VHDL AES128 Encryption/Decryption

By

Gentre Graham & David Leifker

Senior Project Report

EE 452 Senior Capstone Project I
Bradley University
Department of Electrical and Computer Engineering

May 9th, 2004

Abstract

Advanced Encryption Standard (AES), a Federal Information Processing Standard (FIPS), is an approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a block cipher that can encrypt and decrypt digital information. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits, this project implements the 128 bit standard on a Field-Programmable Gate Array (FPGA) using the VHDL, a hardware description language. In June 2003, the National Security Agency (NSA) announced that AES-128 may be used for classified information at the SECRET level and AES-192/256 for TOP SECRET level documents.

Table of Contents

Introduction

AES is an algorithm for performing encryption (and the reverse, decryption) which is a series of well-defined steps that can be followed as a procedure. The original information is known as plaintext, and the encrypted form as cipher text. The cipher text message contains all the information of the plaintext message, but is not in a format readable by a human or computer without the proper mechanism to decrypt it; it should resemble random gibberish to those not intended to read it. The encrypting procedure is varied depending on the key which changes the detailed operation of the algorithm. Without the key, the cipher cannot be used to encrypt or decrypt. In the past, cryptography helped ensure secrecy in important communications, such as those of government covert operations, military leaders, and diplomats. Cryptography has come to be in widespread use by many civilians who do not have extraordinary needs for secrecy, although typically it is transparently built into the infrastructure for computing and telecommunications (Wikipedia). Refer to Appendix A for definitions of key terms used throughout this document.

Functional Description:

Input/Output:

Figure 1: Inputs	
Keyboard	Enter data into system.
Clock	This is the clock used for the core operations.
Reset	Asynchronous reset, that sets the device to a known state.

Figure 2: Outputs	
LCD Display	Displays the output of the core.
LEDs	The eight LEDs display the ASCII code of the last key pressed on the keyboard.

Modes of Operation:

Figure 3: Operational Modes	
ASCII Input Mode	The input from the keyboard is considered to be encoded in ASCII.
Hexadecimal Mode	The input from the keyboard is considered to be encoded in Hexadecimal, thus the only valid characters are 0-9, A-F.
Encryption Mode	Device is configured to receive data and output ciphertext.
Decryption Mode	Device is configured to receive ciphertext and output data.

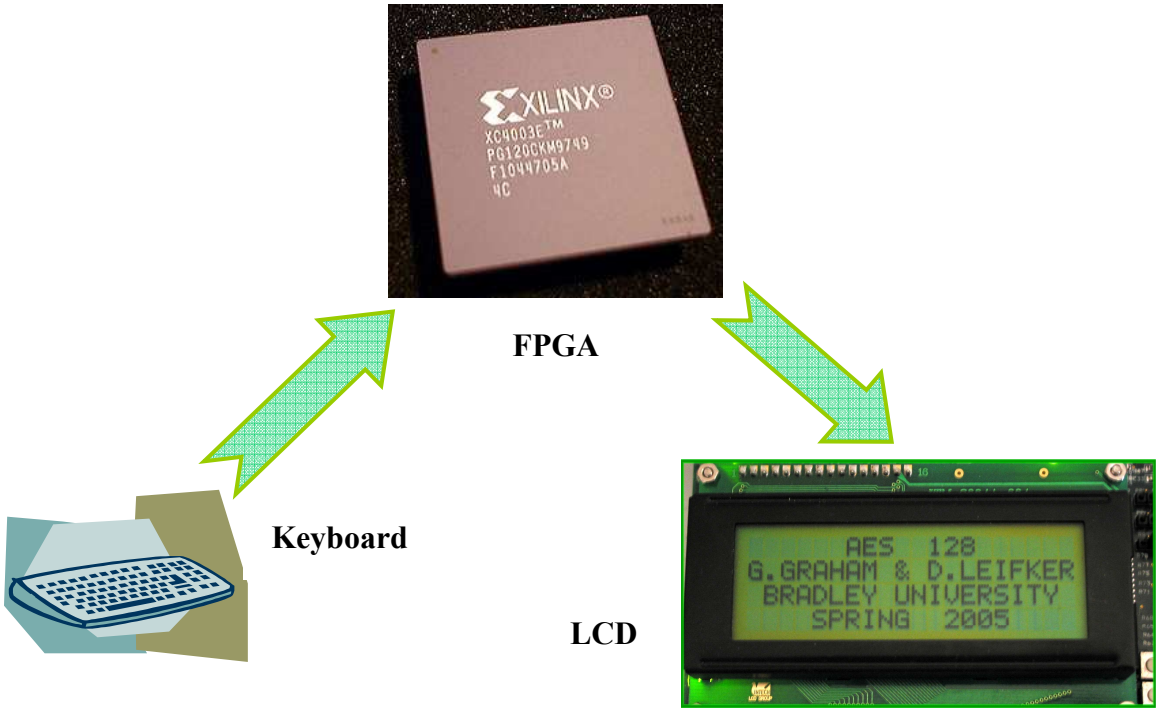


Figure 4: Overall System Diagram

This documentation will cover the architecture of this implementation of the 128bit specification of the Advanced Encryption Standard (AES), detailed in the Federal Information Processing Standard 197 (FIPS197). The design was developed and tested on a Xilinx Spartan III XC3S400 Field-Programmable Gate Array (FPGA), integrated into a NU Horizon development board, model HW-AFX-SP3-400-DB. This document will discuss and explain from a top down perspective how the various operations of the AES specification are implemented in VHDL.

