FPGA Implementation
AES 128
Encryption/Decryption
Sakinder Ali
Presentation Outline

- FPGA
- VHDL
- Design Flow
- AES128 Implementation
FPGA Overview

Field Programmable Gate Array

Basic idea: two-dimensional array of logic blocks and flip-flops with a means for the user to configure:
1. The interconnection between the logic blocks,
2. The function of each block.

The FPGA LUTs are configured to implement Gates

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Empty Look Up Tables

Inputs | Outputs
---|---
A | B | Cin | S | Cout
---|---|---|---|---
0 | 0 | 0 | 0 | 0
0 | 0 | 1 | 1 | 0
0 | 1 | 0 | 1 | 0
0 | 1 | 1 | 0 | 1
1 | 0 | 0 | 1 | 0
1 | 0 | 1 | 0 | 1
1 | 1 | 0 | 0 | 1
1 | 1 | 1 | 1 | 1

Adding More Functionality

DFF next to a 4–Input LUT to form a "Configurable Logic Block" (CLB)

The basic FPGA functional unit: CLB

The interconnect switches are then programmed to implement the net connections
FPGA

➢ Gates
  ➢ 1987: 9,000 gates, Xilinx
  ➢ 1992: 600,000, Naval Surface Warfare Department
  ➢ Early 2000s: Millions

➢ Market size
  ➢ 1985: First commercial FPGA technology invented by Xilinx
  ➢ 1987: $14 million
  ➢ ~1993: >$385 million
  ➢ 2005: $1.9 billion
  ➢ 2010 estimates: $2.75 billion
FPGAs for Space Applications

NASA/Lockheed X-33 Launch Vehicle

The SpaceWire communications system is driven by low-power chips.

VMC: The Venus Monitoring Camera is a wide-angle, multi-channel CCD. The camera includes an FPGA to pre-process image data, reducing the amount transmitted to Earth.

FPGA used to modify the control algorithm to improve testing, explore shutter issues, and further NASA MEMS microshutter array development.

Over the last decade, the company's FPGAs have been onboard more than 100 launches and flown on over 300 satellites and spacecraft, including GPS, Mars Reconnaissance Orbiter, and the Mars Explorer Rovers 1 and 2 (Spirit and Opportunity).
FPGAs for Space Applications
## Xilinx FPGA Devices

- **Virtex-7-HT**
- **Virtex-6**
- **Virtex-5**
- **Virtex-4**
- **Virtex-II Pro**
- **Spartan-6**
- **Extended Spartan-3A**
- **Spartan-3E**
- **Spartan-3**

### 28Gbps Serial Transceiver Technology

#### Xilinx Virtex-7 HT FPGA Highlights

- Enables communication equipment vendors to develop the integrated, high-bandwidth-efficient.
- Built with four to sixteen 28Gbps transceivers complying with OIF CEI-28G (Optical Internetworking Forum’s Common Electrical I/O).
- Up to seventy-two 13.1Gbps transceivers
- Up to 2.8Tbps full duplex throughput

<table>
<thead>
<tr>
<th>Features</th>
<th>Artix-7</th>
<th>Kintex-7</th>
<th>Virtex-7</th>
<th>Spartan-6</th>
<th>Virtex-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Cells</td>
<td>352,000</td>
<td>480,000</td>
<td>2,000,000</td>
<td>150,000</td>
<td>760,000</td>
</tr>
<tr>
<td>BlockRAM</td>
<td>19Mb</td>
<td>34Mb</td>
<td>85Mb</td>
<td>4.8Mb</td>
<td>38Mb</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>1,040</td>
<td>1,920</td>
<td>5,280</td>
<td>180</td>
<td>2,016</td>
</tr>
<tr>
<td>DSP Performance (symmetric FIR)</td>
<td>1.129GMACS</td>
<td>2,450GMACS</td>
<td>6,737GMACS</td>
<td>140GMACS</td>
<td>2,419GMACS</td>
</tr>
<tr>
<td>Transceiver Count</td>
<td>16</td>
<td>32</td>
<td>96</td>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>Transceiver Speed</td>
<td>6.6Gb/s</td>
<td>12.5Gb/s</td>
<td>28.05Gb/s</td>
<td>3.125Gb/s</td>
<td>11.18Gb/s</td>
</tr>
<tr>
<td>Total Transceiver Bandwidth</td>
<td>211Gb/s</td>
<td>800Gb/s</td>
<td>2,784Gb/s</td>
<td>50Gb/s</td>
<td>536Gb/s</td>
</tr>
<tr>
<td>(full duplex)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Interface (DDR3)</td>
<td>1,066Mb/s</td>
<td>1,866Mb/s</td>
<td>1,866Mb/s</td>
<td>800Mb/s</td>
<td>1,066Mb/s</td>
</tr>
<tr>
<td>PCI Express® Interface</td>
<td>Gen2x4</td>
<td>Gen2x8</td>
<td>Gen3x8</td>
<td>Gen1x1</td>
<td>Gen2x8</td>
</tr>
<tr>
<td>Agile Mixed Signal (AMS)/XADC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Configuration AES</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I/O Pins</td>
<td>600</td>
<td>500</td>
<td>1,200</td>
<td>576</td>
<td>1,200</td>
</tr>
</tbody>
</table>

