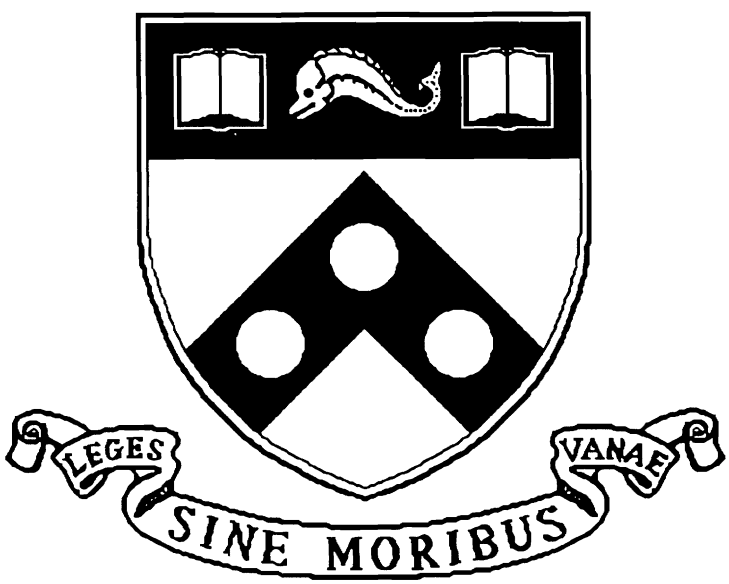
Exploiting Parallelism In Hardware Implementation of the DES

MS-CIS-93-22
DISTRIBUTED SYSTEMS LAB 20

Albert G. Broscius
Jonathan M. Smith



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

February 1993

Exploiting Parallelism in Hardware Implementation of the DES

Albert G. Broscius
Distributed Systems Lab
Dept. of CIS
Univ. of Pennsylvania
Phila PA, 19104-6389 USA
broscius@cis.upenn.edu

Jonathan M. Smith
Distributed Systems Lab
Dept. of CIS
Univ. of Pennsylvania
Phila, PA, 19104-6389 USA
jms@cis.upenn.edu

Abstract

The Data Encryption Standard algorithm has features which may be used to advantage in parallelizing an implementation. The kernel of the algorithm, a single round, may be decomposed into several parallel computations resulting in a structure with minimal delay. These rounds may also be computed in a pipelined parallel structure for operations modes which do not require cryptext feedback. Finally, system I/O may be performed in parallel with the encryption computation for further gain. Although several of these ideas have been discussed before separately, the composite presentation is novel.

1 Introduction

[1] The Data Encryption Standard (DES) is probably the most widely used publicly available secret-key algorithm. Since its introduction by the National Bureau of Standards (NBS) in 1977[FIPS46], DES implementations have improved greatly in encryption rate. Yet, typical computer communication rates have also increased significantly during the same period. Today's high-performance computer networks extend still further the encryption bandwidth needed for adequate performance of secure systems [Giga90]. Thus, we examine means to increase the throughput of a DES implementation to satisfy these demands.

We discuss parallel approaches for several levels of an implementation. At the lowest level, the kernel of the algorithm can be split into several parallel computations for increased speed. By generating subkeys one cycle in advance, the time required can be effectively overlapped with the use of the subkey in the rest of the round operation. An additional overlap can be made of the two stages of exclusive-or (XOR) gates at the expense of increased complexity and gate-count.

One level upward in the hierarchy, the use of multiple round implementations can

[1] This research was supported by NSF and DARPA through the Corporation for National Research Initiatives, and by Bellcore through Project DAWN.

increase computation bandwidth if the DES mode of operation chosen does not require feedback of ciphertext. Of the official modes[FIPS81], this requirement rules out all but the Electronic Code Book (ECB) method. Unfortunately, ECB is known to be susceptible to plaintext frequency-analysis based attacks since multiple identical input blocks result in the same output cryptext block. We discuss in section 3 of this paper a proposed operating method [Feldmeier91] that resists this attack yet does not require feedback of ciphertext.

