AN IMPLEMENTATION OF A PIPELINED ENCRYPTION MULTI-PROCESSING UNIT UTILIZING VHDL AND FIELD PROGRAMMABLE GATE ARRAYS

Magdy Saeb, Samy Salamah
Arab Academy for Science, Technology & Maritime Transport
School of Engineering, Computer Department
Alexandria, Egypt

Abstract:
In this work we present a special-purpose encryption processor. This type of processors is to be used for IP-security applications. The implemented processor can be integrated in the system before the sender router and after the receiver router. In other words, the processor is to be located at the boundary between terminal equipment and the private network, or at the boundary between private equipment and the public network, which is known as virtual private network (VPN).

To implement the design, we have employed Field Programmable Gate Array Technology (FPGA). The hardware description language VHDL, in conjunction with the schematic editor is used to conclude the realization. The instruction set architecture (ISA) is based on judiciously choosing a complete and efficient group of instructions by studying most of the available standard and conventional encryption algorithms. The processor consists of eight registers (R0-R7) including a shifter unit, arithmetic logic unit (ALU), control unit, memory unit, one multiplexer, one instruction decoder, one timing decoder, and one sequence counter.

A pipelined multiprocessing unit is realized using a set of identical processing elements (PE) that are based on the micro-architecture discussed above. The number of PE's depends on the adopted encryption algorithm.

Keywords: Micro-architecture, encryption, FPGA, VHDL, ISA, IP-sec.

1. INTRODUCTION

The Internet Engineering Task Force (IETF) has initiated a process for adopting standards for IP-layer encryption and authentication (IPSEC). Following the IETF guidelines, we present an encryption processor which is appropriate for packet encryption for a proposed security meta-layer. The motivation behind this work is to provide a micro-architecture implementation of a judiciously-selected set of instructions that are used in standard encryption algorithms. This approach, contrary to implementing a special hardware for each algorithm, saves development time while not sacrificing the performance edge of hardware implementations.

In the following few sections we present a micro-architecture implementation of this processor, that is achieved by field Programmable Gate Array Technology (FPGA) and using the Hardware Description Language VHDL. The basic notion behind our approach is to ensure high performance, as provided by a special-purpose design, and at the same time the re-programmability using a carefully selected simple instruction set.

In the following eight sections, we provide a brief description regarding FPGA and (ASICS) implementations, the methodology used in designing our processor, and the details of implementation process. The CAD workstation results are depicted by the timing reports and timing diagrams are also provided. The pipelined unit is discussed in

section 8. Finally, a complete schematic of the processor micro-architecture is shown in Figure 7 at the end of this article.

2. RECONFIGURABLE HARDWARE

Field Programmable Gate Array is an integrated circuit that can be bought off the shelf and reconfigured by the designers themselves. With each reconfiguration, which takes only a fraction of a second, an integrated circuit can perform a completely different function. FPGA consists of thousands of universal building blocks, known as *Configurable Logic Blocks* (*CLBs*), connected using programmable interconnects. Reconfiguration is able to change a function of each CLB and connections among them, leading to a functionally new digital circuit.

For implementing cryptography in hardware, FPGA provide the only major alternative to *custom and semi-custom Application Specific Integrated Circuits* (ASICs), integrated circuits that must be designed all the way from the behavioral description to the physical layout, and sent for an expensive and time-consuming fabrication.

3. THE PROCESSOR ARCHITECTURE

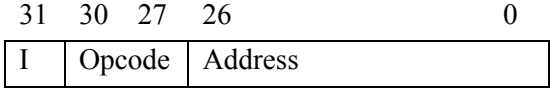
The micro-architecture is designed taking into consideration real-time applications such as packet encryption.

After studying some of the encryption algorithms carefully [1,2,3,5,7,9], we have extracted some of common low-level operations. Accordingly, our processor Instruction set contains these operations: integer addition, bit-wise exclusive or (XOR), AND operation, shift left, shift right, rotate left, rotate right, and compare.

The design procedure starts by deciding which memory word size will be used. Afterwards, the basic register set is selected, and the instruction format is deduced. This procedure is discussed in detail in the next few sections.

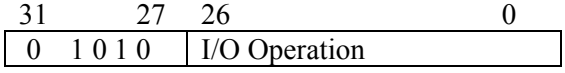
3.1 The Instruction Set and Format

The processor has three instruction code formats, as shown in Fig. 1. Each format has 32 bits. The operation code part of the instruction contains four bits and the remaining 26 bits have different usage depending on the operation code encountered.



