XAPP374 (v1.0) September 26, 2003

CryptoBlaze: 8-Bit Security Microcontroller

Summary

This application note provides a basic outline for creating a cryptographic processor using CoolRunner™-II devices and a CPLD version of the PicoBlaze processor.

Introduction

Standard microprocessors target very general applications. Their instruction sets are chosen for a broad ranges of tasks, with both numerical and data management actions at their centers. Some applications have been less-than-well served by standard instruction sets and architectures. Among these poorly served markets are error correction coding, cryptography, and digital signal processing. Due to a shift away from analog data methods over the last 40 years, digital signal processors have evolved to serve the DSP market needs. Error correction coding and cryptography are less served, but one methodology has received recent attention—the programmable logic based “soft processor.” Many designers (see **References**, page 7) have pined about the lack of specific instruction sets suitable for cryptography and coding, and have recommended more suitable alternatives. This application note discusses these instructions and explores architecture(s) capable of meeting the required needs.

Starting Point—PicoBlaze

While developing soft programmable processors for FPGAs, it became apparent that the same method could be used with CPLDs of sufficient density. Though somewhat restricted in capacity, CPLDs deliver ample speed and are nonvolatile, reprogrammable, and are very difficult to reverse engineer, if properly managed. These qualities are attractive to the cryptography field.

Our starting point is a quick review of the processor design called “PicoBlaze” in the Xilinx literature. Figure 1 shows the PicoBlaze architecture detailed in **XAPP387**. The original design was developed by Ken Chapman (Xilinx) for FPGA products. The key contribution made by **XAPP389** was to streamline the FPGA version, remove FPGA specific constructs, and create a VHDL solution that is directly tied to a cross assembler. It is very easy to add and delete instructions to/from both the architecture and its primary software support. Table 1 is a summary of the PicoBlaze baseline instruction set.

