[[1]](#footnote-1)

Low Area FPGA Implementation of the AES Standard

William Chandler, Elsie Baderinwa, Luke Locastro and Thomas Brown

*Abstract*—This report discusses the design, implementation and performance of a low area FPGA-based implementation of the AES standard, incorporating both encryption and decryption. Pipelining is used to improve performance. The resultant system gives SOMETHING SOMETHING performance with a non-dynamic key and is the same for encryption and decryption.

*Index Terms*—AES, Low-Area, FPGA

# INTRODUCTION

T

HIS report discusses the design decisions, design process and resultant performance of a low area FPGA implementation of the AES standard. It incorporates both encryption and description and attempts to share resources where possible. Pipelining is used to improve the maximum clock frequency achievable by breaking up long sections of combinatorial logic. Previous work LIT REVIEW

# Brief Description of AES

DIAGRAM

LOOP

Key gen

# System Architecture

## Top level design decisions

Data in AES is processed in ‘states’. Each state consists of 16 bytes, usually arranged in 4 columns. Some larger implementations of AES process the whole state concurrently which is very fast but also requires a large area on the FPGA. In order to maintain a low-area it was decided that our design should operate on one byte at a time. This ensures the design will be as small as possible by eliminating the need for duplicate modules.

In a similar fashion, large area designs can ‘unroll’ the loop which means the 10 looped operations are formed by duplicating all the modules 10 times. This allows the designs to be very fast by allowing 10 times as many states to be processed in parallel. For the low area application this is not suitable and the loop is not unrolled. In order to achieve this, a multiplexer is added to the start of the loop which can switch between input data and the output of the loop. Data is only accepted into the loop after 9 full loop iterations whilst the final data is being output.

This low area 8-bit ROLLED? datapath requires more control than the high throughput 128-bit unrolled datapath which means there is a penalty to the throughput per slice. This is a trade off between area and efficiency.

Each of the blocks in the datapath, the key generator and the controller were designed specifically to be low area whilst also striving for high efficiency.

Implementation of the inverse operation in the datapath seems rather inefficient. It consists of a number of multiplexers which change the arrangement of the processes to an almost reversed configuration. ShiftRows and SubBytes do not need to be reversed as ShiftRows is not dependent on byte values and SubBytes only operates on byte values. The delay after MixColumns is also left in that position as it has no effect on the output result. There are other methods for implementing the inverse which were not explored in this project due to time constraints. THESE INCLUDE??

## SubBytes

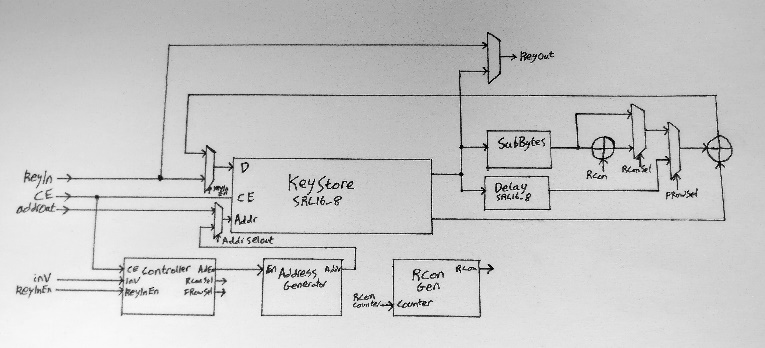
## Shift Rows

## Mix Columns

## Key Generator

Keys are given in the same format at the data, in ‘states’ of 16 bytes. The key generation process consists of and XOR of the N-4 byte with the N-16 byte. Additionally, the first column is rotated, the bytes are substituted and it is XORed with an RCon value. This process is shown in FIGURE. The inverse process requires the final key to be generated by running the forward process 10 times then to generate successive previous keys, the N+4 and N+16 bytes instead to generate the previous key.

Due to the low area constraint, and the ROLLED datapth, storing the keys after generation was not considered an effective solution. This meant that the keys would be generated on-the-fly as they are required by the datapath.

Initial work focused on the area efficiency of the SRL16 shift registers which allow the storage of 16 bits of information in only one LUT with two outputs, one addressable and one final CHECK THIS ENGLISH. These were used to form an SRL16\_8 register which stores 16 bytes of data. 

The SRL16\_8 register was used to form the main key store within input data being switched by a multiplexer between new key inputs and the last generated key. The rotation of the first column was achieved by changing the address given to the register to select the correct bytes.

The two outputs of the register allow both N-4 and N-16 bytes to be selected concurrently. MENTION THAT A SEPARATE SUB BYTES BLOCK WITHOUT INVERSE IS USED

The SubBytes block has 3 cycles delay caused by internal pipelining which means that the addressing starts three cycles before the output is expected. It also necessitates the addition of a delay block (also formed by an SRL16\_8), so that when the first column is complete, further output appears at the final XOR at the appropriate cycle.

The RCon value is zero for all but the first byte, so due to it only being used in an XOR operation where zeros produce no result, only the first value is important. This value is generated using a simple lookup based on a counter passed from the main controller.

