Efficient hardware implementation of the DES

Frank Hoornaert, Jo Goubert, and Yvo Desmedt

Katholieke Universiteit Leuven,
Laboratorium ESAT,
Kardinaal Mercierlaan, 94,
B-3030 Heverlee, Belgium.

Abstract. Several improvements to realize implementations for DES are discussed. One proves that the initial permutation and the inverse initial permutation can be located at the input, respectively the output of each mode in DES. A realistic design for an exhaustive key search machine is presented.

1. Introduction

In [14] the reader find that hardware which is at the same time cheap and fast does not exist [1]. In this paper we propose mainly one efficient chip design for the DES. Nevertheless, depending on the need of the user, different versions are possible. *The proposed version is designed for general purpose.* In section 10 however *a version for exhaustic search of the keys* is illustrated. Important is that all versions can use the same techniques.

The reader not familiar with the DES finds the NBS description of the DES in the literature [9].

1.1. Problems and used techniques

By designing the chip one has to solve several technical problems as:

1. the complexity of the routing
2. the minimization of the needed chip area
3. the maximization of the desired speed
4. the limitation of pin connections.

The main problem in a DES chip is the routing. This can be illustrated by looking at figure 1 which shows a straightforward realization of the DES-algorithm [4]. There we see that the width of the interconnections (e.g. 64, 48 ....) will result in an enormous lost of area and speed.

We can distinct 5 important techniques (*see list*) to solve previous problems.

1. a serial/parallel realization with shiftregisters of the permutations and of memory and internal transport in order to reduce the number of interconnections on chip (sections 3 and 4)
2. the use of equivalent representations of the DES
   in order to optimize speed
   (section 5)
3. pipelining of the different datablocks
   in order to maximize the activity on the chip
   (section 6)
4. rearrangement of the functional elements
   in order to shorten the lenght of interconnections
   (section 7)
5. a modular controller with microprogramming
   in order to ensure the flexibility of the design
   (section 8)

Remark that as consequence of the needed coherence in the hardware solution, a lot of simplifications proposed in previous papers (e.g. [3]) couldn't be used.

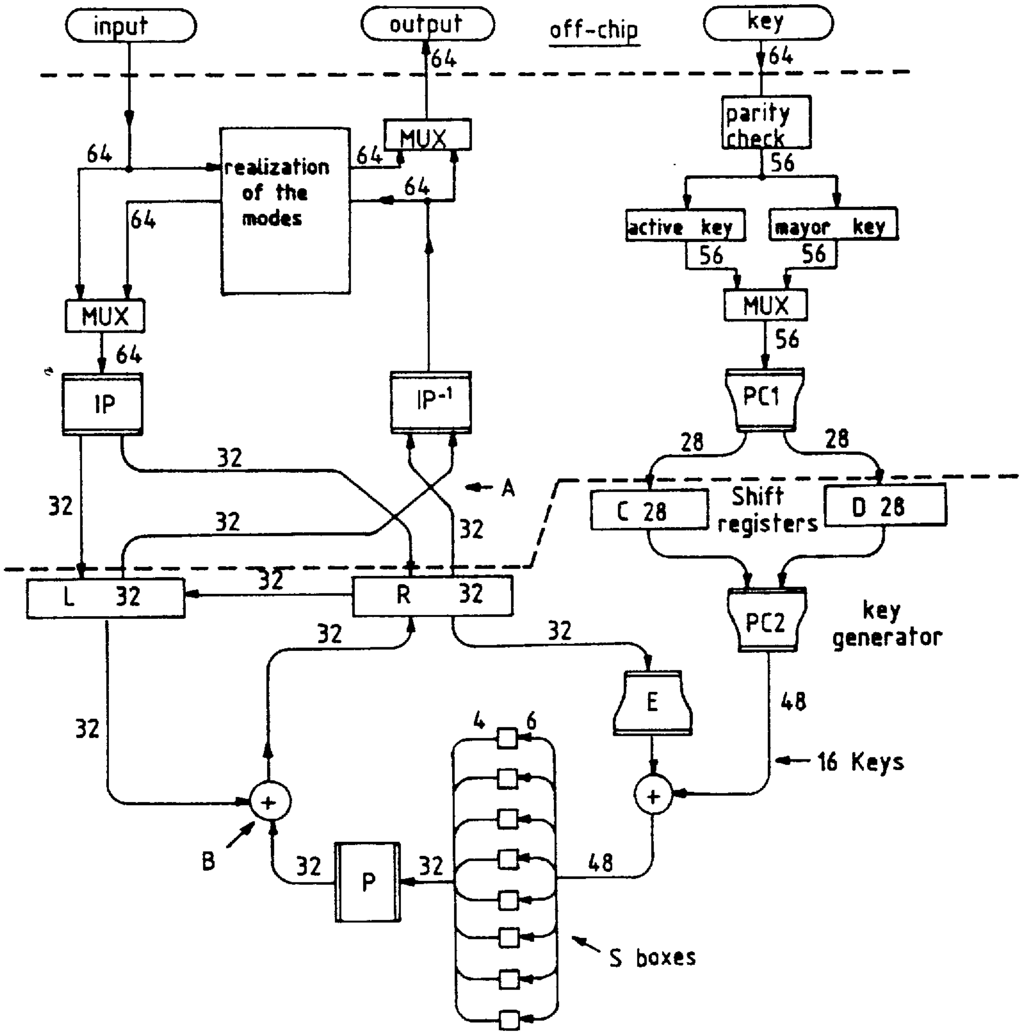
In the next sections the routing problem for the transport part of the chip is solved by using a serial-parallel structure. At the same time, we reduce the area needed for IP, IP^{-1} and PC_1 to about 1/20 of the area compared with a full parallel realization. *Important is that all this solutions do not slow down the datarate.*

The routing problem for the part which carries out the 16 iterations (incl. the subkey generation), is solved by rearranging the different elements. So we shorten the length of the interconnections. This will be explained in extension in section 7 .

1.2. survey of the chip

We divide the chip in 2 important parts. The first called ***transport part,*** supports the datatransport with the environment and also the internal transport between different memories (incl. the permutations IP, $IP^{-1}$and PC_1). The second called ***iteration hardware,*** calculates the 16 iterations of the DES-algorithm without IP and IP^{-1} *(from now on we call these 16 iterations DES^*)* incl. the generation of the subkeys.

We will first explain the basic idea used to simplify the realization of the modes. Later on we will explain the other techniques used.



the upper part of the figure is the transport part
the lowest part shows the iteration hardware
On the first sight, the DES algorithm suffers from an enormous routingproblem. Consequently the main goal of our design was to solve this problem by serialization and rearrangement without slowing down datarate.

Figure 1: the routing problem

2. Equivalent representation for the modes: An introduction

In the Fibs publication [8], four modes of operation for the DES are defined. These modes specify different ways to encrypt and decrypt data. We show that you can reorder these modes so that IP and IP^{-1} appear at an other location in the algorithm. We'll explain first the general idea and the motivation of this transformation. Next we'll apply this idea in the case of the 8 byte modes and the 1 byte modes.

Mostly the execution of permutations in hard- or software slows down the performances. The idea is to put IP respectivily IP^{-1} as close as possible to the input respectivily the output of the mode. So you don't have to carry out IP and IP^{-1} on the data in the feedbackloop. To move the permutations, we'll use the following properties. A permutation followed by a selection, can always be transformed into a selection* followed by a permutation* (figure 2). A similar remark is true for an injection followed up by a permutation. The elementary transformations as explained at crypto 83 will also be used [3].

