**Chapter 3**

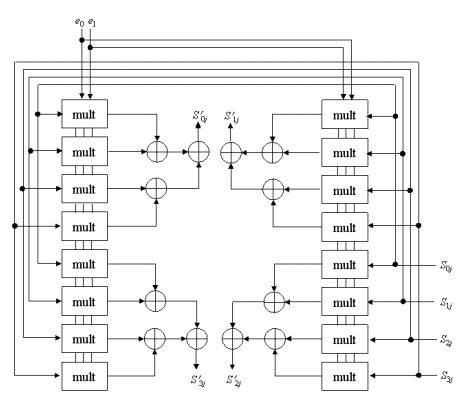
**Implementation Issues**

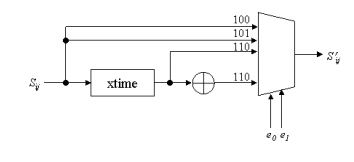
**3.1 Architecture of Basic Components**

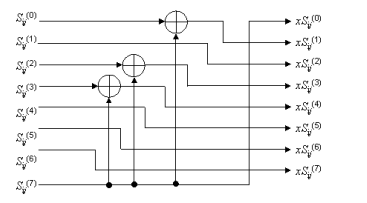
The overall architecture of the AES hardware mirrors the structure of Algorithm discussed in 2.2 . It is a synchronous implementation of both the processes of cipher. It uses 2 128-registers.Every clock transition, these registers are loaded, except dataout1 and dataout2, which is loaded when an input state is completely ciphered. During the encryption process, Register0 is loaded with the input data or the partially encrypted text with the result of the **mixcolumn** and **AddRoundKey** component except first and last round in which mixcolumn operation is skipped , Register2 with the state after applying functions SubBytes and subsequently ShiftRows..The component that implements function AddRoundKey is simply a net of XOR gates that adds in GF(2^8) the key schedule to the current state. The component implementing function SubBytes uses 16 S-boxes stored in a Read-Only Memory (ROM). The obtained state is row-shifted before its storage in Register2. The component architecture is given in Fig. 2.

Function MixColumns is implemented by a massively parallel component that computes all the bytes of the new state in a single clock. It uses four components of the same architecture. This basic component produces one column os the new state. Its architecture is described in Fig. 3, wherein component mult yields the a special product of a given byte from the state times{01}, {02}or{03}.The architecture of component multi presented in Fig. 4.Component xtime computes the xtime operation as defined in[5] and shown in Fig. 5.

Equivalent invMixColumns can be implemented in the same way as MixColumns, , wherein component invmult yields the a special product of a given byte from the state times{0e}, {0b},{0d} or {09}.