Selecting between the first byte output (RCon XOR output), first row output (SubBytes output) and all other bytes is achieved using multiplexers.

The inverse operation can is achieved by feeding in the final key backwards, the control for which is in the main controller.

In order to manage external inputs as well as managing the inverse key reversal, a top controller module was created. This consisted of an external keystore (SRL16\_8) which stores the first or final key for forward and inverse operations respectively. MORE HERE? Some control signals are also changed for the inverse.

Due to some lack of foresight and some changes to the top-level design, significant changes were required. First the datapath was changed from a 16-register cycle to a 32-register cycle which holds two full states before reaching the next cycle. This means that each key must be outputted twice, once for each state. Second, for the inverse operation, the key must still be passed to the datapath forwards. Which means the key needs to be inverted before being output. Third, as the state in the datapath is being output, a new state is being fed in to avoid any bubbles in the pipeline. This means the first key (or final in the inverse case) must be passed to the first XOR in the datapath whist the final key is being passed to the second XOR in the datapath. These changes added significant complexity to the design.

This complexity is mostly handled in the top controller module. An additional SRL16\_8 buffer is used to store the key that is outputted from the key generator. The output to the second XOR in the datapath is then multiplexed between the output of the key generator and the output of the buffer which has the effect of repeating the key. In order to output the inverse key bytes in the forward order, the key is generated whilst the key is being sent from the buffer, a counter is then used to address the key store inside the key generator to select the bytes. The external key store now stores the second key, which is fed back in to the key generator every 10 rounds. A second key store is used to store the first key, which is constantly output to the first XOR in the datapath. Assuming the key is not dynamic, these stores allow the pipeline to avoid bubbles, as well as allowing the inverse operation to be as fast (after setup) as the forward operation. All this extra complexity required a lot of control signals to be generated, so counters were added to keep track of the round as well as the byte within the round. These counters were then used to create control signals.

Much of this complexity is avoidable by using a three or four port RAM.

## Controller

Fortunately, the controller for the datapath did not require much more logic area because of the counters already used by the key generator controller. *keyInReady* and *dataInReady* signals are used to indicate to the serial wrapper that a key or data block can be input to the system. When either is being input, the corresponding *keyInEn* or *dataInEn* signals are set high by the wrapper. When the inverse input signal is set high, the *dataInReady* signal will go low until a key has been input and the final key has been generated. Once dataInEn is set high, signals are generated to start the components at the correct time. The system will stop and wait if no data is given when dataInEn is high. This simplifies the control but does mean that the final state will not be output until new data is given.

# Performance

# Conclusion

The performance of this AES implementation is not as good as some other more developed solutions. It is still functional and gives good results considering the late changes that had to be made to support the inverse operation. Due to the experience gained during this project, future designs will be more thoroughly planned out before starting the design phase. This should prevent unexpected complexities and increase in area.

# References

**There are no sources in the current document.**

TABLE I

Units for Magnetic Properties

|  |  |  |
| --- | --- | --- |
| Symbol | Quantity | Conversion from Gaussian and  CGS EMU to SI a |
| Φ | magnetic flux | 1 Mx → 10−8 Wb = 10−8 V·s |
| *B* | magnetic flux density,  magnetic induction | 1 G → 10−4 T = 10−4 Wb/m2 |
| *H* | magnetic field strength | 1 Oe → 103/(4π) A/m |
| *m* | magnetic moment | 1 erg/G = 1 emu  → 10−3 A·m2 = 10−3 J/T |
| *M* | magnetization | 1 erg/(G·cm3) = 1 emu/cm3  → 103 A/m |
| 4π*M* | magnetization | 1 G → 103/(4π) A/m |
| σ | specific magnetization | 1 erg/(G·g) = 1 emu/g → 1 A·m2/kg |
| *j* | magnetic dipole  moment | 1 erg/G = 1 emu  → 4π × 10−10 Wb·m |
| *J* | magnetic polarization | 1 erg/(G·cm3) = 1 emu/cm3  → 4π × 10−4 T |
| χ*,* κ | susceptibility | 1 → 4π |
| χρ | mass susceptibility | 1 cm3/g → 4π × 10−3 m3/kg |
| μ | permeability | 1 → 4π × 10−7 H/m  = 4π × 10−7 Wb/(A·m) |
| μr | relative permeability | μ → μr |
| *w, W* | energy density | 1 erg/cm3 → 10−1 J/m3 |
| *N, D* | demagnetizing factor | 1 → 1/(4π) |

Vertical lines are optional in tables. Statements that serve as captions for the entire table do not need footnote letters.

aGaussian units are the same as cgs emu for magnetostatics; Mx = maxwell, G = gauss, Oe = oersted; Wb = weber, V = volt, s = second, T = tesla, m = meter, A = ampere, J = joule, kg = kilogram, H = henry.



Fig. 1. Magnetization as a function of applied field. Note that “Fig.” is abbreviated. There is a period after the figure number, followed by two spaces. It is good practice to explain the significance of the figure in the caption.

1. [↑](#footnote-ref-1)