The company claims it is the fastest product rollout of next generation programmable logic devices built with 28nm technology.
The technology leadership continues with the transceivers in 28-nm Stratix® V FPGAs.

Providing up to over 930 Gbps of transceiver bandwidth, Stratix V FPGAs deliver the highest system bandwidth at the lowest power consumption for a wide range of applications and protocols.
How do you make a “Programmable” circuit?

- **One time programmable**
  - Fuses (destroy internal links with current)
  - Anti-fuses (grow internal links)

- **Reprogrammable**
  - EPROM
  - EEPROM
  - Flash
  - SRAM - volatile
One time programmable Fuses (destroy internal links with current)

Programmable Function

Fusible Link Technology
Antifuse based configuration uses a two terminal device that is electrically programmed to change from an electrical open circuit to an electrical short circuit. The operation is the inverse to that of the fuse. This is a one-time process (i.e., permanent) and once blown, cannot be undone.

(i), the no connection exists between the two metal layers. Once programmed, a low-resistance link (ii) exists between the metal layers and connects them together.

- **Disadvantages**
  - Not reprogrammable; links made are permanent

- **Advantages**
  - Small size
  - Relatively low series resistance
  - Low parasitic capacitance
Xilinx CAD software tool encrypts the bitstream using the powerful Triple Data Encryption (DES) algorithm before downloading the configuration inside the FPGA. Triple DES is the standard used by many governments for safe communication and by banks around the world for money transfers. This algorithm uses three 56–bits public keys. The designer can use random keys or choose their own keys.
SRAM-based FPGAs are non-volatile devices.

- Upon power up, they are required to be programmed from an external source.
- Most FPGA vendors do not publish the definition of the bit-stream. It is therefore very difficult to reverse engineer a design from a configuration bit-stream.

**Disadvantages**
- Volatile
- External Permanent Memory Required
- Large Area Required

**Advantages**
- Reprogrammable, easily and quickly
- Requires only standard integrated circuit process technology (as opposed to Antifuse)
Protection level of some circuits

- **Level 0 (Zero)**
  No special security features added to the system.

- **Level 1 (Low)**
  Some security features in place.

- **Level 2 (Medium Low)**
  More expensive tools are required, as well as specialized knowledge.

- **Level 3 (Medium)**
  Special tools and equipment are required, as well as some special skills and knowledge. The attack may become time-consuming but will eventually be successful.

- **Level 4 (Medium High)**
  Equipment is available but is expensive to buy and operate.

- **Level 5 (High)**
  All known attacks have been unsuccessful

<table>
<thead>
<tr>
<th>Integrated Circuit</th>
<th>Security Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional SRAM FPGA</td>
<td>1</td>
</tr>
<tr>
<td>ASIC Gate Array</td>
<td>3</td>
</tr>
<tr>
<td>Cell-based ASIC</td>
<td>3</td>
</tr>
<tr>
<td>SRAM FPGA with bitstream encryption</td>
<td>4</td>
</tr>
<tr>
<td>Flash FPGA</td>
<td>5</td>
</tr>
<tr>
<td>Antifuse FPGA</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1. Security level of classical integrated circuits

Actel - Security [7]
What are the Guts of an FPGA?

- Basic Components
  - LUT (look-up-table)
  - Flip-Flops
  - Multiplexers
  - I/O Blocks
  - Programmable switching matrices
  - Interconnect
  - Clocks

![Diagram of FPGA components](image)
How do you program an FPGA?

