The Advanced Encryption Standard (AES) on an FPGA

Lecture 14

Josh Brake
Harvey Mudd College

Outline

- AES Overview
- AES Implementation Details
  - Block diagram
  - Embedded Block RAMs
  - Timing

Learning Objectives

By the end of this lecture you will…

- Have an operational understanding of the fundamental mathematics used in AES
- Understand the basic process of AES

The Advanced Encryption Standard

AES Overview

From the spec:

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext.

The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

128-bit AES

- The cipher works by taking a plain text message and scrambling it into a cyphertext message in a way that is hard to reverse.
- Scrambling is done 10 times to make it hard. The key changes from round to round.

AES Data Organization

128-bit message is organized as a matrix of 16 bytes (4x4).

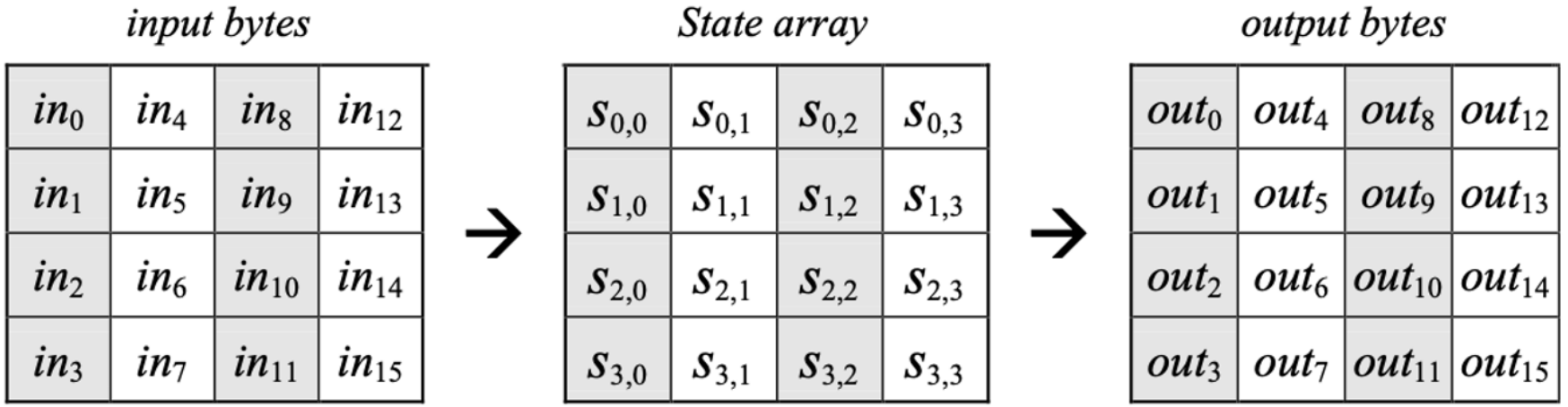

Figure 3. State array input and output.

AES Cipher Process

Each step of the cipher involves the following steps

1. `SubBytes`: take each byte and replace it with a different byte using a randomish lookup table.
2. `ShiftRows`: move bytes around in the rows
3. `MixColumns`: funky Galois multiplication on elements of the columns
4. `AddRoundKey`: XOR with the key for the current round.

AES Cipher Pseudocode

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 5. Pseudo Code for the Cipher.[1]

AES

Overall Block Diagram

1. `SubBytes`: take each byte and replace it with a different byte using a randomish lookup table.
2. `ShiftRows`: move bytes around in the rows
3. `MixColumns`: funky Galois multiplication on elements of the columns
4. `AddRoundKey`: XOR with the key for the current round.
5. Key expansion routine: generates

State matrix

How do we transform the 128-bit input into the state array?

SubBytes

What are the inputs?

What are the outputs?

What is this module doing (in words)?

What does the Verilog look like to accomplish this?

ShiftRows

What are the inputs?

What are the outputs?

What is this module doing (in words)?

What does the Verilog look like to accomplish this?

