http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 4 (3): 262-270, 2005
ISSN 1812-5638

A Multi-FPGA Rapid Prototyping System with the Reusable AES Core

[1,2]Fang-Hsi Kuo, [2]Shou-Te Yen and [2]Chia-Cheng Liu
[1]Department of Information Management,
Jin Wen Institute of Technology, Taiwan, Republic of China
[2]Computer System Laboratory,
Department of Information Engineering and Computer Science,
Feng Chia University, Taiwan, Republic of China

Abstract: In this study, we develop a reconfigurable rapid prototyping system with PCI as interface. Reconfigurable processing unit uses I/O coupling way with general propose processor to work in coordination and to accelerate the execution of the specific task. Use four FPGA chip in order to offer the hardware design environment under multi-FPGA structures systematically at the same time. This system except that the intact hardware is designed and implemented, but also include the setting-up of the driver with offer the application program interface which access the hardware. In order to prove that systematic function of rapid prototyping board is correct, design one IP Core to apply to this system. We implement an Advanced Encryption Standard (AES) hardware circuit for this goal. The focal point designed lies in making optimization to resources of FPGA and AES suitability in reconfigurable computing with multi-FPGA system.

Key words: Reconfigurable rapid prototyping, multi-FPGA system, advanced encrypt standard

INTRODUCTION

In recent years, the scale of the multimedia and communication system have been already more and more complicate and functions offered are more and more diversified. The complexity of per forming the algorithm of the core increases constantly. The system usually has a large amount of work computing, need real-time processing and has ability to computing fast in some specific application.

There are two common approachs for traditional computing. The first is to use general purpose architecture such as microprocessor. It provides a flexible computing platform and capable of perform a large application. By changing the software, the functions of the system task is altered without change the hardware. However, a sequential procedure for fetch, decode and execution instruction with access memory limit the performance that be using this architecture.

The second is use Application Specific In tegrated Circuit (ASIC) to perform the operations in hardware. It is designed specifically to perform a given computing. So, ASIC is very fast and efficient when executing the exact computation. But, ASIC restrict the flexibility of the architecture. If any part of its circuit requires modification then must be redesign and refabrication.

A new paradigm using reconfigurable computing architecture[1-3] promises an intermediate tradeoff between software and hardware, can achieve potentially higher performance than soft ware and higher flexibility than hardware. Usually the reconfigurable computing system is target on SRAM-based Field Programmable Gate Array (FPGA)[4]. It acts as Reconfigurable Processing Unit (RPU) that coupled to a host processor. Therefore, the dedicated hardware couple with a host processor in order to accelerate some computationally intensive tasks.

The application of Multi-FPGA system can be used in emulating for logic circuit design[5-7]. Before it becomes the last products or one customer chip is designed, the behavior operation of the circuit accord with the anticipated design, must verify via some auxiliary ways within it. Traditional method uses software simulation or make prototyping chip. Offer the Computer-Aided Design (CAD) or the Electronic Design Automation (EDA) tools of function simulation can test the execution result of the circuit.

Corresponding Author: Fang-Hsi Kuo, Department of Information Management, Jin Wen Institute of Technology, Taiwan, Republic of China

They include the verification in timing and functional. But the design of the chip is more and more complicated in present; the range that the simulation software can be verified is limited. They include rational execution time; can provide expression of the right test pattern and simulation platform environment with good efficiency, etc. The prototyping chip can get real verification, but making that must be repeated before getting rid of all mistakes. This procedure is very expensive cost. So, use software simulation or make prototyping chip to designing debug repeatedly, it is very high in the time and money cost. If adopt reconfigurable prototyping system to verify the exactness that the circuit is designed, can reduce the cost of the time and money that offer better solution.

We design an IP core of Advanced Encryption Standard circuit to finish a prototyping in reconfigurable computing system. AES is one that has complicated algorithm and a large amount computing of encryption/decryption circuit. Especially, AES is apt to realize on the hardware. We adopt the way in which encryption and decryption design separately. When the main system should carry out the operation work of encrypt or decrypt, then the essential configuration of hardware is in RPU.

THE PCI-mFCU RECONFIGURABLE RAPID PROTOTYPING SYSTEM

The PCI-mFCU (PCI-multiple FPGA Configurable Unit) reconfigurable rapid prototyping system is a platform of reconfigurable computing environment. The general organization is shown in Fig. 1. It is a logic emulator applying to verify IP. The whole system includes a prototyping board, device driver and hardware access API. This prototyping board adopts 32 bits PCI bus interface to connect with host processor. Support PCI r2.1 specification, slave mode and working frequency is 33 MHz. The system uses four FPGAs to realize RPU and communication with mesh interconnection each other.