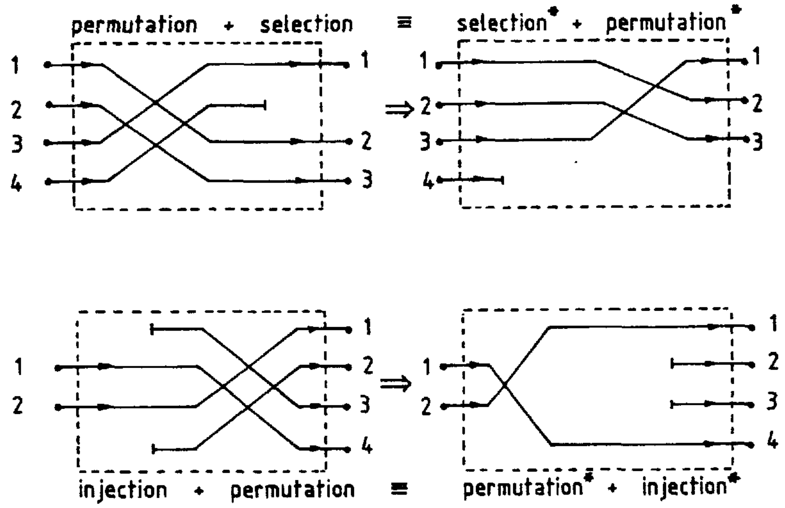
2.1. 8 BYTES MODES

In the case of the ECB mode it's trivial that IP is located at the input and IP^{-1} at the output.
In the CFB mode we can write on location A (figure 3) $IP \cdot IP^{-1}$ and propagate them over the exors, using the elementary transformations of Crypto-83 [3] page 182, we obtain then the desired result.
A similar result is similar to find for the CBC and OFB modes.

2.2. 1 BYTE MODES

In CFB mode, a new input for the DES is formed out of the previous input for the DES and the actual output ciphertext. The 56 most significant bits of the new input for the DES come from the old input shifted 8 times to the left. This can be represented (figure 4) as a selection of 56 bits out of 64 bits (S_1) together with an injection of 56 bits into 64 bits (I_1). The 8 least significant bits of the new input for the DES are the 8 bits output ciphertext. This can be represented by an injection I_2 of those 8 bits into 64 bits. To form the output ciphertext the 8 most significant bits of the DES output are selected by selection S_2, and exored with the plaintext. The content of these selections and injections is showed in table 1 and 2. The selections and injections are similar represented as the permutations of the DES in the NBS norm [9]. Now by applying some transformations one is able to obtain the desired result.

Let us therefore put $IP \cdot IP^{-1}$ at location B in figure 4. Using the property of figure 2 we obtain figure 5, where the effect of $Q_a(S_a(64bits))$ is the same as $S_2(IP^{-1}(64bits))$. In figure 5 we put $Q_a^{-1} \cdot Q_a$ at location C. By moving Q_a, IP and IP^{-1} (using Crypto 83) we obtain figure 6. Using the first property of figure 4, this figure is transformed into figure 7, where $Q_b(S_b(64bits))$ is the same as $S_1(IP(64bits))$ and similar for $I_c(Q_c)$ and



permutation + selection ≡ selection* + permutation*
1
2
3
4
⇒
injection + permutation ≡ permutation* + injection*

Figure 2: properties used in the transformation of the modes

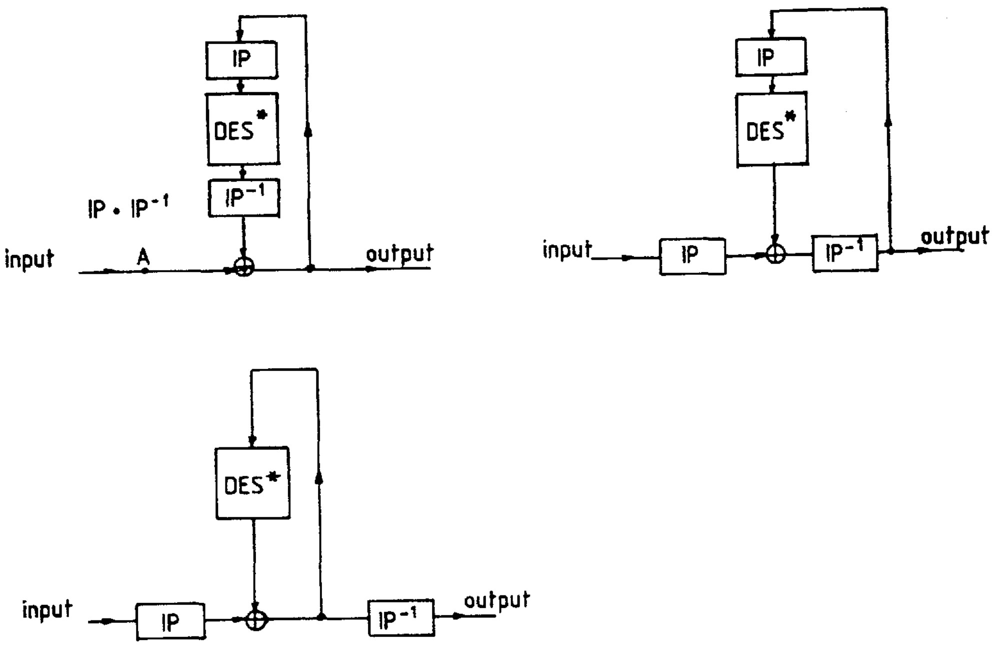
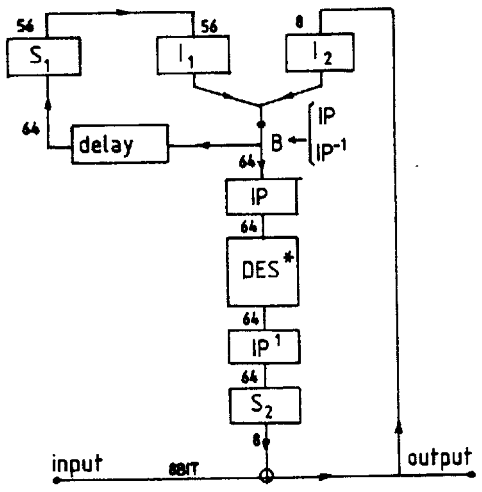


Figure 3: *CFB* 8 byte mode



S = a selection
I = an injection

Figure 4: representation of the 1 byte *CFB* mode with selections and injections

$S_1 =$

9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

$S_2 =$

1	2	3	4	5	6	7	8

Table 1: the selections S_1 ($64 \rightarrow 56$) and S_2 ($64 \rightarrow 8$)

$I_1 =$

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
x	x	x	x	x	x	x	x

$I_2 =$

x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
1	2	3	4	5	6	7	8

" x " *means that* " *nothing* " *is injected at that place*

Table 2: the injections I_1 ($56 \rightarrow 64$)and I_2 ($8 \rightarrow 64$)

