Synopsis of A Pipelined Memoryless AES Decryptor

Synopsis submitted in Fulfillment of the requirements for the degree of

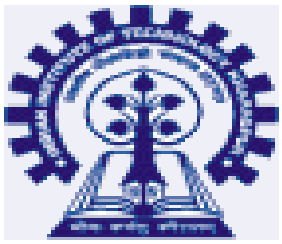
MASTERS OF TECHNOLOGY (HONS.)

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By
Bhaben Deori
03 CS 3020

Under the guidance of
Prof Dipanwita Roychoudhury



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR
KHARAGPUR-721302, INDIA

Abstract:

This document presents a fully pipelined implementation of the Advanced Encryption Standard decryption algorithm with 128-bit input and key length (AES-128), implemented on Xilinx' Spartan-3. The design implements a combinational logic based Rijndael S-Box implementation for the Sub Byte transformation in the Advanced Encryption Standard (AES) algorithm for Field Programmable Gate Arrays (FPGAs).

In the present work, architecture is developed which aims to achieve a high throughput, lesser IO lines, on-chip key generation for physical security. The design has been implemented in Galois Subfield which leads to memory-less architecture. A new architecture has been implemented for mix column and inverse mix column to optimize hardware overhead. Intelligent clocking has been performed to reduce the power. The width of the data input and output buses are 32 bits for easy interface. The key scheduling architecture has been multiplexed to prevent external attack during the normal functioning of the chip. Modules have been designed which would be a part of the design when both encryption and decryption are implemented in a single architecture.

1. INTRODUCTION:

The importance of cryptography is constantly increasing, since the amount of sensitive data being transmitted over open environments is growing at an unprecedented pace. Software-based implementations of cryptographic algorithms fall short of the required performance, as the transmission speeds of core networks reach the gigabits per second (Gbps) range. The significance and applicability of hardware-based implementations of cryptographic algorithms is therefore of interest also to the Field Programmable Gate Array (FPGA) design community.

The implementation of fully unrolled secret-key cryptographic algorithms is feasible on million-gate FPGAs. If the entire algorithm with full inner and outer loop pipelining fits on a single FPGA, the limiting factor for throughput is the achieved clock rate as follows:

$$throughput = blocksize \times frequency$$

Since the block size of AES is fixed at 128 bits, a 100 MHz clock rate implies a throughput of 12.8 Gbps. Clock rates above 100MHz should be achieved in modern FPGAs by partitioning the design into stages and pipelining the entire system.

2. THE AES ALGORITHM

The Advanced Encryption Standard (AES) algorithm is a symmetric block cipher that processes data blocks of 128 bits using cipher keys with lengths of 128, 192 and 256 bits. The AES algorithm is also called the Rijndael algorithm named after its inventors, Joan Daemen and Vincent Rijmen. In the present work, only the 128 bit decryption version is considered. In the following chapters, the description generally concentrates on AES-128, but whenever the description is valid for all variants of AES, the generic abbreviation AES is used.

Data is handled mainly as bytes in the AES algorithm. One byte forms an element in a polynomial representation of Galois Field GF (2^8). A byte can be represented in hexadecimal notation as {ab}, where a represents the four most significant bits (MSB) and b represents the four least significant bits (LSB) of the byte.

In this document, the representation used in the official standard is called F1, formally defined as GF (2) [x] =m(x), where m(x) is an irreducible polynomial

$$\mathbf{m\,(x) = x^8 + x^4 + x^3 + x + 1}$$

Additions are performed as bitwise XORs between operands in polynomial representations of F1. Multiplications in F1 are performed as a multiplication of the regular polynomials. The multiplication result can be a 14-degree polynomial which doesn't fit into a byte. Thus the final multiplication result in F1 is the result of the polynomial multiplication modulo m(x).

128-bit data block and key are considered as a byte array with four rows and four columns. AES-128 consists of ten rounds. One AES encryption round includes four transformations: **SubBytes, ShiftRows, MixColumns and AddRoundKey**. The first and last round differ from other rounds in that there is an additional AddRound- Key transformation at the beginning of the first round and no Mix- Columns transformation is performed in the last round. Key Expansion in the AES algorithm calculates RoundKeys based on the original cipher key. The RoundKeys are needed in AddRoundKeys. In AES-128 encryption, the first RoundKey used in the additional AddRoundKey at the beginning of the first round is always the original key. Intermediate results after every transformation are calledStates. Of all the transformation above, the ByteSub is most computationally heavy.