Finally, at the system level, the processing of I/O concurrent with DES computation provides for continuous operation of the encryption unit. In addition to this buffering, the use of Direct Memory Access (DMA) for encryption allows the host processor to continue other work concurrently with the ongoing encryption.

2 Algorithm Kernel

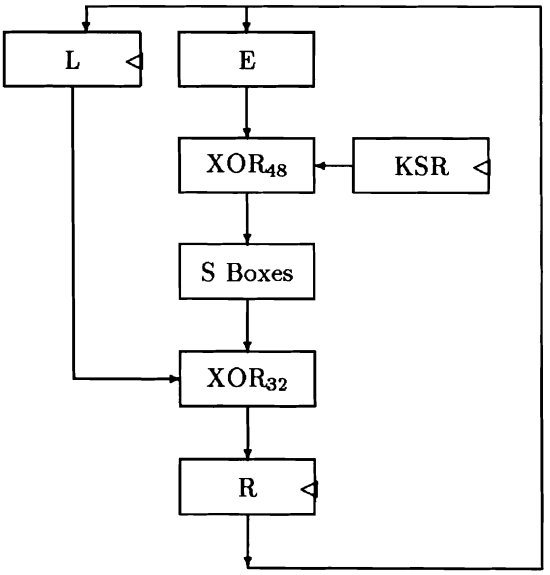


Figure 1: Simple Implementation of Algorithm Kernel

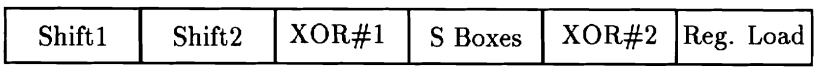


Figure 2: Timing Diagram for Circuit of Fig. 1

The kernel of the DES algorithm consists of four operations: key generation, key mixing, substitution table lookup and data mixing between the R and L words. This kernel

is repeated for sixteen iterations with only one key generation parameter dependent on iteration number. This single parameter specifies one or two shifts of the circular registers from which the current key is derived.

A straightforward implementation of the kernel is depicted in Fig.1. The box labeled *KSR* represents the circular shift registers which hold the key data. These are to be clocked either once or twice depending on the iteration number. Once the key has been shifted, the key-mixing box denoted XOR_{48} outputs the modulo 2 sum of the key data with the extended R data after a propagation delay interval. The *S Boxes* then begin their access time delay interval before output of their results. The box marked XOR_{32} then begins mixing in data from the L register. After a propagation delay of an XOR gate, data are ready at the input to the R register. Once a register setup-time has passed, the R and L registers may be clocked once. A register propagation delay later, the cycle may begin once more.

Timing analysis reveals that the critical timing path results from two shifts of the key registers, the keying XOR array, the S Box table lookup, the R-L mixing XOR array, plus the register loading delays. A simplified timing diagram is shown in Fig. 2. The critical path timing defines the limiting rate at which round computation may proceed. Assuming these delays are minimal, the only way to improve the critical path timing is to modify the circuit so that these sequential processes become concurrent. We will now examine several ways to achieve this concurrency.

2.1 Key Parallelism

Separating the key generation from the remaining three stages of the algorithm kernel can reduce the critical path timing. This approach saves delay by updating the key shift register in anticipation of the next iteration simultaneous with the remaining operations in the current iteration. An additional key latch is introduced to buffer the key value for the current iteration on the input to the key-mixing XOR stage as shown in Fig. 3.

This key parallelism was suggested by Diffie and Hellman [Diffie77] in their timing approach for the proposed DES key-search device. They did not include the additional key latch but instead relied on strict control of key shift timing with respect to the overall R-L clock timing to prevent a race condition. Our introduction of the key latch allows greater tolerance in clock provision by ensuring that the key data input to the key-mixing XOR cannot change during the iteration cycle.