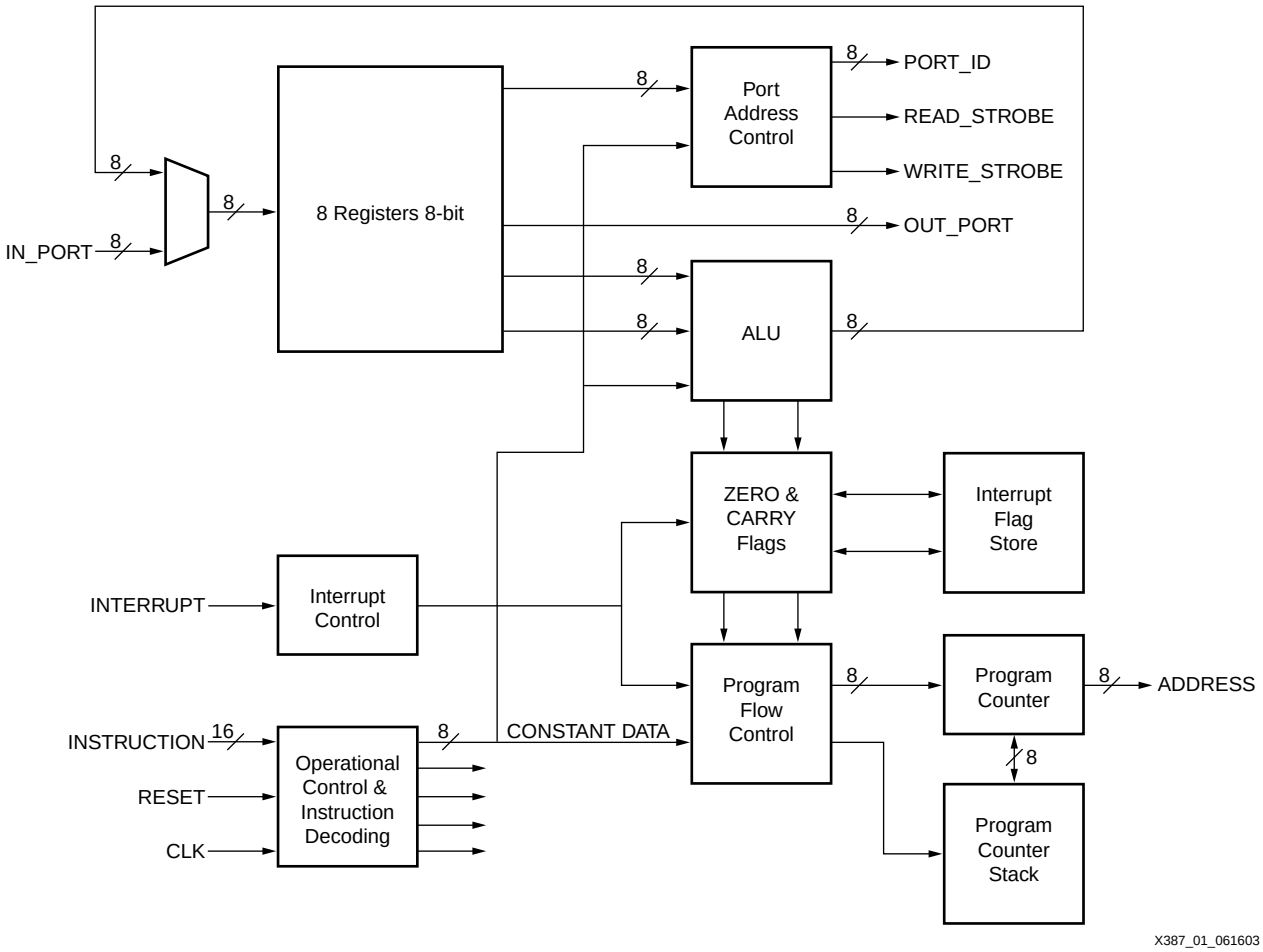

Figure 1: **PicoBlaze Architecture**

Table 1: **PicoBlaze Baseline Instruction Set**

Control	Arith./Logical	Shift/Rotate	Interrupt;I/O
JUMP aa	ADD sX, KK	SR0 sX	INTERRUPT
JUMP Z, aa	ADDCY sX, KK	SRX sX	RETURNI
JUMP NZ, aa	SUB sX, KK	SRA sX	INTP DISABLE
JUMP NC, aa	SUBCY sX, KK	RR sX	INTP ENABLE
Call aa	ADD sX, sY	SL0 sX	ENABLE INTP
Call Z, aa	ADDCY sX, xY	SL1 sX	DISABLE INTP
Call NZ, aa	SUB sX, sY	SLX sX	INPUT sX, PP
Call C, aa	SUBCY sX, sY	SLA sX	INPUT sX, sY
Call NC, aa	LOAD sX, KK	RL sX	OUTPUT sX, PP
RETURN	AND sX, KK		OUTPUT sX, sY
RETURN Z	OR sX, KK		
RETURN NZ	XORsX, KK		

Table 1: **PicoBlaze Baseline Instruction Set *(Continued)***

RETURN C	LOAD sX, sY		
RETURN NC	Or sX, sY		
	AND sX, sY		
	XOR sX, sY		

Note that this is a “baseline” instruction set, because the whole point of a soft processor—similar to microprogramming—is to be able to expand and collapse the instruction set to suit the application you want.

These instructions are fairly straightforward and explained in detail in the PicoBlaze application note. A very useful aspect of PicoBlaze is that the instruction set isn’t locked down. It can be altered. The architecture is written in procedural VHDL specifically *to* be altered. The assembler is also open for modification. That being the case, cryptographic extension instructions can be added to the architecture and the assembler. It is possible to expand the word width as desired. The instruction and data words interact, but are both alterable, also.

New Instructions and a Krypto Kit

So what kind of instructions would be “cryptographic” in nature?

In 1962 Bartee and Schneider were attempting to program BCH coding on a microprogrammable architecture and found a need for Finite Field calculations. Thirty years later, Agnew, Mullin and Vanstone (1993) were attempting to define an architecture for elliptic curve cryptography and found a need for Finite Field calculations. And five years after that, De Waleffe and Quisquater (1998) were defining an architecture for encryption smart cards and found a need for Finite Field calculations. It seems there’s a pattern here.