The PCI-mFCU board has four interconnected Xilinx FPGA Spartan-II XC2S100, a PLX PCI-9052 and configuration ROM. The PCI-9052 is a commercial PCI controller. It provides a compact high performance PCI bus slave interface for this board and that can be programmed to connect directly to the Local bus. Its configuration, which at boot time is loaded from a EEPROM 93LC46. Both I/O and memory mapped and allows PCI transactions through software libraries. By using PLX9052, we can access PCI bus without knowning the PCI specification. The function of Xilinx

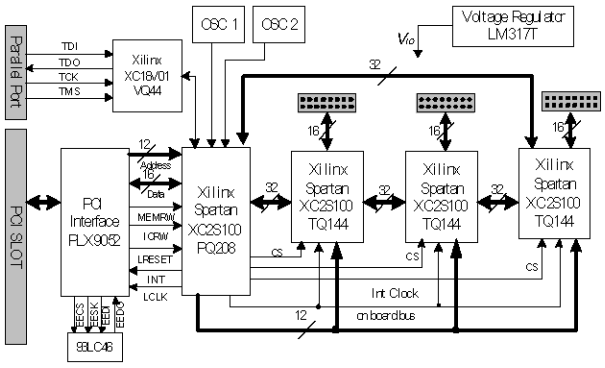

Fig. 1: System block diagram of PCI-mFCU

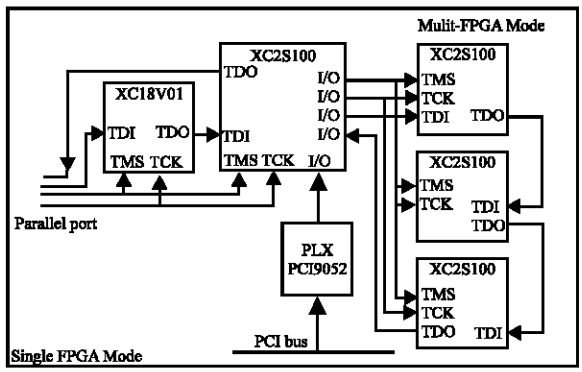

Fig. 2: Configuration bitstream data path

FPGA Spartan-II is to implement user design. One FPGA is connect PCI-9052 directly and can at boot time is loaded from a Xilinx PROM XC18V01 or configure via parallel port. It handles three more FPGA's configuration via PCI bus and all FPGAs be used for user designs.

In prototyping board, the RPU can use two kinds of work mode. The first is realized with single a FPGA, is connected to the local bus on PCI controller directly. The design which applies the circuit is enough to hold in single a FPGA. The second kind is configuring four FPGAs as a group. Become the module of RPU of a Multi-FPGA. It can apply to large circuit design and will cut several parts to put in different FPGA.

The wiring on the circuit board, connect fixed with 32 bits line between FPGA with direct communicate each other. In addition, connect up the bus of 12 bits width on the circuit board. It includes 8 bits data line and control signal that line is to every FPGA. In order to increase the flexibility for system, that each one FPGA includes one 16 bit extension connect header. It can connect other I/O equipment or the outer module like the daughter card with special demand of the circuit design.

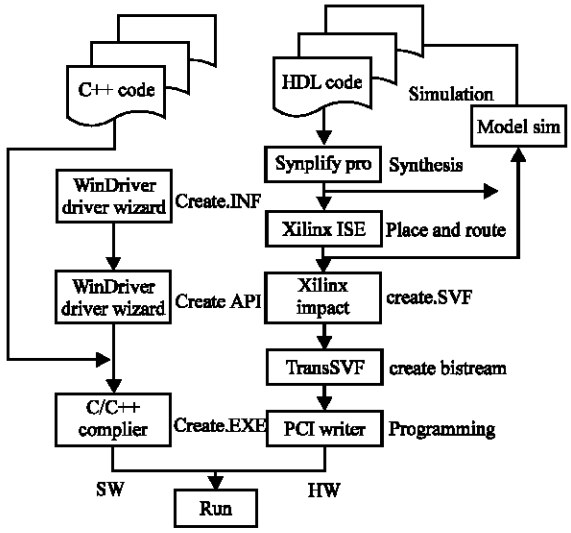

Fig. 3: Hardware/Software design flow

The configuration of the circuit adopts different ways in single FPGA mode and multiple FPGA mode (Fig. 2), but both support programming FPGA by JTAG. Usually JTAG of FPGA is pin of specific but not user's programming, thus connecting directly will be limited bandwidth in PCI Bus. The FPGA connected with PCI controller directly adopts parallel port to be made for the interface of JTAG. So, single FPGA mode used parallel port to come to programming. In multi-FPGA mode, used PCI bus to convey configuration bitstream. The prerequisite is embedded a controller within first FPGA that is connected with PCI interface.

