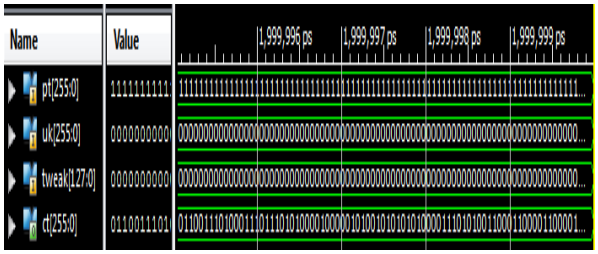


Figure 4.2: Schematic of 8-bit Brent Kung Adder

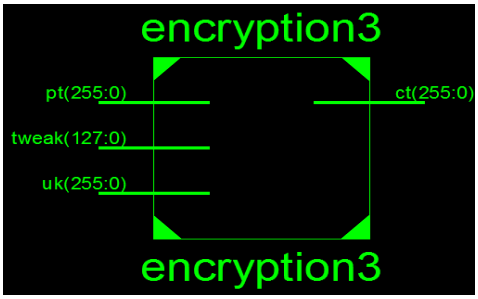
IV. EXPERIMENTAL RESULTS

Ripple Carry Adder Encryption :

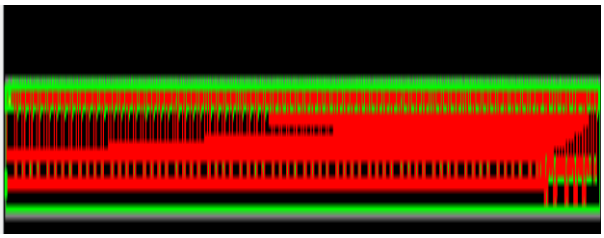
Simulation



Technology Schematic.



RTL Schematic.



Design Summary.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	25147	63400	39%
Number of fully used LUT-FF pairs	0	25147	0%
Number of bonded IOBs	896	210	426%

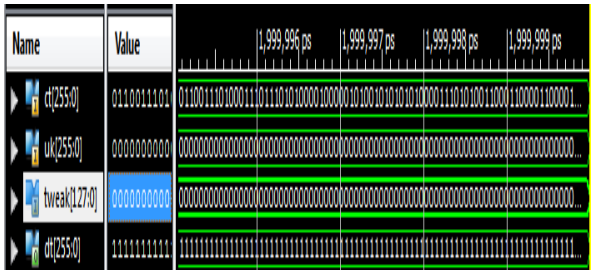
Timing Summary.

```
    LUT5:I4->O            5   0.097   0.314  m143/r11/r8/r3/FA4/Cout1 (m143/r11/r8/c3)
    LUT5:I4->O            3   0.097   0.703  m143/r11/r8/r4/FA2/Cout1 (m143/r11/r8/r4/transfe
    LUT6:I0->O            2   0.097   0.383  m143/r11/r8/r4/FA4/Mxor_Sum_xo<0> (f285<63>)
    LUT2:I0->O            1   0.097   0.279  m143/X1/Mxor_y1_63_xo<0>1 (f286<63>)
    OBUF:I->O                 0.000          ct_191_OBUF (ct<191>)
    ----------------------------------------
    Total                    1466.429ns (231.751ns logic, 1234.678ns route)
                                        (15.8% logic, 84.2% route)

=========================================================================

Cross Clock Domains Report:
--------------------------


=========================================================================


Total REAL time to Xst completion: 359.00 secs
Total CPU time to Xst completion: 358.78 secs

-->

Total memory usage is 1374252 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    2 (   0 filtered)
Number of infos    :    1 (   0 filtered)
```

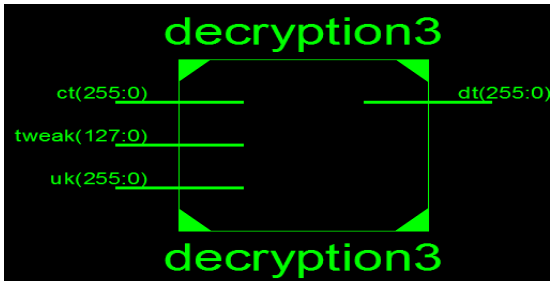
Ripple Carry Adder Decryption.

