Journal of Universal Computer Science, vol. 12, no. 4 (2006), 395-407
submitted: 31/10/05, accepted: 15/03/06, appeared: 28/4/06 © J.UCS

Pareto-Optimal Hardware for Substitution Boxes

Nadia Nedjah
(Department of Electronics Engineering and Telecommunications,
Faculty of Engineering,
State University of Rio de Janeiro, Brazil
nadia@eng.uerj.br)

Luiza de Macedo Mourelle
(Department of Systems Engineering and Computation,
Faculty of Engineering,
State University of Rio de Janeiro, Brazil
ldmm@eng.uerj.br**)**

Abstract: In this paper, we propose a methodology based on genetic programming to automatically generate hardware designs of substitution boxes necessary for many cryptosystems such as DES encryption system. We aim at evolving minimal hardware specifications, which minimise both space (i.e. required gate number), response time (i.e. encryption and decryption time) and dissipated power. We compare our results against existing and well-known designs, which were produced by human designers using conventional methods.

Keywords: S-box, cryptography, genetic algorithms, evolvable hardware, multi-objective optimisation.
Categories: I.2.2, C.3, B.5.2, E.3

1 Introduction

In cryptography, *confusion* and *diffusion* are two important properties of a secure cipher as identified in [Shanon, 49]. Confusion allows one to make the relationship between the encryption *key* and *ciphertext* as complex as possible while diffusion allows one to reduce as much as possible the dependency between the *plaintext* and the corresponding ciphertext. Substituting a symbol in the plaintext by another has been used as a technique of confusion and rearranging the order of the symbols has been used as a mechanism of diffusion. Here we concentrate on confusion using substitution boxes or simply *S-boxes*. Here, we concentrate on evolutionary design of hardware for s-boxes.

Designing a hardware that fulfils a certain function consists of deriving from specific input/output behaviours, an architecture that is *operational* (i.e. produces all the expected outputs from the given inputs) within a specified set of constraints. Besides the input/output behaviour of the hardware, conventional designs are essentially based on knowledge and creativity, which are two human characteristics and too hard to be automated. Evolutionary hardware is a design that is generated

using simulated evolution as an alternative to conventional-based electronic circuit design. *Genetic evolution* [Haupt, 98] is a process that evolves a set of genotypes, i.e. *population*, producing a new population at each iteration process. Here, individuals are hardware designs. The more the design obeys the constraints, the more it is used in the reproduction process. The design constraints can be expressed in terms of hardware area and/or response time requirements. The freshly produced population is yield using some *genetic operators* such as *crossover* and *mutation* that attempt to simulate the natural breeding process in the hope of generating new design that are *fitter*, i.e. respect more the design constraints. Genetic evolution is usually implemented using *genetic algorithms*.

The remainder of this paper is organised in five parts. In Section 2, we introduce symmetric cryptography and the use of substitution boxes. In Section 3, we describe the principles of evolvable hardware. In Section 4, we describe the methodology we employed to evolve new compact, fast and less demanding s-boxes. In Section 5, we compare the discovered novel hardware against existing most popular ones. Finally, in Section 6, we draw some conclusions.

2 Substitution Boxes

*S-Box*es play a basic and fundamental role in many modern block ciphers such as DES [Des, 93]. In block ciphers, they are typically used to obscure the relationship between the plaintext and the ciphertext. Perhaps the most notorious S-boxes are those used in data encryption standard (DES). S-boxes are also used in advanced encryption standard (AES) and Kasumi. All three are Feistel [Menezes, 96] cryptographic algorithms and have the simplified structure depicted in Figure 1.

An S-box can simply be seen as a Boolean function of n inputs and m outputs, often with $n > m$. Considerable research effort has been invested in order to design *resilient* S-boxes that can resist the continuous cryptanalyst's attacks. In order to resist linear and differential cryptanalysis [Matsui, 94a, Matsui, 94b], S-boxes need to be confusing i.e. non-linear and diffusing, i.e. non-differential or non auto-correlated.

Due to the high non-linearity and low auto-correlation of substitution functions, the conventional methods for algebraic expression [Rhyne, 73] have small simplification impact. The corresponding digital circuit is of thus large, slow and power-demanding.

In the rest of the paper, we design efficient design using evolutionary computations. The design are *compact*, i.e. requires minimal hardware area, *efficient*, i.e. propagates the output signal within minimal time and *easy*, i.e. dissipates minimal power.