(a) Memory Reference Instruction

(b) Register Reference Instruction



(c) Input-output Instruction

Figure 1: *Basic computer instruction format*

A memory-reference instruction uses 26 bits to specify an address. The register-reference instructions are recognized by the operation code 1010 with a 1in the leftmost bit of the instruction. Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 1010 with a 0 in the leftmost bit of the instruction.

3.1.1 Instruction set

Memory Reference Instructions:

AND: And a memory word with R0.
ADD: Add a memory word to R0.
XOR: XOR a memory word to R0.
LDA: Load a memory word to R0.
STR: Store contents of R0 to memory.
JMP: Unconditional jump.
JMS: Jump and save return address.

COM: Compare memory word to R0.

Register Reference Instruction:

LDT: Load temporary register with contents of R0.
LDR: Load data register with contents of TMP register
DEC: Decrement the value of R8 by one.
SZC: Exit the loop if the contents of R8 are zero.
SKZ: Skip if the contents of are zero.
CLA: Clear the contents of R0.
CLE: Clear the carry.
CMA: Complement the contents of R0.
CME: Complement the carry.
INC: Increment the value of R0 by one.
SHR: Shift the contents of R0 to right.
SHL: Shift the contents of R0 to left.
CIR: Circulate the contents of R0 to right.
CIL: Circulate the contents of R0 to left.
STP: Stop the processing.

I/O Instructions:

INP: Input the data into R0.
OUT: Output the data from R0.
ION: Interrupt on.
IOF: Interrupt off.

3.1.2 The Instruction cycle

The program is executed by the processor by going through a cycle for each instruction. Each instruction cycle consists of the well-known phases:

Fetch an instruction from memory,
Decode the instruction,
Execute the instruction,
Go back to 1.

3.1.3 Determination of the type of instruction

The control unit determines the type of instruction that was just read from memory; the flow chart in Figure 2 presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding phase. This chart forms the basis for the design of the basic processor cycles and the control unit. Also, using this chart the number of sequence counter bits was estimated since the longest operations are clearly shown. However, this chart does not show the other cycles such as the interrupt cycle. The chart provides the basis for the instruction decoder design.

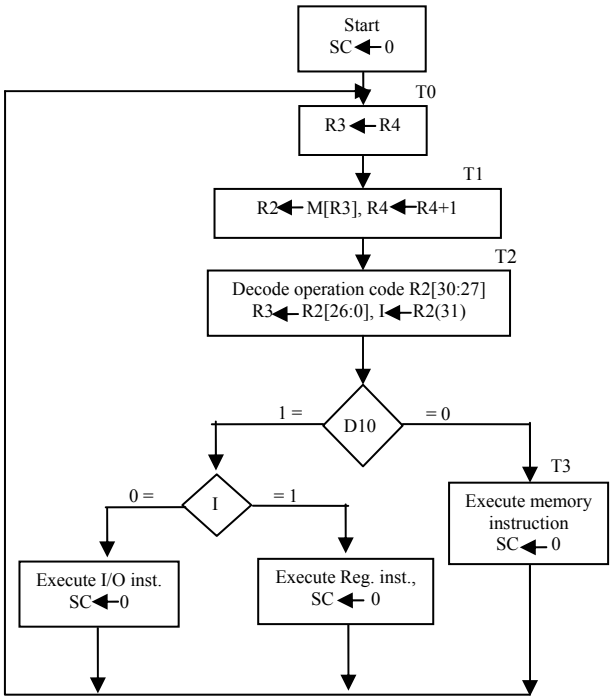


Figure 2: Flow chart of the operation sequence

3.2 The Register set

We have used eight 32-bit registers, one of them is a temporary register and the others are special-purpose registers. These are:
R0: It saves the result of the mathematical and logical operations. It is also used as shifter.
R1: It stores the data from memory unit and passes it to R0 register.
R2: It saves the instruction after the fetching operation.
R3: It gets the address of the current instruction from the program counter.
R4: It is a counter that calculates the address of the next instruction after the execution lf the current instruction is completed.

R5: It is a temporary register.
R6: Input register, which hold the data from user and enter data directly to R0.
R7: It is an output register that gets its value from R0.
R8: It is loop register, it saves the repeat times and it is decremented until zero.