The user design flow in PCI-mFCU system shown in Fig. 3. In hardware design flow, the similar in general FPGA system. User design can be classified into schematic entry tools and Hardware Description Language (HDL). And use any synthesis tools to translate and optimal design that outputting such as netlist file. The next step is implementation to map, place and route the design onto the FPGA via FPGA's vendor provided tools. Because the system adopts PCI bus interface to come to configure FPGA and it unable to support by download tool of Xilinx. Thus, offers a configuration utility program developed by our Lab that it called PCIWriter. A Serial Vector Format file produced by Xilinx download tool that has information with boundary scan. We can transfer a bitstream file and offer PCIWriter to perform configuration. So the whole hardware design flow is making up by some front-end design tools, back-end design tool of Xilinx and our offered utility program.

This system has software libraries provided a C/C++ interface to the PCI-mFCU Board. This main function of libraries and device driver are produced by WinDriver Wizard. The libraries API allow the user to interface to the PCI card from their own C/C++ code with a simple function calls. The API contains all the PCI-9052 chip access function, including the reading/writing of the registers and loading and accessing the user design. A description of each of the some API follows.

- int PCI-mFCU_Check (): checks the PCI-mFCU board has been correctly initialized through the device driver interface.
- int loadDatFile (*filename): this function is used to load a bitstream file into the Xilinx FPGA. The bitstream file is responsible for configuring the circuit in the FPGA and is loaded by a series of boundary scan.
- void PCI-mFCU_Write (addr, dwOffset, data): data writes to the board, the addr can be I/O or memory mapping register address.
- word PCI-mFCU_Read (addr, dwOffset): data read from the board, the return value also can set a byte or double word.
- void PCI-mFCU_Reset (): software reset the board by write control register.
- int PCI-mFCU_Interrupt (): access interrupt on board by read control register.

AES IP CORE DESIGN

The Advanced Encryption Standard[8] is a round based and symmetric block cipher. The algorithm is Rijndael algorithm[9] that process data blocks size of 128 bits using the cipher key of length 128, 192 or 256 bits. The number of algorithm round times depends on the block size and key length. The algorithm operations are performed in a 4x4 array of bytes of State.

In AES algorithm, all bytes are interpreted as elements of the finite field GF (2^8) and using the polynomial representation. The AES are based on four different transformations that are performed repeatedly in a certain sequence. Each of these transformations and hardware implementation are described in the follow.

The cipher operation is described in the pseudo code as follow:

```
Cipher ( byte in[4xNb], byte out[4xNb], word w[Nbx (Nr+1)])
Begin
    byte state[4, Nb];
    state = in;
    AddRoundKey ( state, w[0, Nb-1]);
    for round = 1 step 1 to Nr-1
        SubBytes ( state);
        ShiftRows ( state);
        MixColumns ( state);
        AddRoundKey ( state, w[roundxNb, (round+1)xNb-1]);
    end for
    SubBytes ( state);
    ShiftRows ( state);
    AddRoundKey ( state, w[NrxNb, (Nr+1)xNb-1]);
    out = state;
end
```

Algorithmic implementation

SubBytes/InvSubBytes transformation: These transformations are non-linear bytes substitutions that performed independently on each byte of the State using an S-box table. This table is constructed by composing two transformations. Take the multiplicative inverse in the GF (2^8) and apply affine transformation for each byte. In matrix form, the affine transformation element of the S-box can be expressed as:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Implement each S-box uses a 256x8 bits look-up table as a 2048 bits of memory and to execute 128 bits in parallel need 16 S-box (Fig. 4). S-box was implemented in the FPGA's embedded memory. Two S-boxes or one S-box and its inverse fit into one 4096 bits block RAM of Xilinx Spartan-II device.

In the circuit which combines encryption and decryption, need two S-boxes. One is S-box for encryption and another is inverse S-box for decryption. These two tables are not the same. So, we must build two different memories to store the table. This will cost a lot of area in hardware implementation and desirable to obtain a simplified method to reduce the hardware area.