- Create a circuit design
  - Graphic circuit tool
  - Verilog
  - VHDL
  - AHDL

- Vendor Tools
What is VHDL?

- **VHDL**: VHSIC Hardware Description Language
- **VHSIC**: Very High Speed Integrated Circuit
  - Developed originally by DARPA
    - for specifying digital systems
  - International IEEE standard (IEEE 1076-1993)
  - Practical benefits:
    - A mechanism for digital design and reusable design documentation
    - Model interoperability among vendors
    - Third party vendor support
    - Design re-use.
Why VHDL?

- It allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).

- Allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time.

- VHDL project is multipurpose. Being created once, a calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned (capacity parameters, memory size, element base, block composition and interconnection structure).

- VHDL project is portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies.
Basic VHDL Concepts

- Interfaces
- Behavior
- Structure
- Test Benches
- Analysis, elaboration, simulation
- Synthesis
entity FullAdder is
  port (X, Y, Cin: in bit, -- Inputs
    Cout, Sum: out bit); -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin -- Concurrent Assignments
  Sum <= X xor Y xor Cin after 10 ns;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in  std_logic;
    clk : in  std_logic;
    a : in  std_logic_vector;
    b : in  std_logic_vector;
    q : out std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide
begin -- architecture
  assert (a'length >= b'length)
  report "Port A must be the longer vector if different sizes!"
  severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
  begin
    if (aclr = '1') then
      q_s <= (others => '0');
    elsif rising_edge(clk) then
      q_s <= ('0'&signed(a)) + ('0'&signed(b));
    end if; -- clk'd
  end process;
end signed_adder arch;
Design Flow

Requirements to Define
* Packaging
* Interfaces
  - Power
  - Clocks/Reset
  - Configuration
  - Dataflow/Processing
  - Control
  - Memory
* Configuration
* Dataflow/Processing
* Control
* Testability
* Environmental

Detailed Design Tasks
* Module Block Diagrams (RTL)
* State Machine Diagrams
* Sizing Estimation (Update)
* Power Estimation (Update)
* Clock/Reset Distribution (Update)
* Software Interface Definition (Update)
* Simulation/Test Planning (Update)
* Risks/Mitigations (Update)

Preliminary Design Tasks
* Interface Definition
* I/O Estimation
* FPGA Block Diagram
* Sizing Estimation
* Power Estimation
* Clock/Reset Distribution
* Software Interface Definition
* Design/Simulation/Test Planning
* Risks/Mitigations

Module Level VHDL Coding/Simulation Tasks
* Core Generation
* Dataflow Module Coding
* Dataflow Simulation
  - Module Level
  - Integrated (if Pipelined)
* Power Estimation (Update)
* Control Module Coding
* Control Module Simulation

Top Level VHDL Coding/Simulation Tasks
* Top Level Dataflow Coding
* Top Level Control Coding
* Clock/Reset Module Coding
* Top Level Coding
* Top Level Simulation

Synthesis Tasks
* Synthesis Constraints Generation
* FPGA Data Entry
* Synthesis Tool Options Entry
* Synthesis

Implementation Tasks
* Implementation Constraints Generation
  - Physical Constraints
  - Timing Constraints
* FPGA Data Entry
* Implementation Tool Options Entry
* Implementation

To System Test
Design Entry/RTL Coding
  – Behavioral or Structural Description of Design

RTL Simulation
  – Functional Simulation
  – Verify Logic Model & Data Flow (No Timing Delays)

Synthesis
  – Translate Design into Device Specific Primitives
  – Optimization to Meet Required Area & Performance Constraints

Place & Route
  – Map Primitives to Specific Locations inside Target Technology with Reference to Area & Performance Constraints
  – Specify Routing Resources to Be Used
**Typical FPGA design flow**

- **Timing Analysis**
  - Verify Performance Specifications Were Met
  - Static Timing Analysis

- **Gate Level Simulation**
  - Timing Simulation
  - Verify Design Will Work in Target Technology

- **PC Board Simulation & Test**
  - Simulate Board Design
  - Program & Test Device on Board
System–level debugging
On chip testing

- Logic analysis system provides high-performance, system–level debugging of digital designs.
- Track down cross domain analog and digital problems by time–correlating logic analysis systems.
Advance FPGA design flow

USER CREATED
- IP
- HDL Modules
- Glue-Logic Modules
- Schematic Modules
- DSP Subsystems

3RD PARTY CREATED
- IP
- Subsystems
- Applications Solutions

STANDARD LIBERO IDE DESIGN FLOW
- Synthesis
- Simulation
- Place-and-Route
- Timing Constraints and Analysis
- Power Reduction and Analysis
- Programming File Generation
- Device Programming

Typical FPGA design flow
RTAX–DSP space–flight FPGA

High performance at densities of up to 4 million equivalent system gates and 840 user I/Os for space–based applications.