2.1 The State, the Cipher Key and the number of rounds

State: the intermediate cipher result is called the State. The State can be pictured as a rectangular array of bytes. This array has four rows; the number of columns is denoted by Nb and is equal to the block length divided by 32.The Cipher Key is similarly pictured as a rectangular array with four rows. The number of columns of the Cipher Key is denoted by Nk and is equal to the key length divided by 32. These representations are illustrated in Figure.

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

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

3. RIJNDAEL IN A COMPOSITE FIELD:

Rijndael involves arithmetic on GF (2^8) elements. In a straightforward implementation, inverse, multiplication and substitution are likely to be the operations that determine the overall complexity of the implementation.

3.1 ISOMORPHISM BETWEEN F1 and F2:

Instead of the table implementation in F1 it was decided to perform the SubBytes transformation by calculating the multiplicative inverse of the SubBytes in

$$F2 := GF(2^4)[x] = (x^2 + Ax + B)$$

To make this work a byte representing an element in F1 must be transformed to a byte representing an element in F2. All multiplications in GF (2^4) are performed in GF (2) [y] = (y^4 +y+1). Constants A and B can be chosen freely as long as x^2 +Ax+B is irreducible. The constants are chosen as follows: A = 0b0001 = {1} and B = 0b1000 = {8}. Thus the irreducible polynomial becomes x^2 +x+y^3.

The transformation matrix ø is expressed in matrix form as:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The inverse transformation $ø^{-1}$: $F2 \rightarrow F1$ is also needed $ø^{-1}$ can be defined by inverting the matrix F with the result as follows:

$$\Phi^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

In addition to the multiplicative inverse also other transformations in the AES-128 decryption algorithm are calculated in F2. This makes decryption faster and saves significant amounts of space since the transformations ø and $ø^{-1}$ performed only once. The transformation ø is performed for both the key and data block at the beginning of the decryption and the inverse transformation $ø^{-1}$ is performed for the decrypted data block at the end of the last round. The following subsections describe how the mapping of InvSubBytes, InvMixColumns, AddRoundKey, InvShiftRows and Key Expansion to F2 was performed.

4. The SubByte and InvSubByte in F2:

If a byte is mapped to *F*2 with the transformation F, the multiplicative inverse can be calculated as follows:

$$(bx+c)^{-1} = b\,(b^{2}\lambda + c\,(b+c))^{-1}\,x + (c+b)\,(b^{2}\lambda + c\,(b+c))^{-1}$$

Let $b'' = \mathbf{T}b + c$ be the affine transformation in F1 and

Let $b = \mathbf{T_ø}b_ø + c_ø$ be the affine transformation in F2

$$\text{Now because } b' = ø = ø^{-1}(\mathbf{T_ø}b_ø + c_ø) = ø^{-1}(\mathbf{T_ø}øb + c_ø)$$

$$b' = (øT_øø)b + ø^{-1}c_ø$$

Comparing with above we get,

$$\mathbf{T} = ø^{-1}\mathbf{T_ø}ø \text{ or } \mathbf{T_ø} = ø\,\mathbf{T_ø}ø^{-1}$$

And similarly, $c_ø = ø\,c$

The Affine Transformation in F2 can therefore be expressed as:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Similarly for inverse affine transformation.

5. INVERSE MIX COLUMN:

The InvMixColumns transformation of the AES-128 decryption algorithm must also be mapped to *F*2. The addition in *F*2 is calculated in a similar fashion as in *F*1 (that is, by bitwise XORing the operands), and therefore only the multiplications must be mapped to *F*2.

The InvMixColumns transformation multiplies the polynomial formed by each column of the State with $a^{-1}(x)$ modulo $x^4 + 1$, where

$$\mathbf{a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}}$$

And in Mix Column a column is considered as a polynomial over F1 and multiplied modulo $x^4 + x + 1$ with the polynomial

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

We use different procedure to implement the Inv Mix Column. The Mix Column multiplication will be used to implement the Inv Mix Columns Because F maps {01} to {01} it suffices to map only the multiplications with {02}.Writing,

$$a = a7x^7 + a6x^6 + a5x^5 + a4x^4 + a3x^3 + a2x^2 + a1x + a0$$

multiplication with {02}* a in F1 can be calculated as:

$$\begin{aligned}\{02\} * a &= x\,(a7x^7 + a6x^6 + a5x^5 + a4x^4 + a3x^3 + a2x^2 + a1x + a0) \\ &= a7x^8 + a6x^7 + a5x^6 + a4x^5 + a3x^4 + a2x^3 + \\ &\quad a1x^2 + a0x \bmod x^8 + x^4 + x^3 + x + 1 \\ &= a6x^7 + a5x^6 + a4x^5 + (a3 + a7)x^4 + (a2 + a7)x^3 + \\ &\quad a1x^2 + (a0 + a7)x + a7\end{aligned}$$