Later, a different key-parallel approach incorporating a multiplexer (MUX) was used

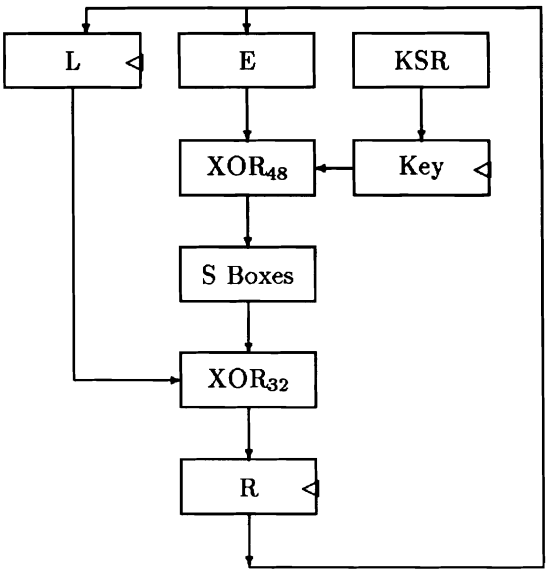


Figure 3: Pipelined Key Generation Algorithm Kernel

	Shift1	Shift2	Key Latch
XOR#1	S Boxes	XOR#2	Reg. Load

Figure 4: Timing Diagram for Circuit of Fig. 3

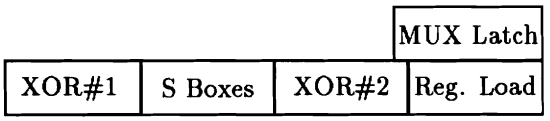


Figure 5: Timing Diagram for Circuit of Fig. 4

by Fairfield et al [Fair84]. Their MUX approach allowed either one or two shifts in either the encryption or decryption direction to be performed in one clocking operation. Additionally, a key loading operation could be selected by the multiplexer. This shortens the time required for the key generation somewhat since the MUX propagation delay is likely to be much lower than a full key shift cycle. More importantly, since the key shift register (KSR) no longer generates intermediate results, as it did when two shifts were required for a given iteration, the extra key latch introduced above to prevent race conditions is no longer necessary so the block diagram reverts to that of Fig.1. A simplified timing diagram for this arrangement is shown in Fig. 5.

2.2 XOR Rearrangement

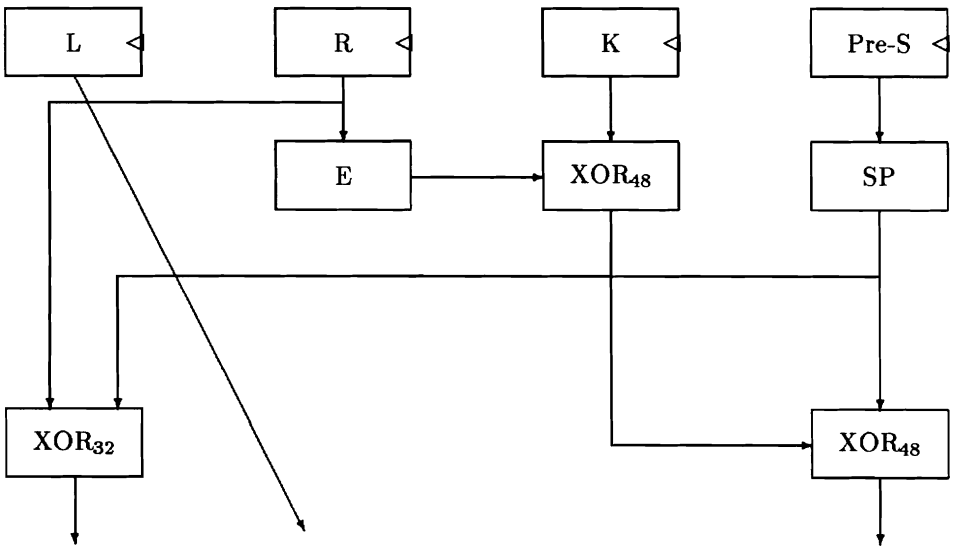


Figure 6: Block Diagram of Datapath with Single XOR Delay in Critical Path

Since the XOR summation is a bitwise linear operation, the order in which XOR operations are performed does not alter their algebraic correctness. Thus, these operations may be grouped (or associated) in any order whatsoever without changing the final results of the combined operations.