The RTAX–DSP family features up to 120 mathblocks, each capable of operating at 125 MHz over the full military temperature range (−55°C to 125°C), for a total throughput of 15 billion multiply/accumulates per second (15 GMACS).

Key Features
• Highly reliable, nonvolatile antifuse technology
• 250,000 to 500,000 ASIC gates (2 to 4 million system gates)
• Up to 120 DSP mathblocks with 125 MHz 18 bit x 18 bit multiply–accumulate
• Up to 540 kbits of embedded memory with optional EDAC protection
• Up to 840 user–programmable I/Os
Are there clear requirements for the effort to be outsourced?

- Interfaces to the system (data, clock, control, and status) including width (serial or parallel)
- Functional blocks in system
- Interfaces between blocks including data width
- Data rates into the system and between blocks (data rates are tied to data widths)
- Clock domains into system and between blocks
- Memory (size, speed, and width)
- Standards/specs
- Timing diagrams (setup, hold, latency tolerance)
- Schedule.
The FPGA is one of the most popular logic circuit components and has revolutionized the way digital systems are designed. Some FPGA advantages include:

- Low-cost
- Fast-turnaround prototype implementation
- Supported by CAD/EDA tools
- High density
- High speed
- Programmable and versatile
- Flexible
- Reusable
- Large amounts of logic gates, registers, RAM and routing resources
- Quick time-to-market
- SRAM FPGA provide the benefits of custom CMOS
ALTERA DE-270

- USB Device Port
- Mic in
- Line In
- Line Out
- Video In 1
- Video In 2
- VGA Out
- Ethernet 10/100M Port
- RS-232 Port
- TV Decoder (NTSC/PAL) X2
- PS2 Port
- VGA 10-bit DAC
- Ethernet 10/100M Controller
- 50Mhz Oscillator
- Expansion Header 2
- Expansion Header 1
- SD Card Slot (SD Card Not Included)
- Altera Cyclone II FPGA with 70K LEs
- IrDA Transceiver
- 8Mbyte Flash Memory
- 8 Green LEDs
- SMA External Clock
- 32Mbyte SDRAMx2
- 26Mhz Oscillator
- 2Mbyte SSRAM
- 4 Push-button Switches

- 12V DC Power Supply Connector
- Power ON/OFF Switch
- USB Host/Slave Controller
- Audio CODEC
- Altera USB Blaster Controller chipset
- Altera EPCS16 Configuration Device
- RUN/PROG Switch for JTAG/AS Modes
- 16x2 LCD Module
- 7-Segment Displays
- 18 Red LEDs
- 18 Toggle Switches
Using Altera Cyclone II For Encryption/Decryption
Design Architecture

128bit AES Encryption CORE

Program Control
- Opcode(0:3)
- Inverse

PS2 Keyboard Interface
- Rx ascii
- Rx Data Ready
- Rx Released
- Rx Read
- Input Accept

LCD Interface
- LCD(0:15)

Data Memory BUS
- ROM(0:7)

Miscellaneous
- Clock
- Reset

Status(0:2)

Ascii Mode

LCD Out

RAM Bus(0:15)

Data Address(0:10)

Load Data

Load Encrypt

Load Key

Leds(0:7)
library ieee;
use ieee.std_logic_1164.all;

entity SBOX_ROM is
  port (address: in std_logic_vector(7 downto 0);
       SBox: in std_logic;
       Output: out std_logic_vector(7 downto 0);
       Read: in std_logic;
       Inverse: in std_logic);
end SBOX_ROM;