3.3 The Arithmetic Logic Unit (ALU)

It has two 32-bit ports and 3-bit port to select between the operations, it performs the following five arithmetic and logical operations: integer addition, bitwise XOR, AND operation, and comparing.

3.4 The Multiplexer

This MUX has seven 32-bit inputs, one 32-bit output and 3-bit port to select between the seven inputs. It is used to control the 32-bit data bus.

3.5 The Instruction Decoder

The instruction decoder has 16 output bits to decode the different instructions.

3.6 The Timing Decoder

The timing decoder has 4-bit input port and 16-bit output ports to derive the processor. These 16 bits signals are used as timing signals.

3.7 The Sequence Counter

The sequence counter is built using a four-bit counter. It is used to generate 16 timing signals.

3.8 The Control Unit

It provides the control signal for the micro-operation that is to be executed. The control functions implemented using VHDL are as shown in table 1. The control functions, shown in front of each micro-operation, were realized using VHDL code and implementing the CU as a macro. This macro was then used in the schematic editor as an essential component of the processor.

Table 3.1
The control functions and micro-operations of the processor

Fetch	$R' T_0 : R_3 \leftarrow R_4$
	$R' T_1 : R_2 \leftarrow M[R_3], R_4 \leftarrow R_4 + 1$
Decode	$R' T_2 : D_0..D_{10} \leftarrow$ Decode R_2 (30-27)
	$R_3 \leftarrow R_2$ (26:0), $I \leftarrow R_2$ (31)
Interrupt	$T'_0 T'_1 T'_2$ (IEN) : $R \leftarrow 1$
	$R T_0 : R_3 \leftarrow 0 , R_5 \leftarrow R_4$
	$R T_2 : M[R_3] \leftarrow R_5 , R_4 \leftarrow 0$
	$R T_2 : R_4 \leftarrow R_4 + 1$, IEN $\leftarrow 0$, R $\leftarrow 0$, SC $\leftarrow 0$
Memory- reference	
AND	$D_0 T_3 : R_1 \leftarrow M[R_3]$
	$D_0 T_4 : R_0 \leftarrow R_0$ & R_1, SC $\leftarrow 0$
ADD	$D_1 T_3 : R_1 \leftarrow M[R_3]$
	$D_1 T_4 : R_0 \leftarrow R_0 + R_1$, SC $\leftarrow 0$
XOR	$D_2 T_3 : R_1 \leftarrow M[R_3]$
	$D_2 T_4 : R_0 \leftarrow R_0$ xor R_1, SC $\leftarrow 0$
LDA	$D_3 T_3 : R_1 \leftarrow M[R_3]$
	$D_3 T_4 : R_0 \leftarrow R_1$, SC $\leftarrow 0$
STR	$D_4 T_3 : M[R_3] \leftarrow R_0$, SC $\leftarrow 0$
JMP	$D_5 T_3 : R_4 \leftarrow R_3$, SC $\leftarrow 0$
JMS	$D_6 T_3 : M[R_3] \leftarrow R_4$, $R_3 \leftarrow R_3 + 1$ $D_6 T_4 : R_4 \leftarrow R_3$, SC $\leftarrow 0$
COM	$D_7 T_3 : R_1 \leftarrow M[R_3]$ $D_7 T_4 : GT \leftarrow R_0$ comp R_1, SC $\leftarrow 0$
LDC	$D_9 T_3 : R_1 \leftarrow M[R_3]$, SC $\leftarrow 0$
Register Reference	
	$D_{10} I T_3$: SC $\leftarrow 0$
LDT	$D_{10} I T_3 IR(14) : R_5 \leftarrow R_0$
LDR	$D_{10} I T_3 IR(13) : R_0 \leftarrow R_5$
DEC	$D_{10} I T_3 IR(12) : R_8 \leftarrow R_8 - 1$
SZC	$D_{10} I T_3 IR(11) : R_4 \leftarrow R_4 + 1$
SKZ	$D_{10} I T_3 IR(10) : R_4 \leftarrow R_4 + 1$
CLA	$D_{10} I T_3 IR(9) : R_0 \leftarrow 0$
CLE	$D_{10} I T_3 IR(8) : E \leftarrow 0$
CMA	$D_{10} I T_3 IR(7) : R_0 \leftarrow \overline{R_0}$
CME	$D_{10} I T_3 IR(6) : E \leftarrow \overline{E}$
INC	$D_{10} I T_3 IR(5) : R_0 \leftarrow R_0 + 1$
SHR	$D_{10} I T_3 IR(4) : R_0 \leftarrow$ shr (R_0)
SHL	$D_{10} I T_3 IR(3) : R_0 \leftarrow$ shl (R_0)
CIR	$D_{10} I T_3 IR(2) : R_0 \leftarrow$ cir (R_0)
CIL	$D_{10} I T_3 IR(1) : R_0 \leftarrow$ cil (R_0)
STP	$D_{10} I T_3 IR(0) : S \leftarrow 0$
Input output	
	$D_{10} I' T_3 : SC \leftarrow 0$
INP	$D_{10} I' T_3 IR(1) : R_0 \leftarrow R_6$
OUT	$D_{10} I' T_3 IR(2) : R_7 \leftarrow R_0$
ION	$D_{10} I' T_3 IR(3) : IEN \leftarrow 1$
IOF	$D_{10} I' T_3 IR(4) : IEN \leftarrow 0$

