Instruction-Level Distributed Processing for Symmetric-Key Cryptography *

AJ Elbirt †
Electrical and Computer Engineering Department
University of Massachusetts Lowell
One University Avenue
Lowell, MA 01854, USA

Christof Paar ‡
Chair for Communication Security
Dept. of Electr. Eng. and Information Sciences
Ruhr-Universitat Bochum
44780 Bochum, Germany

Abstract

Efficient implementation of block ciphers is critical towards achieving both high security and high-speed processing. Numerous block ciphers have been proposed and implemented, using a wide and varied range of functional operations. As a result, it has become increasingly more difficult to develop a hardware architecture that allows the efficient and fast realization of a wide variety of block ciphers. In an effort to achieve such a hardware architecture, a study of a wide range of block ciphers was undertaken to develop an understanding of the functional requirements of each algorithm. This study led to the development of COBRA, a reconfigurable architecture for the efficient implementation of block ciphers. A detailed discussion of the top level architecture, interconnection scheme, and underlying elements of the architecture will be provided. System configuration and on-the-fly reconfiguration will be analyzed, and from this analysis it will be demonstrated that the COBRA architecture satisfies the requirements for achieving efficient implementation of a wide range of block ciphers that meet the 622 Mbps ATM network encryption throughput requirement.

Keywords: cryptography, algorithm-agility, FPGA, block cipher, VHDL

*This research was supported in part through NSF CAREER award #CCR-9733246.

†Email: Adam_Elbirt@uml.edu

‡Email: cpaar@crypto.ruhr-uni-bochum.de

1 Introduction

High throughput encryption and decryption are becoming increasingly important in the area of high speed networking. Many applications demand the creation of networks that are both private and secure while using public data-transmission links. These systems, known as Virtual Private Networks (VPNs), can demand encryption throughputs at speeds exceeding Asynchronous Transfer Mode (ATM) rates of 622 million bits per second (Mbps). Increasingly, security standards and applications are defined to be algorithm independent. Although context switching between algorithms can be easily realized via software implementations, the task is significantly more difficult when using hardware implementations. The advantages of a software implementation include ease of use, ease of upgrade, ease of design, portability, and flexibility. However, a software implementation offers only limited physical security, especially with respect to key storage [9], [25]. Conversely, cryptographic algorithms that are implemented in hardware are by nature more physically secure as they cannot easily be read or modified by an outside attacker when the key is stored in special memory internal to the device [9]. As a result, the attacker does not have easy access to the key storage area and cannot discover or alter its value in a straightforward manner [25].

When using a general-purpose processor, even the fastest software implementations of block ciphers cannot satisfy the required data rates for bulk data encryption for high-end applications [3], [4], [10], [27], [34]. As a result, hardware implementations are necessary for block ciphers to achieve this required performance level. Al-

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

though traditional hardware implementations lack flexibility with respect to algorithm and parameter switching, reconfigurable hardware devices offer a promising alternative for the implementation of block ciphers. One of the potential advantages of block ciphers implemented in reconfigurable hardware is algorithm agility, the switching of cryptographic algorithms during operation. The majority of modern security protocols, such as Secure Sockets Layer (SSL) or IPsec, allow for multiple encryption algorithms whose use is negotiated on a per-session basis. Whereas algorithm agility can be very costly with traditional hardware, algorithm agility through reconfigurable hardware appears to be an attractive possibility [20], [21]. It is also conceivable that fielded devices will be upgraded with a new encryption algorithm which did not exist (or was not standardized) at design time. In particular, it is very attractive for numerous security products to be upgraded for use of the Advanced Encryption Standard (AES) now that the standardization process is complete [1]. Moreover, applications exist which require modification of a standardized algorithm, e.g., by using proprietary S-Boxes or permutations. Such modifications are easily made with reconfigurable hardware. Although typically slower than Application Specific Integrated Circuit (ASIC) implementations, reconfigurable implementations have the potential of running substantially faster then software implementations. The time and costs for developing a reconfigurable implementation of a given algorithm are much lower than for an ASIC implementation (however, for high-volume applications, ASIC solutions usually become the more cost-efficient choice).

What follows is a brief overview of previous block cipher implementations in reconfigurable hardware. A wide range of block ciphers will then be examined to develop an understanding of the functionality required to implement these algorithms in hardware. This examination will lead to a set of requirements for a reconfigurable architecture designed to achieve efficient block cipher implementations. We will then present our proposed solution, the COBRA reconfigurable architecture.

2 Previous Work

The most frequently implemented block cipher algorithms are DES and the AES candidates. The following summarizes block cipher implementations in FPGAs and other reconfigurable logic. There are also a number of reconfigurable hardware based implementations of public-key algorithms, many of which target elliptic curve cryptosystems due to the significantly reduced operand size as compared to other public-key algorithms, such as RSA. However, a discussion of these implementations is beyond the scope of this investigation.

2.1 FPGA Implementations

The first published implementation of DES in an FPGA achieved a throughput of 26.4 Mbps [19]. However, this implementation required generation of key-specific circuitry for the target Xilinx XC4013-14, requiring recompilation of the implementation for each key. Until recently, the best performance an FPGA implementation of DES has achieved is a throughput of 402.7 Mbps when operating in Electronic Code Book (ECB) mode using a Xilinx XC4028EX-3 FPGA [17]. This implementation employed pipeline design techniques to maximize the system clock frequency at the cost of pipeline latency cycles.

