Pipelined AES128 Rijndael with AMBA AXI

Version 0 Revision 0

June 12, 2018 (Jun 12, 2018)

Future Design Systems, Inc.
www.future-ds.com / contact@future-ds.com

Copyright © 2018 Future Design Systems, Inc.

Abstract

This document addresses an implementation of parallel version of AES128 Rijndael.

Table of Contents

1 Overview

The Advanced Encryption Standard (AES) is formal encryption method adopted by the National Institute of Standards and Technology of the US Government, and is accepted worldwide. AES is a symmetric block cipher. Symmetric means that it uses the same key for both encryption and decryption. A block cipher is a method of encrypting text (to produce ciphertext) in which a cryptographic key and algorithm are applied to a block of data.

The Rijndael algorithm, one of AES implementations allows the block and key size of 128, 160, 192, 224, 256 bits. However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128,192,256 bits.

Figure 1 shows overall environment to test AES128[1] core using Future Design Systems CON-FMC, in which AES128 core is implemented in the FPGA and it is controlled by the C program through USB 3.0 channel.

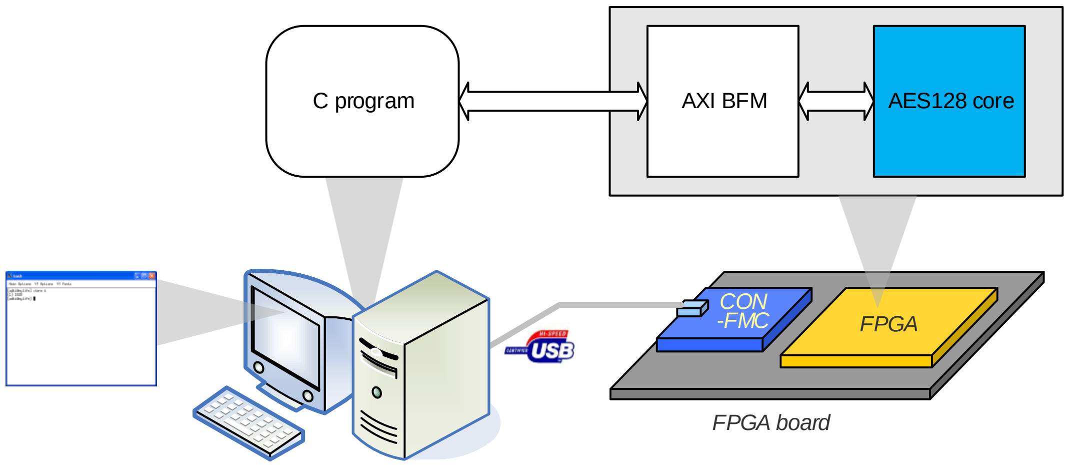

Figure 1: Overall setup

1.1 AES operation

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of its counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes
2. Shift rows

[1] AES128 only uses 128-bit key and 128-bit data in ECB (electronic code book) mode.

3. Mix Columns
4. Add Round Key

The tenth round simply leaves out the Mix Columns stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows
2. Inverse Substitute bytes
3. Inverse Add Round Key