The PicoBlaze includes bit EX-OR operations, which the Finite Field folks call “adding.” Finite Field multiplication over GF(2^m) is another critical operation. Efficient multiplication results in being able to calculate Finite Field inversion (hence, division) and even logarithms. We will focus briefly on the adding Finite Field (Galois) multiplication, with both instructions and hardware.

Other operations that frequently occur include byte transformations, or “substitutions.” In "crypto land" this is called building an S-box. CoolRunner-II CPLDs are particularly efficient in building up these mappings where the CPLD function block is the equivalent of a small EPROM. Multiple S-boxes can be created with a utility that lets you key in the bit mappings, and it spits out the equations in HDL. Block ciphers frequently use S-boxes cascaded with EX-OR operations, rotates, and such. The Advanced Encryption Standard is built from a lot of these little operations.

LFSR creation is very straightforward with CPLD macrocells, each one of which includes both D flip-flops and an EX-OR gate. Creating powerful stream ciphers by cascading LFSR structures in tandem with others, as well as LFSR structures in tandem with S-boxes, permits custom ciphers to be built as desired.

Table 2 summarizes some operations and their capacity consumption, to give an idea of what is needed to build some of these pieces. Tables included in the appendix show how some functions can be calculated once, and stored in a PLA based table. Note that building translation functions in the PLA often doesn’t use up the function block I/Os and may leave half or more of the flip flops for other tasks. LFSRs and S-boxes can even interleave within function blocks very efficiently. One uses mostly the flip-flops and the other uses mostly the gates, while they split the entry and exit sites.

Table 2: **Krypto Kit Resource Usage**

Function	Size	Resource Usage	Comments
GF(2^4) multiplier	4 bit	12 macrocells	Gates+flops
GF(2^8) multiplier	8 bit	24 macrocells	Gates+flops
GF(2^{16}) multiplier	16 bit	48 macrocells	Gates+flops
GF(2^{32}) multiplier	32 bit	96 macrocells	Gates+flops
GF(2^{163}) multiplier	163 bit	340 macrocells	Serial input
AES S-box	8 bit	384 ANDs, 8 ORs	Flops unused
LFSRs	variable	One / stage	ANDs unused
Irreduc.Polynom.	variable	3-5 ANDs *	Flops unused
Log (2^8)	8 bit	383 ANDS, 8 ORs	Flops unused
Exp (2^8)	8 bit	370 ANDS, 8 ORs	Flops unused
GF(2^8) Inverter	8 bit	397 ANDs, 8 ORs	Flops unused

* **Note**: irreducible polynomial assumption here is a trinomial or pentanomial form

Comparing All S/W versus All H/W

As an example trade-off, the code listed in Figure 2 performs finite field multiplication over eight bits, using the baseline instruction set. As you can see, it will take over 100 instructions (counting loops) to complete an arbitrary product of two eight-bit operands.

```
Constant    out_port, 01                ;declare port
                 Namereg s0, opa         ;declare register
                 Namereg s1, opb         ;declare register
                 Namereg s2, poly        ;declare register
                 Namereg s3, result      ;declare register
                 Namereg s4, count1     ;declare register
                 Namereg s5, count2     ;declare register
                 Namereg s6, temp       ;declare register
                 ;
                 Load result, 00            ;init result
                 load opa, 0A                ;init operand a
                 load opb, 0D                ;init operand b
                 load poly, 1D               ;init polynomial
                 load count1, 08            ;init loop counter
Loop1:      SL0 result                     ;shift left
                 call C, xorp          ;do r = r - p if degree(r(t)) = 1
                 ;
                 load temp, opa         ;next code see if a(i) = 1?
                 load count2, count1
                 sub count2, 01
                 Jump Z, and1
loop2:       SR0 temp
                 sub count2, 01
                 Jump NZ, loop2
                 ;
and1:       AND temp, 01                ;if a(i)=1
                 call NZ, xorb ;do r = r + b if a(i) = 1
                 ;
                 sub count1, 01              ;decrease loop counter
                 Jump NZ, loop1             ;goto loop1
                 output result, out_port   ; display result
```

```
                ;
                ;
xorp:        xor result, poly            ;r = r - p
               return
                ;
xorb:         xor result, opb            ;r = r + b
               return
```

Figure 2: **Figure 2 Code Listing For GF(2^3) Multiplier**

The time of this operation depends on the length of the operands, and the number of clock cycles the PicoBlaze processor would need to execute them (typically, two cycles per instruction). If the processor were expanded to include a hardware Finite Field multiplier, the time delay could be reduced substantially. Figure 3 shows the structure of the multiplier for 4 bits. Simply doubling each register and corresponding mux gates results in a working multiplier capable of executing the multiply in eight clock cycles. It only takes four cycles, if implemented with CoolRunner-II dual edge flip flops.