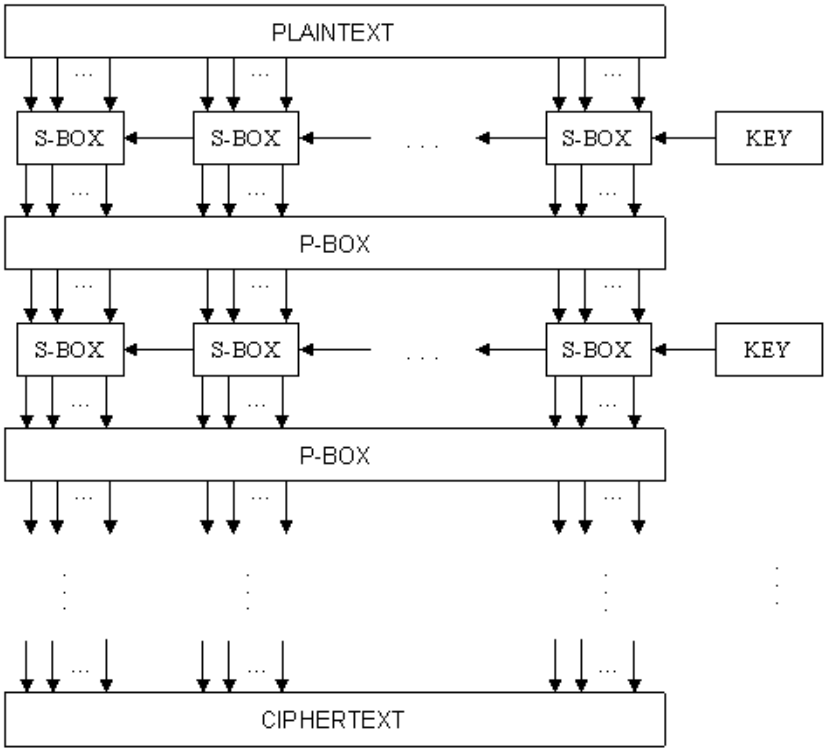

Figure 1: The simplified structure of Feistel cryptographic algorithm

3 Principles of Evolutionary Computation

Starting form random set of solutions, which is generally called *initial population*, evolutionary computation breeds a population of *chromosomes* through a series of steps, called *generations*, using the Darwinian principle of natural *selection*, recombination also called *crossover*, and *mutation*. Individuals are selected based on how much they adhere to the specified constraints. Each evolved solution is assigned a value, generally called its *fitness*, which mirrors how *good* it is in solving the problem in question. Evolutionary computation proceeds by first, randomly creating an initial population of individuals; then, iteratively performing a generation, which consists of going through two main steps, as far as the constraints are not met. The first step in a generation assigns for each chromosome in the current population a fitness value that measures its adherence to the constraints while the second step creates a new population by applying the three genetic operators, which are *selection*, *crossover* and *mutation* to some selected individuals. *Selection* is performed on the basis of the individual fitness. The fitter the individual is, the more probable it is selected to contribute to the new generational population. *Crossover* recombines two chosen solutions to create two new ones using *single-point* crossover or double-point crossover [Haupt, 98]. *Mutation* yields a new individual by changing some randomly

chosen genes in the selected one. The number of genes to be mutated is called *mutation degree* and how many individuals should suffer mutation is called *mutation rate*.

4 Pareto-Optimal Evolvable Hardware

In the context of this paper, an individual is a circuit design of an S-box, which is specified using its truth table form. In the rest of this section, we present the circuit design encoding used, the genetic operators and last but not least, the fitness evaluation of an evolved solution.

4.1 Circuit encoding

We encode circuit schematics using a matrix of cells that may be interconnected. A cell may or may not be involved in the circuit schematics. A cell consists of two inputs or three in case of a MUX, a logical gate and a single output. A cell may draw its input signals from the output signals of gates of previous rows. The gates includes in the first row draw their inputs from the circuit global input signal or their complements. The circuit global output signals are the output signals of the gates in the last raw of the matrix. A chromosome with respect to this encoding is given in Figure 2.

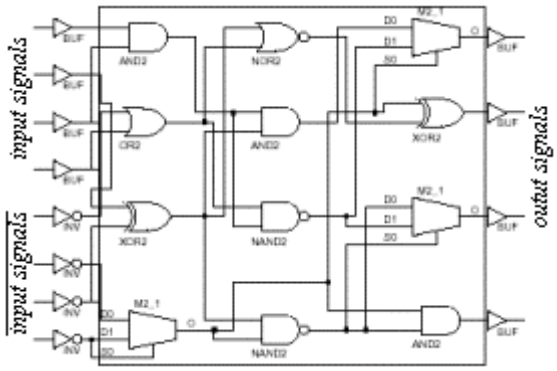

Figure 2: Encoded circuit

4.2 Circuit specification reproduction

Crossover of circuit specification is implemented using a variable four-point crossover [Haupt, 98] as described in Figure 3.

A gene is a circuit gate together with its inputs. So, mutation may occur by changing the gate or one of its input signals, as depicted in Figure 4. In the former case, the gate may be mutated to another of smaller (e.g. AND to NOT), the same (e.g. AND to XOR) or bigger *arity* (e.g. AND to MUX). In the last case, a new signal is randomized to fill in for the new input signal.