Simulation

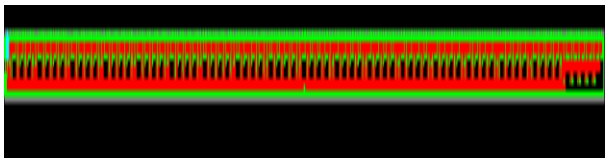
International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018



Technology Schematic.



RTL Schematic.



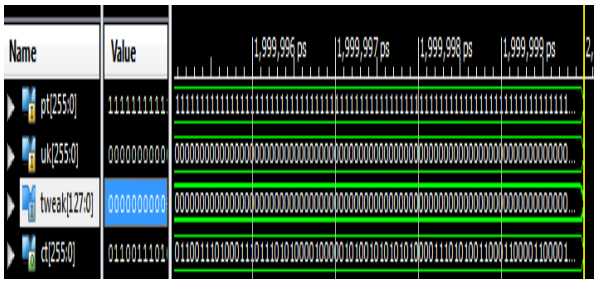
Design Summary.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	22386	63400	35%
Number of fully used LUT-FF pairs	0	22386	0%
Number of bonded IOBs	896	210	426%

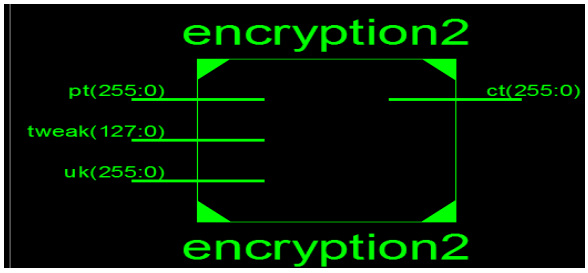
Timing Summary.

```
    LUT5:I1->O            3   0.097   0.693  im143/s64/s8/s3/c431 (im143/s64/s8/s3/c4_bd
    LUT6:I1->O            3   0.097   0.566  im143/s64/s8/s3/c411 (im143/s64/s8/k6)
    LUT6:I2->O            2   0.097   0.383  im143/s64/s8/s4/c431 (im143/s64/s8/s4/c4_bd
    LUT4:I2->O            1   0.097   0.279  iX19/Mxor_s_51_xo<0>1 (dt_51_OBUF)
    OBUF:I->O                 0.000          dt_51_OBUF (dt<51>)
    ----------------------------------------
    Total                     1501.598ns (204.004ns logic, 1297.594ns route)
                                         (13.6% logic, 86.4% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================


Total REAL time to Xst completion: 177.00 secs
Total CPU time to Xst completion: 176.29 secs

-->

Total memory usage is 737580 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    3 (   0 filtered)
Number of infos    :    2 (   0 filtered)
```

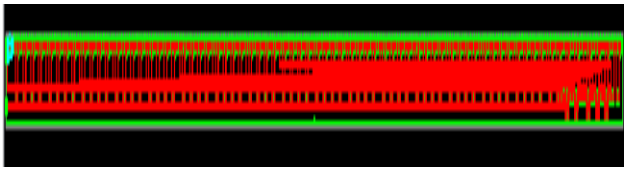
CSA Encryption Simulation



Technology Schematic.



RTL Schematic.



Design Summary.

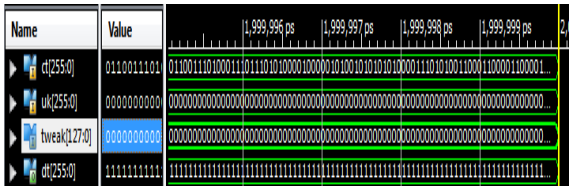
Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	33024	63400	52%
Number of fully used LUT-FF pairs	0	33024	0%
Number of bonded IOBs	896	210	426%

Timing Summary.