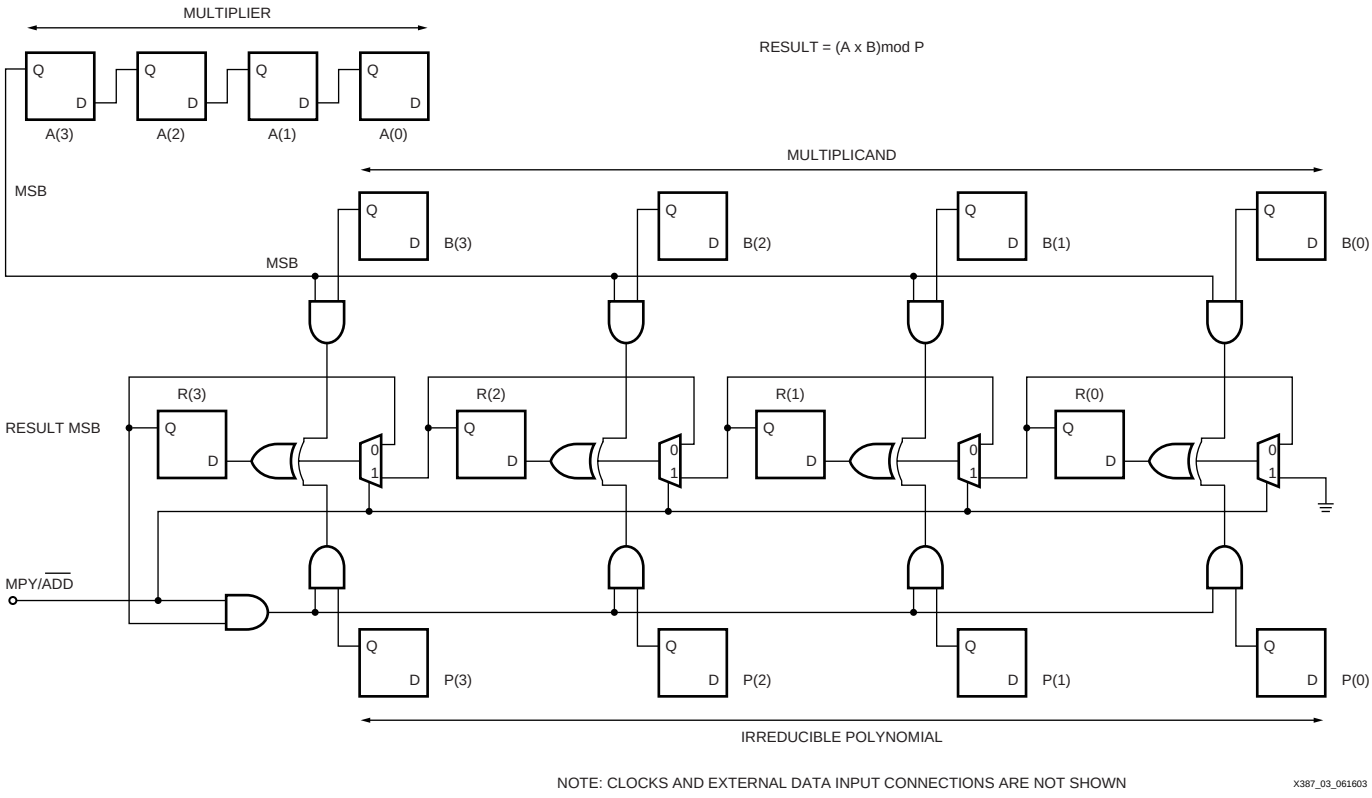

Figure 3: **4-Bit GF (2^2) Multiplier**

Figure 4 shows how the 8-bit Finite Field multiplier could be inserted into the PicoBlaze architecture to deliver a more “cryptographically” appropriate design. Depending on the target application, the design can be modified to tradeoff appropriate instructions or specialty hardware “boxes”. Working details for Figure 3 are provided in **XAPP371**.