With the advent of run-time reconfiguration and more technologically advanced FPGAs, implementations with throughputs in the range of ASIC performance values have been achieved. The most recent DES implementation employing run-time reconfiguration achieved a throughput of 10.752 Gbps when operating in ECB mode using a Xilinx Virtex XCV150-6 FPGA [21]. The use of the Xilinx run-time reconfiguration software application JBits^{TM} allowed for real-time key-specific compilation of the bit-stream used to program the FPGA, resulting in a smaller and faster design (which operated at 168 MHz) as compared to the design in [17] (which operated at 25.19 MHz). Most recently, a DES implementation achieved a throughput of 12 Gbps when operating in ECB mode using a Xilinx Virtex-E XCV300E-8 FPGA [29] but did not make use of run-time reconfiguration.

Multiple FPGA implementation studies have been presented for the AES candidate algorithm finalists [8], [11], [13], [14], [30], the results of which are found in Table 2.1. Note that feedback modes and non-feedback modes of operation are denoted as *FB* and *NFB* respectively in Table 2.1 and that no throughput results are presented in [30]. The studies performed in [11] and [14] used a Xilinx Virtex XCV1000 as the target FPGA. The study performed in [8] used the Xilinx Virtex family of FPGAs but did not specify the target device. Finally, the study performed in [13] used the Altera Flex EPF10K130EQ-1 as the target FPGA.

2.2 Other Reconfigurable Implementations

A number of reconfigurable architectures have been proposed to accelerate the performance of symmetric-key cryptography. Hybrid architectures composed of a microprocessor core combined with reconfigurable function blocks are typically used to accelerate the performance of a general purpose processor. Reconfigurable function blocks may support on-the-fly reconfiguration to provide more optimized implementations and further improve system performance. The mapping of complex functions to adaptable hardware reduces the instruction fetch and ex-

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

Table 1: AES finalists FPGA implementation studies

Alg	NFB Mode Throughput (Mbps) [14]	NFB Mode Throughput (Mbps) [11]	FB Mode Throughput (Mbps) [11]
MARS	•	•	•
RC6	13100	2400	126.5
Rijndael	12200	1940	300.1
Serpent	16800	5040	444.2
Twofish	15200	2400	127.7

Alg	FB Mode Throughput (Mbps) [8]	FB Mode Throughput (Mbps) [14]	FB Mode Throughput (Mbps) [13]
MARS	101.88	61.0	•
RC6	112.87	142.7	•
Rijndael	353.00	414.2	232.7
Serpent	148.95	431.4	125.5
Twofish	173.06	177.3	81.5

ecute bottleneck common to a software implementation [5], but often causes the delay associated with communication between the microprocessor and the reconfigurable logic block to become the bottleneck within the system [6]. This overhead can be reduced by caching multiple configurations within the reconfigurable function blocks at the cost of more expensive and less flexible hardware [15], [18], [31]. Hybrid architectures targeted at accelerating symmetric-key cryptography include ConCISe, Garp, and MorphoSys [16], [18], [26].

Generalized reconfigurable architectures consist of an interconnected network of configurable logic and storage elements where the granularity of the architecture is dependent upon the target application. Architectures are typically distinguished based on the control of processing resources — SIMD, MIMD, VLIW, systolic, microcoded, etc. The ConCISe, CryptoManiac, Garp, MorphoSys, and PipeRench platforms are examples of generalized reconfigurable architectures targeted at accelerating symmetric-key cryptography [16], [18], [26], [28], [35]. Block cipher implementations on these platforms all achieve a significant performance improvement as compared to general purpose processor implementations. However, none of these implementations match the throughput of equivalent FPGA implementations, falling short of the the 622 Mbps ATM network encryption throughput requirement.

3 The COBRA Architecture

When designing the ***Cryptographic*** (***Optimized*** for ***Block*** Ciphers) ***Reconfigurable Architecture*** (COBRA) for efficient block cipher implementation, a number of architectural requirements were established to facilitate achieving an optimized solution. A specialized reconfigurable architecture that is optimized for the implementation of block ciphers results in a system with greater flexibility as compared to custom ASIC or specialized processor solutions, as well as faster reconfiguration time when compared to a commercial FPGA solution. Overhead and off-chip communication are minimized by maintaining most or all of the system's resources on chip. Internal to the chip, it is imperative that the reconfigurable datapath be tightly coupled with the control mechanisms so that the overhead associated with their interface does not become the system bottleneck [15], [23], [26], [32]. Ensuring full support of algorithm-specific operations requires maximizing the functional density of the datapath. This results in the need for a generalized and run-time reconfigurable datapath with functional blocks optimized to efficiently implement the core operations of the target block cipher space [2], [7], [12], [26], [31], [35], [36], [37]. Because block ciphers are dataflow oriented, a reconfigurable element with coarse granularity offers the best solution for achieving maximum system performance when implementing operations that do not map well to more traditional, fine-grained reconfigurable architectures [7], [16], [18], [24], [26], [33]. Finally, an interconnect matrix must also be designed to support both Feistel networks (common to most block ciphers [25]) as well as other networks, such as Substitution-Permutation (SP) networks.

An analysis of block ciphers was performed to develop a hardware architecture optimized for block cipher implementation. The analysis was restricted to block ciphers that operate on block sizes of 64 and 128 bits as they are representative of algorithms in use that meet current and expected future security requirements. The block ciphers examined were Blowfish, CAST, CAST-128, CAST-256, CRYPTON, CS-Cipher, DEAL, DES, DFC, E2, FEAL, FROG, GOST, Hasty Pudding, ICE, IDEA, Khafre, Khufu, LOKI91, LOKI97, Lucifer, MacGuffin, MAGENTA, MARS, MISTY1, MISTY2, MMB, RC2, RC5, RC6, Rijndael, SAFER K, SAFER+, Serpent, SQUARE, SHARK, SKIPJACK, TEA, Twofish, WAKE, and WiderWake. The block cipher analysis resulted in a list of operations that a specialized reconfigurable architecture must support to guarantee efficient implementation over the range of block ciphers studied (a summary of the analysis results is shown in Table 3). It was determined that a reconfigurable cryptographic processor core should support a 32-bit datapath replicated at least four times to allow for the implementation of algorithms requiring either 64-bit or 128-bit block lengths. The system must also support communication between the 32-bit datapaths. From the perspective of implementing the most common atomic operations in hardware (based on their occurrence; e.g. Table 3), the following operations should be implemented as part of any reconfigurable cryptographic processor core:

- Bitwise XOR, AND, or OR.
- Addition/subtraction modulo 2^8, 2^{16}, 2^{32}.
- Fix shift/rotation.
- Variable data-dependent rotation.