3.9 The Memory Unit

It is important to provide a fast, efficient memory hierarchy to keep pace with the processor. Any sequential circuit has memory of a sort, since each flip-flop or latch stores one bit of information. However, the reserved word "memory" refers to bits that are stored in a structured way, usually as a two-dimensional array in which one row of bits is accessed at a time.

The memory has address and control inputs and data outputs, but it also has data inputs. In our system, we have designed a memory unit consisting of 128 32-bit words as shown in figure 3.

The memory has just two defined access operations, read and write operations, each module mentioned above has only one control input, which is write enable (WE), if the value of this bit is 1 then we can write in a certain memory word as specified by the address bus, which consists of 7-bit to address the 128 memory words. If the value of this bit is zero then we can read the content of certain memory word.

The implementation timing reports of this memory module are shown in appendix A.

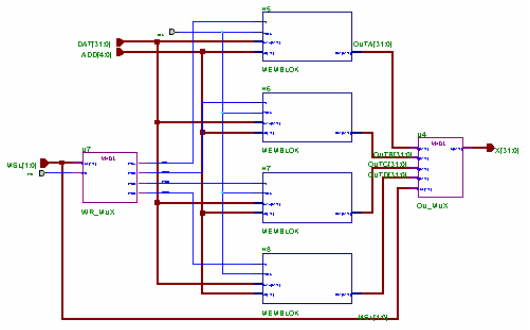


Figure 3: *The memory unit architecture*

We have implemented the design of all major components using VHDL programming language. Afterwards, we integrated these components in the design by using the CAD workstation schematic editor.

4. The Micro-architecture

As shown in Appendix, the encryption processor main building blocks are:

The register set,
The bus system,
The ALU,
The control unit (CU),
The instruction decoder,
The timing decoder and the sequence counter. These and other units are shown with some remarks beside each unit in Figure 8 at the end of this paper.

The memory unit is implemented using four (RAM32x8s) sub-units as found from the system library and connected to each other forming a 32x32 word memory block macro. We used two multiplexers as shown in Figure 3; the WR_MUX is used to pass the write signal to certain module as specified by the address bus (write operation). The OU_MUX is used to pass the data from certain module as specified by the address bus (read operation); the two multiplexers are implemented by VHDL language.

5. VHDL Code

The architecture has been implemented employing VHDL hardware description language and Xilinx workstation for computer-aided design (CAD). The full project code is available from the authors upon request.

6. Simulation Results

Figure 4, shown below, illustrates the processor output for a Vernam encryption algorithm by entering a sequence of characters during the input pad and XOR it with keys stored in memory.