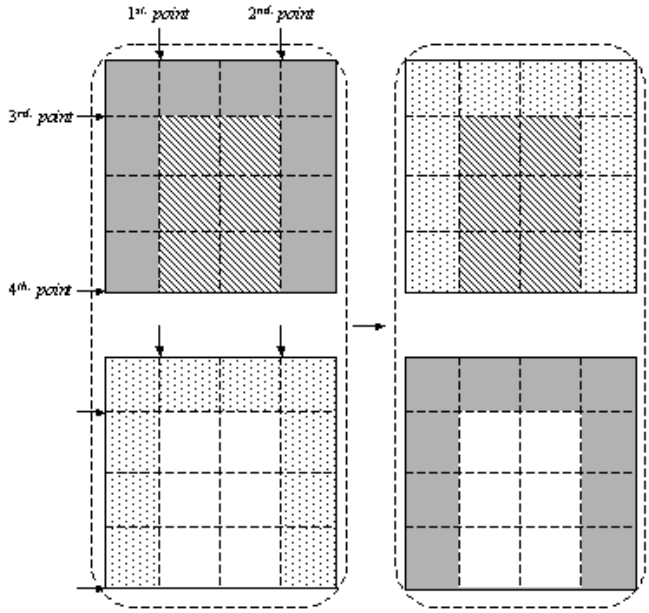

Figure 3: Four-point crossover of circuit schematics

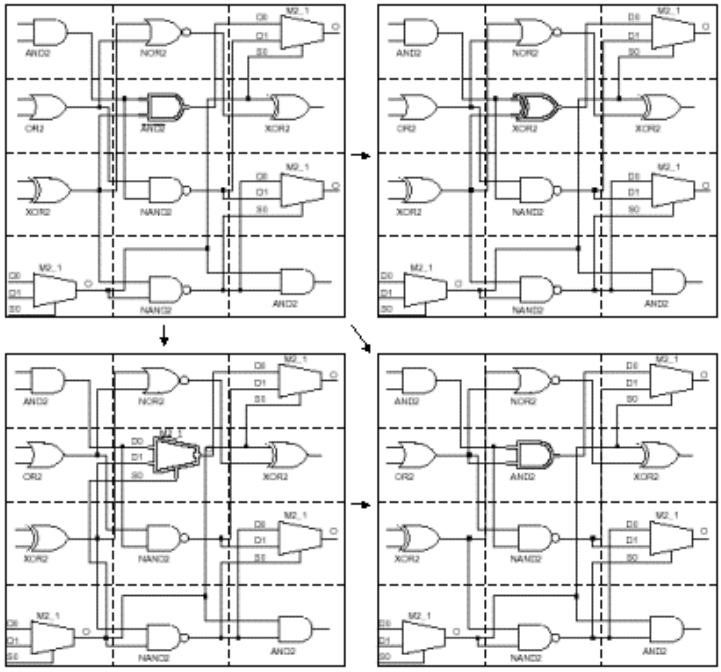

Figure 4: Operand node mutation for circuit specification

4.3 Circuit evaluation

Each circuit within the population is assigned a value, generally called *fitness*. A circuit design is fit if and only if it satisfies the imposed input/output behaviour. In single objective optimisation, a circuit design is considered *fitter* than another if and only if it has a smaller size, shorter response or consumes less power, depending of the optimisation objective size, time or power consumption minimisation respectively. In multi-objective optimisation, however, the concept of fitness is not that obvious. It is extremely rare that a single design optimises all objectives simultaneously. Instead, there normally exist several designs that provide the same *balance*, *compromise* or *trade-off* with respect to the problem objectives. We consider three objectives hardware area (objective *A)*, response time (objective *T*) and power dissipation (objective *P*). Of course, the circuit evolved need to be sound (mandatory objective *S*).

Objective *A* is estimated by the total number of gate equivalent required to implement the evolved circuit and objective *T* by the maximum delay occasioned by it. Objective *P* is evaluated by approximating the switching activity of each gate and the respective fanout [Monteiro, 97].

Name	Gate Equivalent	Delay
NOT	1	0.0625
AND	2	0.2090
OR	2	0.2160
XOR	3	0.2120
NAND	1	0.1300
NOR	1	0.1560
XNOR	3	0.2110
MUX	3	0.2120

Table 1: Gates, size and delay

Let *C* be a digital circuit that uses a subset (or the complete set) of the gates given in Table 1. Let *gates*(*C*) be a function that returns the set of all gates of circuit *C* and *levels*(*C*) be a function that returns the set of all the gates of *C* grouped by level. Notice that the number of levels of a circuit coincides with the cardinality of the set expected from function *levels*. On the other hand, let *B*(*X*) be the Boolean value that the considered circuit *C* propagates for the input Boolean vector *X* assuming that the size of *X* coincides with the number of input signal required for circuit *C*. The fitness function, which allows us to determine how much an evolved circuit adheres to the specified constraints, is given as in (1), wherein *S*(*C*) evaluates the soundness of the evolved circuit *C* and is defined in (2), *A*(*C*) is the occupied hardware area by circuit *C* as defined in (3), *T*(*C*) gives the response time of circuit *C* as defined in (4), and *P*(*C*) evaluates the power dissipated by circuit *C*, which is defined in (5).

$$F(C) = S(C) + \omega_1 A(C) + \omega_2 T(C) + \omega_3 P(C) \quad (1)$$

$$S(C) = \sum_{j=1}^{n} \left(\sum_{i \mid B(x_i) \neq y_{i,j}} \xi \right) \tag{2}$$

$$A(C) = \sum_{g \in gates(C)} GE(g) \tag{3}$$

$$T(C) = \sum_{l \in levels(C)} \max_{g \in l} DE(g) \tag{4}$$

$$P(C) = \sum_{g \in gates(C)} SW(g) FN(g) \tag{5}$$

In (1), x represents the input values of the input signals while y represents the expected output values of the output signals of circuit C, n denotes the number of output signals that circuit C has. For a gate g, functions GE, DE, SW and FN return the number of gate equivalent, propagation delay, number of switches and fanout respectively. For each error in the evolved circuit, the individual pays penalty ξ. Constants ω_1, ω_2 and ω_3 are the weighting coefficients that allow us to consider area, response time and power dissipation to evaluate the performance of an evolved circuit, with $\omega_1 + \omega_2 + \omega_3 = 1$. For implementation issue, we minimised the fitness function below for different values of ω_1, ω_2 and ω_3.