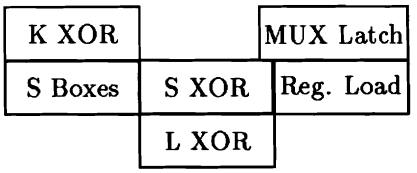


Figure 7: Timing Diagram for Circuit of Fig. 6

If we remove the labeling of R and L in a pair of consecutive rounds of the DES, we observe that there are two stages of XORs, where the second follows the first directly with only the E expansion separating the two. Since E may be commuted with the XOR at the cost of additional bits in the XOR array, we may combine the 48-bit XOR and the 32-bit XOR into a single 48-bit XOR, which would remain in the critical timing path, and a 48-bit XOR which would be computed concurrent with the previous S Box table lookup operation. This transformation also requires the addition of a 32-bit XOR array since the R value is no longer produced in the critical timing path.

3 Multi-Round Parallelism

Using multiple stages in parallel limits the computation of feedback modes of encipherment. Since a parallel implementation begins processing subsequent blocks before completion of a current block's encryption, modes that use the cryptext of a prior block cannot be computed at the full bandwidth that a feedforward mode can achieve. Three of the four modes defined by the NBS for use of the DES require feedback.

A feedforward operation mode proposed by Feldmeier and McAuley appears to overcome the weaknesses of ECB. Their modified ECB mode of operation combines a sequence number with each plaintext block using the XOR operation. This approach should thwart frequency-analysis style attacks since multiple instances of a plaintext block are mapped to different cryptext elements. Using a 64 bit sequence number, cycling of this space would take place in 2^{67} bytes of a data stream. This mode allows independent processing of data elements by avoiding the interdependence of subsequent encryption operations found in feedback modes.

An intermediate alternative between feedback and feedforward modes is the use of multiple interleaved chains. The degree of interleaving can be chosen to allow for as much bandwidth gain through parallelism as needed.