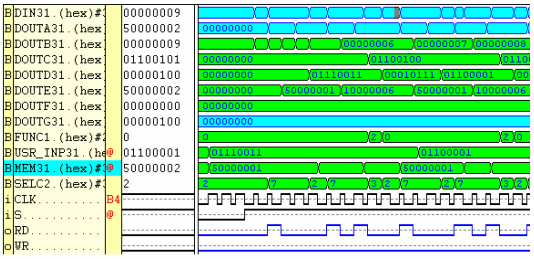


Figure 4: *Vernam Encryption Algorithm Simulation result*

Rc5 Encryption Algorithm simulation

RC5 algorithm can encrypt two plain text data each of 32-bit, the following code depicts the RC5 encryption algorithm, this code and sub-keys are stored in the memory and we entered two plaintext blocks A and B from the input pad, then we added the sub-key S1 to block B, and S2 to block A. Block B is circulated to the left by the value of A and stored in specified memory word, Block A is circulated to the left by the value of B and stored in another memory word. After the completion of the past instructions, the XOR operation is made between the two new blocks and the result is stored in certain memory word. All the past operations implement just one round, we need to repeat the past procedure sixteen times to complete the RC5 algorithm.

```
0000000  INP          "Input block(B)
0000001  ADD 1D h     "Add to S1
0000002  STR  1B h    "Store block B
0000003  INP          "Input block(A)
0000004  ADD 1C h     "Add to S0
0000005  STR  1A h    "Store block As 1Ah
0000006  LDA          "Load block A in R0
0000007  XOR 1B h     "Bitwise XOR A,B
0000008  LDC          "Load R8 by value B
0000009  CIL          "Circulate to the left
000000A  DEC          "Decrement R8
000000B  SZC          "Skip if R8 is zero
000000C  JMP 09 h     "Jump to addr. 09 h
000000D  ADD 01E h    "Add to sub-key S2
000000E  STR  1A h    "Store B in 1Bh
000000F  LDA 1B h     "Load B in R0
0000010 XOR  1A h     "Bitwise XOR B, A
0000011 LDC           "Load R8 by A
0000012 CIL           "Circulate to the left
0000013 DEC           "Decrement R8
0000014 SZC           "Skip if R8 is zero
0000015 JMP  12 h     "Jump to 12 h
0000016 ADD 1F h      "Add to sub-key S3
0000017 STR  1B h     "Store B in addr 1Bh
0000018 STP           "Stop processing
```

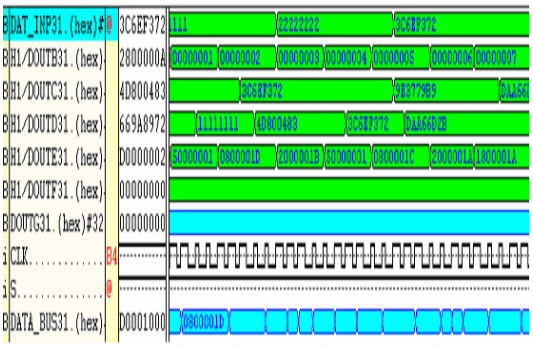


Figure 5: *Rc5 algorithm simulation result*

Figure 5 displays the simulation result of one round of the RC5 encryption algorithm.

Figure 4 and 5 were obtained using Xilinx CAD workstation.

7. IMPLEMENTATION

Using Virtex Xilinx family, and performing the necessary implementation steps, we obtained the following timing reports:

1. The Processor chip:

Maximum period: 56.371 ns at a maximum frequency of 17.740 MHz. A maximum combined path delay of 7.617ns.

2. The Memory Unit chip:

Maximum period: 11.892 ns at a maximum frequency of 84.090 MHz. A maximum combined path delay of 18.152 ns.

8. THE MULTIPROCESSOR PIPELINED UNIT

As shown in Figure 6, the pipelined architecture is built using N units of identical processing elements (PE). Each PE is based on the micro-architecture demonstrated in previous sections and using Virtex Xilinx family. Each PE executes

strictly one instruction for each transition. Therefore for an N-instruction program, we need an N-processing element pipeline.

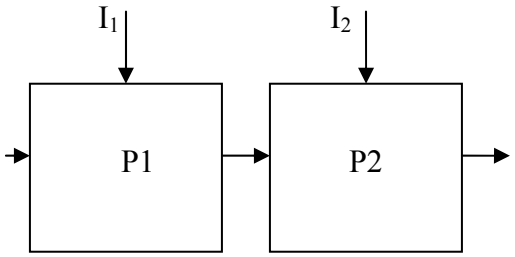


Figure 6: *The Pipelined multiprocessor unit*

In a large number of encryption algorithms, the operations are dependent on each other. Hence, we can benefit from the pipelined unit by passing the result of the first after performing certain operation to the second one to perform another operation as shown in Figure 7. This figure illustrates that the first processor performed an XOR operation and the output is passed to the second one which adds this result to a key stored in the memory.

Figure 7, illustrates that the first processor performed an XOR operation and the output is passed to the second one which adds this result to a key stored in memory.