- Multiplication modulo 2^{16} and 2^{32} and squaring modulo 2^{32}.
- Fixed field constant multiplication in the Galois field GF(2^8).
- Look-up table substitution of the forms:
  - 4-bit to 4-bit with paging mode.
  - 8-bit to 8-bit.
  - 8-bit to 32-bit.

Table 2: Occurrence of block cipher atomic operations

Operation	Occurrences
Boolean	40 of 41
Modular Addition and Subtraction	20 of 41
Fixed Shift	25 of 41
Variable Rotation	10 of 41
Modular Multiplication	7 of 41
Galois Field Multiplication	7 of 41
Modular Inversion	1 of 41
Look-Up Table Substitution	30 of 41

3.1 Block Diagram and Interconnect

Figure 1 details the architecture and interconnect of the COBRA architecture. The COBRA architecture allows for distributed processing across a 128-bit datapath via four interconnected 32-bit datapaths (denoted as columns). Each 32-bit datapath interconnects four Reconfigurable Cryptographic Elements (RCEs) which are the primary processing elements within the COBRA architecture. COBRA hardware resources are managed by the user via microcode to optimize a given block cipher implementation for both speed and efficiency.

The basic building block for the COBRA architecture is the Reconfigurable Cryptographic Elements (RCEs). All RCEs in columns 1 and 3 have an additional built-in functional unit allowing for the performance of modular multiplication and squaring — these elements are denoted as RCE MULs. Each RCE operates upon a 32-bit data stream within a 128-bit block. Two byte shufflers are embedded between rows 0 and 1 and rows 2 and 3 to allow for byte-wise permutations. Data flows from top to bottom, passing through a total of four rows of RCEs and RCE MULs and both byte shufflers. Whitening registers are connected to the outputs of row 3 and the resultant output is both the COBRA output bus and an input to the feedback multiplexor, allowing for iterative operation upon the 128-bit data stream. Whitening registers may be configured to perform either bit-wise XOR or mod 2^{32} addition to support the post encryption/decryption key whitening stages required by many block ciphers. Sixteen embedded RAMs (eRAMs) are provided for use as temporary storage space for intermediate values as well as round keys to be used during encryption or decryption. Half of the eRAMs are allocated to columns 0 and 1 while the other half are allocated to columns 2 and 3.

Note that the COBRA interconnect need not match the functionality of a generic interconnection matrix typically implemented in commercial FPGAs due to the top to bottom flow of data. Implementing a fixed interconnection network significantly decreases the layout complexity of the COBRA architecture. Also note that each RCE or RCE MUL receives the full 128-bit data stream. The 128-bit data stream is partitioned into four 32-bit blocks, with bits 31 to 0 mapping to block 0, bits 63 to 32 mapping to block 1, etc. Block 0 is then denoted as the primary input for all RCE elements in column 0, termed *INA*. Similarly, blocks 1 through 3 map to the primary inputs for RCE or RCE MUL elements in columns 1 through 3, respectively. The other three blocks that make up the remainder of the 128-bit data stream are used as secondary input blocks to the RCE or RCE MUL elements for use in operating upon the primary input block. The secondary input blocks are grouped in ascending numerical order, where the lowest numbered block is termed *INB*, the next middle block is termed *INC*, and the highest numbered block is termed *IND*. The eRAM input is termed *INER*. These terms are used in the COBRA assembly language.

3.2 RCEs

Figures 2 and 3 detail the functionality of the RCE and RCE MUL. The elements of the RCE structure are:

- *A*: Bit-wise XOR, AND, or OR.
- *B*: Add or subtract mod 2^8, 2^{16}, or 2^{32}.
- *C*: Look-Up-Tables (LUTs) — four 8-bit to 8-bit or eight pages of eight 4-bit to 4-bit mappings.
- *D*: Multiply mod 2^{16} or 2^{32} or Square mod 2^{32}.
- *E*: Shift left, shift right, or rotate left. Shift and rotate values may be data dependent.
- *F*: GF(2^8) fixed field constant Galois field multiplier.
- *M*: Multiplexor.
- *REG* : Register.

While the data flow through the elements within the RCE structures is fixed, each element may be selectively disabled via the microcode. Therefore, to allow for multiple useful and efficient configurations, the location and ordering of the elements within the RCE structures was carefully engineered based on the results of the block cipher study. For example, *E* elements are placed at the front, rear, and middle of the structures to allow for flexibility in manipulation of the data.