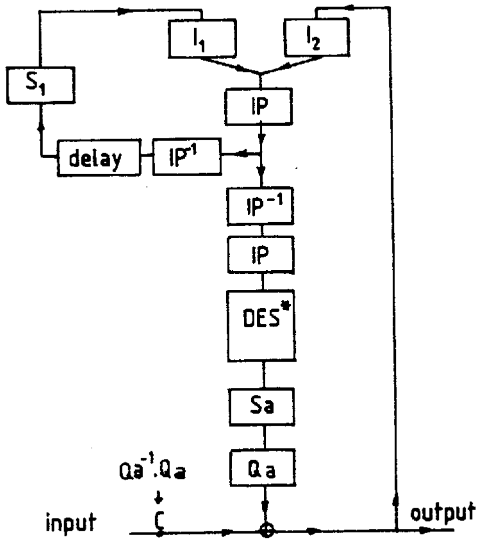


Figure 5: intermediate result of the CFB 1 byte mode derived from figure 4

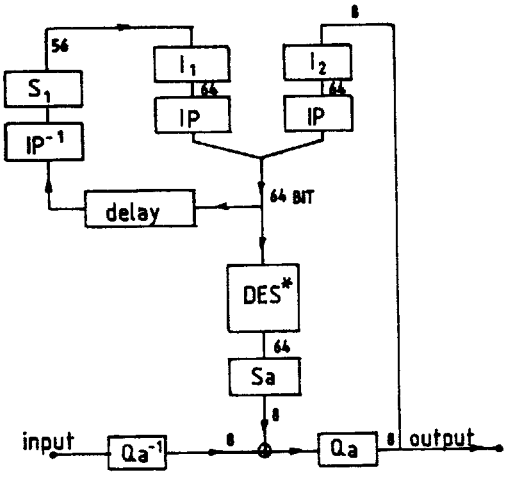


Figure 6: intermediate result of the CFB 1 byte mode derived from figure 5

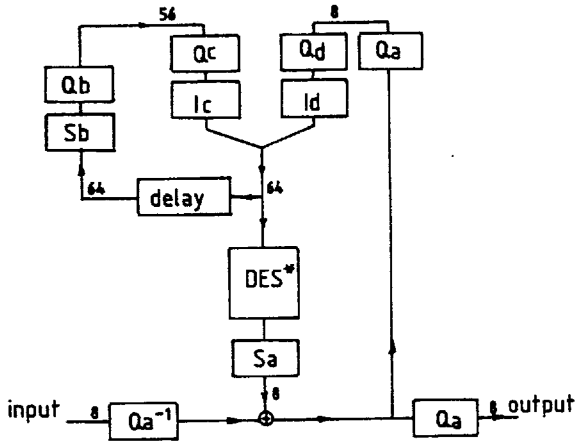


Figure 7: intermediate result of the CFB 1 byte mode derived from figure 6

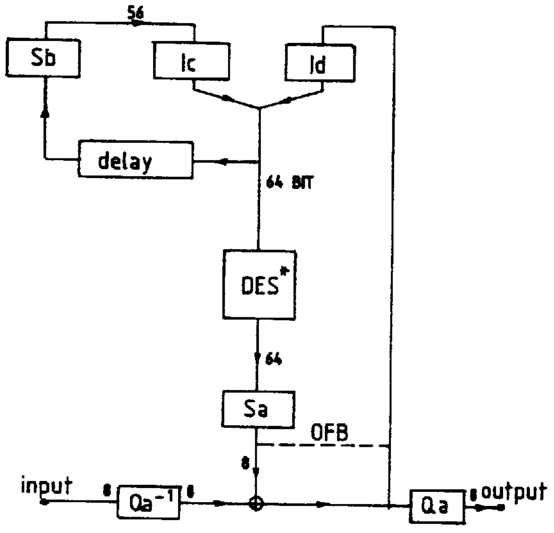


Figure 8: final result of the CFB 1 byte mode derived from figure 7 without permutations in the feedbackloop

$$S_a = \left[\begin{array}{cccccccc} 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 \end{array} \right]$$

$$S_b = \left[\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{array} \right]$$

Table 3: the selections S_a (64 →8) and S_b (64 →56)

$$Q_a = \left[\begin{array}{cccccccc} 5 & 1 & 6 & 2 & 7 & 3 & 8 & 4 \end{array} \right] \quad Q_a^{-1} = \left[\begin{array}{cccccccc} 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 \end{array} \right]$$

Table 4: the permutations Q_a (8 →8) and Q_a^{-1} (8 →8)

$$Q_b = \left[\begin{array}{cccccccc} 35 & 7 & 42 & 14 & 49 & 21 & 56 & 28 \\ 34 & 6 & 41 & 13 & 48 & 20 & 55 & 27 \\ 33 & 5 & 40 & 12 & 47 & 19 & 54 & 26 \\ 32 & 4 & 39 & 11 & 46 & 18 & 53 & 25 \\ 31 & 3 & 38 & 10 & 45 & 17 & 52 & 24 \\ 30 & 2 & 37 & 9 & 44 & 16 & 51 & 23 \\ 29 & 1 & 36 & 8 & 43 & 15 & 50 & 22 \end{array} \right]$$

$$Q_c = \left[\begin{array}{ccccccc} 50 & 42 & 34 & 26 & 18 & 10 & 2 \\ 52 & 44 & 36 & 28 & 20 & 12 & 4 \\ 54 & 46 & 38 & 30 & 22 & 14 & 6 \\ 56 & 48 & 40 & 32 & 24 & 16 & 8 \\ 49 & 41 & 33 & 25 & 17 & 9 & 1 \\ 51 & 43 & 35 & 27 & 19 & 11 & 3 \\ 53 & 45 & 37 & 29 & 21 & 13 & 5 \\ 55 & 47 & 39 & 31 & 23 & 15 & 7 \end{array} \right] \qquad Q_d = \left[\begin{array}{cccccccc} 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 \end{array} \right]$$

Table 5: the permutations Q_b (56 →56), Q_c (56 →56) and Q_d (8 →8)

$$I_c = \left[\begin{array}{cccccccc} x & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ x & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ x & 15 & 16 & 17 & 18 & 19 & 20 & 21 \\ x & 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ x & 29 & 30 & 31 & 32 & 33 & 34 & 35 \\ x & 36 & 37 & 38 & 39 & 40 & 41 & 42 \\ x & 43 & 44 & 45 & 46 & 47 & 48 & 49 \\ x & 50 & 51 & 52 & 53 & 54 & 55 & 56 \end{array} \right] \qquad I_d = \left[\begin{array}{cccccccc} 1 & x & x & x & x & x & x & x \\ 2 & x & x & x & x & x & x & x \\ 3 & x & x & x & x & x & x & x \\ 4 & x & x & x & x & x & x & x \\ 5 & x & x & x & x & x & x & x \\ 6 & x & x & x & x & x & x & x \\ 7 & x & x & x & x & x & x & x \\ 8 & x & x & x & x & x & x & x \end{array} \right]$$

Table 6: the injections IC ($56 \rightarrow 64$) and ID ($8 \rightarrow 64$)

$I_d(Q_d)$. By calculating all these tables we can proof that $Q_c(Q_b) = I$ and $Q_d(Q_a) = I$, with I the identity permutation.

In figure 8 we find the final result. We see that IP is no longer used in the feedback loop, and that two $8 \rightarrow 8$ permutations have appeared at the input and output of the data. The content of the new permutations, injections and selections is showed in table 3, 4, 5 and 6. Thy must be read as in the NBS representation [9].

A similar result for the OFB mode can be obtained using similar techniques as for the CFB mode.

Remark that this equivalent representation of the modes can also speed up the software, as explained in [14].

3. a fast serial/parallel realization for the permutations

The transport part of the chip will communicate with the environment using 8 bit buses. The main reason for this is the limitation on the amount of connection pins of the chip. However these buses allow us also to realize the permutations IP, IP^{-1} and PC_1 in an elegant way.

The idea is that by shifting data from a bus into a shiftregister, you carry out a permutation in a hidden way. If you put an $8 \rightarrow 8$ permutation between the bus and the shiftregister (see figure 9), you can realize a whole set of permutations. We found that IP, IP^{-1} and PC_1 can be realized using this set of permutations.

3.1. permuting with shiftregisters: an introduction

When you send a block of 64 bit over an 8 bit bus, you'll send it byte after byte. If we place at the end of the bus 8 shiftregisters of each 8 bit, we can enter sequentially these 8 bytes (8×8) (see figure 10). Normally, we'll call the first shiftregister the first byte of our memory, the second shiftregister the second byte etc. (indicated with italic numbers e.g. *1 2*).

What we see is that the numbers of the memory locations (*1, 2, 3, ... , 64*) and the numbers of the data bits (**1, 2, 3, ... , 64**) don't agree. Conclusion, we have carried out a permutation. This permutation is represented by the vector at the foot of figure 10 (for the interpretation of the vector notation, we refer to the Fibs publication of the data encryption standard [9]). From now on we'll call this permutation SR.

In the same way, we can show (see figure 11) that when we read out information of 8 shiftregisters into a bus, we carry out another permutation. You can easily check that we get in that way the permutation SR^{-1}.

Now that we know SR and SR^{-1}, we'll search for which 8→8 permutations we need to realize IP, IP^{-1} and PC_1.

3.2. realization of IP and IP^{-1}

If we want to have IP after we have shifted in, we have to do a 64→64 permutation before SR, satisfying the following equation:

$$IP = SR \text{ after } IP^{*} \quad \text{or} \quad IP^{*} = SR^{-1} \text{ after } IP$$

The result is shown in table 7. If we write e.g. bit 18 as the 2' bit of byte 3 (or 2_3) we see that IP^{*} can be realized by carrying out on each byte the same 8→8 permutation IP^{**} (see table 8).

So the realization of IP has become very simple as shown in figure 12. This realization is much smaller than a hardware connection path realizing the permutation. At the same time, the execution of IP doesn't consume extra time because it is carried out during the input from the environment.

A similar method is used to realize IP^{-1}.

$$IP^{-1} = IP^{-1*} \text{ after } SR^{-1} \quad \text{or} \quad IP^{-1*} = IP^{-1*} \text{ after } SR$$

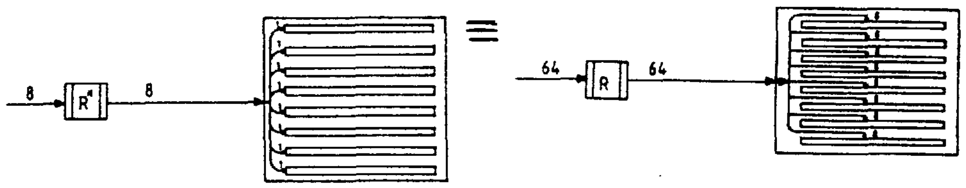
Note that it is possible to use the same shiftregisters to carry out IP and IP^{-1}. This can be done simultaneously by reading 1 byte out every time you read 1 byte in (see figure 12).

