High-performance FPGA implementation of DES using a novel method for implementing the key schedule

Article *in* IEE Proceedings - Circuits Devices and Systems · November 2003

DOI: 10.1049/ip-cds:20030574 · Source: IEEE Xplore

2 authors:

Maire Mcloone
Queen's University Belfast
91 PUBLICATIONS **1,671** CITATIONS
SEE PROFILE

J.V. Mccanny
Queen's University Belfast
208 PUBLICATIONS **2,794** CITATIONS
SEE PROFILE

High-performance FPGA implementation of DES using a novel method for implementing the key schedule

M. McLoone and J.V. McCanny

Abstract: A generic, parameterisable key scheduling core is presented, which can be utilised in pipelinable private-key encryption algorithms. The data encryption standard (DES) algorithm, which lends itself readily to pipelining, is utilised to exemplify this novel key scheduling method and the broader applicability of the method to other encryption algorithms is illustrated. The DES design is implemented on Xilinx Virtex FPGA technology. Utilising the novel method, a 16-stage pipelined DES design is achieved, which can run at an encryption rate of 3.87 Gbit/s. This result is among the fastest hardware implementations and is a factor 28 times faster than software implementations.

1 Introduction

The rapid developments in communication systems in the last century and the growth of the Internet in the last decade have meant that the need for effective security and reliability of data communication, processing and storage is greater than ever. Encryption algorithms have been developed as a mechanism for providing this security and there is a need to perform these algorithms on data in real time.

Theoretical studies carried out on encryption algorithms and protocols one year will appear in actual products and standards the following year. However, further research may reveal unsuspected weaknesses in the previous year's studies and the algorithms and standards may need to be updated or altered. Also, secure communications systems often require the capacity to encrypt messages with several different algorithms in addition to the need to change keys regularly [1]. Hence, the concept of reusable, parameterisable cryptographic cores is ideally suited to meeting these requirements.

This paper presents a reusable, parameterisable key scheduling implementation or core, which can be utilised in any pipelined private-key encryption algorithm. The design supports the use of different keys every clock cycle, thus improving overall security since users are not restricted to using the same key during any one session of data transfer. It also supports both electronic codebook (ECB) and output feedback modes of operation. This design is particularly suited to substitution–permutation and fiestel-structured algorithms. A substitution–permutation (SP) algorithm is one composed of a number of stages each involving substitutions and permutations. A fiestel cipher is an iterated algorithm that maps a $2t$-bit plaintext (L_0,R_0), for t-bit blocks L_0 and R_0, to a ciphertext (R_r,L_r), through an r-round process where $r \geq 1$ [2]. This is illustrated in Fig. 1.

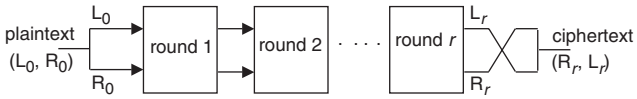

Fig. 1 *Fiestel structure*

The data encryption standard (DES) is used as a design example. This algorithm is the best known and most widely used encryption algorithm. IBM developed DES in the mid 1970 s as a modification of an earlier system known as Lucifer. In 1976, DES was adopted by the National Bureau of Standards (NBS), now the National Institute of Standards & Technology (NIST), as a federal standard and authorised it for use on all classified government communications. It has been the standard algorithm for banking and other applications since 1977. Although it has been replaced by the advanced encryption standard (AES) algorithm, DES will still remain in the public domain for a number of years because of legacy requirements. It provides a basis for comparison for new algorithms and is also used in IPSec protocols, ATM cell encryption, the secure socket layer (SSL) protocol and in TripleDES, adopted to improve DES in the X9.17 and ISO 8732 standards [3, 4].