MixColumns

What are the inputs?

What are the outputs?

What is this module doing (in words)?

What does the Verilog look like to accomplish this?

AddRoundKey

What are the inputs?

What are the outputs?

What is this module doing (in words)?

What does the Verilog look like to accomplish this?

Key Expansion

What are the inputs?

What are the outputs?

What is this module doing (in words)?

What does the Verilog look like to accomplish this?

AES as FSM

- What does the state transition diagram look like for performing AES?
- Hint: Think about controller and datapath.

Embedded Block RAMs

3.1.5. sysMEM Embedded Block RAM Memory

Larger iCE40 UltraPlus device includes multiple high-speed synchronous sysMEM Embedded Block RAMs (EBRs), each 4 kbit in size. This memory can be used for a wide variety of purposes including data buffering and FIFO.

sysMEM Memory Block

The sysMEM block can implement single port, pseudo dual port, or FIFO memories with programmable logic resources. Each block can be used in a variety of depths and widths as listed in Table 3.4.

Table 3.4. sysMEM Block Configurations

Block RAM Configuration	Block RAM Configuration and Size	WADDR Port Size (Bits)	WDATA Port Size (Bits)	RADDR Port Size (Bits)	RDATA Port Size (Bits)	MASK Port Size (Bits)
SB_RAM256x16 SB_RAM256x16NR SB_RAM256x16NW SB_RAM256x16NRNW	256x16 (4 k)	8 [7:0]	16 [15:0]	8 [7:0]	16 [15:0]	16 [15:0]
SB_RAM512x8 SB_RAM512x8NR SB_RAM512x8NW SB_RAM512x8NRNW	512x8 (4 k)	9 [8:0]	8 [7:0]	9 [8:0]	8 [7:0]	No Mask Port
SB_RAM1024x4 SB_RAM1024x4NR SB_RAM1024x4NW SB_RAM1024x4NRNW	1024x4 (4 k)	10 [9:0]	4 [3:0]	10 [9:0]	4 [3:0]	No Mask Port
SB_RAM2048x2 SB_RAM2048x2NR SB_RAM2048x2NW SB_RAM2048x2NRNW	2048x2 (4 k)	11 [10:0]	2 [1:0]	11 [10:0]	2 [1:0]	No Mask Port

Note: For iCE40 UltraPlus, the primitive name without “Nxx” uses rising-edge Read and Write clocks. “NR” uses rising-edge Write clock and falling-edge Read clock. “NW” uses falling-edge Write clock and rising-edge Read clock. “NRNW” uses failing-edge clocks on both Read and Write.

Embedded Block RAM: Block Diagram

RAM4k Block

Figure 3.4 shows the 256x16 memory configurations and their input/output names. In all the sysMEM RAM modes, the input data and addresses for the ports are registered at the input of the memory array.