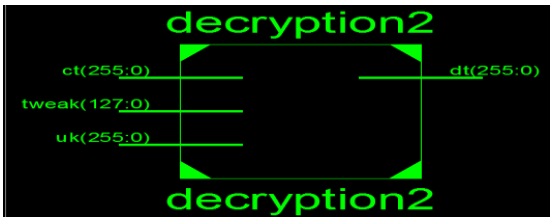
```
    LUT3:I2->O            3   0.097   0.305  m144/C11/C8/C3/m0/Mmux_out12 (m144/C11/C8/c3)
    LUT5:I4->O            3   0.097   0.693  m144/C11/C8/C4/rca1/fa1/cout1 (m144/C11/C8/C4/rc
    LUT6:I1->O            2   0.097   0.383  m144/C11/C8/C4/rca1/fa3/h2/Mxor_sum_xo<0>3 (f287
    LUT2:I0->O            1   0.097   0.279  m144/X1/Mxor_y1_63_xo<0>1 (f288<63>)
    OBUF:I->O                 0.000          ct_63_OBUF (ct<63>)
    ----------------------------------------
    Total                     1728.501ns (292.588ns logic, 1435.913ns route)
                                         (16.9% logic, 83.1% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================


Total REAL time to Xst completion: 626.00 secs
Total CPU time to Xst completion: 625.92 secs

-->

Total memory usage is 1612140 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :  363 (   0 filtered)
Number of infos    :    2 (   0 filtered)
```

CSA Decryption. Simulation

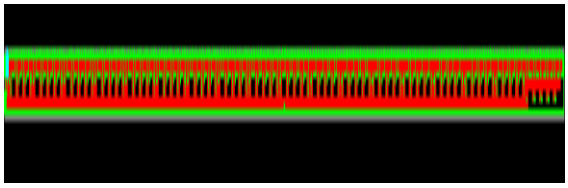
International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018



Technology Schematic.



RTL Schematic.



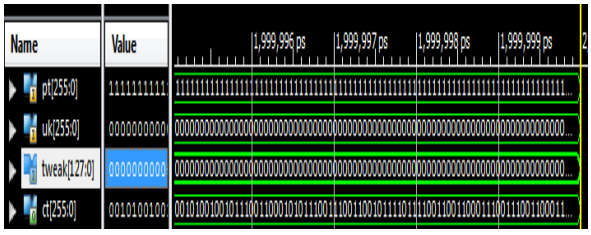
Design Summary.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	23453	63400	36%
Number of fully used LUT-FF pairs	0	23453	0%
Number of bonded IOBs	896	210	426%

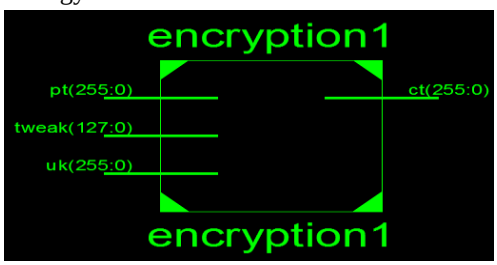
Timing Summary.

```
    LUT5:I1->O            3   0.097   0.693  im143/s64/s8/s3/c431 (im143/s64/s8/s3/c4_
    LUT6:I1->O            3   0.097   0.566  im143/s64/s8/s3/c411 (im143/s64/s8/s6)
    LUT6:I2->O            2   0.097   0.383  im143/s64/s8/s4/c431 (im143/s64/s8/s4/c4_
    LUT4:I2->O            1   0.097   0.279  iX19/Mxor_s_51_xo<0>1 (dt_51_OBUF)
    OBUF:I->O             0.000              dt_51_OBUF (dt<51>)
    ----------------------------------------
    Total                      1523.084ns (205.945ns logic, 1317.139ns route)
                               (13.5% logic, 86.5% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================


Total REAL time to Xst completion: 197.00 secs
Total CPU time to Xst completion: 197.87 secs

-->

Total memory usage is 730988 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :   76 (   0 filtered)
Number of infos    :    3 (   0 filtered)
```