The DES algorithm design example presented in the paper is implemented on Xilinx Virtex FPGA technology. The advantages of implementing cryptographic algorithms on FPGAs include not only algorithm agility, where the same FPGA can be reprogrammed at run time to support different algorithms, but also scalable security through different versions of the same algorithm (DES and TripleDES) and alterable architecture parameters, where features such as variable s-boxes or different modes of operation can be realised [5]. Although software implementations can easily convert between encryption algorithms, they suffer from poor data rates. The fastest DES software implementation achieves a throughput of 127 Mbit/s on a 300 MHz Alpha 8400 processor [6]. ASICs offer the highest performance implementations but lack flexibility. The

IEE Proceedings online no. 20030574

doi:10.1049/ip-cds:20030574

Paper first received 8th December 2000 and in revised form 7th April 2003

The authors are with the DSiP™ Laboratories, School of Electrical and Electronic Engineering, Queen's University of Belfast, Stranmillis Road, Belfast, Northern Ireland BT9 5AH, UK

Sandia National Laboratories (SNL) DES ASIC implementation is the fastest known DES implementation, capable of running at 9.28 Gbit/s [7].

The fastest FPGA DES implementation [8], which runs a data rate of 10.7 Gbit/s, utilises Jbits on a Virtex XCV150-6 device. Jbits provides a Java-based Application Programming Interface (API) for the run-time creation and modification of the configuration bitstream. This design is not a single-chip implementation of the full DES algorithm since the key schedule is computed in software. Also, it can only accommodate one key per data transfer session.

2 DES algorithm description

DES is a private key (symmetric) algorithm. An outline of DES is shown in Fig. 2. It is a block cipher operating on 64-bit blocks of plaintext utilising a 64-bit key. Every eigth bit of the 64-bit key is used for parity checking and otherwise ignored. After an initial permutation, the 64-bit input is split into a right (R_o) and left half (L_o), each 32 bits in length. DES has 16 iterations or rounds. In each round a function f is performed in which the data is combined with a 48-bit permutation of the key. After the 16th iteration, the right (R_{16}) and left (L_{16}) halves are concatenated and a final permutation, which is the inverse of the initial permutation, completes the algorithm.

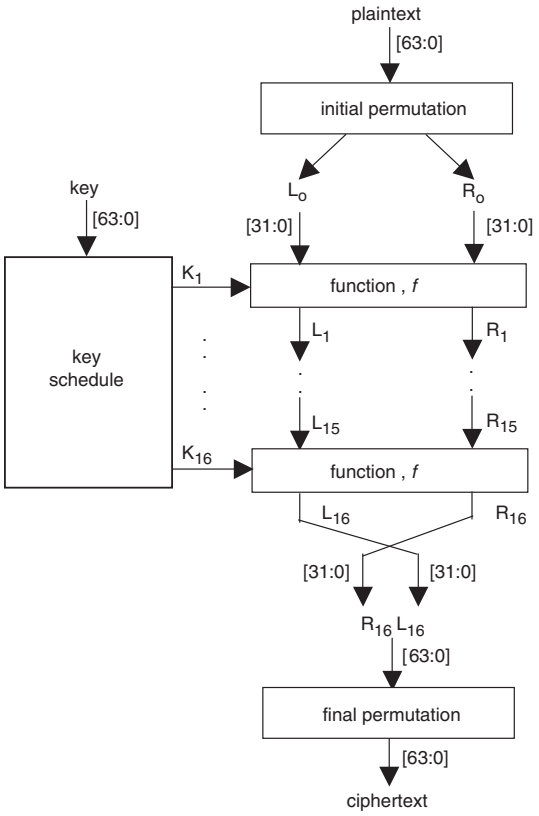

Fig. 2 *Outline of DES encryption algorithm*

2.1 Function f of DES algorithm

The function f of the DES algorithm is made up of four operations. Firstly, the 32-bit right half of the plaintext R_o is expanded to 48-bits and then XORed with a 48-bit sub-key K_1. The result is fed into eight substitution boxes (s-boxes), which transform the 48-bit input to a 32-bit output. Finally, a straight permutation (P-permutation) is performed, the output of which is XORed with the initial left half, L_o to obtain the new right half R_1. The original right half R_o becomes the new left half L_1. This is outlined in Fig. 3.