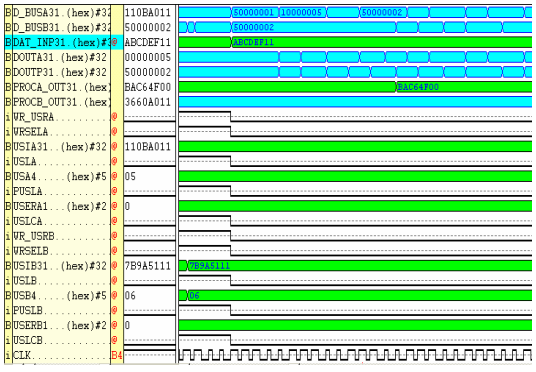


Figure 7: *The test for the pipelined multiprocessor system*

SUMMARY AND CONCLUSION

We have studied a large number of standard and modern encryption algorithms regarding the basic operations that are required in each one of them. It turned out that these basic operations form a simple set of low-level instructions. Our suggested architecture is based on a judiciously selected repertoire of instructions, shown in table 1 that provides minimum FPGA implementation area and acceptable minimum average connection delay. The processor organization discussed in sections 2, 3, and 4 is adopted for each PE employed in the pipelined micro-architecture. For an N-instruction encryption program, one uses N processing elements.

Based on implementation results, the maximum operating frequency for each PE was found to be 500 MHz. This "hybrid approach" of FPGA-based implementation of encryption algorithms jointly with added feature of re-programmability provides the necessary performance advantage while at the same time keeps the design effort to a minimum. We think that this hybrid approach is an advantageous compromise. This compromise deals with two alternatives: the first one is using a general-purpose processor with the added layers of software that hinders performance. The second alternative is designing a special-purpose processor for each adopted encryption algorithm with the added effort and time associated with such an approach.

Based on the implementation timing reports, one concludes that the multiprocessor pipelined architecture provides more than adequate performance that is required for packet-level encryption.

REFERENCES

[1] VICTOR FISCHER, "Realization of the Round 2 AES Candidates using Altera FPGA," www.micronic.sk, 2000.
[2] Kris Gaj and Pawel Chodowies, George Mason University, "Comparison of the hardware performance of the AES

candidates using reconfigurable hardware," http://ece.gmu.edu/crypto-text.htm, 2000.
[3] Ronald Rivest, " The RC5 Encryption Algorithm," Dr. Dobb's Journal, #226, pp.146-148, January, 1995.
[4] John F. Wakerly., Digital Design Principles & Practices, Prentice Hall, 2001.
[5] William Stalling., Cryptography and Network Security: Principles and Practices, Prentice Hall, 1999.
[6] M. Morris Mano., Computer System Architecture, Prentice Hall, 1993.
[7] Alfred J. Menzes., Handbook of Applied Cryptography, CRC Press, 1997.
[8] Peter J. Asbenden., The Designer's Guide to VHDL, Morgan Kaufmann.
[9] Bruce Schneier, Applied Cryptography: Protocols Algorithms & Source code in C, second edition, John Wiley & Sons, 1996.

APPENDIX A:

The timing diagram and CIBS Xilinx Reports.
A part of the timing implementation results is given as follows:

Design specification:
Target Device: xv100
Target package: bg256
Target Speed: -6
Mapper version: Virtex -- C.16

The Processor Module:
Timing summary:
The average connection Delay for this design is : 1.939 ns.
The maximum pin Delay is: 7.625 ns.
The average connection Delay on the 10 worst Nets is: 6.302 ns.

Design statistics:
Minimum period: 46.60 ns (maximum frequency: 21.456 MHz).
Maximum combinational path delay 7.62 ns.
Device utilization summary:
Number of Slices: 795 out of 1,200 66 %
Slice Flip Flops: 520
Slice Latches: 71
Four input LUTs: 1,286
Number of Slices containing unrelated logic : 0 out of 771 0 %
Number of bonded IOBs: 174 out of 180 96%
Number of GCLKs: 4 out of 4 100%
Number of GCLKIOBs: 1 out of 4 25%
Number of RPM macros: 2
Total equivalent gate count for design: 11,998
Additional JTAG gate count for IOBs: 8,064

Memory Module:
Timing summary:
The average connection Delay for this design is : 4.066 ns.
The maximum pin Delay is: 9.555 ns.
The average connection Delay on the 10 worst Net is 6.587 ns.