5 Evolutionary vs. Conventional Designs for S-Boxes

For comparison purposes, we evolved the S-boxes of the data encryption standard (DES) and obtained the characteristics (area, time and power) of the evolved circuit. However, for existing work on designing hardware for DES S-boxes, we could only obtain the size in terms of gate equivalent. In Table 2, we give the characteristics of the S-boxes of the fastest implementation of DES known as *bitslice DES* [Kwan, 00]. In Table 3, we present the characteristics of the evolved DES S-boxes.

	area	***time***	***power***
S1	167	2.2010	981
S2	149	3,8290	761
S3	153	2.4675	992
S4	119	1.5505	571
S5	161	2.1170	884
S6	162	2.2395	831
S7	148	2.6180	716
S8	152	2.7915	1009

Table 2: Characteristics of the bitslice DES S-boxes

The parameters used in the evolutionary algorithms were 0.9 as mutation rate, 16 as mutation degree and a population of 100 circuits. It took us about a couple of hours to evolve the designs of DES S-Boxes S1, S2, S3 and S4, given in the appendix. However, we believe that given time, the circuit designs for the S-Boxes will much more efficient in all of the three aspects: hardware area, response time and power consumption (switching activity only).

	area	***time***	***power***
S1	124	1.2880	1071
S2	117	1.1005	981
S3	102	1.7145	412
S4	92	0.7660	771
S5	126	1.2760	514
S6	111	1.9115	959
S7	108	1.2220	801
S8	137	0.9895	897

Table 3: Characteristics of the evolved DES S-boxes

The chart of Figure 5 relates the performance factor of the bitslice DES S-Boxes versus those obtained by the evolutionary process described. The performance factor is the product *area×time×power*. It is clear that the evolutionary S-boxes designs are far better than those designed using conventional methods.

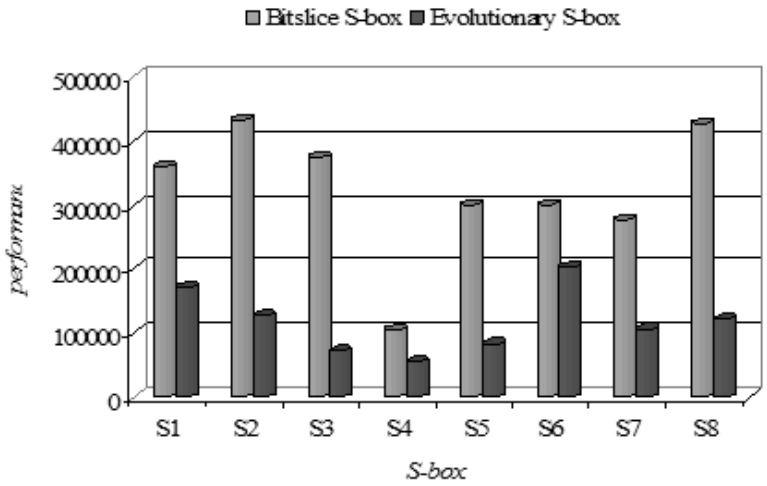

Figure 5: Performance factor of DES S-boxes: bitslice DES S-boxes vs. evolutionary S-boxes

6 Conclusion

In this paper, we proposed a methodology based on evolutionary computation to automatically generate data-flow based specifications for hardware designs of substitution boxes usually used in modern cryptography such as data encryption system (DES) and advanced encryption system (AES). Our aim was evolving minimal hardware specifications, i.e. hardware that minimises the three main characteristics of a digital circuit, which are space (i.e. required gate number), time (i.e. encryption and decryption time) and power dissipation. We compared our results against the fastest existing design. The S-boxes hardware evolved is more compact in terms of the required hardware area.

References

[Shanon, 49] Shanon, C. E., *Communication theory of secrecy systems*, Bell Sys. Tech. J. vol. 28, no. 4, pp. 656-715, 1949.

[Haupt, 98] Haupt, R. L. and Haupt, S. E., *Practical genetic algorithms*, John Wiley and Sons, 1998.

[Des, 93] National Institute of Standards and Technology, *Data Encryption Standard (DES)*, FIPS 46-2 edition, December 1993.

[Menezes, 96] Menezes A. J., Van Oorschot, P. C. and Vanstone, S. A., *Handbook of Applied Cryptography*, CRC Press, 1996.

[Matsui, 94a] Matsui, M., *Linear cryptanalysis method for DES cipher*, T. Helleseth (ed.), Advances in Cryptology, vol. 765, Lecture Notes in Computer Science, pp. 386-397, Springer-Verlag, 1994.

[Matsui, 94b] Matsui, M., *The first experimental cryptanalysis of the Data Encryption Standard*, Y. Desmedt (Ed.), Advances in Cryptology, vol. 839, Lecture Notes in Computer Science, pages 1-11, Springer-Verlag, 1994.

[Rhyne, 73] Rhyne, V. T., *Fundamentals of digital systems design*, Prentice-Hall Electrical Engineering Series, 1973.