3.3. realization of PC_1

For PC_1 we have a more complicated solution because of the irregularity in PC_1 [3]. This can be shown by rewriting the notation of PC_1 (table 9). If we could turn the second part of the permutation PC_1, we get the permutation SR (for 7 shiftregisters of 1 byte). This is exactly the way we will realize the permutation.

First we realize SR(7x8) with 7 shiftregisters during the input of the key in a similar way as for IP (see the ——— path in figure 13). Then we rearrange these seven registers in two registers of 28 bit. The first register of 28 bit goes from the first byte up to half the fourth byte. The second register goes in reversed order from the last byte up to half the fourth byte (see the - - - - - path in figure 13). This reversed order of the second 28 bit register permits to turn the second part of the key at the moment that those 2 registers of 28 bit are loaded in a following memory unit of 2 times 28 bit (see figure 13).

This realization consumes a little bit more time, but this isn't a drawback because you don't change the key very often. A variant is possible which change the 2 keys e.g. in 1 clockperiod but this consumes a little bit more place. The fact of using 2 memory-units for 2 keys can be used for the multiple key mode. We'll take the first memory as the major-key register and the second register as the active-key register. Therefore we only have to add 2 feedbacklines which carry back the content of the active-key register to the



R^* and R are permutations

Figure 9: With this shiftregister structure it is possible to carry out a set of 64→64 permutations.

$$IP^* = \left[\begin{array}{cccccccc} 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 \\ 10 & 12 & 14 & 16 & 9 & 11 & 13 & 15 \\ 18 & 20 & 22 & 24 & 17 & 19 & 21 & 23 \\ 26 & 28 & 30 & 32 & 25 & 27 & 29 & 31 \\ 34 & 36 & 38 & 40 & 33 & 35 & 37 & 39 \\ 42 & 44 & 46 & 48 & 41 & 43 & 45 & 47 \\ 50 & 52 & 54 & 56 & 49 & 51 & 53 & 55 \\ 58 & 60 & 62 & 64 & 57 & 59 & 61 & 63 \end{array} \right]$$

Table 7: $IP^* = SR^{-1}$ after IP

$$IP^* = \left[\begin{array}{cccccccc} 2_1 & 4_1 & 6_1 & 8_1 & 1_1 & 3_1 & 5_1 & 7_1 \\ 2_2 & 4_2 & 6_2 & 8_2 & 1_2 & 3_2 & 5_2 & 7_2 \\ 2_3 & 4_3 & 6_3 & 8_3 & 1_3 & 3_3 & 5_3 & 7_3 \\ 2_4 & 4_4 & 6_4 & 8_4 & 1_4 & 3_4 & 5_4 & 7_4 \\ 2_5 & 4_5 & 6_5 & 8_5 & 1_5 & 3_5 & 5_5 & 7_5 \\ 2_6 & 4_6 & 6_6 & 8_6 & 1_6 & 3_6 & 5_6 & 7_6 \\ 2_7 & 4_7 & 6_7 & 8_7 & 1_7 & 3_7 & 5_7 & 7_7 \\ 2_8 & 4_8 & 6_8 & 8_8 & 1_8 & 3_8 & 5_8 & 7_8 \end{array} \right]$$

$$IP^{**} = \left[\begin{array}{cccccccc} 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 \end{array} \right]$$

Table 8: $IP^* \equiv 8$ times IP^{**}

the input is:

byte1= 1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
byte2= 9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
byte3=17	**18**	**19**	**20**	**21**	**22**	**23**	**24**
byte4=25	**26**	**27**	**28**	**29**	**30**	**31**	**32**
byte5=33	**34**	**35**	**36**	**37**	**38**	**39**	**40**
byte6=41	**42**	**43**	**44**	**45**	**46**	**47**	**48**
byte7=49	**50**	**51**	**52**	**53**	**54**	**55**	**56**
byte8=57	**58**	**59**	**60**	**61**	**62**	**63**	**64**

the result is:

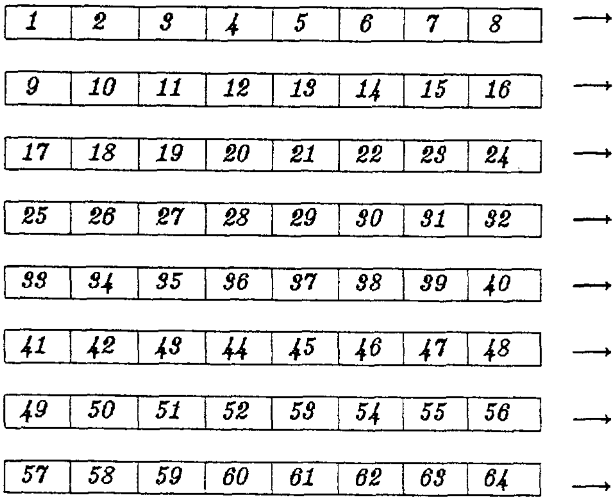
$\longrightarrow$ **57** *1*	**49** *2*	**41** *3*	**33** *4*	**25** *5*	**17** *6*	**9** *7*	**1** *8*
$\longrightarrow$ **58** *9*	**50** *10*	**42** *11*	**34** *12*	**26** *13*	**18** *14*	**10** *15*	**2** *16*
$\longrightarrow$ **59** *17*	**51** *18*	**43** *19*	**35** *20*	**27** *21*	**19** *22*	**11** *23*	**3** *24*
$\longrightarrow$ **60** *25*	**52** *26*	**44** *27*	**36** *28*	**28** *29*	**20** *30*	**12** *31*	**4** *32*
$\longrightarrow$ **61** *33*	**53** *34*	**45** *35*	**37** *36*	**29** *37*	**21** *38*	**13** *39*	**5** *40*
$\longrightarrow$ **62** *41*	**54** *42*	**46** *43*	**38** *44*	**30** *45*	**22** *46*	**14** *47*	**6** *48*
$\longrightarrow$ **63** *49*	**55** *50*	**47** *51*	**39** *52*	**31** *53*	**23** *54*	**15** *55*	**7** *56*
$\longrightarrow$ **64** *57*	**56** *58*	**48** *59*	**40** *60*	**32** *61*	**24** *62*	**16** *63*	**8** *64*

the realized permutation

$$SR = [\begin{array}{cccccccc} 57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 \\ 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 \\ 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\ 60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 \\ 61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 \\ 62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 \\ 63 & 55 & 47 & 39 & 31 & 23 & 15 & 7 \\ 64 & 56 & 48 & 40 & 32 & 24 & 16 & 8 \end{array}]$$

the *italic* number indicates a memory location and the **boldface** number a data bit

Figure 10: If you shift 8 byte from a bus into 8 shiftregisters, you get the permutation SR (a permutation is represented in a similar way as in the NBS norm of DES)



the output is:

byte1=8	**16**	**24**	**32**	**40**	**48**	**56**	**64**
byte2=7	**15**	**23**	**31**	**39**	**47**	**55**	**63**
byte3=6	**14**	**22**	**30**	**38**	**46**	**54**	**62**
byte4=5	**13**	**21**	**29**	**37**	**45**	**53**	**61**
byte5=4	**12**	**20**	**28**	**36**	**44**	**52**	**60**
byte6=3	**11**	**19**	**27**	**35**	**43**	**51**	**59**
byte7=2	**10**	**18**	**26**	**34**	**42**	**50**	**58**
byte8=1	**9**	**17**	**25**	**33**	**41**	**49**	**57**

the realized permutation

$$SR^{-1} = [\begin{array}{cccccccc} 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 \\ 7 & 15 & 23 & 31 & 39 & 47 & 55 & 63 \\ 6 & 14 & 22 & 30 & 38 & 46 & 54 & 62 \\ 5 & 13 & 21 & 29 & 37 & 45 & 53 & 61 \\ 4 & 12 & 20 & 28 & 36 & 44 & 52 & 60 \\ 3 & 11 & 19 & 27 & 35 & 43 & 51 & 59 \\ 2 & 10 & 18 & 26 & 34 & 42 & 50 & 58 \\ 1 & 9 & 17 & 25 & 33 & 41 & 49 & 57 \end{array}]$$