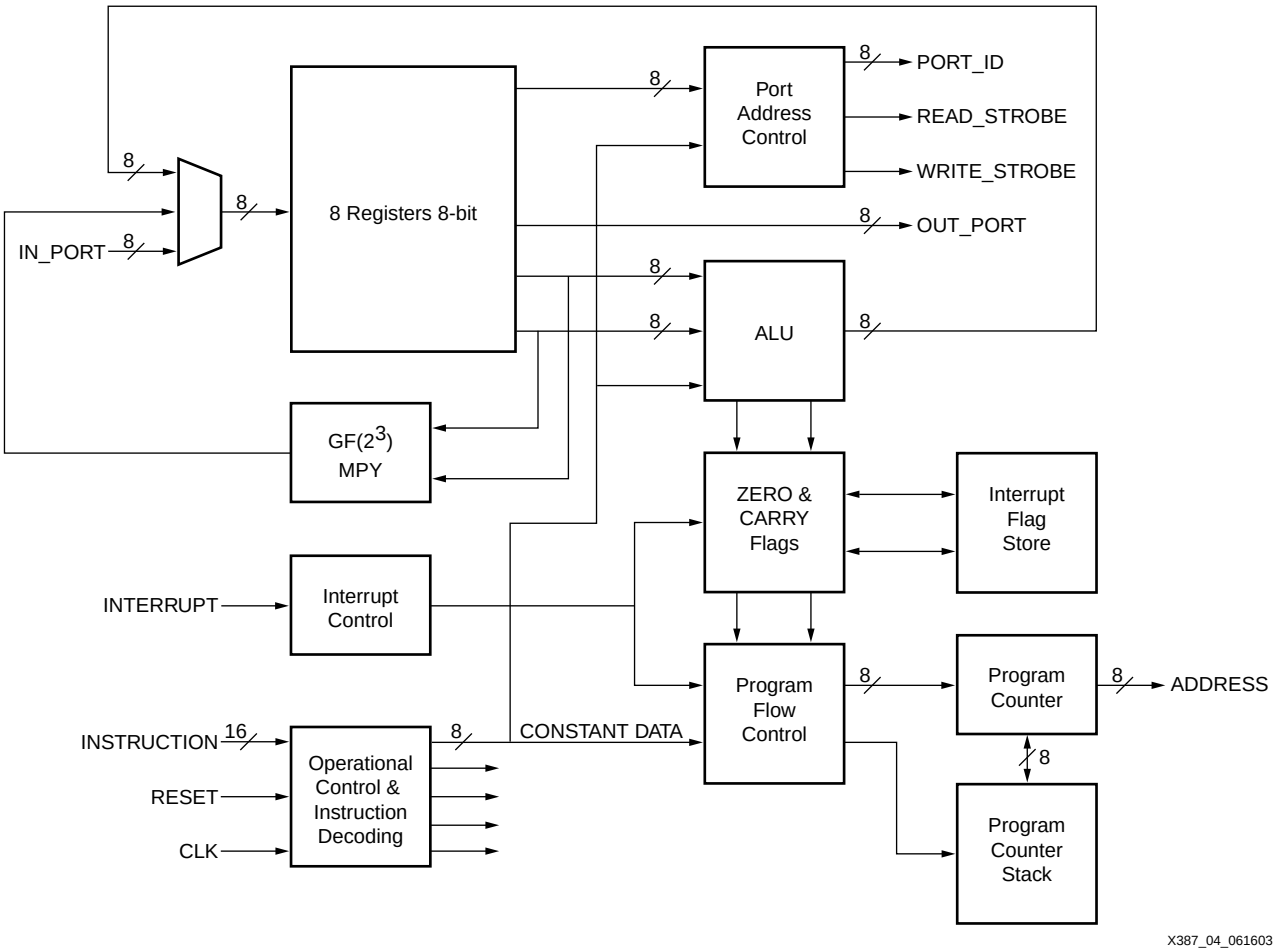

Figure 4: **Embedded GF (2^3) Multiplier**

Power Analysis and Tempest

Cryptographers always worry about “attacks.” Attacks are either real or potential ways for intruders to gain access to the information the crypto folks seek to hide. Simple brute force attacks, such as lots of trials of passwords, are what many people think about when they think about crypto attacks. Most systems today are far beyond that level of concern. However, newer dimensions have arisen in recent years, whereby designers must be aware that “side channel” attacks do exist. Side channel attacks examine behavior outside the code program and look to identify electrical behavior indicating some degree of “what’s going on inside the chip.” The two common categories of side channel attacks are "Power Analysis" and "Tempest."

Power Analysis falls into two further categories—Simple Power Analysis (SPA) and Differential Power Analysis (DPA). Tempest attacks are similar in nature to Power Analysis, but focus on measuring the fields emanating from the cryptographic devices. For our purposes we will focus on Power Analysis.

It is possible to determine elements of internal behavior for a chip by simply examining the current flow into it over time. Loop behavior will deliver a repetitious current draw (usually a voltage profile), and branch behavior can be witnessed from similar profiles. It has been shown that “power attackers” can determine the bits of passwords or keys by delivering them in succession and then determining by the branching and looping whether each bit was “accepted” or “rejected.” Given the binary nature of the item, it is just a matter of tenacity to determine a key—it’s much like listening to the tumblers of a lock as the dial spins.

Microprocessors which have published instruction sets are particularly susceptible to this sort of attack.

Although CryptoBlaze does not claim to be impervious to Power Analysis, it is possible to change the architecture to either include “bogus” logic to alter current flow into the CoolRunner-II, or alternately change the structure to “homogenize” the current flow, making it more difficult to determine what is going on within the chip.