[Monteiro, 97] Monteiro, J., Devadas, D., Gosh, A., Keutzer, K. and White, J., *Estimation of average switching activity in combinational logic circuits using symbolic simulation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 1, pp. 121-127, January 1997.

[Kwan, 00] Kwan, M., *Reducing the gate count of bitslice DES*, http://eprint.iacr.org/2000/051.ps, 2000.

Appendix

The evolved DES S-boxes (S1, S2, S3 and S4) specifications are given below. The 6-bit input signal is $X = \langle x_5x_4x_3x_2x_1x_0\rangle$ and the 4-bit output signal is $Y = \langle y_3y_2y_1y_0\rangle$.The most significant two bits x_5 and x_4 are used as the row indices.

S-box S1 specification

$t_0 \Leftarrow$ XOR(NOR(NOR($\bar{x}_1$, x_0),XOR($\bar{x}_0$, x_2)),NAND(OR(XOR($\bar{x}_2$, $\bar{x}_3$), x_1), x_3))

$t_1 \Leftarrow$ MUX(XNOR(x_1, x_3),NAND(OR(XOR($\bar{x}_2$, $\bar{x}_3$), x_1),NAND(NOR($\bar{x}_1$, x_0), XOR($\bar{x}_0$, x_2))),NAND(NAND(NOR($\bar{x}_1$, x_3), x_0), x_0))

$t_2 \Leftarrow$ MUX(AND(OR(XOR($\bar{x}_2$, $\bar{x}_3$), x_1),NAND(NOR($\bar{x}_1$, x_0),XOR($\bar{x}_0$,x_2))),NAND(NAND(NAND(NOR($\bar{x}_1$, x_3), x_0), XOR($\bar{x}_0$, x_2)), NAND($\bar{x}_1$,OR(XOR($\bar{x}_2$, $\bar{x}_3$), x_1))), NAND(XOR(x_1, x_3), NAND(NOR($\bar{x}_1$, x_3), x_0)))

$t_3 \Leftarrow$ XOR(NOR(XOR($\bar{x}_0$, x_2), $\bar{x}_0$),NAND(NAND(NAND(NOR($\bar{x}_1$, x_3), x_0), x_0), NAND(XOR(x_1, x_3),NAND(NOR($\bar{x}_1$, x_3), x_0))))

$t_4 \Leftarrow$ AND(MUX($\bar{x}_2$,NAND($\bar{x}_2$, $\bar{x}_3$),XOR(NAND(x_1, x_3), x_0)),OR(XOR(NAND(x_1, x_3), x_0),MUX($\bar{x}_3$, NOR($\bar{x}_1$,AND($\bar{x}_2$, $\bar{x}_3$)), x_1)))

$t_5 \Leftarrow$ NAND(OR(XOR(NAND(x_1, x_3), x_0),MUX($\bar{x}_3$,NOR($\bar{x}_1$,AND($\bar{x}_2$, $\bar{x}_3$)), x_1)),MUX(MUX($\bar{x}_3$, NOR($\bar{x}_1$,AND($\bar{x}_2$, $\bar{x}_3$)), x_1),MUX(x_3, $\bar{x}_1$,MUX($\bar{x}_0$, x_1, $\bar{x}_3$)),MUX($\bar{x}_2$,NAND($\bar{x}_2$, $\bar{x}_3$), XOR(NAND(x_1, x_3), x_0))))

$t_6 \Leftarrow$ XNOR(MUX(XOR(MUX($\bar{x}_0$, x_1, $\bar{x}_3$),NOR($\bar{x}_0$, x_2)), $\bar{x}_3$, x_2), NAND(NOR($\bar{x}_1$,AND($\bar{x}_2$, $\bar{x}_3$)),MUX($\bar{x}_0$, x_1, $\bar{x}_3$)))

$t_7 \Leftarrow$ MUX(XOR(MUX($\bar{x}_0$, x_1, $\bar{x}_3$),NOR($\bar{x}_0$, x_2)), x_2,MUX(x_3, $\bar{x}_1$,MUX(x_3, $\bar{x}_1$, MUX($\bar{x}_0$, x_1, $\bar{x}_3$))))

$t_8 \Leftarrow$ MUX(MUX($\bar{x}_1$, x_1, $\bar{x}_3$),XNOR($\bar{x}_2$,NAND(XOR(x_0, $\bar{x}_1$), $\bar{x}_3$)), $\bar{x}_0$)

$t_9 \Leftarrow$ XOR(NOR($\bar{x}_1$, x_0),MUX($\bar{x}_2$,MUX(x_2, $\bar{x}_1$,XOR(x_0, $\bar{x}_1$)), $\bar{x}_3$))

$t_{10} \Leftarrow$ NAND(NAND(MUX(OR($\bar{x}_0$, x_1), x_2, $\bar{x}_3$),NOR(NOR($\bar{x}_3$, $\bar{x}_2$), $\bar{x}_0$)), OR(NOR(NOR($\bar{x}_3$, $\bar{x}_2$), $\bar{x}_0$),MUX(x_1, $\bar{x}_1$, $\bar{x}_3$)))

$t_{11} \Leftarrow$ MUX(AND(MUX(OR($\bar{x}_0$,x_1), x_2, $\bar{x}_3$),XOR(x_0, $\bar{x}_1$)),XNOR(MUX(x_2, $\bar{x}_1$,XOR(x_0, $\bar{x}_1$)), NOR(NOR(NOR($\bar{x}_3$, $\bar{x}_2$), $\bar{x}_0$), $\bar{x}_2$)),MUX(x_1, $\bar{x}_1$, $\bar{x}_3$))