The design goal of this project was to create a demonstration of the AES for the end user and not for integration into a communication or data storage device; however this design could be modified to such ends. Since the goal was to create

a demo, two human interface devices (HIDs), a regular PS2 keyboard and a 4x20 line LCD comprise the inputs and outputs to the system. A top level block diagram is shown in Figure 4 & 5 to illustrate. The VHDL code that describes this structure is located in file devicelevel.vhd accompanying this documentation.

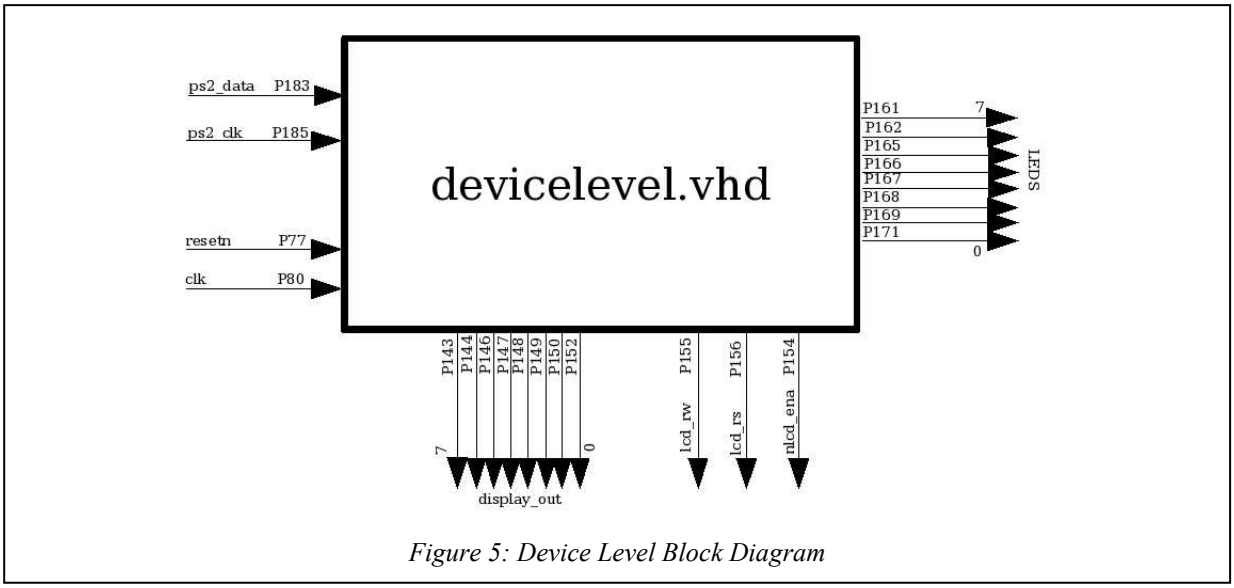


Figure 5: Device Level Block Diagram

The inputs to the system are the clock, clk, tested at 50 MHz & 100 MHz, resetn, and the PS2 keyboard inputs, ps2_data and ps2_clk. Technically, ps2_data and ps2_clk are bidirectional ports however the device never sends commands to the keyboard, thus these pins are limited to inputs of the system. The LEDs display information about the last key press encoded in ASCII and displayed in binary. Characters that are not recognized, because of the limited ASCII table implemented in this design, are displayed as FF hex. The reset is active low since it is configured to use a push button on the development board. The pin numbers are included as well for the particular NU Horizon development board. The remaining outputs display_out, lcd_rs, lcd_rw, and nlcd_ena are connected to the 4x20 LCD display.

The file devicelevel.vhd, contains the component level descriptions of the major sub-systems, a detailed diagram of these systems are displayed in Figure 6. Figure 6 also shows the VHDL files specific to each of these subsystems. A detailed description of each sub-system will follow. Please refer to Figure 6 as this is the only Figure which details the interconnections of the sub-systems. The directions of the interconnection arrows are accurate, bidirectional ports are indicated with arrows at both terminals. Dots represent an electrical connection and buses larger than 4 bits wide are numbered for convenience. The following discussion of the sub-systems will exclude resetn and clk, since they were previously discussed.