architecture Inside_SBOX of SBOX_ROM is
  type box is array (0 to 15) of std_logic_vector(7 downto 0);
  constant SBOX box : box := (
    "00", "01", "02", "03", "04", "05", "06", "07",
    "08", "09", "0A", "0B", "0C", "0D", "0E", "0F",
    "10", "11", "12", "13", "14", "15", "16", "17",
    "18", "19", "1A", "1B", "1C", "1D", "1E", "1F",
    "28", "29", "2A", "2B", "2C", "2D", "2E", "2F",
    "30", "31", "32", "33", "34", "35", "36", "37",
    "38", "39", "3A", "3B", "3C", "3D", "3E", "3F",
    "40", "41", "42", "43", "44", "45", "46", "47",
    "48", "49", "4A", "4B", "4C", "4D", "4E", "4F",
    "50", "51", "52", "53", "54", "55", "56", "57",
    "58", "59", "5A", "5B", "5C", "5D", "5E", "5F",
    "60", "61", "62", "63", "64", "65", "66", "67",
    "68", "69", "6A", "6B", "6C", "6D", "6E", "6F",
    "70", "71", "72", "73", "74", "75", "76", "77",
    "78", "79", "7A", "7B", "7C", "7D", "7E", "7F",
    "80", "81", "82", "83", "84", "85", "86", "87",
    "88", "89", "8A", "8B", "8C", "8D", "8E", "8F",
    "90", "91", "92", "93", "94", "95", "96", "97",
    "B0", "B1", "B2", "B3", "B4", "B5", "B6", "B7",
    "B8", "B9", "BA", "BB", "BC", "BD", "BE", "BF",
    "C8", "C9", "CA", "CB", "CC", "CD", "CE", "CF",
    "D0", "D1", "D2", "D3", "D4", "D5", "D6", "D7",
    "D8", "D9", "DA", "DB", "DC", "DD", "DE", "DF",
    "E8", "E9", "EA", "EB", "EC", "ED", "EE", "EF",
    "F0", "F1", "F2", "F3", "F4", "F5", "F6", "F7",
    "F8", "F9", "FA", "FB", "FC", "FD", "FE", "FF"
  );

  constant INVERSE_SBOX : box := (
    "00", "01", "02", "03", "04", "05", "06", "07",
    "08", "09", "0A", "0B", "0C", "0D", "0E", "0F",
    "10", "11", "12", "13", "14", "15", "16", "17",
    "18", "19", "1A", "1B", "1C", "1D", "1E", "1F",
    "28", "29", "2A", "2B", "2C", "2D", "2E", "2F",
    "30", "31", "32", "33", "34", "35", "36", "37",
    "38", "39", "3A", "3B", "3C", "3D", "3E", "3F",
    "40", "41", "42", "43", "44", "45", "46", "47",
    "48", "49", "4A", "4B", "4C", "4D", "4E", "4F",
    "50", "51", "52", "53", "54", "55", "56", "57",
    "58", "59", "5A", "5B", "5C", "5D", "5E", "5F",
    "60", "61", "62", "63", "64", "65", "66", "67",
    "68", "69", "6A", "6B", "6C", "6D", "6E", "6F",
    "70", "71", "72", "73", "74", "75", "76", "77",
    "78", "79", "7A", "7B", "7C", "7D", "7E", "7F",
    "80", "81", "82", "83", "84", "85", "86", "87",
    "88", "89", "8A", "8B", "8C", "8D", "8E", "8F",
    "90", "91", "92", "93", "94", "95", "96", "97",
    "B0", "B1", "B2", "B3", "B4", "B5", "B6", "B7",
    "B8", "B9", "BA", "BB", "BC", "BD", "BE", "BF",
    "C8", "C9", "CA", "CB", "CC", "CD", "CE", "CF",
    "D0", "D1", "D2", "D3", "D4", "D5", "D6", "D7",
    "D8", "D9", "DA", "DB", "DC", "DD", "DE", "DF",
    "E8", "E9", "EA", "EB", "EC", "ED", "EE", "EF",
    "F0", "F1", "F2", "F3", "F4", "F5", "F6", "F7",
    "F8", "F9", "FA", "FB", "FC", "FD", "FE", "FF"
  );

begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      if (Read = '1') then
        if (Inverse = '0') then
          Output <= SBox(Select_SubByte(Address(7 downto 4))).Select_SubByte(Address(3 downto 0));
        else
          Output <= INVERSE_SBox(Select_SubByte(Address(7 downto 4))).Select_SubByte(Address(3 downto 0));
        end if;
      end if;
    end if;
  end process;
end Inside_SBOX;
when x"13e" => ascii <= x"2a"; -- *,
when x"155" => ascii <= x"2b"; -- +,
when x"041" => ascii <= x"2c"; -- .,
when x"04e" => ascii <= x"2d"; -- -,
when x"049" => ascii <= x"2e"; -- ,
when x"04a" => ascii <= x"2f"; -- /,
when x"045" => ascii <= x"30"; -- 0,
when x"016" => ascii <= x"31"; -- 1,
when x"01e" => ascii <= x"32"; -- 2,
when x"026" => ascii <= x"33"; -- 3,
when x"025" => ascii <= x"34"; -- 4,
when x"02e" => ascii <= x"35"; -- 5,
when x"036" => ascii <= x"36"; -- 6,
when x"03d" => ascii <= x"37"; -- 7,
when x"03e" => ascii <= x"38"; -- 8,
when x"046" => ascii <= x"39"; -- 9,
when x"14c" => ascii <= x"3a"; -- :,
when x"04c" => ascii <= x"3b"; -- ;
when x"141" => ascii <= x"3c"; -- <
when x"055" => ascii <= x"3d"; -- =,
when x"149" => ascii <= x"3e"; -- >,
when x"14a" => ascii <= x"3f"; -- ?
when x"11e" => ascii <= x"40"; -- @,
when x"11c" => ascii <= x"41"; -- A,
when x"132" => ascii <= x"42"; -- B,
when x"121" => ascii <= x"43"; -- C,
when x"123" => ascii <= x"44"; -- D,
when x"124" => ascii <= x"45"; -- E
library ieee;
use ieee.std_logic_1164.all;
use work.global.all;

description ENTITY KEY_RAM IS
    port (address : in std_logic_vector( 6 downto 0 );
    KEY_bus : inout std_logic_vector( 15 downto 0 );
    clk : in std_logic;
    read : in std_logic;
    write : in std_logic;
    enable : in std_logic);
end KEY_RAM;

architecture internals of KEY_RAM is
    type mem_array is array
    (integer range 0 to 15, integer range 0 to 7) of
    std_logic_vector(15 downto 0);
    constant KEYSCHED : mem_array := ((
        x"0001", x"0203", x"0405", x"0607", x"0809", x"0a0b", x"0c0d", x"0e0f"),
        (x"d5aa", x"74fd", x"d2af", x"72fa", x"dafa", x"78f1", x"d6ab", x"76fe"),
        (x"b592", x"cf0b", x"643d", x"bdf1", x"b90b", x"c500", x"e030", x"b3e"),
        (x"b5ff", x"744e", x"d2c2", x"c9bf", x"6c59", x"0cbf", x"0469", x"bf41"),
        (x"47f7", x"f7bc", x"9535", x"3e03", x"f6bc", x"32bc", x"fd05", x"8dfe"),
        (x"3ca8", x"a3e8", x"a99f", x"9deb", x"50f3", x"af57", x"a9f6", x"22aa"),
        (x"5e39", x"017d", x"f7a6", x"929e", x"a755", x"3dca", x"0aa8", x"1f6b"),
        (x"14f9", x"701a", x"e35f", x"e20c", x"440a", x"df4a", x"4e8f", x"c026"),
        (x"4743", x"8735", x"a41c", x"65b9", x"e016", x"ba4f", x"ae8f", x"7ad2"),
        (x"5499", x"32d1", x"f005", x"5760", x"1093", x"e9c0", x"be2c", x"974e"),
        (x"1311", x"1d7f", x"e394", x"4a17", x"f307", x"a78b", x"d2b", x"30c5"));

begin

process(clk, enable)
begin
    if(enable='0') then
        KEY_bus<=(others=>'Z');
    elsif(clk='1'and enable='0') then
        if(read='1'and write='0') then
            KEY_bus<=KEYSCHED(Select_SubByte(address(6 downto 3)),Select_SubByte(address(2 downto 0)));
        end if;
    end if;
end process;

end internals;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MixColumns is
  Port (SYS_CLK, RST : in STD_LOGIC;
        DATA_IN : in STD_LOGIC_VECTOR(127 downto 0);
        DATA_OUT : out STD_LOGIC_VECTOR(127 downto 0));
end MixColumns;