And multiplication {04} * a results in the following equation:

$$\begin{aligned}\{04\} * a = {} & a5x7 + a4x6 + (a3+a7)\,x5 + (a2 + (a6+a7))\,x4 \\ & + (a1+a6)\,x3 + (a0+a7)\,x2 + (a6+a7)\,x + a6\end{aligned}$$

In matrix form the above multiplication is expressed as

$$\{02\} \bullet \mathbf{a} = M_2\mathbf{a} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}$$

The matrices $\mathbf{M_{ø2}}$ can be expressed in F2 and can be calculated from $\mathbf{M_2}$ as follows:

$$M_{\phi 2} = \Phi M_2 \Phi^{-1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

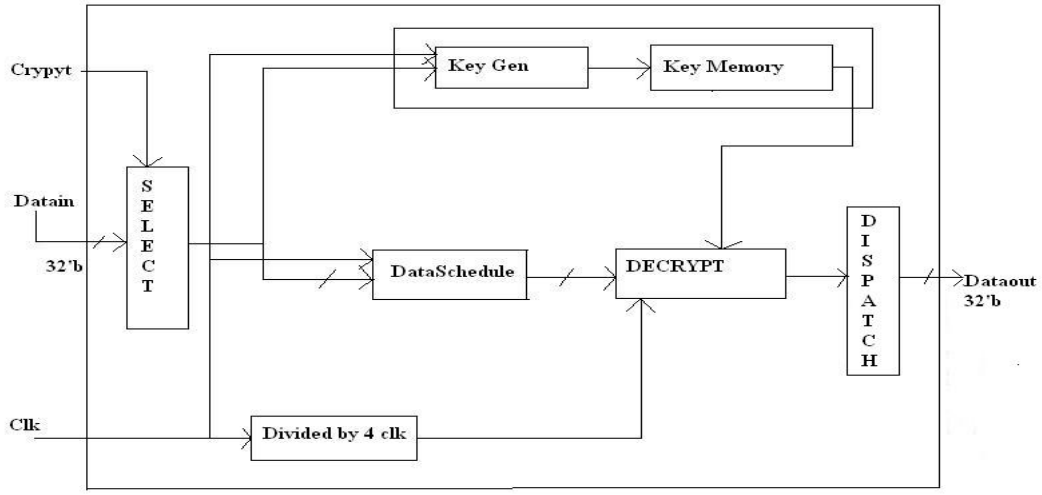
6. ADD ROUND KEYS AND INVSHIFT ROWS IN F2

Because addition is calculated as a bitwise XOR in both *F*1 and in *F*2 there is no need for changes in the AddRoundKey transformation. Also the ShiftRows transformation remains unchanged, because no calculations are required there

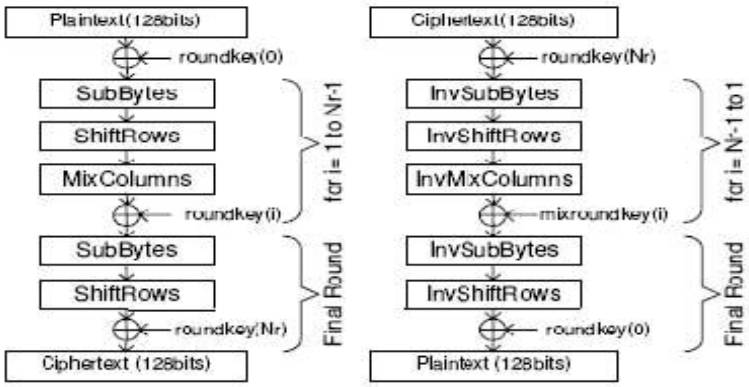
7. KEY EXPANSION IN F2:

In the Key Expansion, the function SubWord() and the round constant rcon[i] must be mapped to *F*2. SubWord(), which consists of four SubBytes, is mapped as described in Section 3.1.The rcon[i] values (powers of *x*) are mapped to *F*2 by multiplying them with the matrix ø. All the transformations of the AES-128 decryption algorithm have now been mapped from *F*1 to *F*2. The decryption can be implemented as follows: first both the 128-bit data block and the 128-bit key are mapped to *F*2 with the transformation ø and then the decryption is carried out as described above. At the end of the last round the encrypted data is mapped back to F1 with the inverse transformation $ø^{-1}$.