Table 1: Opcode Table

Opcodes	***Function***
“0000”	Reset internal variables & handles keyboard input.
“0010”	Store contents of INT_REGS in RAM.
“0011”	Expand key in RAM & one round of Key Expansion Routine.
“0100”	Add Round Key
“0101”	Byte Substitution Operation
“0110”	Shift Rows Operation
“0111”	Mix Columns Operation
“1010”	Inverse Shift Rows Operation
“1011”	Inverse Mix Columns Operation
“1111”	NOP

Table 2: Status Word Table

Status Code	***State***
“000”	Idle
“100”	Busy
“101”	Error
“010”	Operation Complete

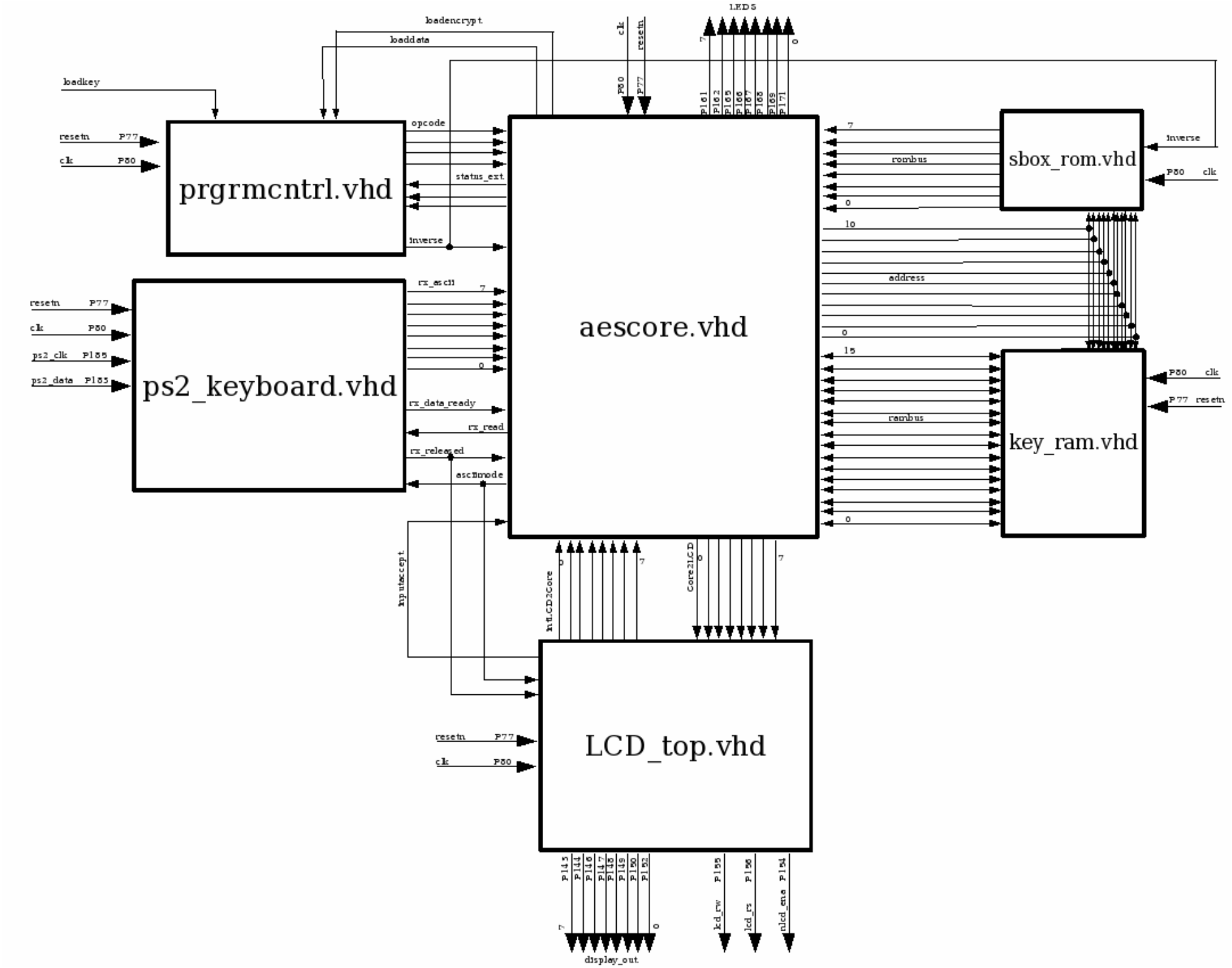


Figure 6: VHDL Sub-system Block Diagram

The first sub-system or module contains the sequence of steps to perform to execute all three major operations: key expansion, encryption and decryption. Please refer to FIPS 197 for details. The sequential steps are micro-programmed and are simply output through the opcode bus; see Table 1 for the opcodes. Three control lines, loadkey, loaddata, and loadencrypt determine which of the three major routines to execute, key expansion, encryption, and decryption respectively. Each routine consists of multiple opcode instructions. The start of each routine is initialized by the control lines. Once an operation is initialized the control lines are ignored until the aescore.vhd has completed all instructions associated with the routine and optionally after a timeout. After the routine is initialized, the sequential opcodes are sent to the opcode bus after aescore.vhd indicates an operation complete code on the status_ext bus. A complete list of status modes for the aescore.vhd is included in Table 2. The remaining output, inverse, is only high when the decryption routine is running. The status_ext bus is updated on the rising edge of the clk while the prgrmcntrl.vhd module triggers on the falling edge of the same clock so that the next opcode is immediately available for the next rising edge.