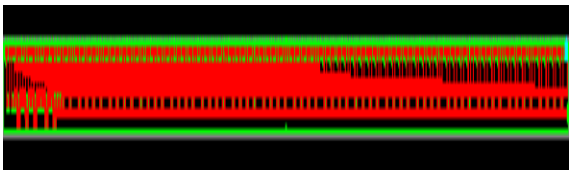
Brent Kung Adder Encryption.
Simulation.



Technology Schematic.



RTL Schematic.



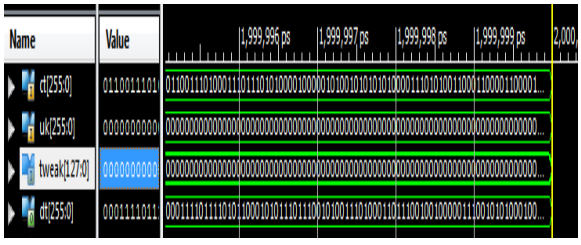
Design Summary.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	25592	63400	40%
Number of fully used LUT-FF pairs	0	25592	0%
Number of bonded IOBs	896	210	426%

Timing Summary.

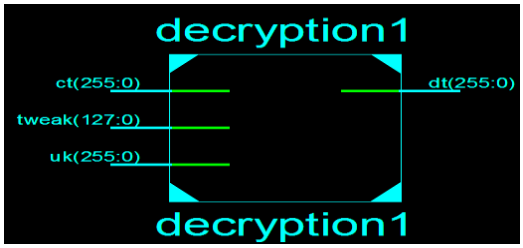
```
    LUT6:I0->O            2   0.097   0.697  m140/X1/Mxor_y1_16_xo<0>1 (f280<16>)
    LUT6:I0->O            7   0.097   0.539  m141/k11/k8/zk4/c1/d3/c0 (m141/k11/c4)
    LUT6:I3->O            4   0.097   0.570  X19/Mxor_s_101_xo<0>1 (k19<101>)
    LUT4:I0->O            4   0.097   0.309  m144/k11/k7/k2/c1/d1/c01 (m144/k11/k7/k2/c1/d3/u1
    LUT5:I4->O            1   0.097   0.511  m144/k11/k7/k2/c1/d3/c01 (m144/k11/k7/c2)
    LUT6:I3->O            2   0.097   0.697  m144/k11/k7/k3/po1/x1/Mxor_out_xo<0>1 (f287<40>)
    LUT6:I0->O            1   0.097   0.279  m144/X1/Mxor_y1_40_xo<0>1 (f288<40>)
    OBUF:I->O             0.000              ct_40_OBUF (ct<40>)
    ----------------------------------------
    Total                      139.283ns (20.953ns logic, 118.330ns route)
                               (15.0% logic, 85.0% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================


Total REAL time to Xst completion: 378.00 secs
Total CPU time to Xst completion: 377.61 secs

-->

Total memory usage is 1231148 kilobytes
```

Brent Kung Adder Decryption.
Simulation.

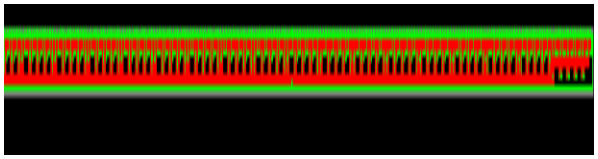


Technology Schematic.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018



RTL Schematic.



Design Summary.

Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	22567	63400	35%
Number of fully used LUT-FF pairs	0	22567	0%
Number of bonded IOBs	896	210	426%

Timing Summary.

```
    LUT6:I1->O            3   0.097   0.566  im143/s64/s8/s3/c411 (im143/s64/s8/k6)
    LUT6:I2->O            2   0.097   0.383  im143/s64/s8/s4/c431 (im143/s64/s8/s4/c4_bdd:
    LUT4:I2->O            1   0.097   0.279  iX19/Mxor_s_51_xo<0>1 (dt_51_OBUF)
    OBUF:I->O             0.000              dt_51_OBUF (dt<51>)
    ----------------------------------------
    Total                 1500.516ns (204.102ns logic, 1296.414ns route)
                                     (13.6% logic, 86.4% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

=========================================================================


Total REAL time to Xst completion: 189.00 secs
Total CPU time to Xst completion: 189.10 secs

-->

Total memory usage is 751916 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :   77 (   0 filtered)
Number of infos    :    3 (   0 filtered)
```

COMPARISON TABLE

Encryption comparison:

contents	Carry save adder	Ripple carry adder	Brunt Kung adder
No.of LUT tables	33204	25147	25592
delay	1728.501ns	1466.429ns	139.283

Decryption comparison:

contents	Carry save adder	Ripple carry adder	Brunt Kung adder
No.of LUT tables	23453	22386	22567
delay	1523.084ns	1501.599ns	1500.516ns

CONCLUSION

Adders are core and essential block in many modules which involves computation and adders play a vital role in the design of multipliers using adder based logic, hence the design and implementation of the adders is a prime concern, In this paper we have designed, modeled the different adders like RCA, KSA, BKA using different design style and coated comparative results obtained. From the results it is clear that the adders designed using GDI design style give less delay and consumes less number of gate count, CMOS design style give less power consumption, as these the performance parameters are prime concerned while designing a module. Hence the choice has to be made and as per the desire of the designer and the prime concern of performance measure needed at that point in time.

REFERENCES

[1] American Bankers Association, "Keyed Hash Message Authentication Code," ANSI X9.71, 2000.

[2] J. Aumasson, C. Calik, W. Meier, O. Ozen, R. Phan, and K. Varici, "Improved Cryptanalysis of Skein" http://www.131002.net/papers.html, submitted to the IACR eprint server, September 2009.

[3] E. Barker, D. Johnson, and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)," NIST Special Publication SP 800-56A, Mar 2007.

[4] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," NIST Special Publication SP 800-90, Mar 2007.

[5] M. Bellare, "New Proofs for NMAC and HMAC: Security without Collision-Resistance," Advances in Cryptology—CRYPTO '06 Proceedings, Springer-Verlag, 2006, pp. 602–619.

[6] M. Bellare, R. Canetti and H. Krawczyk, "Keying hash functions for message authentication,"

Advances in Cryptology—CRYPTO '96 Proceedings, Springer-Verlag, 1996 , pp. 1–15.
[7] M. Bellare, R. Canetti, and H. Krawczyk, "Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security," Proceedings of the 37th Symposium on Foundations of Computer Science, IEEE Press, 1996, pp. 514–523.
[8] M. Bellare, J. Kilian, and P. Rogaway. "The Security of Cipher Block Chaining," Advances in Cryptology—CRYPTO '94 Proceedings, Springer-Verlag, 1994, pp 341–358. [9] M. Bellare, T. Kohno, S. Lucks, N. Ferguson, B. Schneier, D. Whiting, J. Callas, and J. Walker, "Provable Security Support for the Skein Hash Family," Version 1.0, Apr 2009, http://www.skein-hash.info/sites/default/files/skein-proofs.pdf.
[10] M. Bellare and T. Ristenpart, "Multi-Property-Preserving Hash Domain Extension and the EMD Transform," Advances in Cryptology—ASIACRYPT '06 Proceedings, Springer-Verlag, 2006, 299–314.
[11] M. Bellare and B. Yee, "Forward Security in Private Key Cryptography, " Topics in Cryptology—CT-RSA, Springer-Verlag, 2003, pp. 1–18.
[12] D.J. Bernstein, "Cache-Timing Attacks on AES," April 2005, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.