3.1 Pipeline

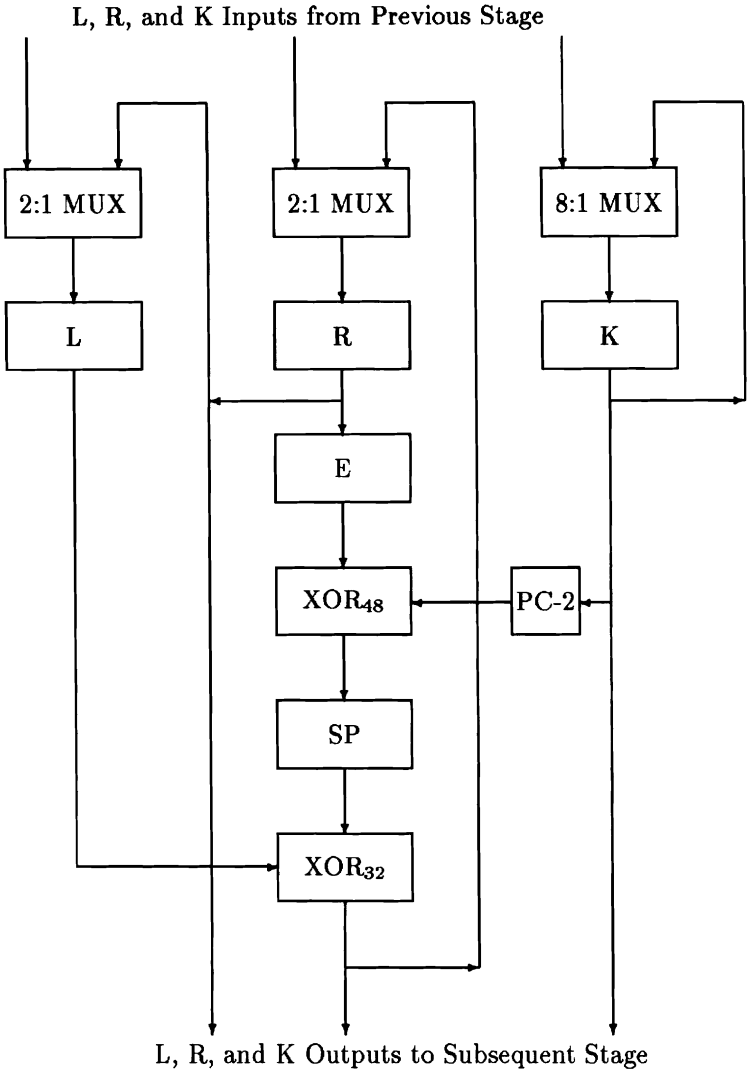


Figure 8: Pipeline Segment for DES with Key Transport

The parallel computation elements may each be configured to implement a fraction of the rounds in a pipeline approach or each element may operate independently in a computation farm approach which we discuss in section 3.2. A pipeline of elements may be configured from two, four, eight or sixteen round implementations. Each element would operate on a 64-bit block for an equal number of cycles needed to partition the algorithm computation. Thus, a two round pipeline would execute eight cycles on each of the processors in the pipeline. Similarly, a four round pipeline would execute four cycles

on each processor.

When keying needs to be updated frequently, the pipeline style allows a matching of the datapath flow with a parallel keypath. In this way, data blocks are accompanied by their key throughout the computation. Switching keys between successive datablocks without flushing the pipeline is made possible since the key and data streams are synchronized. Each stage of the pipeline has the structure depicted in Fig. 8. Note that the 8:1 key MUX actually selects between four different shifted versions of either the current key or the input key from the previous stage.

For infrequent key changes, the tradeoff in keying interconnection may not be worthwhile as compared to maintaining separate key registers for each stage of the pipeline. Each round would then maintain its own key load (shadow) register in this approach. The standard's key shifting sequence would be partially executed on each stage. Since a partial execution of the key schedule would not result in a complete cycling of the keytext, the key would be reloaded from the shadow register when it had completed its share of the computation on a data block.

A limitation of the pipeline approach is the bandwidth ceiling imposed by the number of rounds in the algorithm, sixteen. This means that a single pipeline of processors cannot provide more than sixteen times the encryption bandwidth of a single processor. Additionally, the pipeline suffers in scalability since the number of stages possible is restricted to be a factor of sixteen. This deficit is most notable when considering an upgrade of a pipeline: to gain any increase in bandwidth requires a doubling in computation resources.

3.2 Computation Farm

Instead of a pipeline approach, multiple devices may be configured as a computation farm with each given subsequent plaintext elements to process. This configuration allows for smooth increase in available encryption bandwidth since the number of rounds used need not be a power of two as in the case of the pipeline parallelism.

Managing the farm requires logic similar to that used in FIFO buffers. A counter to keep track of the next available processor and the last busy processor are required. A generalized depiction of the interconnection is shown in Fig.9, using an Input Manager and Output Manager to coordinate operations.