The S-box and the inverse S-box can be computed by implementing the affine and inverse affine transformation and together with a look-up table for multiplicative inverse. Therefore, an additional multiplexer is used to switch the data path for encryption or decryption. The block diagram is look Fig. 5.

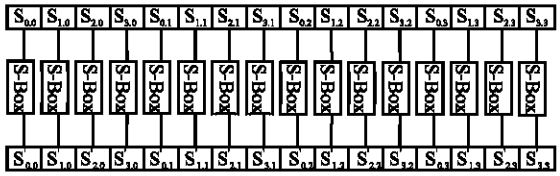

Fig. 4: Parallel implement SubBytes () Transformation

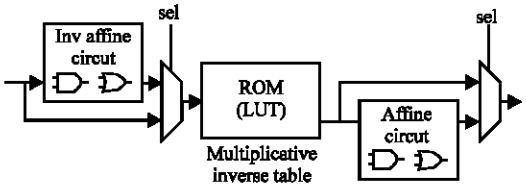

Fig. 5: Both SubBytes and InvSubBytes transformation

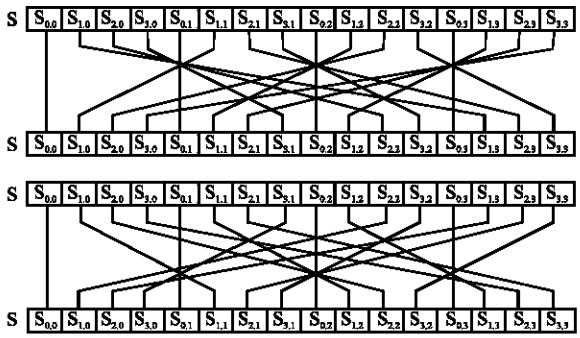

Fig. 6: ShiftRows and InvShiftRow implementation

ShiftRows/InvShiftRows transformation: In these transformations, the bytes in the last three rows of the State are cyclically shifted over different direction and numbers of offsets. For second, third and fourth rows are shifted over one, two and three bytes. The bytes are left shifted for ShiftRows and are right shifted for InvShiftRows. They involve only changing the order of signals and can be implemented using routing only (Fig. 6) but don't require any logic resources.

If both are used ShiftRows and InvShiftRow transformation, a multiplexer is need to select the one will be executed. But only the second and fourth rows need selection, because the first and third rows are shifted by the same offset for both transformation.

MixColumns/InvMixColumns transformation: These transformations performed on the State columns-by-columns as a four-term polynomial and each term is an element of GF (2^8). MixColumns transformation multiplies each column by fixed polynomial:

$$a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\} \text{ modulo } x^4+1.$$

We can be writing as a matrix multiplication:

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

So, we can reduce effectively then MixColumns processing given:

$$S'_{0,c} = \text{xtime}(S_{0,c} \oplus S_{1,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus S_{3,c}$$
$$S'_{1,c} = \text{xtime}(S_{1,c} \oplus S_{2,c}) \oplus S_{0,c} \oplus S_{2,c} \oplus S_{3,c}$$
$$S'_{2,c} = \text{xtime}(S_{2,c} \oplus S_{3,c}) \oplus S_{0,c} \oplus S_{1,c} \oplus S_{3,c}$$
$$S'_{3,c} = \text{xtime}(S_{0,c} \oplus S_{3,c}) \oplus S_{0,c} \oplus S_{1,c} \oplus S_{2,c}$$

InvMixColumns transformation multiplies each column by fixed polynomial:

$$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\} \text{ modulo } x^4+1.$$

We can be writing as a matrix multiplication:

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

Similar, InvMixColumns also processing given:

$$\text{etime} = \text{xtime}(\text{xtime}(\text{xtime}(S_{0,c} \oplus S_{1,c} \oplus S_{2,c} \oplus S_{3,c})))$$
$$\text{ftime} = \text{xtime}(\text{xtime}(S_{0,c} \oplus S_{2,c}))$$
$$\text{ftime} = \text{xtime}(\text{xtime}(S_{1,c} \oplus S_{3,c}))$$
$$S'_{0,c} = \text{etime} \oplus \text{ftime} \oplus \text{xtime}(S_{0,c} \oplus S_{1,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus S_{3,c}$$
$$S'_{1,c} = \text{etime} \oplus \text{ftime} \oplus \text{xtime}(S_{1,c} \oplus S_{2,c}) \oplus S_{0,c} \oplus S_{2,c} \oplus S_{3,c}$$
$$S'_{2,c} = \text{etime} \oplus \text{ftime} \oplus \text{xtime}(S_{2,c} \oplus S_{3,c}) \oplus S_{0,c} \oplus S_{1,c} \oplus S_{3,c}$$
$$S'_{3,c} = \text{etime} \oplus \text{ftime} \oplus \text{xtime}(S_{0,c} \oplus S_{3,c}) \oplus S_{0,c} \oplus S_{1,c} \oplus S_{2,c}$$