architecture Behavioral of MixColumns is
  COMPONENT Column_of_Matrix
    Port (SYS_CLK : in std_logic;
          RST : in std_logic;
          COLUMN_IN : in std_logic_vector(31 downto 0);
          COLUMN_OUT : out std_logic_vector(31 downto 0)
    )
  BEGIN
  END COMPONENT;
BEGIN
  Column_of_Matrix_0: Column_of_Matrix PORT MAP(
    SYS_CLK => SYS_CLK,
    RST => RST,
    COLUMN_IN => DATA_IN(127 downto 96),
    COLUMN_OUT => DATA_OUT(127 downto 96)
  );
  Column_of_Matrix_1: Column_of_Matrix PORT MAP(
    SYS_CLK => SYS_CLK,
    RST => RST,
    COLUMN_IN => DATA_IN(95 downto 64),
    COLUMN_OUT => DATA_OUT(95 downto 64)
  );
  Column_of_Matrix_2: Column_of_Matrix PORT MAP(
    SYS_CLK => SYS_CLK,
    RST => RST,
    COLUMN_IN => DATA_IN(63 downto 32),
    COLUMN_OUT => DATA_OUT(63 downto 32)
  );
  Column_of_Matrix_3: Column_of_Matrix PORT MAP(
    SYS_CLK => SYS_CLK,
    RST => RST,
    COLUMN_IN => DATA_IN(31 downto 0),
    COLUMN_OUT => DATA_OUT(31 downto 0)
  );
end Behavioral;

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

\[
\begin{align*}
s'_{0,c} &= (02 \cdot s_{0,c} \oplus (03 \cdot s_{1,c} \oplus s_{2,c} \oplus s_{3,c}) \\
s'_{1,c} &= s_{0,c} \oplus (02 \cdot s_{1,c} \oplus (03 \cdot s_{2,c}) \oplus s_{3,c}) \\
s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (02 \cdot s_{2,c} \oplus (03 \cdot s_{3,c}) \\
s'_{3,c} &= (03 \cdot s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus (02 \cdot s_{3,c}).
\end{align*}
\]
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity AddRoundKey is
  Port ( Data_IN : in  STD_LOGIC_VECTOR (127 downto 0);
         Key_IN  : in  STD_LOGIC_VECTOR (127 downto 0);
         Data_OUT : out  STD_LOGIC_VECTOR (127 downto 0);
         SYS_CLK, RST : in  STD_LOGIC);
end AddRoundKey;

architecture Behavioral of AddRoundKey is

begin
  XOR_PARTIALKEY_WITH_PLAINTEXT : PROCESS(SYS_CLK)
  begin
    IF (SYS_CLK'event AND SYS_CLK = '1') then
      IF RST = '1' then
        Data_OUT <= (OTHERS => '0');
      ELSE
        Data_OUT <= Data_IN XOR Key_IN;
      END IF;
    END IF;
  END PROCESS;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ShiftRows is
  Port ( ShiftRows_In : in STD_LOGIC_VECTOR (127 downto 0);
         ShiftRows_Out : out STD_LOGIC_VECTOR (127 downto 0);
         Sys_Clk,RST : in STD_LOGIC);
end ShiftRows;

architecture Behavioral of ShiftRows is

begin
  SHIFT_ROWS_MIXING : PROCESS(SYS_CLK)
  begin
    IF (SYS_CLK'event AND SYS_CLK = '1') then
   IF RST = '1' then
      ShiftRows_Out <= (OTHERS => '0');
   ELSE
      ShiftRows_Out <= ShiftRows_In(127 downto 120) &
                       ShiftRows_In(87 downto 80) &
                       ShiftRows_In(47 downto 40) &
                       ShiftRows_In(7 downto 0) &
                       ShiftRows_In(95 downto 88) &
                       ShiftRows_In(55 downto 48) &
                       ShiftRows_In(15 downto 8) &
                       ShiftRows_In(103 downto 95) &
                       ShiftRows_In(63 downto 56) &
                       ShiftRows_In(23 downto 16) &
                       ShiftRows_In(111 downto 104) &
                       ShiftRows_In(71 downto 64) &
                       ShiftRows_In(31 downto 24) &
                       ShiftRows_In(119 downto 112) &
                       ShiftRows_In(79 downto 72) &
                       ShiftRows_In(39 downto 32);
   END IF;
  END IF;
end PROCESS;

end Behavioral;
REFERENCES

- http://www.altera.com/
- http://www.xilinx.com/
- http://www.home.agilent.com/agilent
- http://www.esa.int/SPECIALS/Venus_Express/
QUESTIONS?