When not operating in short mode, the entire 32-bit input to a *C* element is divided into either four 8-bit blocks or eight 4-bit blocks depending on the mode of operation. The resultant output blocks from the LUTs retain their ordering when recombined to form the final 32-bit output. To provide secondary inputs to the RCE structure datapath, the *M8* and *M[6:1]* elements accept four 32-bit inputs, *B*, *C*, *D*, and *ER*, and provide either a 32-bit or a 5-bit output. The 5-bit output occurs in the cases of *M6*

Figure 1: COBRA architecture and interconnect

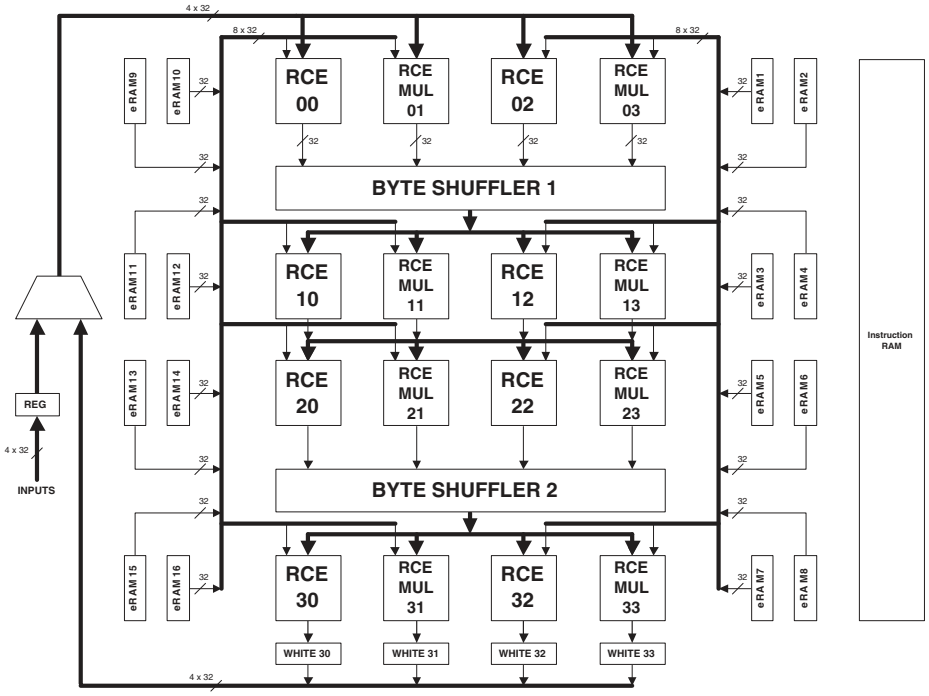
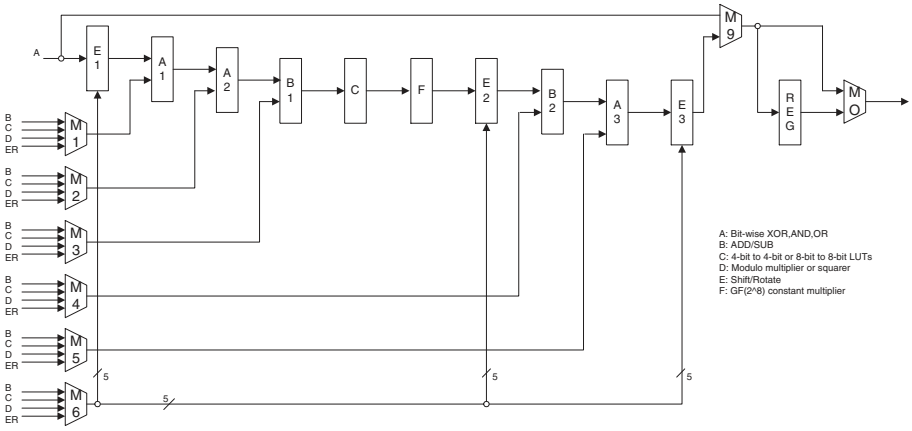


Figure 2: COBRA RCE



All connections are 32-bit bus lines unless otherwise noted.

in an RCE and *M8* in an RCE MUL — all other *M* elements provide 32-bit outputs. The *REG* element provides a registered version of the input and may be enabled or disabled as needed. Note that the RCE structures may be pipelined by enabling the *REG* element in multiple rows for a particular 32-bit datapath.

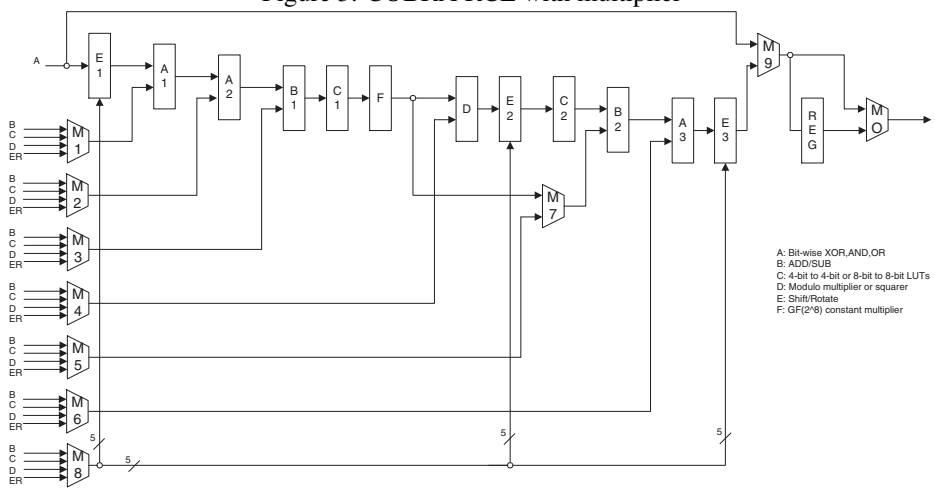
3.3 Instruction Mapping

COBRA operates via a Very Long Instruction Word (VLIW) format. Each instruction is 80 bits and the instruction bus is connected to the control mechanisms for the RCEs, RCE MULs, byte shufflers, eRAMs, whitening registers, and the feedback multiplexor. The instruction bus is driven via the instruction RAM (iRAM) which operates independent from the datapath. This configuration allows the iRAM to reconfigure the datapath during operation. The iRAM is a 12-bit $\times$ 80-bit memory which supports programs of up to 4096 total instructions. Instructions are classified as follows:

- *Configure RCE or RCE MUL*:
- *Enable/disable RCE or RCE MUL outputs*.
- *Configure byte shuffler*.
- *Configure input multiplexor*.
- *Configure eRAM*:

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

Figure 3: COBRA RCE with multiplier



- *Control whitening elements*:
- *Read/write flag register*.
- *Jump to specified address*.
- *No operation (NOP)*.