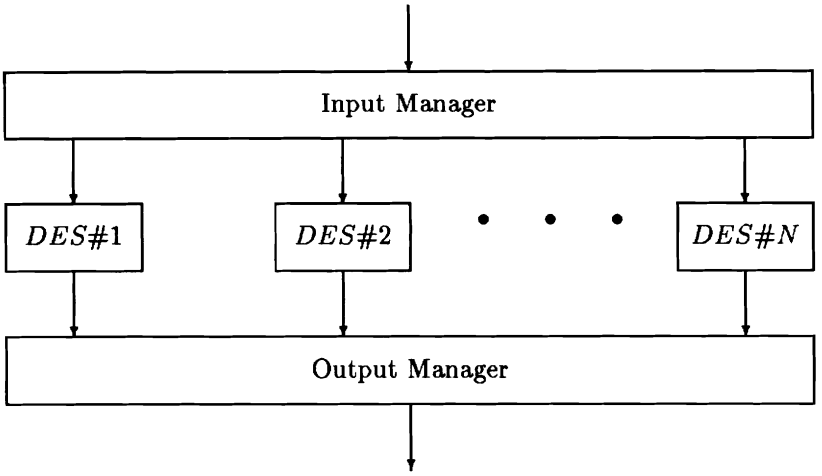


Figure 9: Computation Farm Block Diagram

4 System-Level Parallelism

To maintain constant throughput rates requires careful consideration of the encryption system's interface or input/output section. Overlap of the input, output and encryption processes of subsequent text blocks provides high throughput [Ver88]. Similarly, DMA support decouples the host processor from the encryption function to allow CPU processing of other tasks to proceed in parallel with the encryption request[Anderson87].

5 Conclusion

Parallel aspects of the DES may be exploited at three levels: within the algorithm kernel, through duplication of the algorithm kernel, and in the encryption processor I/O design. Consideration of operation mode also impacts the maximum performance attainable – nonstandard or hybrid operations modes should be studied further as a means of increasing bandwidth without compromising security.

As part of our work with the Aurora network testbed [Giga90], we have developed a DES board [Broscius91] using SSI TTL and MSI PALs using the MUX key register approach. Testing of the wirewrapped prototype indicated an encryption rate of 93 Mbps. Further work on a DMA interface to the MicroChannel interface bus of the IBM RS/6000 is planned. However, recent announcement by VLSI Technology of their VM007 encryption processor [VLSI91] with 192 Mbps performance obsoletes our discrete

approach and will most likely be used in our final version.

References

[Anderson87] David P. Anderson and P. Venkat Rangan, *High-Performance Interface Architectures for Cryptographic Hardware*, EUROCRYPT '87 Proceedings, Springer-Verlag, Amsterdam, 1987

[Broscius91] Albert G. Broscius, *Hardware Analysis and Implementation of the NBS Data Encryption Standard*, MSE Thesis, CIS Dept., Univ. of Penn., May 1991

[Davio83] Marc Davio, Yvo Desmedt, Marc Fosseprez, Rene Govaerts, Jan Hulsbosch, Patrik Neutjens, Philippe Piret, Jean-Jacques Quisquater, Joos Vandewalle and Pascal Wouters, *Analytical Characteristics of the DES* Advances in Cryptology: Proceedings of CRYPTO 83, Plenum Press, New York, 1984

[Denning82] Dorothy Denning, *Cryptography and Data Security* , Addison-Wesley (1982)

[Diffie77] Whitfield Diffie and Martin E. Hellman, *Exhaustive Cryptanalysis of the NBS Data Encryption Standard* , IEEE Computer Vol. 10 No. 6, June 1977 pp. 74-84

[Fair84] R.C.Fairfield, A. Matsuevich and J. Plany, *An LSI Digital Encryption Processor*, Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, New York, 1985

[FIPS46] National Bureau of Standards, *Federal Information Processing Standard #46: The Data Encryption Standard*

[FIPS81] National Bureau of Standards, *Federal Information Processing Standard #81: Operational Modes of the DES*

[Feldmeier91] Anthony McAuley and David C. Feldmeier, *Minimizing Protocol Ordering Constraints to Improve Performance* , Submitted for publication, Available via anonymous *ftp* from Internet host *thumper.bellcore.com*

[Giga90] Anonymous, *Gigabit Network Testbeds* , IEEE Computer, Vol. 23 No. 9

[Ver88] Ingrid Verbauwhede, Frank Hoornaert, Joos Vandewalle and Hugo de Man, *Security and Performance Optimization of a New DES Data Encryption Chip* , IEEE Journal of Solid State Circuits Vol. 23, No. 3, pp. 647-656, June 1988

[VLSI91] VLSI Technology, Inc., *VM007 Data Encryption Processor* , Tempe, AZ 1991