Design statistics:
Minimum period: 11.892 ns (maximum frequency: 84.090 MHz).
Maximum combinational path delay: 18.152 ns.
Device utilization summary:
Number of Slices: 162 out of 1,200 13%
4 input LUTs: 68
32x1 RAMs: 128
Number of Slices containing unrelated logic: 0 out of 162 0%
Number of bonded IOBs: 72 out of 180 40%
Number of GCLKs: 1 out of 4 25%
Number of GCLKIOBs: 1 out of 4 25%
Total equivalent gate count for design: 33,656
Additional JTAG gate count for IOBs: 3,504

APPENDIX B:

The complete Micro-architecture of each processing element is shown in Figure 8.

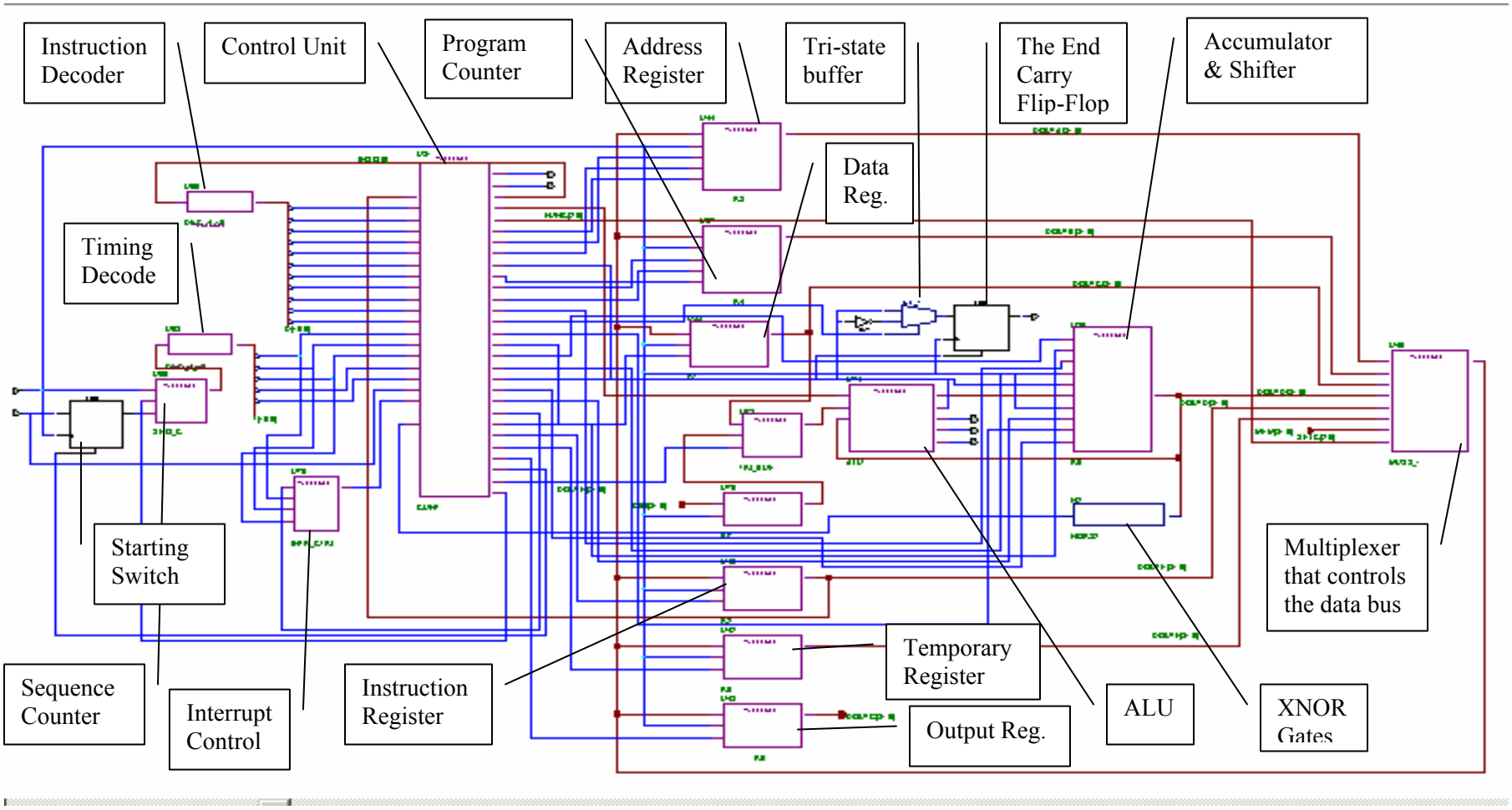


Figure 8: *The Complete Processing Element Micro-architecture*