$t_{12} \Leftarrow$ XNOR($\bar{x}_1$,MUX(OR(XNOR(NOR($\bar{x}_3$, $\bar{x}_1$), $\bar{x}_0$),XOR(NOR($\bar{x}_3$, $\bar{x}_1$), $\bar{x}_0$)),NOR($\bar{x}_0$,AND($\bar{x}_3$, XNOR($\bar{x}_2$, $\bar{x}_1$))),OR(MUX(x_3,XNOR($\bar{x}_2$, $\bar{x}_3$), $\bar{x}_1$), AND($\bar{x}_3$,XNOR($\bar{x}_2$, $\bar{x}_1$)))))

$t_{13} \Leftarrow$ XNOR(XNOR(NOR($\bar{x}_3$, $\bar{x}_1$), $\bar{x}_0$),MUX(AND($\bar{x}_3$,XNOR($\bar{x}_2$, $\bar{x}_1$)), NAND($\bar{x}_1$,XNOR($\bar{x}_2$, $\bar{x}_1$)),NOR(x_0,XOR($\bar{x}_2$, $\bar{x}_3$))))

$t_{14} \Leftarrow$ MUX(MUX($\bar{x}_2$, x_2,AND($\bar{x}_3$,XNOR($\bar{x}_2$, $\bar{x}_1$))),MUX(x_3,XNOR($\bar{x}_2$, $\bar{x}_3$), $\bar{x}_1$), $\bar{x}_0$)

$t_{15} \Leftarrow$ MUX(XOR($\bar{x}_2$, $\bar{x}_3$),XNOR($\bar{x}_2$, $\bar{x}_1$),NOR(XNOR(NOR($\bar{x}_3$, $\bar{x}_1$), $\bar{x}_0$), XNOR(NOR($\bar{x}_3$, $\bar{x}_1$), $\bar{x}_0$)))

$y_0 \Leftarrow$ MUX(MUX(t_0, t_4, x_4),MUX(t_9, t_{13}, x_4), x_5)

$y_1 \Leftarrow$ MUX(MUX(t_1, t_5, x_4),MUX(t_{10}, t_{14}, x_4), x_5)

$y_2 \Leftarrow$ MUX(MUX(t_2, t_6, x_4),MUX(t_{11}, t_{15}, x_4), x_5)

$y_3 \Leftarrow$ MUX(MUX(t_3, t_7, x_4),MUX(t_{12}, t_{16}, x_4), x_5)

S-box S2 specification

$t_0 \Leftarrow$ XOR(XNOR(NOR(MUX(x_0,NOR(x_0,x_2),x_3),MUX($\bar{x}_2$, $\bar{x}_1$, $\bar{x}_0$)),
OR(NOR(MUX(X0,NOR(x_0,x_2), x_3),MUX(x_0,x_3, $\bar{x}_1$)),NOR(x_0,x_2))), XOR($\bar{x}_2$,x_1))

$t_1 \Leftarrow$ MUX(x_0,NOR(MUX(x_0,NOR(x_0,x_2),x_3),AND(x_0,MUX(MUX(x_2,x_1, $\bar{x}_0$),
AND(x_1, $\bar{x}_3$), $\bar{x}_3$))),NAND(x_1, $\bar{x}_3$))

$t_2 \Leftarrow$ XOR(MUX(x_0, x_3, $\bar{x}_1$),NAND(OR(x_0, x_1),XNOR(NAND(x_1, $\bar{x}_3$),XOR($\bar{x}_2$,x_1))))

$t_3 \Leftarrow$ NOR(NOR(x_0,XOR($\bar{x}_2$,x_1)),AND(x_0,MUX(MUX(x_2, x_1, $\bar{x}_0$),AND(x_1, $\bar{x}_3$), $\bar{x}_3$)))

$t_4 \Leftarrow$ MUX(XNOR($\bar{x}_2$, x_0),MUX(NOR(x_2, $\bar{x}_3$),XOR($\bar{x}_2$, x_0),NAND(x_1, x_3)),MUX(x_0, x_3, $\bar{x}_1$))

$t_5 \Leftarrow$ XNOR(XNOR($\bar{x}_0$, $\bar{x}_1$),NAND($\bar{x}_2$,MUX($\bar{x}_3$,AND(x_1, x_3),MUX(x_0, x_3, $\bar{x}_1$))))

$t_6 \Leftarrow$ AND(x_1, x_3)

$t_7 \Leftarrow$ MUX(MUX($\bar{x}_3$, x_3, x_2),MUX(x_1, $\bar{x}_1$, $\bar{x}_3$), $\bar{x}_0$)

$t_8 \Leftarrow$ XNOR(XNOR($\bar{x}_1$,x_0),NAND(OR($\bar{x}_1$,XOR($\bar{x}_3$,x_2)),NAND($\bar{x}_2$, $\bar{x}_1$)))

$t_9 \Leftarrow$ XOR(XNOR($\bar{x}_1$,x_0),NOR($\bar{x}_3$,NOR(XNOR(x_2, x_1), x_0)))