Figure 11: by reading out of a shiftregister you realize the permutation SR^{-1}

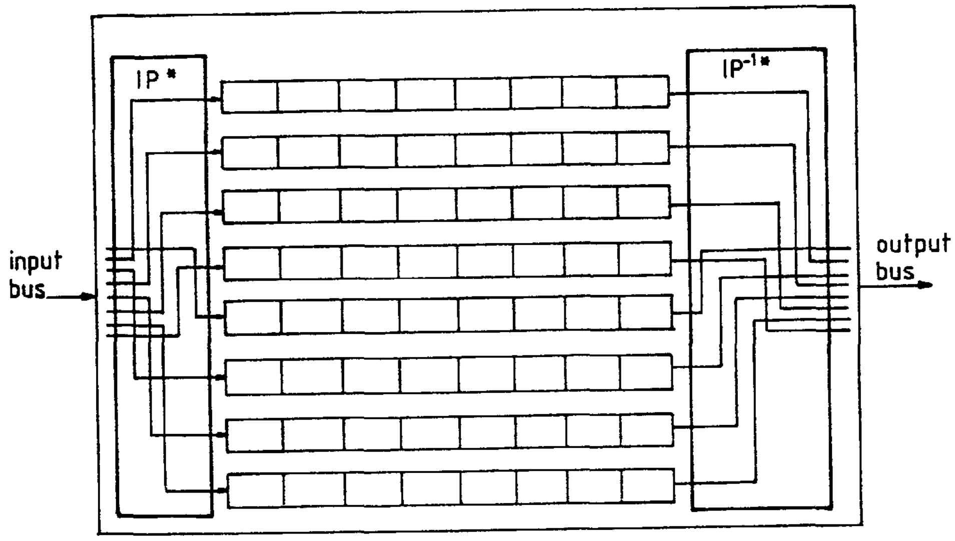


Figure 12: realization of IP and IP^{-1} with the same shiftregister

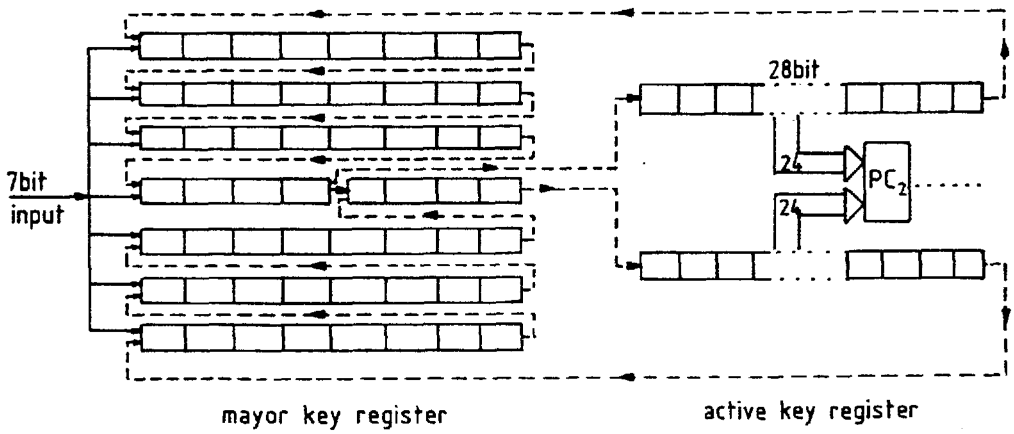
major-keyregister so that no key is lost by the transport from the major-key register to the active-key register (see figure 13). This configuration can also expanded with a third keyregister which is very interesting for multiple encipherment of the active key [12]

4. a fast serial/parallel realization for memory and transport

We'll explain our internal interconnection system built out of shiftregisters and multiplexers. This system is small, very fast and flexible enough for the DES–algorithm.

$$\begin{array}{cccccccccccccccccc}
PC_1 = & & & & & & & & & & SR(7\text{x}8)= & & & & & & & \\
{[}\,57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 & & & {[}\,57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 \\
58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 & \rightarrow & & 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 \\
59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 & & & 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\
60 & 52 & 44 & 36 & & & & & & & 60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 \\
63 & 55 & 47 & 39 & 31 & 23 & 15 & 7 & & & 61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 \\
62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 & \searrow & & 62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 \\
61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 & \nearrow & & 63 & 55 & 47 & 39 & 31 & 23 & 15 & 7\,] \\
 & & & & 28 & 20 & 12 & 4\,] & & & & & & & & & &
\end{array}$$

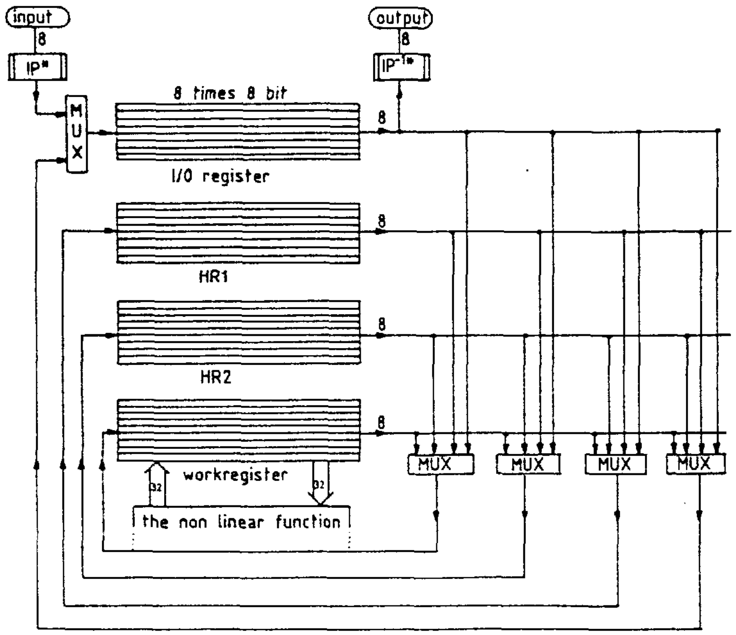
Table 9: the structure in PC_1 and the relation with SR



——— = the path to read in a new key

- - - - - = the path to change the 2 key's.

Figure 13: realization of PC_1 by 2 subsequent memory transports



HR1 and HR2 are additional registers. The workregister is the memory unit in the iteration hardware (see section 9).

Figure 14: the fast internal transport organization. It's used to calculate the modes (the exors are omitted from the figure for simplicity)

A common way to organize transport is with one databus and an addressbus. Then the memory usually consists out of RAM and ROM units. The advantage of this solution is the flexibility of addressing and the possibility of transport between many devices far from each other.

We took another approach because we don't need those advantages. We chose to organize the memory in 4 shiftregister units of 8 byte (8×8) each. Instead of one bus, we made a connection path from every output to each of the four inputs (included his own input) (see figure 14). For our design this structure is faster, small and enough flexible.

It's faster because of two reasons. First, it isn't necessary to specify a new address for every byte. Second, it can transport 4 bytes simultaneously. So the maximum capacity is 32 byte in 8 clockperiods. We'll use this structure to calculate the feedbackmodes. This is a time critical job because it isn't possible to pipeline it with something else (see section 6).

It is difficult to explain why this structure is very small. On the first sight you may think to discover a new routing- problem. This isn't so because the length of the connections can be kept small by a good floorplan. In section 9.2 we describe a floorplan for a nMOS realization. Most interconnections between these registers aren't longer than $150\mu m$. It was even possible to place all the interconnections and multiplexers on $0,5mm^2$ (8×8 version).

This structure has a smaller flexibility because we always have to transport all 8 bytes in the same sequence from one unit to the other. Transporting less will break up every byte by partly shifting it out and partial shifting something else in. It is clear that this partial transport is a " wrong " way to transport entire bytes. But as we see in section 5.1, we can use this " wrong " way of transport to execute the 1 byte modes in a very simple way. Remark that to execute the modes we also need to incorporate some exors in the structure. This can be done in many ways depending on the used technology. One method well suited for nMOS is adding 8 fixed exors between the workregister and the input output register and also 8 fixed exors between the workregister and the HR1 (= additional register 1). The output of those exors are connected to each multiplexer again. It can be shown that this configuration is sufficient to calculate the modes.