We can find a relation between the MixColumns matrix and InvMixColumns matrix. It is notation becomes:

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} = \begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \times \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

So, the InvMixColumns can be implemented a simple preprocessing step that after MixColumns Step. This preprocessing step is given:

$$S'_{0,c} = \text{xtime}(\text{xtime}(S_{0,c} \oplus S_{2,c})) \oplus S_{0,c}$$
$$S'_{1,c} = \text{xtime}(\text{xtime}(S_{1,c} \oplus S_{3,c})) \oplus S_{1,c}$$
$$S'_{2,c} = \text{xtime}(\text{xtime}(S_{0,c} \oplus S_{2,c})) \oplus S_{2,c}$$
$$S'_{3,c} = \text{xtime}(\text{xtime}(S_{1,c} \oplus S_{3,c})) \oplus S_{3,c}$$

The xtime () function, which multiplies a polynomial by {02}, was used repeatedly.

Combine these two step, can be design an efficient MixColumns circuit that used in encryption and decryption. It also can reduce area of design. The block diagram is look Fig. 7.

AddRoundKey transformation: In this transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of word from the key schedule and added into the columns of the State (Fig. 8). The bitwise XOR operation is the same for the encryption and decryption process.

Key Schedule: This consist of two components is Key Expansion and Key Selection. The function of the Key Expansion is to produce the round key from initial key. The following will describe the pseudo code.

```
KeyExpansion (byte key[4*Nk], word w[Nbx (Nr+1)],Nk)
begin
    word temp;
    i = 0;
    while (i < Nk)
        w[i] = word (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);
        i=i+1;
    end while
    i = Nk;
    while (i < Nb* (Nr+1))
        temp = w[i-1];
        if ( i mod Nk = 0)
        temp = SubWord (RotWord (temp)) xor Rcon[i / Nk];
    else if ( Nk > 6 and i mod Nk = 4)
        temp = SubWord (temp);
    end if
        w[i] = w[i-Nk] xor temp;
        i = i + 1;
    end while
end
```

The Key Expansion processing key is a word. It performed word (), RotWord (), SubWord () and addition with Rcon[] transformation to produce the new word. Repeat operation until the round key are full produce then ended. Different key length has different repeat number of times. The action of the SubWord () transformation is a function that takes a four byte input word and applies the S-Box to each of the four bytes to produce an output word. The action of the RotWord () transformation function takes SubBytes () output word as input, performs a cyclic permutation and return the

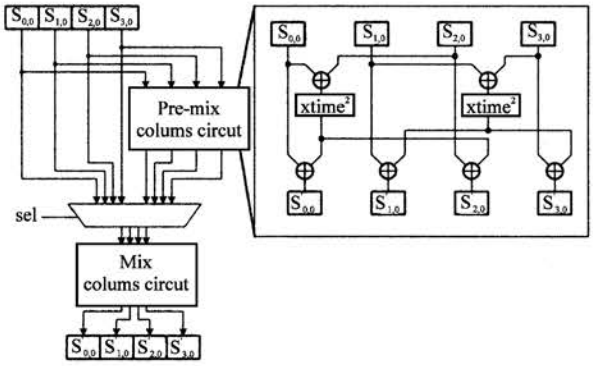

Fig. 7: MixColumns and InvMixColumns integration

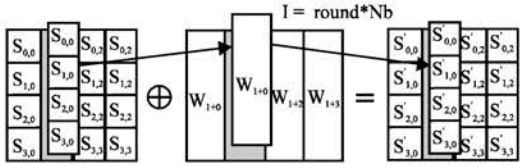

Fig. 8: AddRoundKey transformation

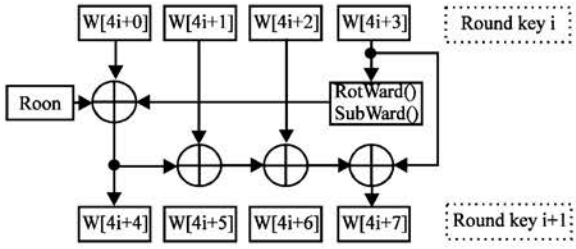

Fig. 9: Key Expansion implementation

new word. The Rcon[] is round constant word array that contains the values given by fixed constant in GF (2^8).