The second component is the ps2_keyboard.vhd module. The clk input to the ps2_keyboard.vhd sub-system must be ~50 MHz for proper timing. If another clock is used the constants in this file must be updated. The 8-bit ASCII code is output on the rx_ascii bus on a make and break signal from the keyboard. The rx_data_ready signal is high when a make or break operation has updated the contents of rx_ascii. If the keyboard signal is a break signal then rx_released is

also high. Once the aescore.vhd has acknowledged the data on rx_ascii by setting rx_read high, both rx_data_ready and rx_released are reset. The signal asciimode determines whether to accept ASCII characters or only hexadecimal values 0-9, A-F. A user defined 8-bits is placed on rx_ascii when operating in hex mode. In this state the bit vector on rx_ascii is not ASCII, see the end of ps2_keyboard.vhd for those values. Several keys have special functions in the aescore.vhd module. These keys are sent as unprintable ASCII codes. The keys are F1, F2, and Esc. The F1 and F2 keys instruct the core to signal the encryption and decryption routines respectively. The 'Esc' key is used to switch between ASCII input mode and hexadecimal input mode.

The third sub-system which controls the display is LCD_top.vhd. The input and output of aescore.vhd which is to be displayed on the LCD is stored in a 4x4 byte array. LCD_top.vhd sends an unsigned integer from 0 to 15 on intLCD2Core and the core replies with the appropriate byte of the array on Core2LCD. A more detailed description of the internals of LCD_top.vhd will follow the analysis of the aescore.vhd operation.

Sub-system key_ram.vhd stores the key schedule. The key schedule is calculated by the Key Expansion routine every time a new key is entered. Read and write operations are performed on the falling edge of the clock. The three most significant bits of the address bus are control lines for both key_ram.vhd and sbox_rom.vhd. The read control line for key_ram.vhd is bit 9 and write is bit 8. The remaining 8-bits comprise the address pins for the RAM and ROM. For AES-128 the size of the RAM is forty four 32-bit words or 176 bytes, the first 16 bytes is the key itself.

The sbox_rom.vhd component stores the substitution byte array defined in the specification. There are two arrays, one for the encryption routine and the other for the decryption operation, each array consists of 256 bytes. Bit 10 of the address bus is the read control line for the ROM. The additional control line, inverse, determines which of the two arrays should be used.

The final component, aescore.vhd, has been described through its relation with the other subsystems. All inputs and outputs to and from the core have been explained via the various interconnections with the other components. The following will describe the internal aescore.vhd code with reference to the AES specification as well as the nuances of the core that are not directly related to the specification.

All operations are preformed on a set of sixteen 1-byte registers arranged in a 4x4 array. This array is called INT_REGS(x, x) and has two indexes both 3 down to 0. Refer to the source code, aescore.vhd, to understand the context of these signals in the code itself. The first routine, the Key Expansion, is executed with two instructions 0010 and 0011, refer to Table 1. The first instruction stores INT_REGS in RAM to the location pointed to by WORD_EXTREGPTR. The pseudo

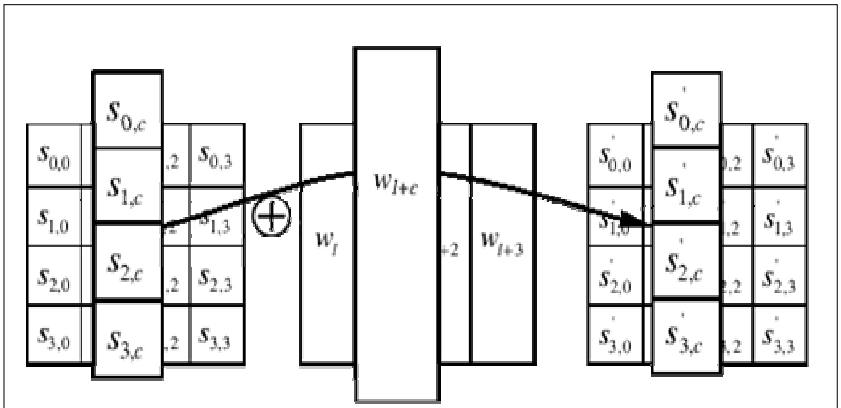


Figure 7: Add Round Key Operation

code for the Key Expansion, Figure 11 in the AES specification, implements the function on one 32-bit word at a time. The implementation for this project processes 4 32-bit words or all 128bits of the INT_REGS in one instruction.