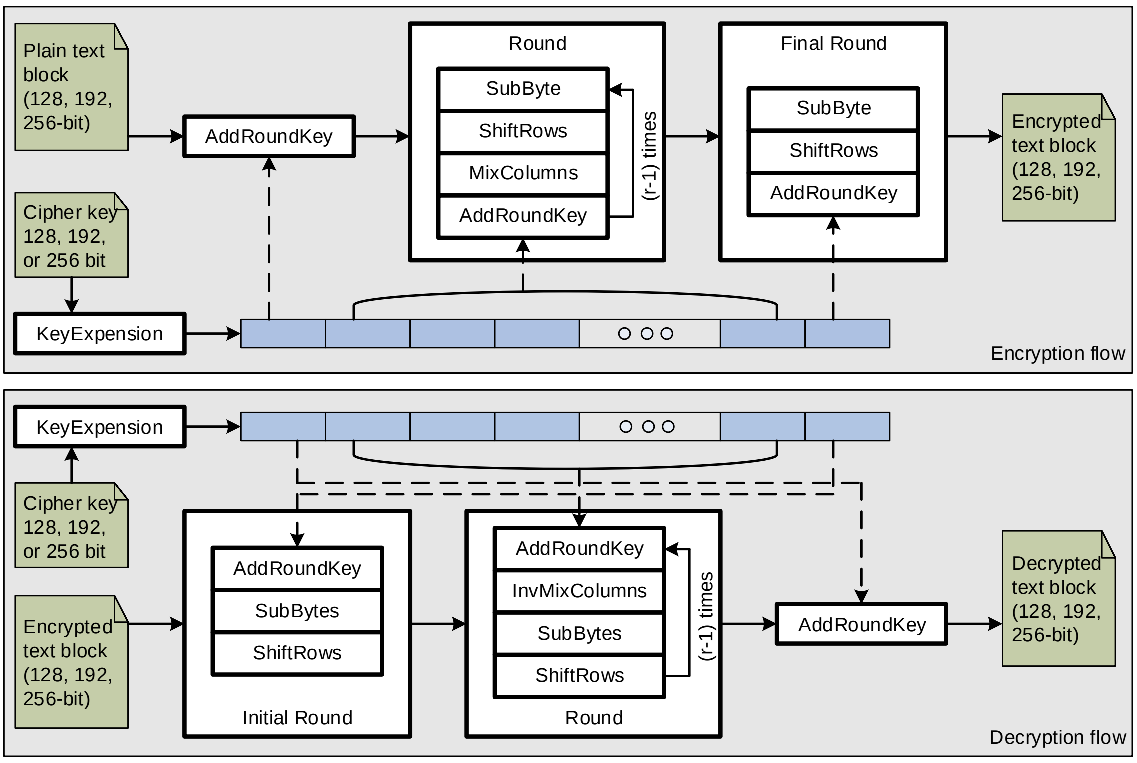

Figure 2: AES128 operation

2 AES128 core with AMBA AXI interface

AES128 AXI provides AMBA AXI interface as shown in Figure 3.

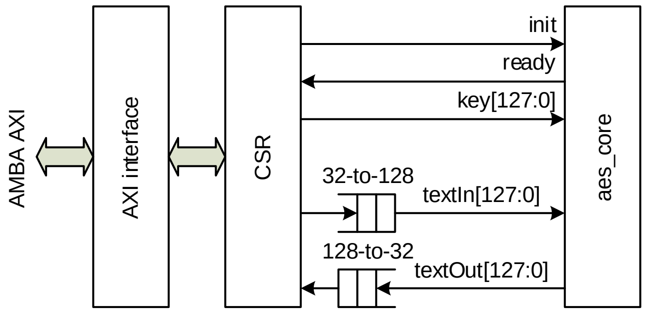

Figure 3: AES128 AXI block diagram

2.1 AES128 AXI interface

It takes care of AMBA AXI interface.

2.2 AES128 CSR

It provides CSR (Control and Status register) as shown in Figure 4. All registers are little-endian register.