2.2 Key scheduling

There are two methods of implementing the DES key procedure. The initial step in the first method is to remove the parity check bits in the 64-bit key. Every eigth bit is used for parity checking, leaving 56-bits. A different 48-bit sub-key is now generated for each of the 16 rounds of DES. The sub-keys are determined by first splitting the 56-bits into two 28-bit lengths of data. Then both halves are shifted left by either one or two bits depending on the round number. The second method simply involves the implementation of the resulting 48-bit permutations.

2.3 Pipelining the DES algorithm

The iterative nature of the DES algorithm makes it ideally suited to pipelining. The DES algorithm implementation presented in this paper is based on the ECB mode. Although the ECB mode is less secure than other modes of operation, it is commonly used and its operation can be pipelined [9].

The fully pipelined DES implementation will also operate in counter mode. Counter mode is a simplification of output feedback (OFB) mode and involves updating the input plaintext block as a counter, $I_{j+1} = I_j + 1$, rather than using feedback. Hence, the ciphertext block i is not required in order to encrypt plaintext block $i+1$ [2]. Counter mode provides more security than ECB mode and operation of either mode involves trading security for high throughput.

In order to pipeline the algorithm, the function f block must be instantiated 16 times. Registers are then placed at the left and right outputs of each function f block to allow the data to be sequenced.

3 Key scheduling procedure

The conventional method of implementing the DES key schedule involves carrying out a shift operation during each stage of the 16-stage pipeline. This is illustrated in the block diagram in Fig. 4.

The 64-bit input key is passed through an initial key permutation which removes the parity check bits. The resulting 55-bits enter the first function f block. It is split into two 28-bit words and a cyclic shift left operation is carried out on each half. The two halves are then concatenated and operated on by a final key permutation to obtain the sub-key required by the first function f block. The output from the shift operation provides the input to the second function f block where once again a cyclic shift left operation is carried out and key permutation performed. This process continues for each stage of the 16-stage pipeline. In rounds 1, 2, 9 and 16 of the DES algorithm the halves are shifted one position to the left and for all other rounds two positions to the left.

3.1 Novel implementation of DES algorithm key schedule

In the implementation of the DES algorithm key schedule [10] employed here, the sub-keys are pre-computed and hence, for a 16-stage pipelined DES design, it is necessary to control the time at which the sub-keys are available to each function f block. This is accomplished by the addition of a skew that delays the individual sub-keys by the required amount. An outline of this key scheduling method is provided in Fig. 5.

3.2 Sub-key creation core

The sub-key creation block constructs the key permutations required for each iteration of the DES algorithm. Each sub-key permutation has the same 64-bit key input (the initial