The encryption routine uses the following opcodes: 0100, 0101, 0110, and 0111. The first opcode 0100 performs an operation that is common to both the encryption and decryption operations, Add Round Key. Figure 7 depicts the Add Round Key instruction. Add Round Key XORs each column of the state stored in

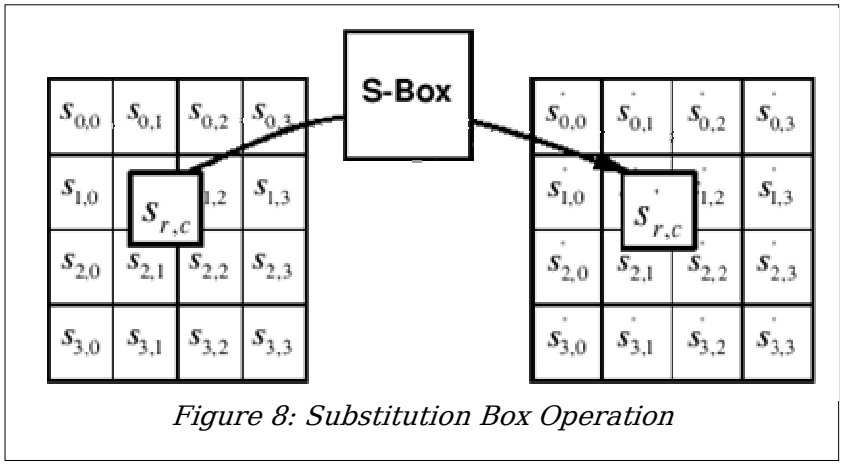


Figure 8: Substitution Box Operation

INT_REGS with a word from the Key Schedule, stored in RAM and pointed to by WORD_EXTREGPTR. The second opcode 0101 performs the substitute byte operation. The operation is also the same for encryption and decryption except that separate lookup tables are used. The sbox_rom.vhd will automatically use the correct table based on the inverse signal thus the operation call is identical whether encrypting data or decrypting data. This operation also operates on the entire state array byte by byte. See Figure 8.

The two unique functions for the encryption routine are 0110 and 0111, which are Shift Rows and Mix Columns operations respectively. Shift Rows simply operates on the state array, INT_REGS, by shifting the rows in the array by the amount specified in Figure 9. The reverse operation 1010 performs the inverse of

the operation illustrated in Figure 5 for the decryption routine.

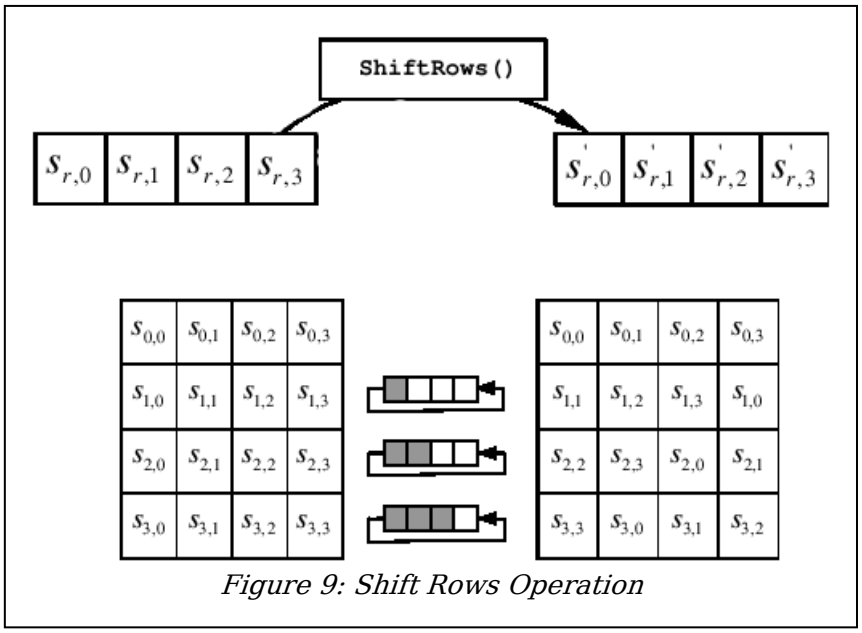


Figure 9: Shift Rows Operation

Mix Columns and the Inverse Mix Columns instructions, 0111 and 1011, operate column by column on the state array. Each column is replaced after being multiplied by a constant array. See Figure 10 & 11. The operation in equation form is show in Figures 12 & 13. The operation is reduced to simple shift and XOR operations refer to the source code.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 10: Mix Columns Constant Array

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 11: InvMixColumns Constant Array

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Figure 12: MixColumns ()

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Figure 13: Inverse MixColumns ()