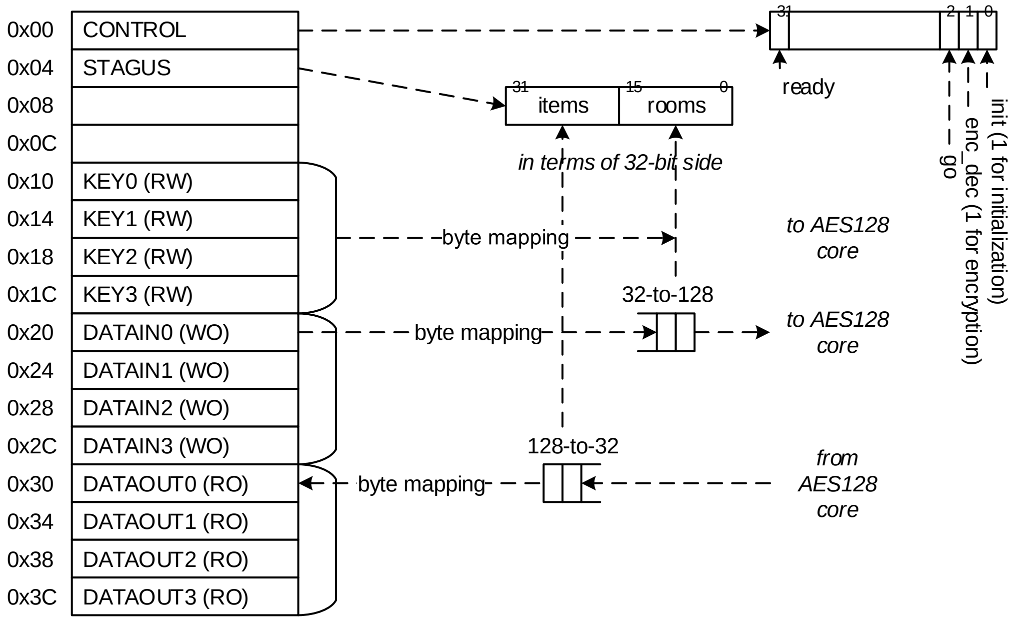

Figure 4: SHA_256_AXI CSR

As AES128 core uses big-endian style byte mapping, key and data to/from AES128 core are swapped as shown in Figure 5.

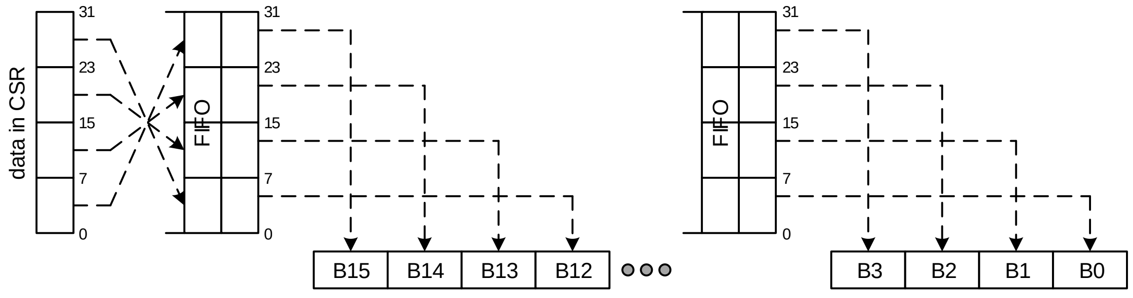

Figure 5: Byte mapping

2.2.1 Operations

There are three distinct operations: key scheduling, data pushing, and data popping.

```
// key scheduling
dataW = (enc_dec&0x1)<<1) | 0x1; // initialization
AXI_WRITE(CSR_CONTROL, dataW);
AXI_WRITE_BURST(CSR_KEY0, key, 4); // key update
dataW = (0x1<<2) | (enc_dec&0x1)<<1) | 0x0; // de-assert init
do { AXI_READ(CSR_CONTROL, dataR);
} while ((dataR&0x80000000)==0); // wait until ready
```

```
// data pushing – say (wnum/4) encryptions
do { AXI_READ(CSR_STATUS, dataR);
} while ((dataR&0xFFFF)<wnum); // wait for sufficient rooms
AXI_WRITE_BURST(CSR_DATAIN0, plain, wnum); // use fixed-address mode
```

```
// data popping
do { AXI_READ(CSR_STATUS, dataR);
} while ((dataR>>16)<wnum); // wait for results
AXI_READ_BURST(CSR_DATAOUT0, cyper, wnum); // use fixed-address mode
```

2.3 AES128 FIFO controller

Since AES128 core requires 128-bit data at a time, 32-bit AXI interface uses 32-bit to 128-bit asymmetric FIFO between interface and the core. The FIFO uses the first word fall-through feature.

2.4 AES128 core

'aes_init' causes key scheduling, which ends with 'aes_ready'. This key scheduling takes 260 cycles as shown in Figure 6.

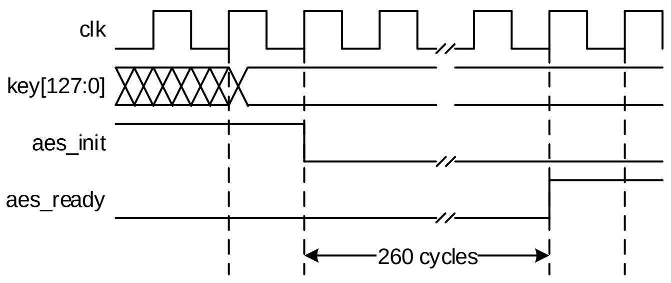

Figure 6: key scheduling timing

Each 128-bit block takes 10-cycles to produce result and this AES128 can take contiguous data stream.