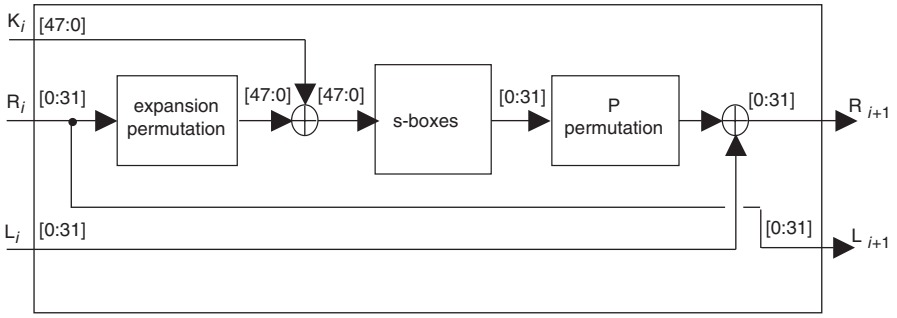

Fig. 3 *Function f of the DES algorithm*

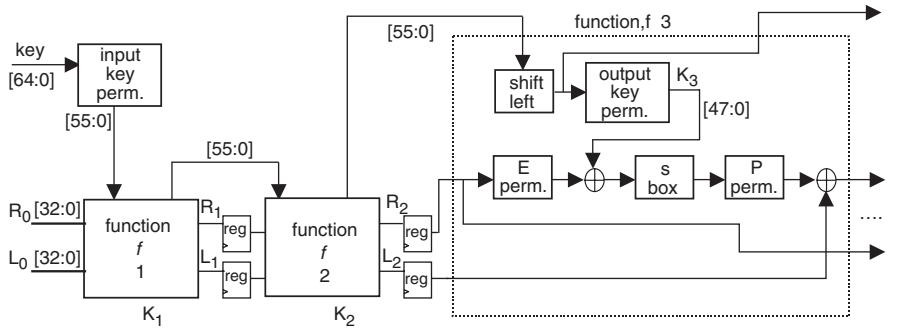

Fig. 4 *Outline of conventional key scheduling method*

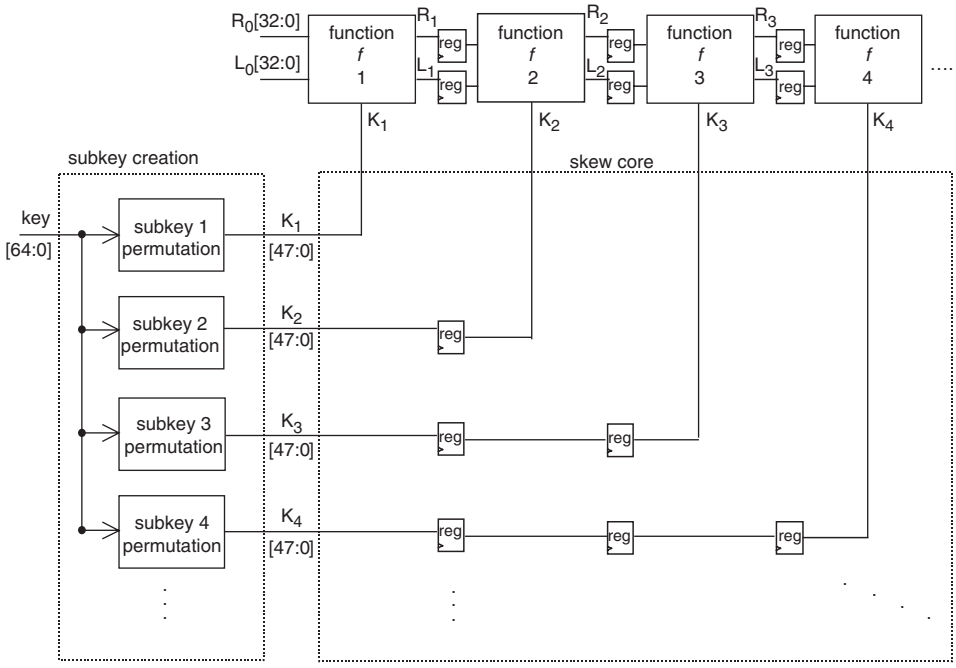

Fig. 5 *Outline of novel key scheduling method*

key input) but a different 48-bit key output and simply involves a rearrangement of the input data.

For example, the permutation required to create the first subkey during encryption is outlined in Fig. 6. From this table it can be seen that bit 2 of the 64-bit input key becomes bit 9 of the 48-bit permutated key, bit 62 becomes bit 46, bit 63 becomes bit 40, and so on.

The key scheduling design described also supports decryption. When decrypting data the keys for each round or iteration are used in reverse order. Therefore, sub-key permutation 1 will be used to create the first sub-key K_1 in the encryption process and the final sub-key K_{16} in the decryption process.

3.3 Skew core

For the 16-stage pipelined DES design it is necessary to control the time at which the sub-keys are available to each function *f* block. This is accomplished by the addition of a skew that delays the individual sub-keys by the required amount.