4.1. a possible modification for smaller, but slower devices

Up to now, we've used units of 8×8 shiftregisters for the input output registers and the other registers of the fast internal transport. However it's also possible to construct an equivalent version with 4 shiftregisters of 16 bits long [5].

The difference between the two is that the interconnection hardware is only half of the size for the 4×16 version. On the other hand the 4×16 version is also 2 times slower for the transport of data. Therefore the 4×16 version is only usefull for a slow and small version (≤ 5 Mbit and $\pm 4mm^2$).

5. Equivalent representations

5.1. modes

As explained in section 3.2, we can easily realize IP when we transport data from an 8 bit bus into a shiftregister. This consumes no extra time when it can be combined with the input and output from and to the environment. However, this method needs a lot of time when you have to use it for data already on chip. Therefor it's very usefull to move the permutations IP and IP^{-1} out of the feedback loop of the modes to the input and output of the chip.

Figure 3 shows clearly that for the 8 byte modes the solution is found. However for the 1 byte modes, a little bit more explanation is needed to show why the result in figure 8 is usefull. The permutations Q_a and Q_a^{-1} of figure 8 are the permutations IP^* and IP^{-1*} we used to read in and out (section 3). The realization of the selections and injections is very simple with our internal tansport structure (section 4). We saw there that you can only transport entire bytes if you transport all 8 bytes together by shifting 8 times the shiftregisters. However if you shift those registers only once, you will eject the last bit of every byte ($=S_a$). The other seven bits of every byte ($=S_b$) will be shifted to the seven last places of every byte ($=I_c$) and there will enter a new bit on the first place of every byte ($=I_d$). This new bit is the ejected bit exored with the correspondent bit of the incoming byte of data.

5.2. the permutation P

Because the permutation P will be realized hardware with wires, it's obvious that the needed area can be diminished by a well choosen modified P [3]. How you get the optimal

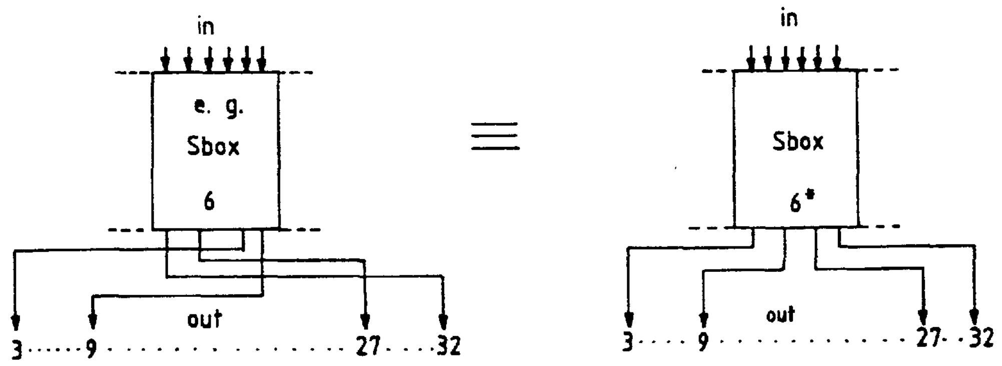


Figure 15: optimization of P for a hardwired realization

modified P is shown in figure 15.

5.3. optimization of the 16 iterations

Starting from the official representation of the DES algorithm, we see that each iteration starts with the evaluation of the non-linear function, and ends with reversing the 2 datablocks. If you reorder [13] the algorithm as in figure 16, it becomes clear that the exchange can be done at the same moment as the evaluation of the non-linear function. In this way, the time to execute DES* (=DES without IP and IP^{-1}) is almost equal to the time needed for 16 evaluations without losing time by exchanging the 2 datablocs. As consequence, the evaluation of the non-linear function is the time critical path of the algorithm. The only small drawback is that at the end an extra exchange of the right and left block is required.

6. Pipelining

The aim of the pipelining is to prevent that datarate is slowed down by the time used for datatransport between chip and the environment. This is very important because this communication is slow.

In our design we distinct 3 sections able to work simultaneously: an input section, a DES* section and an output section. While the input section is entering the next datablock, the DES* is working on the actual datablock and the outputsection is busy to releaze the previous datablock. When these 3 sections have finished, there is a large amount of data which has to be exchanged between the sections. The operation doing this job is called the transfert. The transfert is executed with the structure described in section 4 . This operation calculates also the modes.

To illustrate the functioning of the device, especially to show how the modes are realized, we have in figure 17 a representation of the activity in function of time e.g. for the modes ECB and CBC. Note that HR1 and HR2 (additional memory 1 and 2) are memories needed in the feedbackloop of the modes.

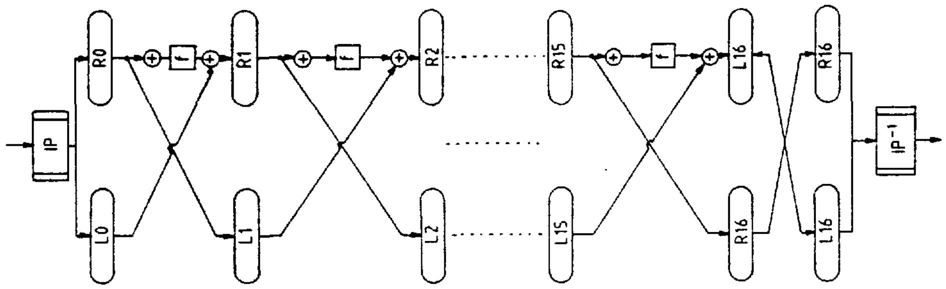


Figure 16: optimization of the 16 iterations

7. Rearrangement of the functional elements

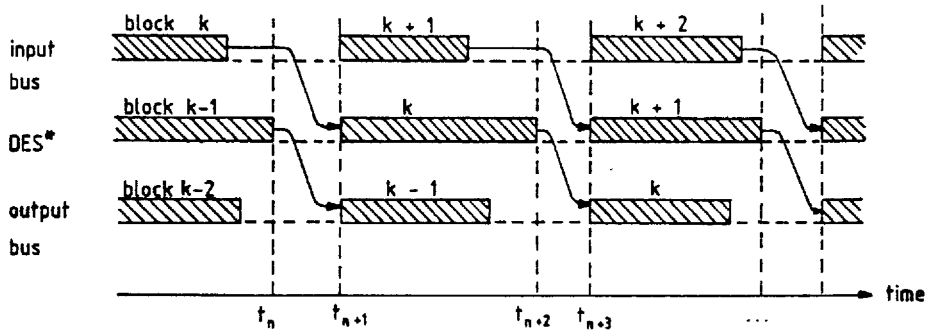
The aim of rearranging is not to avoid the interconnections but to shorten the length.

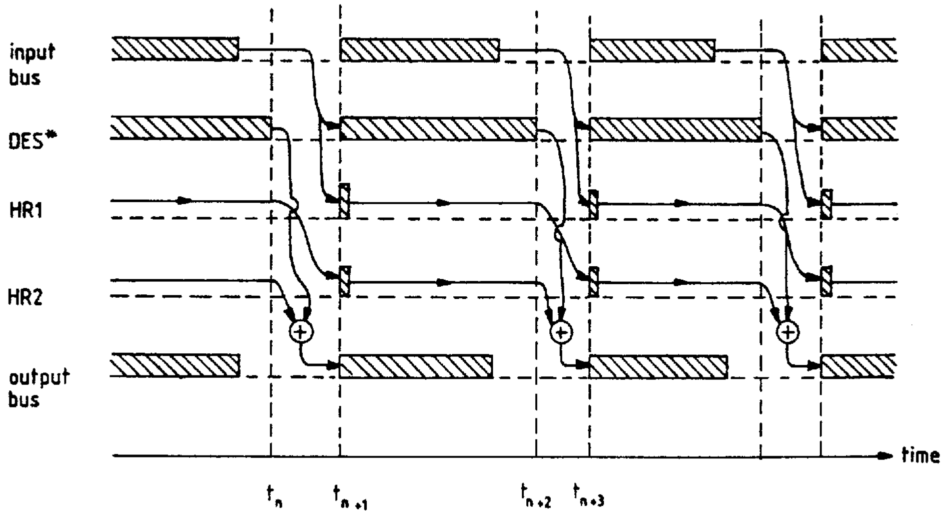
A large part of the routing is shortened by mixing the memory cells in the iteration hardware. If you number the cells from 1 to 64 and put them in the following order: (1 33), (2 34), (3 35), (4 36), ... (29 61), (30 62), (31 63), (32 64), the connected cells come next to each other. So the connections become shorter than $100\mu m$. This structure can be build with 32 cells each containing a schiftregister of 2 bit and 1 exor.