$t_{10} \Leftarrow$ MUX(NAND($\bar{x}_1$,x_3),AND(NAND($\bar{x}_2$, $\bar{x}_1$),XOR($\bar{x}_3$,x_2)),NAND(XNOR($\bar{x}_0$,x_2),NAND($\bar{x}_3$,x_1)))

$t_{11} \Leftarrow$ NAND(NAND(NAND(XNOR($\bar{x}_0$,x_2),NAND($\bar{x}_3$,x_1)),AND(NAND($\bar{x}_2$, $\bar{x}_1$),
XOR($\bar{x}_3$,x_2))),MUX(NAND($\bar{x}_1$,x_3),NAND($\bar{x}_3$,x_1),NAND($\bar{x}_1$, $\bar{x}_0$)))

$t_{12} \Leftarrow$ MUX(MUX(x_0,MUX(x_1, $\bar{x}_1$, $\bar{x}_3$),MUX(x_2,x_1, $\bar{x}_3$)), NAND(XNOR($\bar{x}_0$,x_1),NAND($\bar{x}_3$, $\bar{x}_2$)),
NOR(x_2,NOR($\bar{x}_1$, $\bar{x}_0$)))

$t_{13} \Leftarrow$ OR(AND(MUX(x_1, $\bar{x}_1$, $\bar{x}_3$),NOR(XNOR($\bar{x}_0$,x_1),x_2)),AND(XNOR($\bar{x}_0$,x_1),NAND($\bar{x}_3$, $\bar{x}_2$)))

$t_{14} \Leftarrow$ NAND(XNOR(AND(x_2, $\bar{x}_3$),NOR($\bar{x}_1$,x_0)),NAND(x_3,NOR(x_2,NOR($\bar{x}_1$, $\bar{x}_0$))))

$t_{15} \Leftarrow$ XNOR(x_0,MUX(MUX(x_2, x_1, $\bar{x}_3$),AND(XNOR($\bar{x}_0$,x_1),NAND($\bar{x}_3$, $\bar{x}_2$)),AND(x_2, $\bar{x}_3$)))

$y_0 \Leftarrow$ MUX(MUX(t_0, t_4, x_4),MUX(t_9, t_{13}, x_4), x_5)

$y_1 \Leftarrow$ MUX(MUX(t_1, t_5, x_4),MUX(t_{10}, t_{14}, x_4), x_5)

$y_2 \Leftarrow$ MUX(MUX(t_2, t_6, x_4),MUX(t_{11}, t_{15}, x_4), x_5)

$y_3 \Leftarrow$ MUX(MUX(t_3, t_7, x_4),MUX(t_{12}, t_{16}, x_4), x_5)

S-box S3 specification

$t_0 \Leftarrow$ MUX($\bar{x}_0$,MUX($\bar{x}_2$, x_0, x_3),XNOR($\bar{x}_3$,XNOR(x_1, $\bar{x}_2$)))

$t_1 \Leftarrow$ MUX(x_3,MUX(MUX(MUX($\bar{x}_2$, x_0, x_3), x_2,XOR($\bar{x}_3$, x_1)),NOR($\bar{x}_0$, x_1), $\bar{x}_3$),
OR(NAND($\bar{x}_0$, $\bar{x}_2$),MUX(MUX(x_2, $\bar{x}_0$, x_3), $\bar{x}_2$,XOR($\bar{x}_3$, x_1))))

$t_2 \Leftarrow$ AND(MUX(x_2,OR($\bar{x}_0$, x_1),MUX($\bar{x}_2$, x_0, x_3)),NAND(XNOR($\bar{x}_0$, $\bar{x}_2$), x_1))

$t_3 \Leftarrow$ XOR(XOR($\bar{x}_3$, x_1),NAND(XNOR(x_1, $\bar{x}_2$), x_0))

$t_4 \Leftarrow$ AND(NAND(OR($\bar{x}_1$, $\bar{x}_0$),XOR(NOR(x_2, x_0), $\bar{x}_3$)), NAND(XNOR(XNOR(NAND($\bar{x}_3$, x_0), $\bar{x}_2$),
XNOR(x_3, $\bar{x}_1$)), XOR(OR($\bar{x}_1$, $\bar{x}_0$), x_3)))

$t_5 \Leftarrow$ XOR($\bar{x}_3$,NOR(MUX(x_0,NOR(XNOR(x_1, x_0),XNOR(NAND($\bar{x}_3$, x_0), $\bar{x}_2$)),
OR($\bar{x}_2$,XNOR(x_3, $\bar{x}_1$))),AND($\bar{x}_2$, $\bar{x}_1$)))

$t_6 \Leftarrow$ MUX(MUX(XNOR(NAND($\bar{x}_3$, x_0), $\bar{x}_2$),NOR(XNOR(x_1, x_0),XNOR(
NAND($\bar{x}_3$, x_0), $\bar{x}_2$)),OR($\bar{x}_1$, $\bar{x}_0$)),AND(OR($\bar{x}_1$, $\bar{x}_0$), XOR(NOR(x_2, x_0), $\bar{x}_3$)),NOR(x_0, x_1))

$t_7 \Leftarrow$ MUX(XOR(NAND($\bar{x}_3$, x_0), $\bar{x}_2$),XOR(x_3, $\bar{x}_1$), $\bar{x}_0$)

$t_8 \Leftarrow$ XOR(NOR(NOR(NOR(XOR(x_2, x_3), x_0), $\bar{x}_2$), x_0),XNOR($\bar{x}_2$, x_1))