8. TOP LEVEL ARCHITECTURE:



9. THE DECRYPTING UNIT



The AES algorithm

9.1 INV MIX COLUMN:

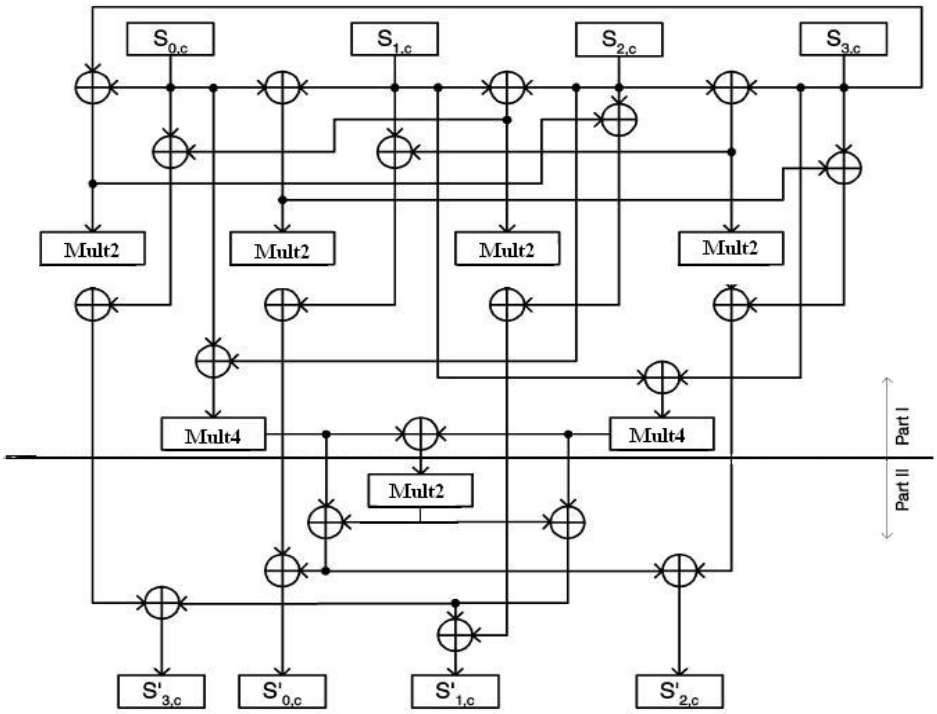
In matrix form, the InvMixColumns transformation can be expressed by

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{0e\}_{16} & \{0b\}_{16} & \{0d\}_{16} & \{09\}_{16} \\ \{09\}_{16} & \{0e\}_{16} & \{0b\}_{16} & \{0d\}_{16} \\ \{0d\}_{16} & \{09\}_{16} & \{0e\}_{16} & \{0b\}_{16} \\ \{0b\}_{16} & \{0d\}_{16} & \{09\}_{16} & \{0e\}_{16} \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$
$$0 \leq c < 4.$$

This can be rewritten as:

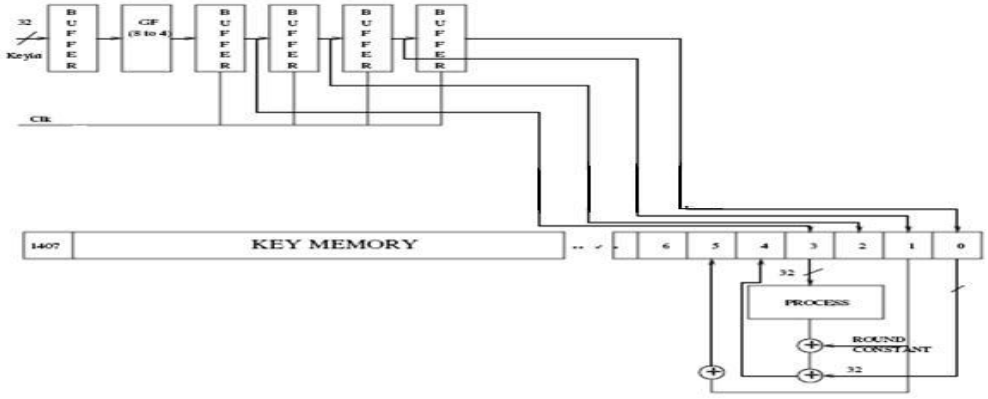
$$\begin{aligned}
S'_{0,c} &= (\{02\}_{16}(S_{0,c} + S_{1,c}) + (S_{2,c} + S_{3,c}) + S_{1,c}) \\
&\quad + (\{02\}_{16}(\{04\}_{16}(S_{0,c} + S_{2,c}) \\
&\quad + \{04\}_{16}(S_{1,c} + S_{3,c})) + \{04\}_{16}(S_{0,c} + S_{2,c})) \\
S'_{1,c} &= (\{02\}_{16}(S_{1,c} + S_{2,c}) + (S_{3,c} + S_{0,c}) + S_{2,c}) \\
&\quad + (\{02\}_{16}(\{04\}_{16}(S_{0,c} + S_{2,c}) \\
&\quad + \{04\}_{16}(S_{1,c} + S_{3,c})) + \{04\}_{16}(S_{1,c} + S_{3,c})) \\
S'_{2,c} &= (\{02\}_{16}(S_{2,c} + S_{3,c}) + (S_{0,c} + S_{1,c}) + S_{3,c}) \\
&\quad + (\{02\}_{16}(\{04\}_{16}(S_{0,c} + S_{2,c}) \\
&\quad + \{04\}_{16}(S_{1,c} + S_{3,c})) + \{04\}_{16}(S_{0,c} + S_{2,c})) \\
S'_{3,c} &= (\{02\}_{16}(S_{3,c} + S_{0,c}) + (S_{1,c} + S_{2,c}) + S_{0,c}) \\
&\quad + (\{02\}_{16}(\{04\}_{16}(S_{0,c} + S_{2,c}) \\
&\quad + \{04\}_{16}(S_{1,c} + S_{3,c})) + \{04\}_{16}(S_{1,c} + S_{3,c})).
\end{aligned}$$