Figure 9 generating round key word in a single cycle. It targets 128 bits of key length. The round key is selected from the expanded key. In AES-128, ten rounds require 11 round keys. There are two ways of generating the key schedule. First is on-thy-fly scheme, while one round is being executed. In this scheme, Key Expansion calculates that round key that will be used in the next round. It only requires round key words of register to store the key. The others scheme is pre-computing. This way requires storing all round key word in a memory and it costly.

Architectural implementation: There are several architectures to implement the AES circuit[10]. A loop unrolled or unfolded architecture can process all round

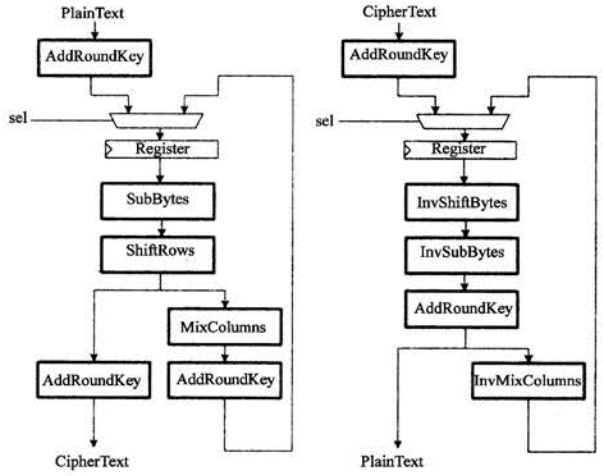

Fig. 10: Iterative loop architecture

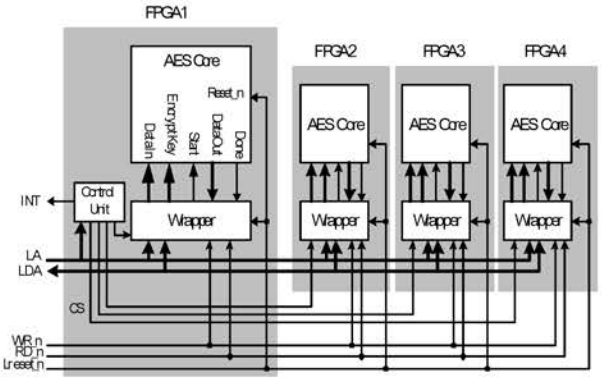

Fig. 11: Multiple AES System

of cipher block at a clock cycle. But the critical path is the longest and area costly. For architectural optimization, we have to require group of clock cycles equal to the number of rounds in the cipher to complete the encryption or decryption process. And also need to reduce design area.

Figure 10 shows an iterative loop architecture that implements the AES round unit. This hardware for the single round should contain a signal set of registers. The input to the round should be multiplexed between the cipher block of data input and the output from the previous round. The design area can large amount of reducing in this architecture. When the application requiring minimum area that the architecture is desired.

In round based architecture can be divided into several stages. Can inserts registers among combinational logic. This is way of inner pipeline that always bring speedup.

We can realize an AES system with multi chip in PCI-mFCU (Fig. 11). The AES Core is to configuration in four FPGA chip on the board respectively. So, there are

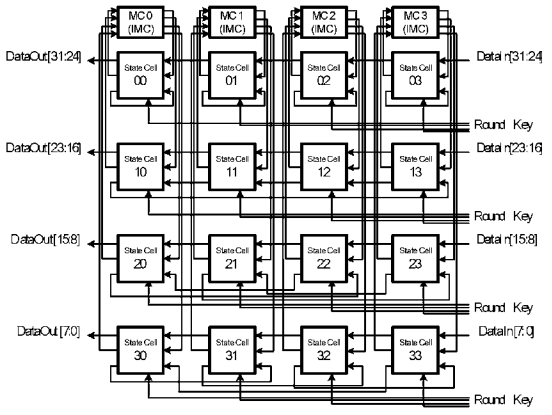

Fig. 12: The encryption/decryption of AES core consists of 16 so-called State Cell and four MC/IMC units