The skew consists of a 'dffarray' sub-component, which generates a sequence of latches as required. The code for this sub-component is outlined in Fig. 7. The 'Depth' parameter is generic and indicates the desired length of the array. If Depth = 0, the process, S1 is used to create one latch. If Depth > 1, for example if Depth = 2 (effectively the Depth parameter begins at 0 and counts up to 2), the S1

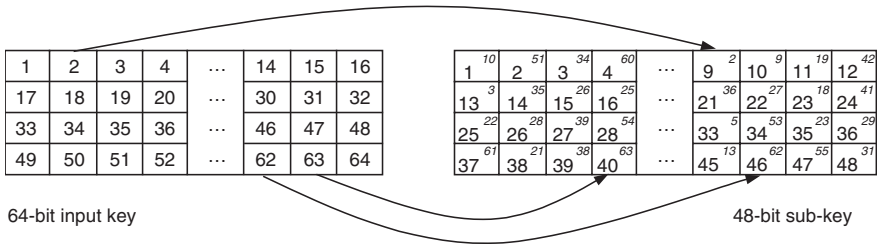

Fig. 6 *Round 1 encryption key permutation*

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.TYPES.all;

--KEY is of typestd_logic_vector(0 to 47)
entity dffarray is
     generic ( Depth    : POSITIVE);
     port      ( Keyin    : in KEY;
                 clk      : in std_logic;
                 reset    : in std_logic;
                 D_Key    : out KEY);
end dffarray;

architecture synth of dffarray is

component dff1
     port      ( D        : in std_logic_vector( 0 to 47);
                 clk      : in std_logic;
                 reset    : in std_logic;
                 Q        : out std_logic_vector( 0 to 47));
end component;

type MIDKEY is array (0 to Depth) of KEY;
```

```
signal A   : MIDKEY;
begin
     G1: for i in 0 to depthgenerate
              begin
              S1: if i = 0 generate
                       FF1: dff1
                       port map (   D      => Keyin,
                                    clk    => clk,
                                    reset  =>reset,
                                    Q      => A(i));
              end generate S1;

              S2: if i /= 0 generate
                       FF2 : dff1
                       port map (   D      => A(i-1),
                                    clk    => clk,
                                    reset  => reset,
                                    Q      => A(i));
              end generate S2;

     end generate G1;

     D_ Key <= A(Depth);

end synth;
```

Fig. 7 *Code for 'dffarray' component*

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.TYPES.all;

--SKEWBUS is array (0 to 15) of  std_log_vect(0 to 47)
entity skew is
     port    (   SkewKeyin   : in SKEWBUS;
                 clk         : in std_logic;
                 reset       : in std_logic;
                 SkewD_Key   : out SKEWBUS);

end skew;
architecture synthof skewis

component dffarray

     generic  ( Depth        : POSITIVE := 15);
     port       (keyin       : in KEY;
                 clk         : in std_logic;
                 reset       : in std_logic;
                 D_Key       : out KEY);

end component;
```

```
begin

     G2:  for  i  in  0  to  14  generate

          S1: dffarray
          generic map (Depth =>i)
          port map     ( Keyin    => SkewKeyin(i+1),
                         clk      => clk,
                         reset    => reset,
                         D_Key    => SkewD_Key(i+1));

     end generate G2;

     SkewD_Key(0) <= SkewKeyin(0);

end synth;
```

Fig. 8 *Code for 'skew' component*

process is used to create the first latch in the array and the S2 process creates the remaining two latches. The 'skew' component generates an array of latches of varying lengths. It uses the 'dffarray' sub-component to produce the correct number of latches required at each round of the DES algorithm. The code for the 'skew' component is shown in Fig. 8. Since the DES algorithm consists of 16 rounds, the skew core is set to loop 15 times (for $i = 0$ to 14) since a latch is not required to delay the first sub-key. The value of i determines the 'Depth' of the array to be generated by the 'dffarray' component. When $i = 0$, one latch is created. If $i = 2$, for example, three latches are created. Hence an array of latches of varying lengths is generated as illustrated in Fig. 9.