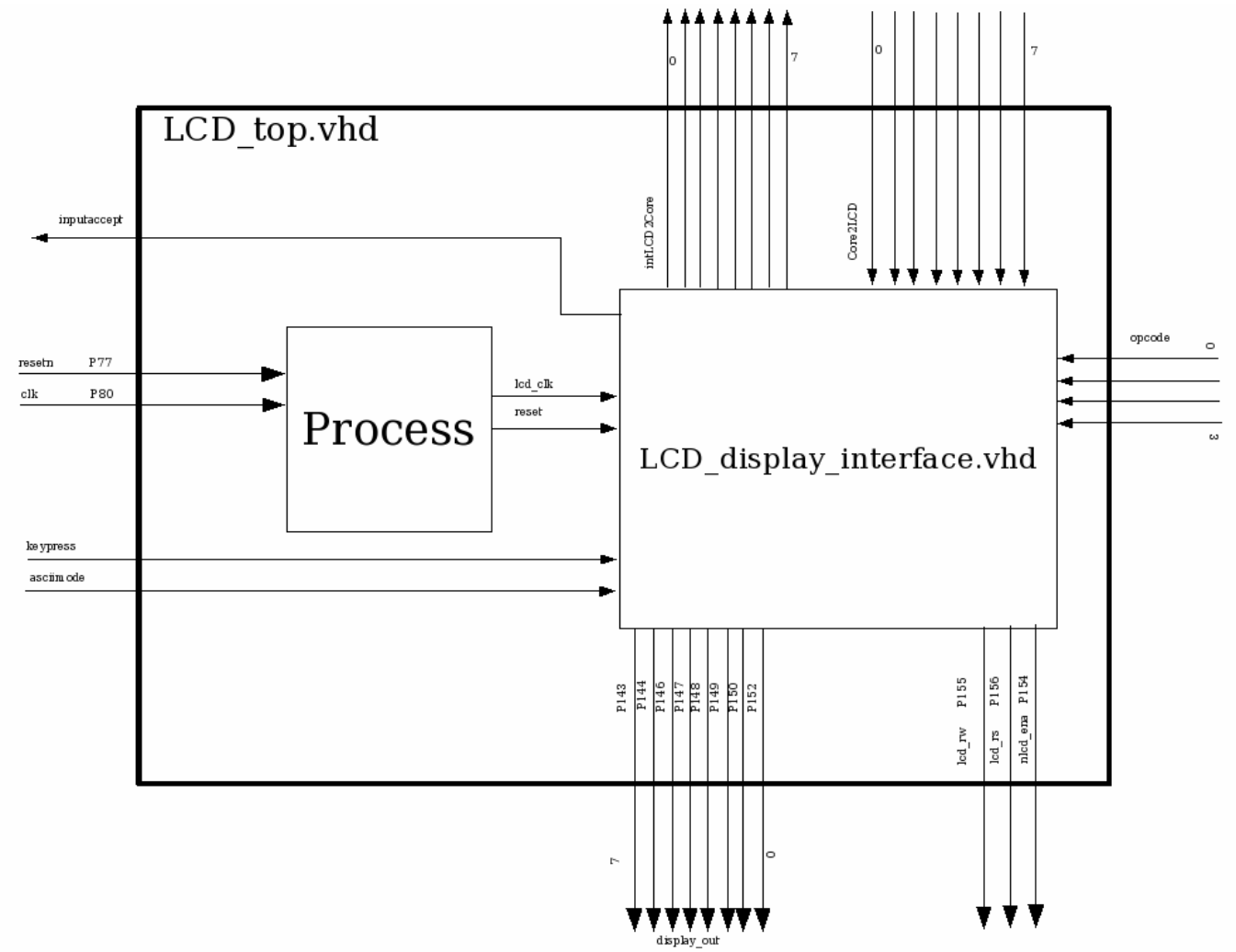


Figure 14: LCD System Block Diagram

The LCD_top.vhd file contains to main sub system blocks a Process and the lcd_display_interface.vhd. The LCD_top connects the lcd_display_interface.vhd to the entire system. The Process receives the system clock and the resetn signals and uses both signals to create the lcd_clk ouput by use of a counter. The lcd_dislplay_interface.vhd receives the lcd_clk and the reset input from the Process. The lcd_clk is the clock used to drive the state machine within the lcd_display_interface and the reset signal resets the system to an initial state. The asciimode signal is received from the keyboard to determine the display mode to be used by the lcd_display_interface. There are two modes used by the

lcd_display_interface one is plain text or ASCII and the second is Hex. The keypress input moves the current data from the output to the input. The core2LCD signal is received for the AEScore which is the data read from memory to be displayed on the LCD. The 4 bit opcode inputs current program status of the system to the display_interface.