The second way to shorten lines relies on the fact that a memorycell is much larger than an interconnection. So we'll put the lines from the subkey, the lines from the memory and the lines from the S-boxes wired next to each other. In this way we have designed a floorplan for the iteration hardware that minimize the length between the memory and the S-boxes that is part of the time critical path (cfr. section 5.3).

8. Modular construction of the controller

Microprogramming is known as a good but slow way of controlling. The speed problem can be solved by using a lot of small units. Every unit is able to carry out one class of tasks. Above this units for the tasks is one unit to coordinate the cooperation between the units. The communication between this coordination unit and the task units happens with microcode. There is no communication between the task units so that modularity and testability is assured.





▨ :shows that this part is active

:shows the transfert of data between the different memories.

Remark for CBC-decipher that the calculation of DES* introduces a supplementary delay of the data in the main path so that in the feedback path 2 delays are needed instead of 1.

Figure 17: the activity in function of time for the modes ECB and CBC-decipher

9. The entire design

9.1. the architecture

In figure 18 the survey of the total architecture is shown. Depending on the design strategy, used technology and the desired features some elements may have to change. The solution of the figure is very well suited for an nMOS chip with a speed of about 14mbit/sec. .

As seen in section 4, there are 4 data memories of 64 bit (=8 × 1 byte). The memory called workregister is a special one because it can work in 2 different ways. It can be switched as 8 shiftregisters of 1 byte, or as 32 registers of 2 bit to perform the 16 iterations (see section 7).

There are also 2 key memories of 56 bit(=7 × 1 byte). The parity check is carried out on each key byte that is entered for the first time. With this configuration you can memorize 2 keys e.g. a major and an active key. When you enter a new key, you'll overwrite the key which was in the input key register.

In the chip we have the following three transports:

1. the internal transport, mainly used for realizing the modes
2. an inputbus
3. an outputbus

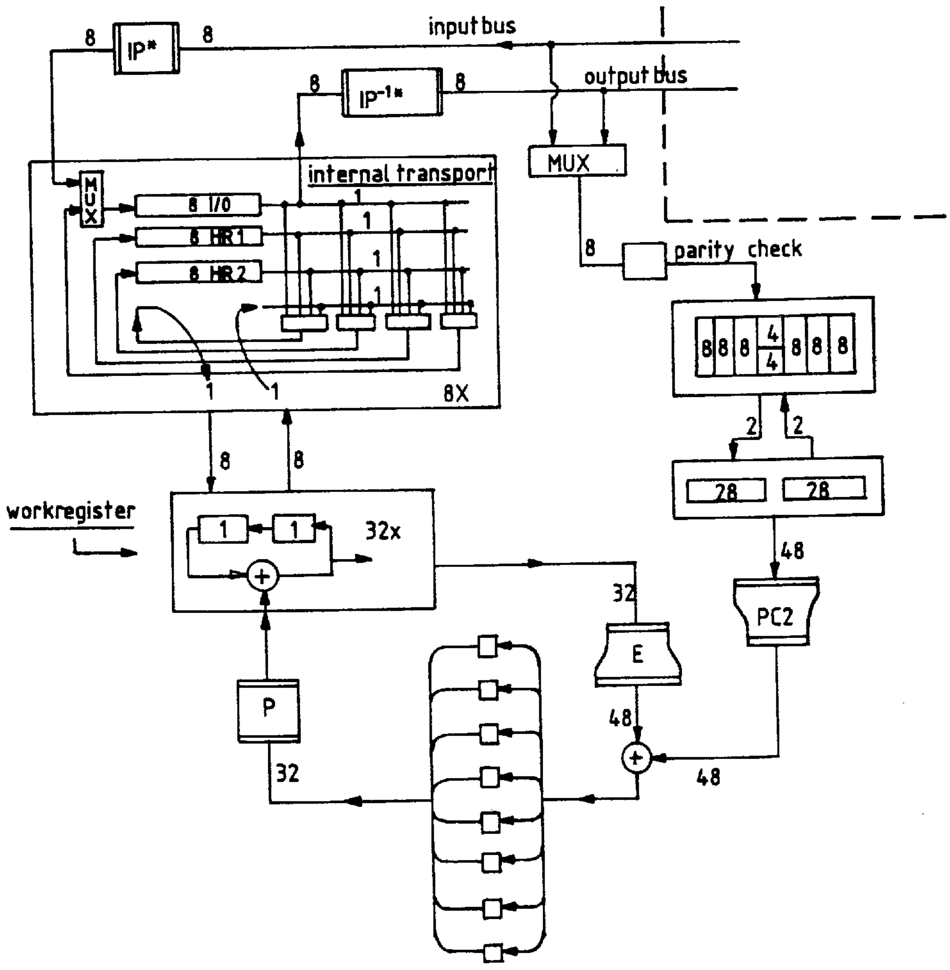
The first serves to transport data in a fast way between the 4 data memories (see section 4). The second and the third serve for datatransport between the environment, one fixed register of 64 bit (called the input output register) and the active key register (see figure 12 and 13).

Maybe it is now a good moment to take attention on the conformity of the used techniques. E.g. reading in the data needs shiftregisters to realize at the same time *IP*; with those shiftregisters a very fast transport is possible; we need that fast transport to allow the pipelining and the execution of the modes; to allow this way of transport, the workregister has to be a shiftregister; on the other hand, the iterations can be carried out very fast using cells of a shiftregister,... etc.. It is with this conformity that we could design a chip which at the same time is very fast and very small.

9.2. the floorplan

In figure 19 you find a floorplan of an nMOS design. The total area used is about 9 mm^2 and the transistor density is more than a thousand transistors on 1 mm^2. In the floorplan you can destinct 4 important parts.

1. The datapad doing the 16 iterations (+ the subkey generation)
2. The memories and multiplexers of the fast internal transport
3. The input-output bus
4. The controllers