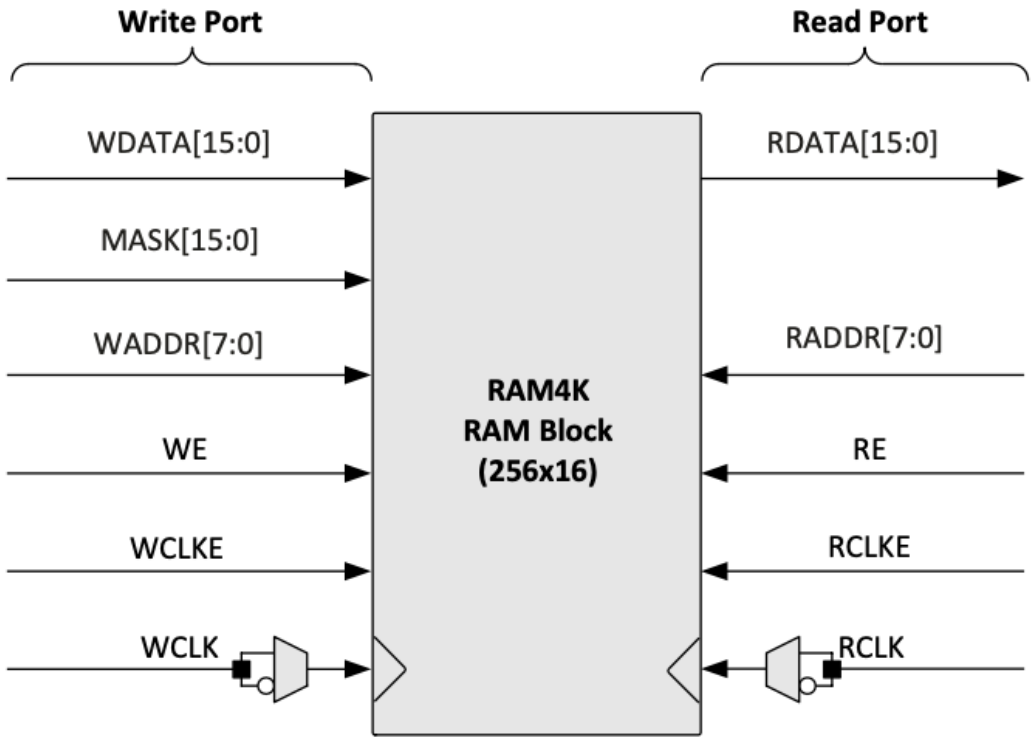

Figure 3.4. sysMEM Memory Primitives

Cascading Memories

4.3.1. Address Cascading (or Depth Cascading)

Address/Depth cascading is useful when the memories are required to have the capacity of storing *more* words while keeping the data width the same. In this case additional user logic is needed to decode the address.

Figure 4.1 shows an example of the depth cascading of a 32K x 16 SPRAM. Additional logic is required that guides the data to the correct memory block using Muxes and Demuxes. The rest of the signals (that are not shown), should be connected to both the memory blocks without any other logic requirements.

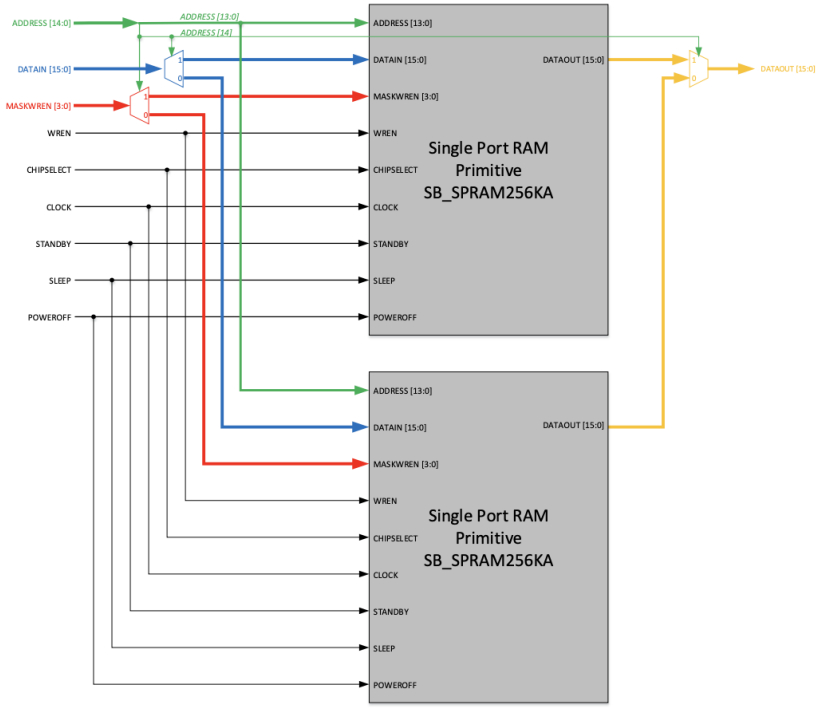

Figure 4.1. Address/Depth Cascading Example for 32K x 16 SPRAM using Primitive