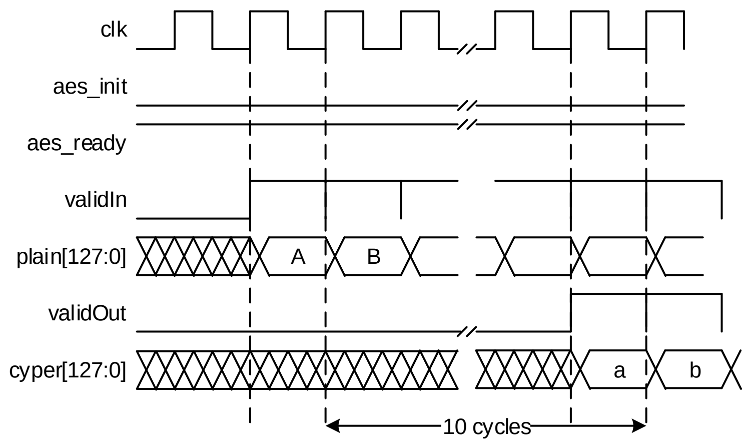

Figure 7: AES128 timing

3 Verification

3.1 Simulation

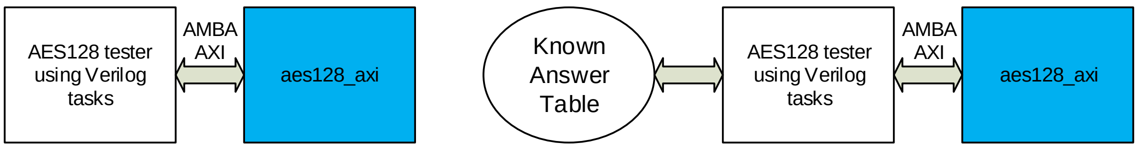
AES128 tester using Verilog tasks
AMBA AXI
aes128_axi
Known Answer Table
AES128 tester using Verilog tasks
AMBA AXI
aes128_axi

Figure 8: Verification using HDL simulation

3.2 FPGA verification

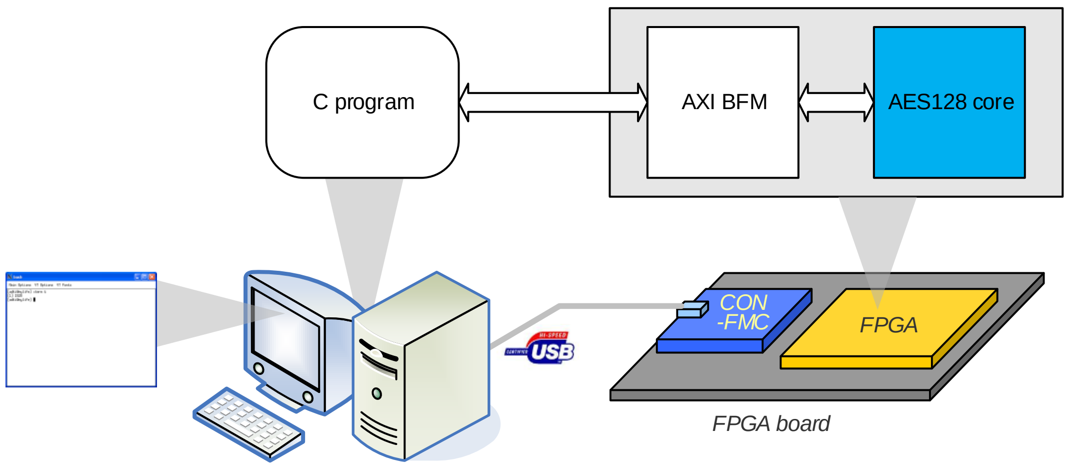

Figure 9: FPGA verification

4 Summary and future work

This document addresses an implementation of AES128 Rijndael algorithm.

5 References

[1]
[2]

Wish list

- ☐

Revision history

- ☐ 2018.06.12: Started by Ando Ki (adki@future-ds.com)

– End of document –