You see that the amount of interconnections is diminished compared with figure 1

Figure 18: survey of the new architecture

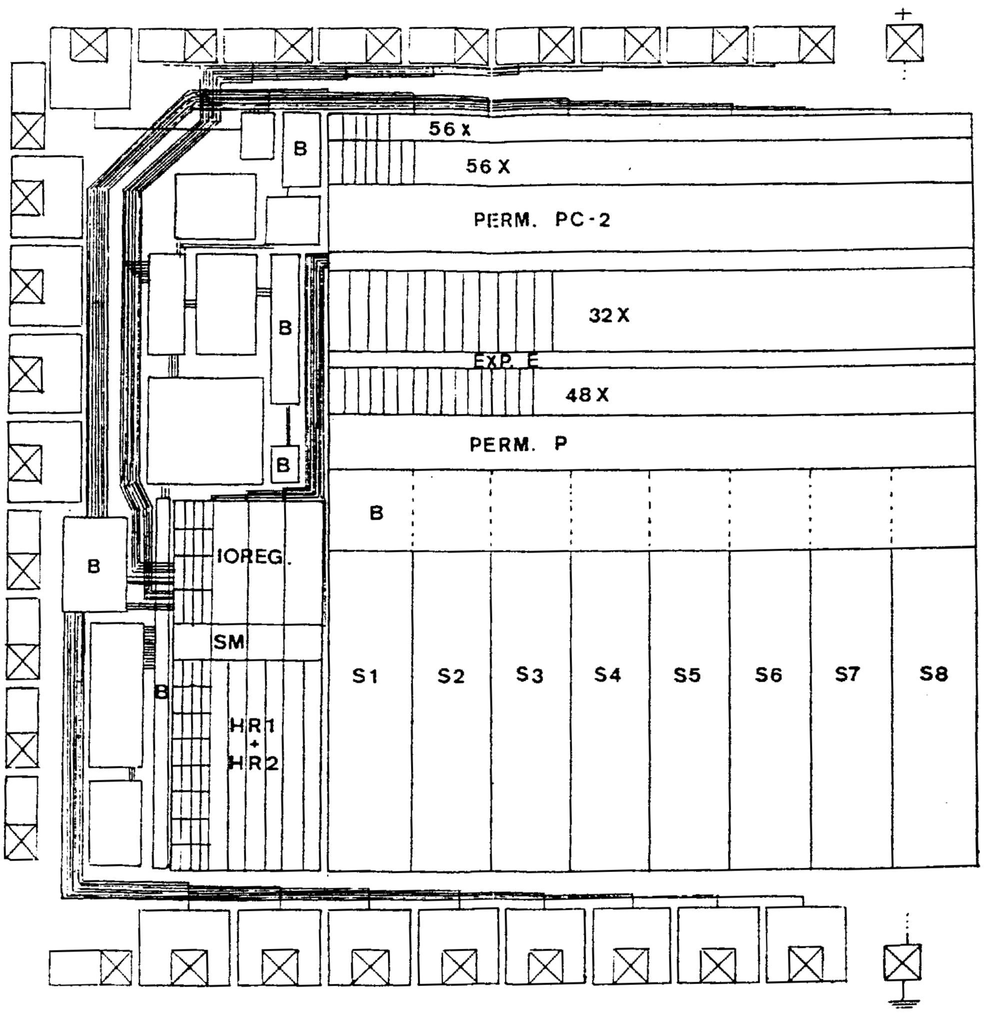


Figure 19: floorplan for a nMOS version

10. a modified design for exhaustic search of the keys

The idea to cryptanalyze the DES by an exhaustive search was proposed by Diffie and Hellman [15]. An improvement is now presented, which mainly solves the problem of the complexity of the machine and its cooling.

A chip builded for exhaustic search of the keys must have 2 properties. First it has to be very fast and second it must work with a minimum of communication with the environment.

To make it fast we'll use the property that an exhaustic search of the key can be realized using only the ECB mode. Therefore we will divide the path calculating the non-linear function in e.g. 3 section and pipeline those sections. There is however the problem that we need the result of this non-linear function to calculate the new input for the non-linear function. This will be solved by calculating simultaneous DES for 3 different keys. To do that we need three workregisters and three keyregisters. In that way the speed can be improved by a factor 3.

To minimize the communication with the environment, we'll generate the subsequent keys on chip and we'll do the check of the result also on chip. To generate the subsequent keys, we enter once a start value for the key in a counter on chip and then augment this value each time by 1. By giving each chip a good startvalue, the whole key space will be checked. To check the result, we enter only once the 2 datablocks for which we search the key. The first block will serve as input for the DES algorithm and the result of the DES algorithm will be compared with the second datablock on chip. If the result is equal to the second block, an interrupt signal is given. In this way, only a μcomputer and a big power supply is needed to command e.g. 10,000 chips.

Important is at which speed this device can work! To calculate three outputs you need:
48 (=16 iterations) +2 (=delay in the pipeline) +3 (=time for in and output) = 53 clockcycli.
At a clockfrequency of 20 Mhz you can check $\pm 1.13 \cdot 10^6$ keys in one second. Suppose that one device (incl. connection) costs 40\$ and that you spend 1,000,000 \$, you can with $2.5 \cdot 10^4$ devices calculate $2.8 \cdot 10^{10}$ keys in one second, or $1 \cdot 10^{14}$ keys in one hour. So you calculate $1.7 \cdot 10^{16}$ keys in only 1 week. In total there are $7.2 \cdot 10^{16}$ keys. On the avarage you will find the key after you've tried $3.6 \cdot 10^{16}$ keys and for this you need about two weeks. If a choosen plaintext attack is possible, the time needed to find the key is devided by 2 [11]. It may also be necessary to make allowance for more than one key satisfying: cypherblock(64bit) = DES(plaintext,key) [7].

The proposed hardware can be designed for CMOS such that no power problems would exist.

11. Obtained results

Without exaggerating, we may expect that a speed up to 20 Mbit/sec. is possible to obtain. Maybe higher speeds should be possible, but this should certainly cost a large amount of power and area.

It's easy to calculate the speed by hand. Because of the pipelining, the chip is sometimes doing many tasks simultaneously (see figure 17). The chip is so designed that the

clock	datarate	area
3Mhz	3,4Mbit	2x2 mm^2
5Mhz	5,5Mbit	
10Mhz	11 Mbit	3x3 mm^2
14Mhz	15,5Mbit	
18Mhz	20 Mbit	4x4 mm^2

Table 10: datarate in function of the clockfrequency (with the expected area in $3\mu m$ technology)

evaluation of the 16 iterations takes normally most of the time. So the time needed for encrypting or decrypting each datablock is the time for 16 iterations augmented with the time for the transfert. In our design, 1 iteration needs 3 clockcycli. Finally, we also need 2 clockcycli to allow the controller to jump from one to an other task unit (section 8). Together we get:

Transfert:	8 clockperiods
DES*	:48 clockperiods
jumps	: 2 clockperiods
Total	:58 clockperiods

This means 64/58 bits in one clockperiod. To know the total speed you have to estimate the clockfrequency. In our design there are 2 to 3 gatelevels for every 1/2 clockperiod (2 phase clock)[10]. So a high clockspeed can surely be obtained. In table 10 we give some frequencies and the corresponding speed. We also indicate the possibility to exchange speed versus area (and power). One can agree that the design is very compact compared with his performances. Smaller technology should futher minimize the area.

12. Conclusions

In this paper we presented several improvement in order to realize faster and smaller chips.

We also proved that the initial permutation and the inverse initial permutation can always be located at the input, respectively the output of each mode.

To use DES in a strong way one has to change frequently the active key (e.g. every 10 seconds) and this active key must be multiple enciphered with different major keys before transmision.

Acknowledgment

The authors are grateful to Dr. J.-J. Quisquater and Prof. M. Davio (Philips Research Laboratory, Brussels, Belgium), for discussions about this paper.

[1] R. Cushman, "Data encryption chips provide security, is it false security," *EDN*, February 1982, pp. 39 - 42.

[2] A. Konheim, "*Cryptography: A Primer*," John Wiley, Toronto, 1981.

[3] M. Davio, Y. Desmedt, M. Fosseprez, R. Govaerts, J. Hulsbosch, P. Neutjens, P. Piret, J.-J. Quisquater, J. Vandewalle, P. Wouters, "Analytical Characteristics of the DES," *Advances in Cryptology, Proc. Crypto 83*, August 1983, Plenum, pp. 171 - 202.

[4] D. Davies, "The Data Encryption Standard," *International Course on Cryptography and Computer Security*, ESAT, K.U.Leuven, Belgium, November 29 - December 2, 1983.

[5] J. Goubert and F. Hoornaert, "Study about an efficient chip implentation of the DES (in Dutch: Studie van een efficiente implementatie van het DES algoritme op chip)," Final work, K.U.Leuven, Department Elektrotechniek, June 1984.

[6] H. De Man, L. Reynders, M. Bartholomeus and J. Cornelissen, "Plasco: A silicon compiler for nMOS and CMOS PLA," *Proc. IFIP: VLSI 83*, Trondheim, Norway, August 1983, pp. 171 - 181.

[7] Y. Desmedt, J.-J. Quisquater and M. Davio, "Dependence of output on input in DES: Small avalanche characteristics," *Advances in Cryptology, Proc. Crypto 84*, August 1984, Springer-Verlag.

[8] FIPS publication, "DES modes of operation," *Federal Information Processing Standard*, no.81, National Bureau of Standards, U. S. Department of Commerce, Washington D. C. , U. S. A. , December 2, 1980.

[9] "Data Encryption Standard," *FIPS* (NBS Federal Information Processing Standards Publ.), no. 46, January 1977

[10] C.Mead and Conway, "An introduction to VLSI systems," Addison Wesley, Reading, 1980.

[11] M. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig and P. Schweitzer, "Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard," *Electrical Engineering Dep. Stanford Univ.* , SEL 76-042.

[12] American National Standard Institute, "Financial Institution Keymanagement" Draft April 1984, N216.

[13] C. H. Meyer and S. M. Matyas, "*Cryptography: A New Dimension in Computer Data Security*," J. Wiley, New York, 1982.

[14] M. Davio, Y. Desmedt, J. Goubert, F. Hoornaert, and J.-J. Quisquater, "Efficient hardware and software implementations to the DES," Abstract, *Advances in Cryptology, Proc. Crypto 84*, August 1984, Springer-Verlag.

[15] W. Diffie and M. E. Hellman, "Exhaustive cryptanalysis of the NBS Data Encryption Standard," *Computer*, vol. 10, no. 6, pp. 74 - 84, June 1977.