Using substructure sharing, the InvMixColumn can be implemented by the architecture illustrated below. The “mult4” block computes the constant multiplication of {04} transformed by Ø, can be implemented by two serially concatenated “mult2” block.



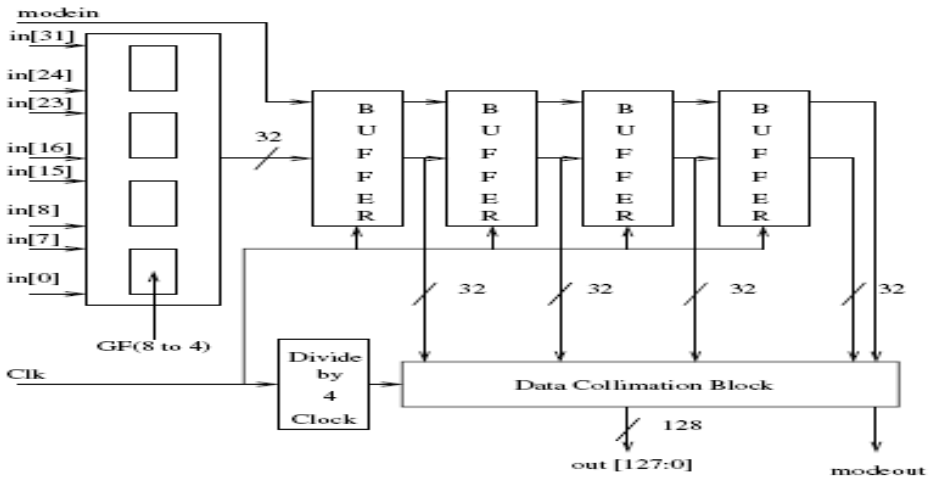
9.2 *Key Scheduler:*

The KeyScheduling unit has been multiplexed with the databus *Figure 1*. The *CRYPT* signal when high sets the keygeneration mode whereby the input data is streamed into the scheduler and creates all the subkeys. The multiplexing successfully reduce the pin count as no extra pins are required for the input key. Further, when the AES core is used to process (decrypt) the data the keymemory is not accessible from the external world through the IO pins. Thus the round keys are secured from external attack during the normal operation of the device.



9.3 The Data Scheduler

The *Data Scheduling Unit* converts a GF(2^8) element into a GF(2^4)^2 element and buffers in the data at each system clock. At the fourth clock when 128 bit (32 x 4) of data has arrived the unit *dumps* the data into a 128-bit register . When a valid block is being decrypted, a corresponding signal of high modein is passed in. This signal is synchronized with the data block, so that a high modeout indicates that the result is corresponding to a valid input.



9.4 The Dispatch Unit

The *Dispatch Unit* converts the output word is to GF(2^8) and streams out as a 32-bit wide stream of processed data.

10. Synthesis Report:

The design was implemented in Xilinx virtex 2vp30ff1152-7 device and simulated by Xilinx ISE simulator.

==========

* Final Report *

==========

Device utilization summary:

Selected Device : 2vp30ff1152-7

Number of Slices:	10155	out of 13696	74%
Number of Slice Flip Flops:	3383	out of 27392	12%
Number of IOs:	69		
Throughput (Gbps)	13.8		
Clock frequaency (MHz)	107.8		