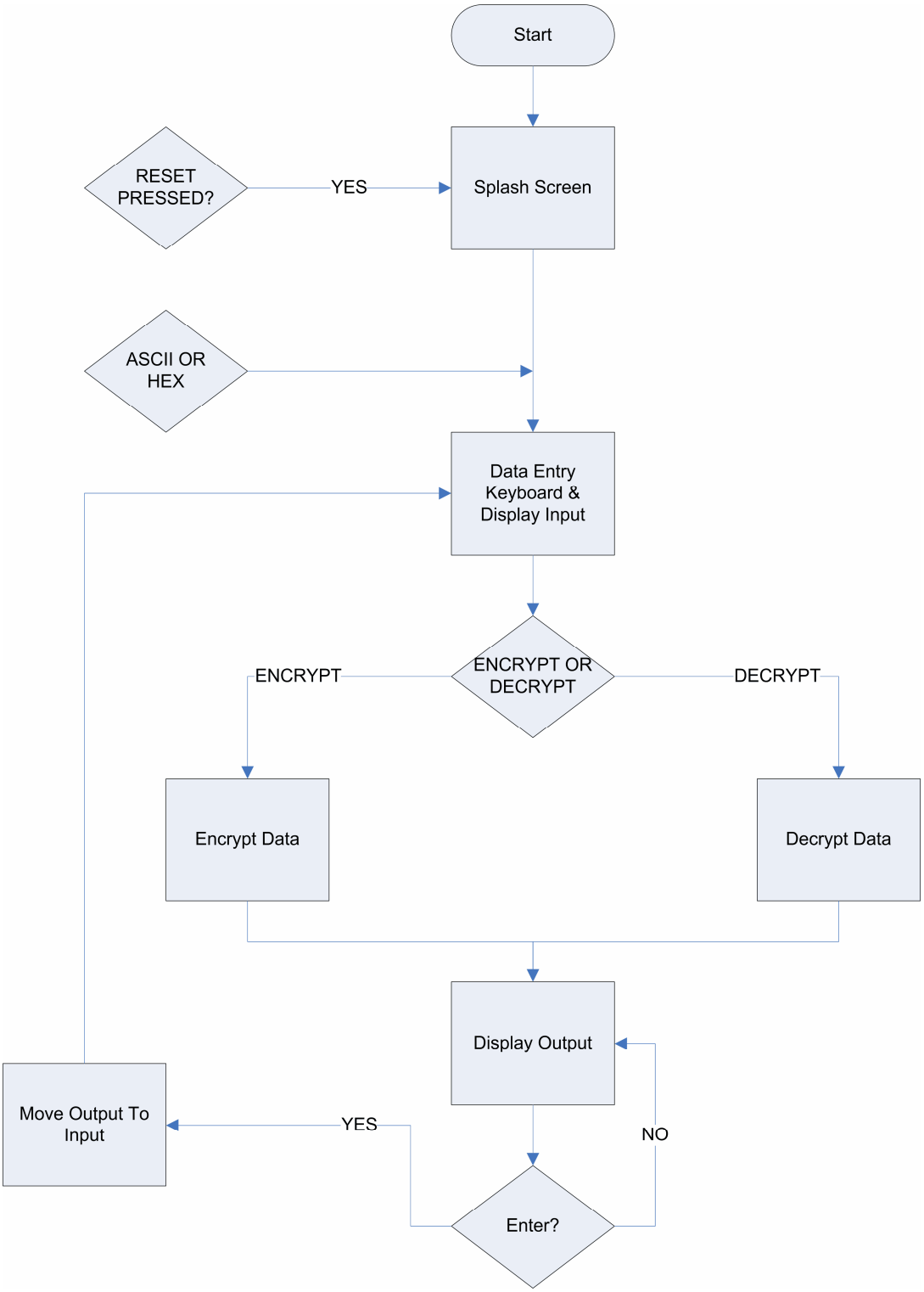
The inputaccept output sends a signal to the AEScore to allow inputs from the keyboard to be saved in the current memory. The intLCD2Core signal outputs an 8 bit vector to the AEScore to assign the output vector to the core2LCD input signal. The diplay_out 8 bit vector is sent to the LCD RAM along with the lcd_rw, lcd_rs, and nlcd_ena signal to drive the LCD. The lcd_rw, lcd_rs, and nlcd_ena control the LCD display while the display_out signal tells the display which character from the CGRAM memory to display.