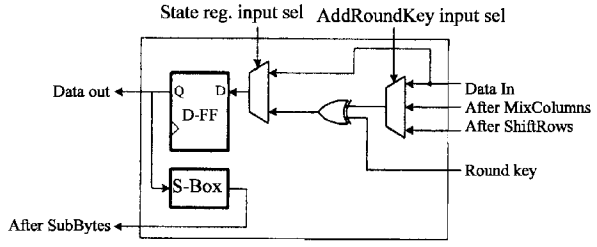

Fig. 13: Encryption core of State Cell

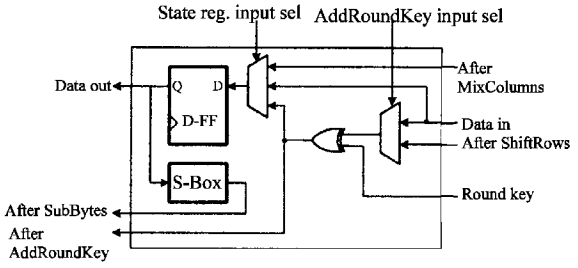

Fig. 14: Decryption core of State Cell

four AES Chips to work at the same time on the system that can be improving throughput. We use the way of I/O polling and write into or read back the data of encrypted or decrypted. AES chip receive or send data to cause with chip select signal at the same time.

We present architecture in addition that structure is regular and related to the definition of the AES State. The architecture uses similarities of encryption and decryption while keeping the core size small. Because the regular structure is especially reached by keeping combinational paths balanced[11]. Another property of the architecture is its scalability. In this architecture, the key size can easily be changed. However, the overall architecture does not change for different require version and key sizes.

An MC/IMC of the architecture is a circuit capable of performing the MixColumns and InvMixColumns transformation for column by column of State Cell (Fig. 12). The State Cell store eight bits per cell and perform SubBytes/InvSubBytes and AddRoundKey transformation.

The AES encryption unit or decryption unit are based on 32 bits I/O interface. In order to load a cipher block, the input data is shifted column by column from the right side into the State Cell. The initial AddRoundKey transformation is done in the fourth clock cycle at the same time as the last column is loaded. In each round, the 128 bits result is finally stored in the registers of the State Cell. Then, the result is shifted columns by columns to the left. The number of clock cycles are required to perform an AES-128 is 17. Eight clock cycles are required for the I/O and include initial round and final round. And nine clock cycles to perform the nine normal rounds.

The design of the State Cell is major for the overall architecture of the encryption/decryption unit. Each State Cell consists of the following components (Fig. 13 and 14):

- Flip-flop: each State Cell used a D-type flip-flop store one byte of the current AES state.
- XOR Gate: each State Cell requires eight XOR gates to performed AddRoundKey transformation in parallel in this architecture.
- Multiplexers: each State Cell consists of multiplexers to select which input is loaded into flip-flop or XOR gate. The current selection is according to control unit.
- S-Box: each State Cell has an S-Box to performed SubBytes or InvSubBytes transformation.

RESULTS

PCI-mFCU board layout uses OrCAD to finish, adopts manufacture methods of PCB circuit board of four layers. The passive component such as resistance and electric capacity adopt SMT component to reduce the use area of the circuit board. The actual size of the circuit board is 195 mm×105 cm (Fig. 15).

Both indicate local register on board maps into PCI memory and I/O space on host. So, we can use memory mapping or I/O mapping to access configure register on PCI board. The remap into I/O space, the base address is 200 h and range is 00h to FFh means have up to 256 ports to access hardware. The remap into memory

Table 1: Synthesis report of AES encryption core

	AES128_encrypt_core	
IP core	Xilinx Spartan-II XC2S200-5	
Target device		
Version type	Version 1	Version 2
Timing		
Max. frequency	74.6 MHz	67.9 MHz
Area		
I/O	258/129	162/33
Block RAMs	14 (100%)	14 (100%)
Slices	842 (35%)	984 (41%)
LUTs	1553 (33%)	1723 (36%)
Equivalent Gate Count	265,301	265,730
Cycle Count	12	17
Throughput	795.723 Mbps	511.092 Mbps
Throughput/Slices	0.945	0.519

Table 2: Synthesis report of AES decryption core

	AES128_decrypt_core	
IP core	Xilinx Spartan-II XC2S200-5	
Target device		
Version type	Version 1	Version 2
Timing		
Max. frequency	65.9 MHz	61.9 MHz
Area		
I/O	259/129	163/33
Block RAMs	14 (100%)	14 (100%)
Slices	1068 (45%)	1150 (48%)
LUTs	1991 (42%)	2177 (46%)
Equivalent Gate Count	283,739	284,007
Cycle Count	12	17
Throughput	702.586 Mbps	466.16 Mbps
Throughput/Slices	0.658	0.405

Fig. 15: Photo of PCI-mFCU Board

space, the address range is 000h to FFFh means have up to 4096×16 bit memory space to access hardware.