4 Performance results

The conventional method of implementing the DES algorithm key schedule utilises logic cyclic shift operations at each stage of the 16-stage pipeline to create the sub-keys. The method of implementing the key schedule presented in this paper simply utilises permutations to create the

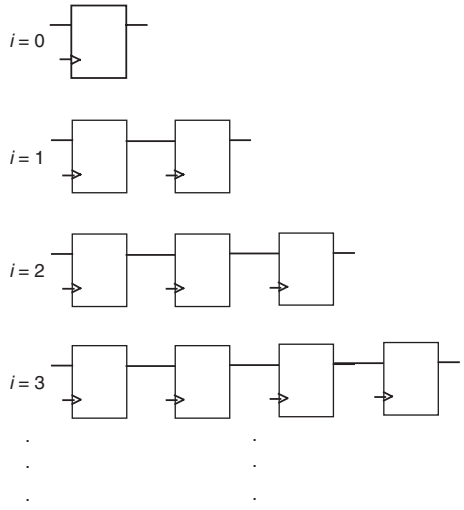

Fig. 9 *Array of latches of varying lengths generated by skew core*

sub-keys from the input key. The sub-keys are delayed by the required amount using the necessary array of latches. Since no actual logic is used, a faster implementation is achieved.

The pipelined DES design is a large design as it contains 16 instantiations of the function *f* component, hence the targeted FPGA device is the largest in the Virtex family, the XCV1000. A study by Haskins [11] indicates that using ROM blocks provides the most efficient implementation for the s-boxes of the DES algorithm. Thus two LUTs within a slice on the Virtex FPGA device can be combined to create a 32 × one-bit synchronous RAM [12], initialised and used to implement the s-boxes. Eight 32 × one-bit RAM blocks are required for each s-box. The design is described in VHDL, simulated using Modelsim XE and synthesised using Synplify Pro and Xilinx Foundation Series 1.2i software on a XCV1000-4bg560 device. Data blocks can be accepted every clock cycle and after an initial delay of 16 clock cycles the respective encrypted/decrypted data blocks appear on consecutive clock cycles. The design also supports a key change at full speed, i.e. a different key can be used with every block of plaintext.

The DES design incorporating the novel key scheduling method utilises 6446 CLB slices which is 52% of the total number of CLB slices available on this device. Of IOBs, 188 out of 404 (46%) are used. The critical path of the design lies in the function *f* component and the design uses a system clock of 59.5 MHz. The data rate achieved is 3.8 Gbit/s [13].

It is possible to further enhance these performance figures by optimisation of the algorithm specific to the requirements of the FPGA device on which the design is to be implemented. However, this would result in the design being less easy to migrate to other devices and technologies.

5 Discussion

A new key scheduling method for pipelined implementations of symmetric-key encryption algorithms has been presented. It is a simple, easy-to-follow method, which involves the pre-computation and delayed presentation of the algorithm sub-keys. The algorithm used to demonstrate the key scheduling method in this paper is the DES algorithm. DES has been a Federal Information Processing Standard (FIPS) since 1977 and although it has recently been replaced, it will be required to retain compatibility with old products and will remain as a benchmark for all new algorithms in the future. In algorithms similar to DES, where the sub-keys may alternatively be obtained by carrying out a permutation and substitution method, the core will support the use of different keys every clock cycle. If this is not possible the core may still be utilised provided one key is used per session.

A fully pipelined DES algorithm implementation is presented in this paper, incorporating the new key scheduling method. This key scheduling method is generic and parameterisable and hence is migratable to any algorithm which can be pipelined in its implementation. For the purposes of this paper, the design presented is demonstrated by implementation in FPGA technology. It can also easily be implemented in other silicon technologies such as enhanced Virtex devices and ASIC technology, from which higher throughputs can be obtained.

Table 1 summarises some recent DES hardware and software implementations.

The DES design described in this paper achieves the very high encryption/decryption rate of 3.8 Gbit/s, which is approximately 27 times faster than equivalent software implementations. It also compares favourably with existing hardware implementations, including commercial ASIC designs such as Sandia Laboratories' 9.28 Gbit/s encryptor/decryptor core. The design reported by Patterson operates at 10.7 Gbit/s. However, in this design the key schedule is computed in software and can only support one key per data transfer session. The performance of the presented DES design is among the fastest hardware implementations currently available and is one of the fastest single-chip FPGA designs.