The instruction word comprises the *operation code*, *slice address*, *element address*, *LUT address*, and *configuration data* fields. The *operation code* type are indicates the operation to be performed. The *operation code* groups similar instructions and use some or all of the *slice address* to indicate the specific COBRA element to be configured. In the case of an RCE or RCE MUL, the *element address* is used to indicate which specific components within the RCE or RCE MUL are to be configured. The *LUT address* is used when configuring either a LUT block or a Galois field fixed field constant multiplier within an RCE or RCE MUL. COBRA elements are configured by instructions that write control words to the associated elements' control registers, allowing for on-the-fly reconfiguration. As an example, to reconfigure the elements within an RCE, an instruction must be executed that contains the control information for each element within the RCE with the individual RCE components' control words forming the *configuration data* field of the instruction.

3.4 Configuration and Control

The COBRA architecture control logic performs a variety of functions to guarantee proper system operation. On power-up, the architecture idles until the iRAM and datapath clocks have been synchronized and the external system indicates that the iRAM has been loaded, automatically initiating the loading and executing of instructions within the iRAM. Generic flags may be used in the microcode to indicate to the external system when to provide data required for key scheduling or other tasks.

Upon completion of key scheduling, the *ready* flag must be raised by the microcode, indicating to the external system that COBRA is ready to begin either encryption or decryption. COBRA halts upon detection of the *ready* flag and waits for the external system to initiate the encryption or decryption process by raising the *go* signal. Upon detection of the *go* signal, the *busy* flag must be raised by the microcode, indicating to the external system that an encryption or a decryption is in progress. Upon completion of an encryption, the *data_valid* flag must be raised by the microcode, indicating to the external system that the COBRA output data is valid. The microcode must reconfigure COBRA as necessary for the start of a new encryption or decryption operation and then jump to the idle point at the start of the process. If COBRA detects that the *go* signal is still active, a new encryption or decryption operation will commence. COBRA will idle while the *go* signal is inactive.

Reconfiguration is achieved via a dual clocking scheme. The maximum operating frequency of the datapath is calculated based upon how it is to be programmed to implement each function required by the given block cipher — key scheduling, encryption, or decryption. The implementation of each of these functions results in a different critical delay path depending on the configuration of the COBRA datapath. The iRAM clock is then set to twice the maximum operating frequency based on a worst case delay analysis performed across these functions. This ensures that the datapath will have settled after the loading and execution of an instruction given that this process requires two iRAM clock cycles to complete.

To determine the datapath clock frequency, the programmer must determine the optimal number of instructions (not necessarily the maximum number) that must

be executed within a datapath clock cycle (the *instruction window*) by examining the number of *overfull* and *underfull* instruction cycles. An *overfull* instruction cycle occurs when the total number of instructions that must be executed within the *instruction window* is greater than the number of instructions that can be executed in a datapath cycle. An *overfull* instruction cycle is completed by disabling the RCE outputs before the end of the datapath cycle and enabling the outputs when reconfiguration is completed. An *underfull* instruction cycle occurs when the total number of instructions that must be executed within the *instruction window* is less than the total number of instructions that can be executed in a datapath cycle, requiring the insertion of NOP instructions.

In most applications, the time associated bulk data encryption or decryption far outweighs the time associated with key scheduling. Therefore, to achieve optimal performance, the programmer must carefully analyze the encryption/decryption segment of the program when choosing the optimal *instruction window* size. This selection is made to minimize both the *overfull* and *underfull* instruction cycles and usually corresponds to the number of instructions required to reconfigure the datapath to compute the output of the current round of the cipher. It is important to note that the first and last round of a cipher typically require additional overhead instructions to perform operations such as the enabling/disabling of key whitening or the addition/removal of particular functions. Therefore, the programmer must focus on the intermediate rounds when choosing the *instruction window* size, typically selecting the *window size* such that none of the intermediate rounds require an *overfull* instruction cycle. Once the *window size* has been selected, the datapath clock frequency is calculated as $F_{DP} = F_{iRAM}/(2 \times windowsize)$.

4 Results

The COBRA architecture was implemented in VHDL using a bottom-up design and test methodology. Key scheduling and encryption were either coded in COBRA assembly language and assembled into microcode or written directly as microcode. System operation was controlled via a VHDL test bench that served to load the iRAM with the microcode, apply the control signals to the architecture, and examine the encrypted data for validity. The iRAM clock frequency was set in the test bench based on the output of the timing analyzer. The datapath clock frequency was set after examining the microcode and optimizing the instruction stream to minimize both *underfull* and *overfull* instruction cycles.