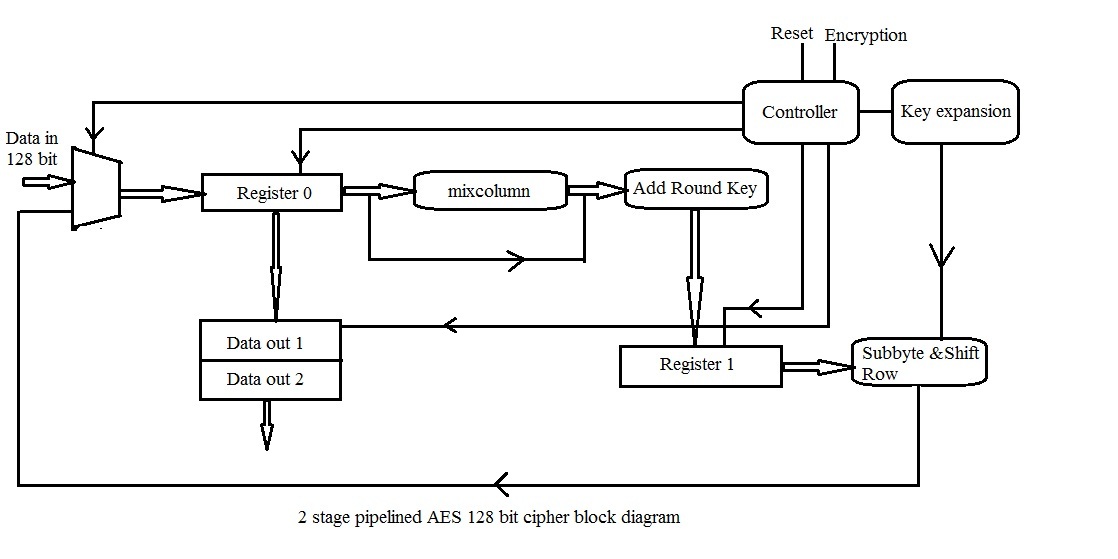
**3.2 Pipelining**

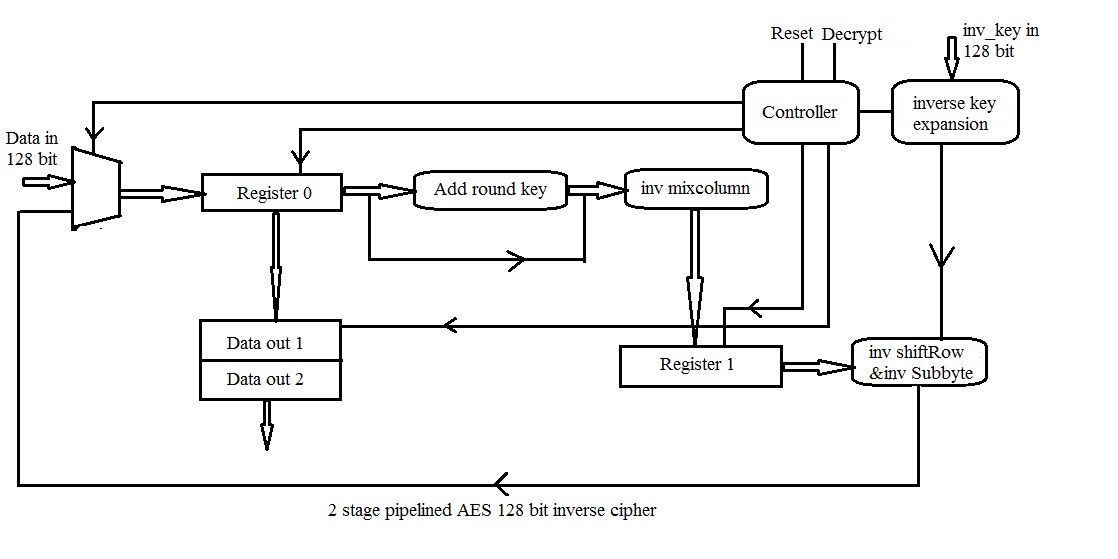
2 stage pipelined architecture is shown in fig. it two 128 bit data block in first two clock cycles . There were two possible ways of pipelining we decided upon. In one case, there is a possibility of using one stage of pipelining for each of the 11 stages of the encryption process, thereby increasing the throughput. The other is the one described in [2]. A 11 stage pipeline would obviously have a 3 to 4 times increase in throughput over a 3-stage one: but it requires 11 128-bit registors, thereby taking up too much space and a possibility of overflow of space in the FPGA. So we stuck with the implementation of the 2 stage pipeling. The following are the design issues we faced:

* Encryption routine and key expansion routine run parallelly in our implementation. Each round key remain valid for two clock cycles to operate on two 128 bit data blocks.
* In first two clock cycles data is input into the cipher by asserting the control signal rw 10 .There are 10 rounds So after 22 clock cycles output data1 is ready . on the 22th clock cycle control signal rw is asserted 01 and data is read from the memory. It is maintained for two clock cycles.so in 23 clock cycles 2 outpus leading to a throughput of

T.p=(2\*128)/(23\*clock cycles)