Differential Power Analysis is viewed as being the most devastating of attacks, as it goes beyond SPA by adding into the analysis repeated statistical correlation. Attempts have been made to alter behavior to avoid correlation directly attributed to “key” bits in algorithms, but DPA has frequently prevailed in the past. Behavior alteration is usually instruction sequencing in nature. Part of the issue is that the analysis can relate to standard identifiable instructions in the processor repertoire. It could be more difficult to analyze the behavior of an arbitrary state machine performing transitions that might not so easily correlate to “instructions,” as with the insertion of a field multiplier that switches elements on both flip flop edges. Nonetheless, little study has been published on DPA attacking programmable logic devices versus attacking programmable processors. It is possible that processors like CryptoBlaze which combine elements of both might be more resistant to this attack method.

Additional Organizations

Although CryptoBlaze is an expansion of PicoBlaze, it can be altered in both the instruction width and the data width. The depth of program counter can be expanded, as can all dimensions of altering the data. The design can migrate up and down the CoolRunner-II density spectrum, as chosen. However, there is another dimension that can be exploited. Rather than making a standalone processor, it is possible to create a co-processor that embodies just the set of KryptoKit functions needed, and that works with a standard off-the-shelf microprocessor. This improves processor performance by doing the functions in hardware, at multiple hundred megahertz clocking, instead of leaving them as slow instructions within the micro. As expected, it brings just the key functions to the system, without substantially increasing the system power consumption.

Conclusions

We have presented the basics for creating your own cryptographic processor. The PicoBlaze design is available and can be downloaded from the Internet. The CoolRunner-II Design Kit is an appropriate physical package for downloading a working design into a 256-macrocell part. If needed, the board can accept a larger capacity part, to do more. We have only pointed the way for such a development, and you are welcome to design your own, using this as a starting point. Go for it.