A subset of the block ciphers studied was chosen for implementation in the COBRA architecture. In particular, ciphers were chosen based on the orthogonality of their core functions in an effort to exercise as many of the architecture's features as possible. The 128-bit block ciphers RC6, Rijndael, and Serpent were selected for implementation based on their core element requirements spanning the set of required functionality for the COBRA architecture. Given the large number of implementations of DES and IDEA, both block ciphers were also considered for implementation but proved to be poorly suited to the COBRA architecture in its current form. DES requires both an initial and a final permutation, each of which is a bit-wise reordering of the 64-bit data. While bit-wise shifts and rotations are possible, bit-wise permutations are extremely difficult to implement. A 32-bit bit-level shuffler was considered for addition to the RCE or RCE MUL structures, the decision was made to avoid the implementation of special purpose hardware for specific ciphers design and to maintain the coarse-grained nature of the COBRA architecture for the initial design. In the case of IDEA, all core operations are supported via standard COBRA functionality except for the $mod\, 2^{16} + 1$ multiplication. This type of operation, while easily implemented as a special function block, would be highly specific to IDEA , and was therefore not implemented. However, given the large number of currently deployed implementations of DES and IDEA, addition of the special purpose hardware required for improved performance of these ciphers is an enhancement that may become necessary in future generations of the COBRA architecture.

4.1 Performance Analysis

Table 4.1 details the performance of the COBRA implementations of RC6, Rijndael, and Serpent. Up to two rounds of RC6, two rounds of Rijndael, or one round of Serpent may be mapped to the COBRA architecture in its current form. Data for implementations with more rounds assumes implementation within an architecture expanded by increasing both the iRAM address space and the number of rows, byte shufflers, and eRAMs with corresponding additions to the instruction set to support configuration of the new hardware elements. Timing estimates for the RCEs and RCE MULs were obtained from Synopsys Design Compiler targeting a 0.35 micron library.

The cycle counts in Table 4.1 are for pipelined implementations operating on multiple blocks of data in non-feedback mode where the round function is used as the atomic unit of the pipeline and the pipeline latency is assumed to be met. Note that when all rounds of an algorithm are implemented within the COBRA architecture, the instructions required for on-the-fly reconfiguration are eliminated, resulting in a greatly reduced cycle count for the implementation. Moreover, as evidenced by the sixteen round Serpent implementation, it is possible that the

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

cycles required to output the blocks in the pipeline overshadows the performance gain achieved through operating on multiple blocks of data simultaneously.

COBRA performance nears that of equivalent implementations targeting a Xilinx Virtex XCV1000 FPGA [11], with the performance gap decreasing as the number of rounds implemented for a given block cipher increases. Note that the data provided in [14] only details peak performance results and does not include performance data for intermediate degrees of partial pipelining and was therefore not used for this comparison. Also note that the clock frequencies for COBRA implementations remain constant for each block cipher as the number of rounds increases. These implementations use the block cipher round function as the atomic unit of the pipeline and the implementations do not increase the pipeline depth within the round functions.

Table 3: COBRA encryption performance comparison

Alg	Rnds	Clock Cycles	Clock Freq (MHz)	Throughput (Mbps)	Equivalent FPGA Throughput (Mbps) [11]
RC6	1	145	60.975	53.83	250.0
RC6	2	73	60.975	106.92	497.4
RC6	4	38	60.975	205.39	891.3
RC6	5	30	60.975	260.16	1067.0
RC6	10	15	60.975	520.32	2397.9
RC6	20	2	60.975	**3902.40**	•
Rijndael	1	57	102.041	229.14	294.2
Rijndael	2	22	102.041	593.69	575.3
Rijndael	5	22	102.041	593.69	1165.8
Rijndael	10	9	102.041	**1451.25**	•
Serpent	1	273	54.054	25.34	77.0
Serpent	8	35	54.054	197.68	1241.6
Serpent	16	56	54.054	123.55	•
Serpent	32	3	54.054	**2306.30**	5035.0

4.2 Hardware Resource Requirements

The VHDL used to implement the COBRA architecture was synthesized using Exemplar Logic's LeonardoSpectrum Level 3 version v2001_1d.46 targeting the ADK TSMC 0.35 micron library. Table 4.2 summarizes the gate counts for each configurable element within a COBRA RCE or RCE MUL. From Table 4.2 it is clear that the dominant element in terms of area is the *C* element. Four 128×4 look-up-tables and four 256×8 look-up-tables are implemented within the *C* element, resulting in a total of 10,240 storage bits. The gate count listed for the elements in Table 4.2 is the total gate count for all elements of that type, resulting in a total gate count of nearly 6.7 million gates for the COBRA architecture. Note however that the gate counts for memory elements are significantly inflated due to the synthesis tool using D flip-flops to implement memory elements as opposed to SRAM blocks. It is estimated that memory element gate counts will decrease by a factor of three should SRAM blocks be used (using an estimate of four gates per SRAM bit [22]), resulting in a total of approximately 2.5 million gates.

The value added by the COBRA architecture versus the FPGA implementations in [11] occurs in the areas of scalability, cost, and design cycle time. The COBRA architecture easily scales up or down by adding rows or columns to the base architecture as tiling is readily performed at the RCE level. From a cost perspective, while the COBRA architecture requires 2.5 times the logic resources as the FPGA used in [11], the COBRA layout is significantly less complex due to the use of a fixed interconnect. We believe that the use of a fixed interconnect will more than offset the cost of the additional logic resources, resulting in the design and manufacturing costs of the COBRA architecture being far smaller than those associated with the FPGA used in [11]. Finally, the time required to implement and verify algorithms mapped to the COBRA architecture is expected to require significantly less time as compared to the schematic and language based design methodologies used to target FPGAs due to the assembly language design methodology and the specialized cryptographic elements that COBRA employs.