6 Acknowledgments

The research undertaken had been funded by Amphion Semiconductor Ltd. and by a University Research Studentship, which incorporates funding by the European Social Fund.

Table 1 Specifications for recent DES hardware and software implementations

Manufacturer	Device used	CLB slices	System clock (MHz)	Data rate (Mbit/s)
Wong *et al.* [14]	XC4020E	438	10	26.7
Biham [6] (software)	Alpha 8400	—	300	127
Kaps and Paar [5]	XC4028EX	741	25.18	402.7
Free-DES [15]	XCV400	5263	47.7	3052
McLoone, McCanny	XCV1000	6446	59.5	3808
Sandia Laboratories [7]	ASIC	—	—	9280
Patterson (Jbits) [8]	XCV150	1584	168	10752

7 References

1 Leonard, J., and Mangione-Smith, W.H.: 'A case study of partially evaluated hardware circuits: key-specific DES'. Proc. field programmable logic and applications, FPL '97, London, UK, Sept. 1997 (Springer-Verlag, 1997), pp. 151–160

2 Menezes, A., Oorschot, P., and Vanstone, S.: 'Handbook of applied cryptography' (CRC Press, 1997)

3 ANSI X9.17 (Revised): 'American National Standard for financial institution key management (wholesale)' American Bankers Association, 1986

4 ISO DIS 8732: 'Banking – key management (wholesale)' Association for Payment Clearing Services, London, December 1987

5 Kaps, J.P., and Paar, C.: 'Fast DES implementations for FPGAs and its application to a universal key-search machine'. Proc. 5th Annual Workshop on Selected areas in cryptography - SAC '98, Ontario, Canada, August 1998 (Springer-Verlag, 1998), pp. 234–247

6 Biham, E.: 'A fast new DES implementation in software'. Proc. 4th Int. Workshop on Fast software Encryption, FSE '97, Haifa, Israel, Jan. 1997 (Springer-Verlag, 1997), pp. 260–271

7 Wilcox, D.C., Pierson, L.G., Robertson, P.J., Witzke, E.L., and Gass, K.: 'Sandia National Laboratories : A DES ASIC suitable for network encryption at 10 Gps and beyond'. Proc. First Int. Workshop on Cryptographic hardware and embedded systems, CHES '99, Worcester, MA, USA, August 1999 (Springer-Verlag, 1999), pp. 37–48

8 Patterson, C. (Xilinx Inc.): 'High performance DES encryption in virtex FPGAs using Jbits'. Proc. IEEE Symp. on Field-programmable custom computing machines, FCCM '00, Napa Valley, CA, USA, April 2000 (IEEE Comput. Soc., CA, USA, 2000), pp. 113–121

9 Van Der Lubbe, J.C.A.: 'Basic methods of cryptography' (Cambridge University Press, 1998)

10 McLoone, M., and McCanny, J.V.: 'Data encryption apparatus'. UK Patent Application 0023409.6, October 2000

11 Haskins, G.M.: 'Securing asynchronous transfer mode networks'. Masters Thesis, Worcester Polytechnic Institute, Worcester, MA, USA, May 1997

12 Xilinx Virtex™ 2.5 V FPGA data sheet: URL: http://www.xilinx.com January 2000

13 McLoone, M., and McCanny, A.: 'A high performance implementation of DES'. Proc. IEEE Workshop on Signal processing systems design and implementation, SiPS2000, Lafayette, LA, USA, October 2000, pp. 374–383

14 Wong, K., Wark, M., and Dawson, E.: 'A single-chip FPGA implementation of the data encryption standard (DES) algorithm'. Proc. IEEE Globecom Communications Conf., Sydney, Australia, Nov. 1998, pp. 827–832

15 Free-DES Core (2000), URL: http://www.free-ip.com/DES/, March 2000