References

1. An Electronic Decoder for Bose-Chaudhuri-Hocquenghem Error-Correcting Codes, Thomas C. Bartee, David I. Schneider, IRE Transactions on Information Theory, Vol. IT-8, pp s17-s24, Sept. 1962
2. An Implementation of Elliptic Curve Cryptosystems Over F_2^{155}, G.B. Agnew, R.C. Mullin, and S. A. Vanstone, IEEE Journal on Selected Areas in Communications, Vol. II, June 1993
3. CORSAIR: A Smart Card for Public Key Cryptosystems, Dominique de Waleffe, J.J. Quisquater, 1998, Springer-Verlag
4. Design and Implementation of a Crypto Processor and Its Application to Security System, H.W. Kim, Y.J. Choi, M.S. Kim, 2002 International Technical Conference on Circuits/Systems, Computers and Communications, July, 2002.
5. Differential Power Analysis, Paul Kocher, Joshua Jaffe, Benjamin Jun, Advances in Cryptology – Proceedings of Crypto ’99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, 1999.

6. 6. Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards, Rita Mayer-Sommer, pp. 78-92, CHES 2000 Proceedings.
7. "Techniques of Side Channel Cryptoanalysis, James A Muir, Master's Thesis, available on the web at: www.math.uwaterloo.ca/wjamuir/sidechannel.htm.

Further Reading

Application Notes

http://www.xilinx.com/xapp/xapp371.pdf (Galois Field GF (2^m) Multiplier)
http://www.xilinx.com/xapp/xapp372.pdf (Smart Card Reader)
http://www.xilinx.com/xapp/xapp373.pdf (Avoiding CPLD Brownout)
http://www.xilinx.com/xapp/xapp374.pdf (CryptoBlaze)
http://www.xilinx.com/xapp/xapp375.pdf (Timing Model)
http://www.xilinx.com/xapp/xapp376.pdf (Logic Engine)
http://www.xilinx.com/xapp/xapp377.pdf (Low Power Design)
http://www.xilinx.com/xapp/xapp378.pdf (Advanced Features)
http://www.xilinx.com/xapp/xapp379.pdf (High Speed Design)
http://www.xilinx.com/xapp/xapp380.pdf (Cross Point Switch)
http://www.xilinx.com/xapp/xapp381.pdf (Demo Board)
http://www.xilinx.com/xapp/xapp382.pdf (I/O Characteristics)
http://www.xilinx.com/xapp/xapp383.pdf (Single Error Correction Double Error Detection)
http://www.xilinx.com/xapp/xapp384.pdf (DDR SDRAM Interface)
http://www.xilinx.com/xapp/xapp387.pdf (PicoBlaze Microcontroller)
http://www.xilinx.com/xapp/xapp388.pdf (On the Fly Reconfiguration)
http://www.xilinx.com/xapp/xapp389.pdf (Powering CoolRunner-II CPLDs)
http://www.xilinx.com/xapp/xapp393.pdf (8051 Microcontroller Interface)
http://www.xilinx.com/xapp/xapp394.pdf (Interfacing with Mobile SDRAM)
http://www.xilinx.com/xapp/xapp395.pdf (Using DataGATE)
http://www.xilinx.com/xapp/xapp398.pdf (CompactFlash Card Interface)

CoolRunner-II Data Sheets

http://direct.xilinx.com/bvdocs/publications/ds090.pdf (CoolRunner-II Family Datasheet)
http://direct.xilinx.com/bvdocs/publications/ds091.pdf (XC2C32 Datasheet)
http://direct.xilinx.com/bvdocs/publications/ds092.pdf (XC2C64 Datasheet)
http://direct.xilinx.com/bvdocs/publications/ds093.pdf (XC2C128 Datasheet)
http://direct.xilinx.com/bvdocs/publications/ds094.pdf (XC2C256 Datasheet)
http://direct.xilinx.com/bvdocs/publications/ds095.pdf (XC2C384 Datasheet)
http://direct.xilinx.com/bvdocs/publications/ds096.pdf (XC2C512 Datasheet)

CoolRunner-II White Papers

http://www.xilinx.com/publications/products/cool2/wp_pdf/wp165.pdf (Chip Scale Packaging)
http://www.xilinx.com/publications/whitepapers/wp_pdf/wp170.pdf (Security)
http://www.xilinx.com/publications/whitepapers/wp_pdf/wp197.pdf (Cipher Stream Protocol)

http://www.xilinx.com/publications/whitepapers/wp_pdf/wp198.pdf (Cell Phone Handsets)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/26/03	1.0	Initial Xilinx release.