APPENDIX A

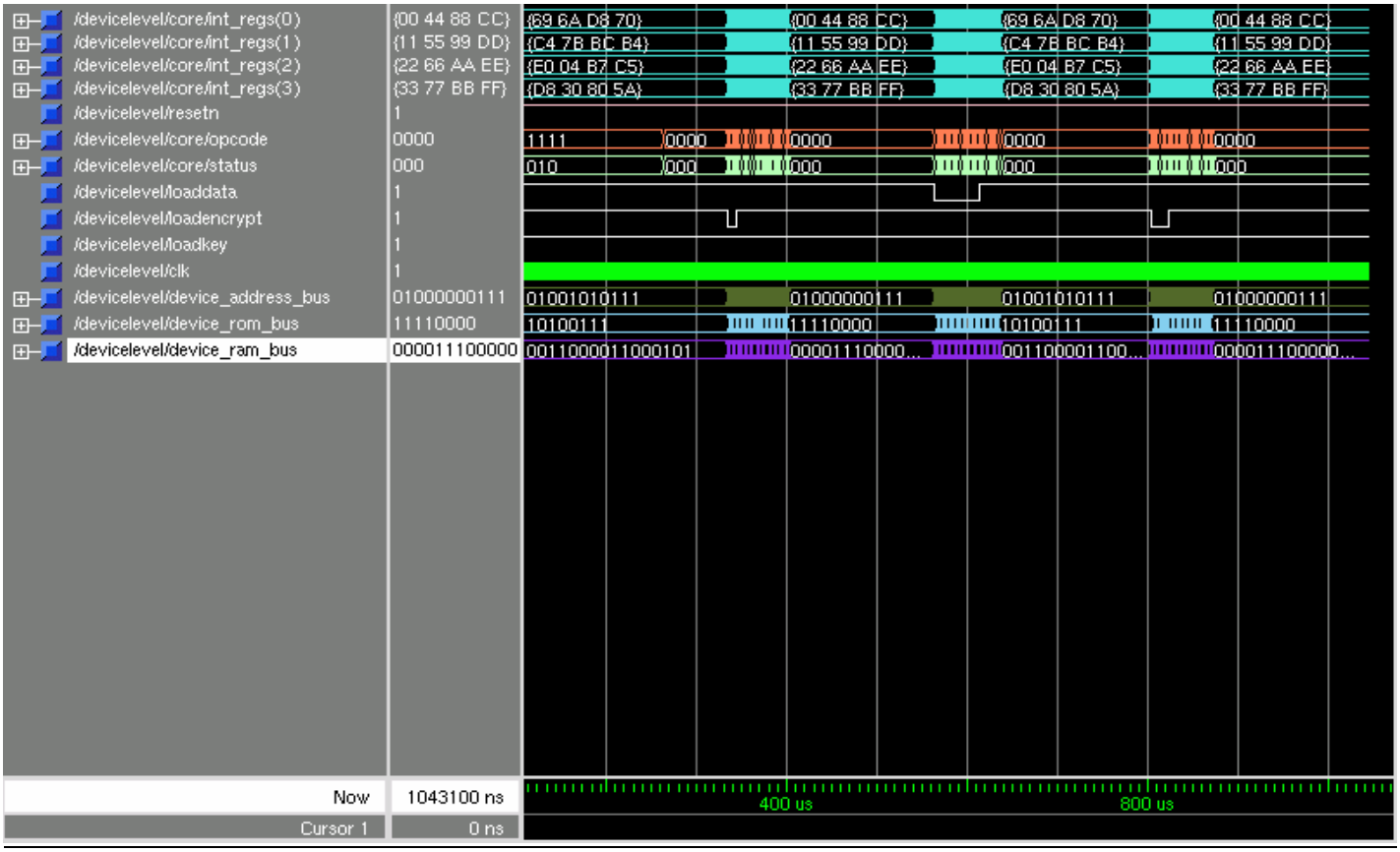
Definitions: [1]

AES	Advanced Encryption Standard
Bit	A binary digit having a value of 0 or 1.
Block	Sequence of binary bits that comprise the input and output. The length of a sequence is the number of bits it contains. Blocks are also interpreted as arrays of bytes.
Byte	A group of eight bits that is treated either as a single entity or as an array of 8 individual bits.
Cipher	Series of transformations that coverts plaintext to ciphertext using the Cipher Key.
Cipher Key	Secret, cryptographic key that is used by the Key Expansion routine to generate a set of Round Keys; can be pictured as a rectangular array of bytes, having four rows and n columns.
Ciphertext	Data output from the Cipher or input to the Inverse Cipher.
Inverse Cipher	Series of transformations that converts ciphertext to plaintext using the Cipher Key.
Key Expansion	Routine used to generate a series of Round Keys from the Cipher Key.
Plaintext	Data input to the Cipher or output from the Inverse Cipher.
Rijndael	Cryptographic algorithm specified in this Advanced Encryption Standard (AES).
Round Key	Round keys are values derived from the Cipher Key using the Key Expansion routine; they are applied to the State in the Cipher and Inverse Cipher.
State	Intermediate Cipher result that can be pictured as a rectangle array of bytes, having four rows and m columns.
Word	A group of 32 bits that is treated either as a single entity or as an array of 4 bytes.

APPENDIX B
Software Flow Chart



APPENDIX C
Computer Simulation (ModelSim 5.8c)



Bibliography

[1] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] http://www.nstissc.gov/Assets/pdf/fact%20sheet.pdf

[3] http://en.wikipedia.org/wiki/AES