Table 4: Reconfigurable element gate counts

Configurable Element	Gates
A	172
B	1,012
C	98,624
D	5,243
E	887
F	10,606
4-to-1 Multiplexor, Grouping of 32	160
4-to-1 Multiplexor, Grouping of 5	26
2-to-1 Multiplexor, Grouping of 32	83
32-Bit Register	267

Table 5: COBRA architecture gate counts

Element	Gates
RCE/RCE MUL Array	2,692,840
Byte Shufflers	8,556
Input Multiplexors	332
Whitening Blocks	3,128
Embedded RAMs	1,210,640
Instruction RAM	2,773,184
Datapath Overhead	2,464
Chip Overhead	370
Total	**6,691,514**

A cycle-gates (CG) product similar to the classical time-area (TA) product may be calculated. The normalized CG products for the RC6, Rijndael, and Serpent implementations are shown in Table 4.2. As previously noted, up to two rounds of RC6, two round of Rijndael, or one round of Serpent may be mapped to the COBRA architecture in its current form. Gate counts for implementations with more rounds assumes implementation within an architecture expanded by increasing both the iRAM address space and the number of rows, byte shufflers, and eRAMs with corresponding additions to the instruction

set to support configuration of the new elements.

Table 6: COBRA encryption CG product

Alg	Rnds	Clock Cycles	Gates	CG Prod	Norm CG Prod
RC6	1	145	6,691,514	970,269,530	13.477
RC6	2	73	6,691,514	488,480,522	6.785
RC6	4	38	9,544,240	362,681,120	5.038
RC6	5	30	11,197,598	335,927,940	4.666
RC6	10	15	19,464,388	291,965,820	4.055
RC6	20	2	35,997,968	71,995,936	1.000
Rijndael	1	57	6,691,514	381,416,298	2.591
Rijndael	2	22	6,691,514	147,213,308	1.000
Rijndael	5	22	13,970,782	307,357,204	2.088
Rijndael	10	9	27,783,940	250,055,460	1.699
Serpent	1	273	6,691,514	1,826,783,322	5.140
Serpent	8	35	29,736,440	1,040,775,400	2.928
Serpent	16	56	59,315,256	3,321,654,336	9.346
Serpent	32	3	118,472,888	355,418,664	1.000

In the case of both RC6 and Serpent, the best CG product occurs when all rounds of a cipher can be implemented in the COBRA architecture. However, intermediate degrees of unrolling do not always result in an improved CG product, as evidenced by the five round implementation of Rijndael and the sixteen round implementation of Serpent. In each of these cases, the decrease in encryption cycles realized through loop unrolling is overshadowed by the increase in required hardware resources. However, in the case of Rijndael, the two round implementation exhibits a slightly better CG product versus the ten round implementation. In this case, the increase in performance gained through the use of a full-length pipeline does not justify the required increase in hardware resources. However, a significant argument in favor of full-length pipeline implementations is that these implementations meet the ATM network encryption throughput requirement of 622 Mbps for all three algorithms.

5 Conclusions

An investigation of block cipher implementations in reconfigurable hardware has been presented and a wide range of block ciphers were examined. This examination led to an understanding of the functionality required to implement these algorithms through the characterization of their key components. This characterization led to a set of requirements used to develop COBRA, a reconfigurable architecture designed to achieve efficient block cipher implementations. A detailed discussion of the top level architecture, interconnection scheme, and underlying elements of the architecture was provided along with an examination of system configuration and on-the-fly reconfiguration. Algorithms were mapped to the COBRA architecture and implemented using the COBRA assembly language and microcode format. Performance data was gathered in terms of cycle counts so as to evaluate the implementations of the targeted block ciphers. This evaluation demonstrated that the COBRA architecture achieved efficient implementation of a wide range of of block ciphers that meet the 622 Mbps ATM network encryption throughput requirement and approach the performance levels of custom hardware implementations.

6 Acknowledgement

We would like to thank Yusuf Leblebici from the Swiss Federal Institute of Technology for the useful discussions we had with him at the beginning of the COBRA project.

References

[1] Advanced Encryption Standard. At http://www.nist.gov/aes.

[2] C. Alippi, W. Fornaciari, L. Pozzi, M. Sami, and P. L. D. Vinci. Determining the Optimum Extended Instruction-Set Architecture for Application Specific Reconfigurable VLIW CPUs. In *Proceedings of the 12th International Workshop on Rapid System Prototyping, RSP 2001*, pages 50–56, Monterey, California, USA, June 25–27 2001.

[3] K. Aoki and H. Lipmaa. Fast Implementations of AES Candidates. In *The Third Advanced Encryption Standard Candidate Conference*, pages 106–122, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.

[4] L. Bassham, III. Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standard. In *The Third Advanced Encryption Standard Candidate Conference*, pages 136–148, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.

[5] K. Bondalapati and V. K. Prasanna. Reconfigurable Computing: Architectures, Models and Algorithms. *Current Science*, 78(7):828–837, 2000.

[6] K. K. Bondalapati. *Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures*. PhD thesis, University of Southern California, Los Angeles, California, USA, August 2001.

[7] D. C. Chen and J. M. Rabaey. A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths. *IEEE Journal of Solid-State Circuits*, 27(12):1895–1904, December 1992.

[8] A. Dandalis, V. K. Prasanna, and J. D. P. Rolim. A Comparative Study of Performance of AES Final Candidates Using FPGAs. In Ç. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS 1965, Worcester, Massachusetts, USA, August 2000. Springer-Verlag.

[9] R. Doud. Hardware Crypto Solutions Boost VPN. *Electronic Engineering Times*, (1056):57–64, April 12 1999.

[10] J. Dray. NIST Performance Analysis of the Final Round Java^{TM} AES Candidates. In *The Third Advanced Encryption Standard Candidate Conference*, pages 149–160, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.

[11] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(4):545–557, August 2001.