The system use two positive edge triggered interrupt controller input to latch the interrupt and a software reset asserts LRESET signal.

The AES IP core and its functional unit were described in Verilog HDL. The circuit was synthesized with Synplify Pro and was implemented using Xilinx ISE tools. The target device was a Xilinx Spartan-II XC2S200, speed grade-5. The Spartan-II is fabricated in a 0.22 μm process and is optimized for low cost and low silicon area. The synthesis result shown in Table 1 and 2. All tool flows were set to use maximum effort. Testing was performed in ModelSim Simulation using Verilog test bench file.

In synthesis report, the version 1 shows iterative loop architecture design and version 2 shows regular and scalable architecture design. All design requires 14 BlockRAM. The version 2 requires CLBs is more then the version. The version 2 owns complicated control unit and multiplexer on State Cell. However, version 2 design area can keep regular and easy to maintain. The version 1 is can pipeling to improve throughput. But design area will increase, because use a large number registers.

The proposed iterative loop architecture processes a block of data in 12 clock cycles and regular and scalable architecture in 17 clock cycles. Spend clock cycle in I/O interface more. The encryption core of iterative loop architecture has a critical path delay of 13.4 ns; throughput and latency are related by:

$$\text{Throughput} = \frac{\text{block}}{\text{latency}} = \frac{128\ \text{bits}}{12\ \text{cycles} \times 13.4\text{ns}}$$

$$= 795.723\text{Mbps}$$

CONCLUSIONS

This study described a design and implementation of reconfigurable rapid prototyping system. This RPU of reconfigurable computing system uses I/O coupling to cooperation with host processor. That is to say, the PCI-mFCU is a PCI bus compatible board. This platform used four FPGAs that act a RPU role or used for verifying IP under fixed wires resource. The system provides software API to access hardware and to configure FPGA via PCI Bus.

This study designs the AES IP Core operation in this system in order to verify this exactness of prototyping of systematic function. We have adopted two ways on IP Core design. The first is round based of iterative loop architecture. The second is regular and scalable architecture. The design supreme working frequency can reach 74.6 MHz and throughput is 795.723 Mbps.

ACKNOWLEDGEMENT

This work was supported by the National Science Council, Taipei, Taiwan, Republic of China, Project No. NSC 92-2213-E-035-028.

REFERENCES

1. Scott, H., 1998. The future of reconfigurable systems. Proceeding of 5th Canadian Conference on Field Programmable Devices.

2. Kiran, B. and K.P. Viktor, 2002. Reconfigurable computing systems. Proceedings of the IEEE, Vol. 90, No. 7.
3. Katherine, C. and H. Scott, 2002. Reconfigurable computing: A survey of systems and software. ACM Computing Surveys, 34: 171-210.
4. Scott, H., 1998. The roles of FPGAs in reporgrammable systems. Proceedings of the IEEE., 86: 615-639.
5. Laurent, M., H. Alan and S. Mark, 1999. Sepia: scalable 3D compositing using PCI Pamette. Field-Programmable Custom Computing Machines, 1999. (FCCM '99) Proceedings. Seventh Annual IEEE Symposium, pp: 146-155.
6. Oskar, M., M. Martin and J.F. Michael, 1998. PAM-Blox: High Performance FPGA Design for Adaptive Computing. FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium, pp: 167-174.
7. Ronald, L., R.R. Taylor and S. Herman, 1999. PCI-PipeRench and the SWORDAPI: A System for Stream-based Reconfigurable Computing. Field Programmable Custom Computing Machines, 1999. (FCCM '99) Proceedings. Seventh Annual IEEE Symposium, pp: 200-208.
8. National Institute of Standards and Technology, 2001. Advanced Encryption Standard (AES). Federal Information Processing Standard (FIPS).
9. Joan, D. and R. Vincent, 1999. AES Proposal. Rijndael, Document Version 2.
10. Xinmiao, Z. and P.K. Keshab, 2002. Implementation approaches for the advanced encryption standard algorithm. Circuits and Systems Magazine, IEEE, 2: 24-46.
11. Stefan, M., A. Manfred and D. Sandra, 2003. A highly regular and scalable AES hardware architecture. IEEE Transactions on Computers, 52: 483-491.