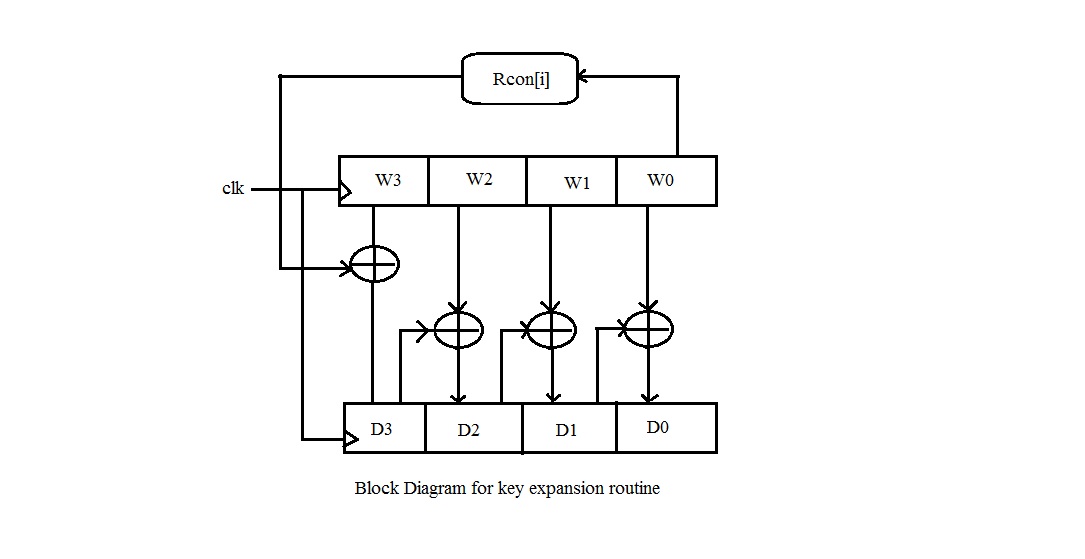
The inverse cipher operate in the same way with operations are performed in reverse order.



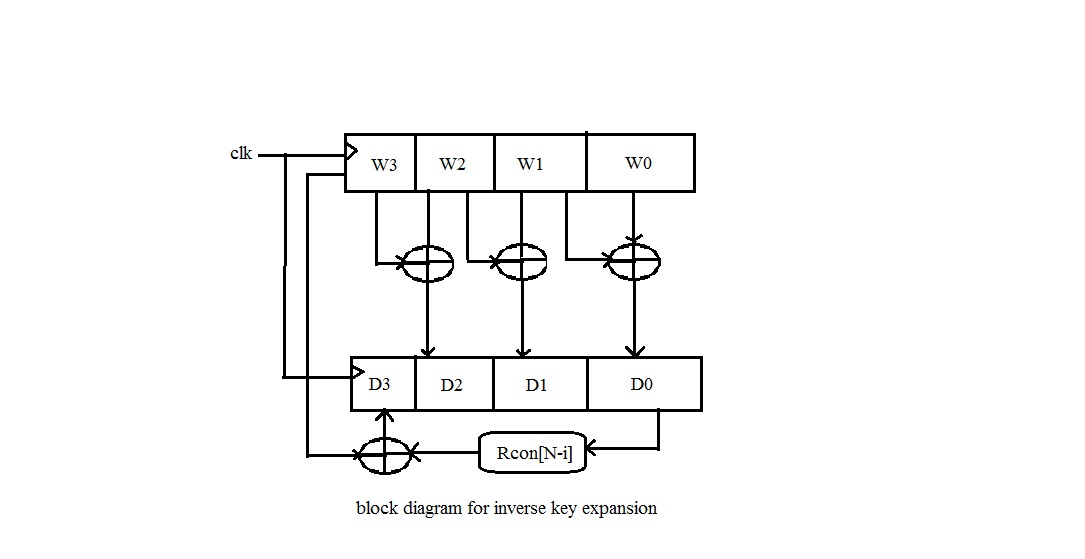


**3.3 KeyExpansion and invKeyExpansion routine**

As only 128-bits are used at each stage in the encryption process, the key schedule for each stage can be generated dynamically. The implemetation is shown in Figure. With each clock cycle, the previous value is loaded on the register. As is evident from our pipelined design, the key for stage i has to be held for two continuous clock cycles (as there are two stages in the pipeline–the same key acts on two different inputs in two clock cycles), and we employed a counter to achievethe same.



The key expansion routine for the inverse cipher can be generated by the method as shown in the following fig



**3.4 Key Length Requirements**

An implementation of the AES algorithm shall support at least one of the three key of lengths 128, 192, or 256 bits (i.e., Nk = 4, 6, or 8, respectively). Implementations may optionally support two or three key lengths, which may promote the interoperability of algorithm implementations.

**Chapter 4**

**Conclusion**