[12] J. G. Eldredge and B. L. Hutchings. Density Enchancement of a Neural Network Using FPGAs and Run-Time Reconfiguration. In D. A. Buell and K. L. Pocek, editors, *Second Annual IEEE Symposium on Field-Programmable*

Custom Computing Machines, FCCM '94, pages 180–188, Napa Valley, California, USA, April 10-13 1994. IEEE, Inc.

[13] V. Fischer. Realization of Round 2 AES Candidates Using Altera FPGA. World Wide Web — http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html, 2000.

[14] K. Gaj and P. Chodowiec. Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays. In *RSA Security Conference*, San Francisco, California, USA, April 8–12 2001. Springer-Verlag.

[15] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimaera Reconfigurable Function Unit. In *Fifth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '97*, pages 87–96, Napa Valley, California, USA, April 16–18 1997.

[16] J. Hauser and J. Wawrzynek. Garp: A MIPS Processor With A Reconfigurable Coprocessor. In *Fifth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '97*, Napa Valley, California, USA, April 16–18 1997.

[17] J.-P. Kaps and C. Paar. DES auf FPGAs (DES on FPGAs, in German). *Datenschutz und Datensicherheit*, 23(10):565–569, 1999. invited contribution.

[18] B. Kastrup, A. Bink, and J. Hoggerbrugge. ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator. In *Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '99*, pages 92–101, Napa Valley, California, USA, April 21-23 1999. IEEE, Inc.

[19] J. Leonard and W. Magione-Smith. A Case Study of Partially Evaluated Hardware Circuits: Keyspecific DES. In W. Luk, P. Cheung, and M. Glesner, editors, *Seventh International Workshop on Field-Programmable Logic and Applications, FPL '97*, London, UK, September 1–3 1997. Springer-Verlag.

[20] C. Patterson. A Dynamic Implementation of the Serpent Block Cipher. In Çetin K. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS 1965, pages 142–155, Worcester, Massachusetts, USA, August 17–18 2000. Springer-Verlag.

[21] C. Patterson. High Performance DES Encryption in Virtex^{TM} FPGAs Using JBits^{TM}. In K. L. Pocek and J. M. Arnold, editors, *Eighth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '00*, pages 113–121, April 2000.

[22] B. Penner. What is Gate Count? What Are Gate Count Metrics for Virtex/Spartan-II/4K Devices? Electronic Mail Personal Correspondance, January 2003. Xilinx Inc.

[23] C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. M. Arnold, and M. Gokhale. The NAPA Adaptive Processing Architecture. In *Sixth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '98*, pages 28–37, Napa Valley, California, USA, April 15-17 1998. IEEE, Inc.

[24] G. Sassatelli, G. Cambon, J. Galy, and L. Torres. A Dynamically Reconfigurable Architecture for Embedded Systems. In *Proceedings of the 12th International Workshop on Rapid System Prototyping — RSP 2001*, pages 32–37, Monterey, California, USA, June 25–27 2001.

[25] B. Schneier. *Applied Cryptography*. John Wiley & Sons Inc., New York, New York, USA, 2nd edition, 1996.

[26] H. Singh, M. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho. MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. *IEEE Transactions on Computers*, 49(5):465–481, May 2000.

[27] A. Sterbenz and P. Lipp. Performance of the AES Candidate Algorithms in Java^{TM}. In *The Third Advanced Encryption Standard Candidate Conference*, pages 161–168, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.

[28] R. Taylor and S. Goldstein. A High-Performance Flexible Architecture for Cryptography. In Ç. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 1999*, volume LNCS 1717, pages 231–245, Worcester, Massachusetts, USA, August 1999. Springer-Verlag.

[29] S. Trimberger, R. Pang, and A. Singh. A 12 Gbps DES Encryptor/Decryptor Core in an FPGA. In Çetin K. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS 1965, pages 156–163, Worcester, Massachusetts, USA, August 17–18 2000. Springer-Verlag.

[30] N. Weaver and J. Wawrzynek. A Comparison of the AES Candidates Amenability to FPGA Implemenation. In *The Third Advanced Encryption Standard Candidate Conference*, pages 28–39, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.

[31] M. J. Wirthlin and B. L. Hutchings. A Dynamic Instruction Set Computer. In *Third Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '95*, pages 99–107, Napa Valley, California, USA, April 19-21 1995. IEEE, Inc.

[32] R. Wittig and P. Chow. OneChip: An FPGA Processor With Reconfigurable Logic. In *Fourth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '96*, 1996.

[33] A. Wolfe and J. P. Shen. Flexible Processors: A Promising Application-Specific Processor Design Approach. In *Proceedings of the 21st Annual Workshop on Microprogramming and Microarchitecture — MICRO '21*, pages 30–39, San Diego, California, USA, November 1988.

[34] T. Wollinger, M. Wang, J. Guajardo, and C. Paar. How Well Are High-End DSPs Suited for the AES Algorithms? In *The Third Advanced Encryption Standard Candidate Conference*, pages 94–105, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.

[35] L. Wu, C. Weaver, and T. Austin. CryptoManiac: A Fast Flexible Architecture for Secure Communication. In B. Werner, editor, *Proceedings of the 28th Annual International Symposium on Computer Architecture — ISCA-2001*, pages 110–119, Goteborg, Sweden, June 30–July 4 2001.

[36] A. K. Yeung and J. M. Rabaey. A 2.4 GOPS Data-Driven Reconfigurable Multiprocessor IC for DSP. In *Proceedings of the 1995 IEEE International Solid-State Circuits Conference*, pages 108–109, 346, San Francisco, California, USA, February 15–17 1995.

[37] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey. A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications. In *Proceedings of the 2000 IEEE International Solid-State Circuits Conference*, pages 68–69, 448, San Francisco, California, USA, February 7–9 2000.