$t_9 \Leftarrow$ MUX(MUX(x_1, $\bar{x}_1$,NOR(x_3, $\bar{x}_0$)),NOR(XOR(x_2, x_3), x_0), $\bar{x}_0$)

$t_{10} \Leftarrow$ AND(NAND(NOR(XOR(x_2, x_3), x_0),XNOR($\overline{x}_2$, x_1)), MUX(MUX(x_1, $\overline{x}_1$,NOR(x_3, $\overline{x}_0$)), NAND($\overline{x}_2$, x_0), x_3))

$t_{11} \Leftarrow$ XNOR(MUX(x_1, $\overline{x}_1$,NOR(x_3, $\overline{x}_0$)),NOR(NOR(XOR(x_2, x_3), x_0), $\overline{x}_2$))

$t_{12} \Leftarrow$ MUX(XOR($\overline{x}_1$, $\overline{x}_0$),XNOR(x_2,XNOR($\overline{x}_1$, $\overline{x}_0$)), x_3)

$t_{13} \Leftarrow$ XNOR(NAND($\overline{x}_3$, $\overline{x}_1$),XOR(NAND(NAND($\overline{x}_1$, $\overline{x}_2$), x_0), x_2))

$t_{14} \Leftarrow$ XOR(NOR(NOR(x_1,NAND($\overline{x}_1$, $\overline{x}_2$)),XOR($\overline{x}_1$,NAND($\overline{x}_3$, $\overline{x}_0$))),NAND(XOR($\overline{x}_2$, AND(NAND($\overline{x}_1$, $\overline{x}_2$), x_3)),XNOR($\overline{x}_1$, $\overline{x}_0$)))

$t_{15} \Leftarrow$ XNOR(x_0,MUX(NAND(XNOR($\overline{x}_1$, $\overline{x}_0$), x_3), x_3, $\overline{x}_2$))

$y_0 \Leftarrow$ MUX(MUX(t_0, t_4, x_4),MUX(t_9, t_{13}, x_4), x_5)

$y_1 \Leftarrow$ MUX(MUX(t_1, t_5, x_4),MUX(t_{10}, t_{14}, x_4), x_5)

$y_2 \Leftarrow$ MUX(MUX(t_2, t_6, x_4),MUX(t_{11}, t_{15}, x_4), x_5)

$y_3 \Leftarrow$ MUX(MUX(t_3, t_7, x_4),MUX(t_{12}, t_{16}, x_4), x_5)

S-box S4 specification

$t_0 \Leftarrow$ OR(NOR($\overline{x}_1$,MUX($\overline{x}_0$, x_0, $\overline{x}_2$)),NOR(XNOR(MUX(x_0, x_3, x_2), x_1),AND($\overline{x}_2$, x_3)))

$t_1 \Leftarrow$ MUX(NAND(NAND($\overline{x}_2$, $\overline{x}_1$),MUX($\overline{x}_0$, x_0, $\overline{x}_2$)), AND(XNOR(MUX(x_0, x_3, x_2), x_1),NAND($\overline{x}_2$, $\overline{x}_1$)),OR(AND($\overline{x}_2$, x_3),NOR($\overline{x}_2$, $\overline{x}_1$)))

$t_2 \Leftarrow$ XNOR(MUX(MUX($\overline{x}_0$, $\overline{x}_3$, x_2),NAND($\overline{x}_0$,AND(x_2, $\overline{x}_3$)),OR($\overline{x}_3$,NOR($\overline{x}_2$, $\overline{x}_1$))), MUX(OR(x_1, x_2), $\overline{x}_3$, $\overline{x}_0$))

$t_3 \Leftarrow$ MUX(XOR(x_0,NAND(XNOR(MUX(x_0, x_3, x_2), x_1),NAND($\overline{x}_2$, $\overline{x}_1$))), NOR(AND($\overline{x}_0$,x_1),AND(x_2, $\overline{x}_3$)),MUX(MUX($\overline{x}_0$, $\overline{x}_3$, x_2), NAND($\overline{x}_0$,AND(x_2, $\overline{x}_3$)), OR($\overline{x}_3$,NOR($\overline{x}_2$, $\overline{x}_1$))))

$t_4 \Leftarrow$ MUX(x_0,XNOR(NOR(OR($\overline{x}_1$, $\overline{x}_0$), x_2),MUX($\overline{x}_3$, x_3, $\overline{x}_2$)),NAND(x_2, $\overline{x}_1$))

$t_5 \Leftarrow$ XOR(NAND(x_2, $\overline{x}_1$),MUX(OR(XNOR(x_1, $\overline{x}_0$),NOR($\overline{x}_2$, x_3)), MUX(x_2, $\overline{x}_3$,OR($\overline{x}_1$, $\overline{x}_0$)),NAND($\overline{x}_0$, $\overline{x}_2$)))

$t_6 \Leftarrow$ AND(OR(MUX(AND($\overline{x}_0$, $\overline{x}_2$), x_0,XOR(NOR(OR($\overline{x}_1$, $\overline{x}_0$), x_2), MUX($\overline{x}_3$, x_3, $\overline{x}_2$))),NAND($\overline{x}_3$,NAND(x_2, $\overline{x}_1$))),OR(XNOR(x_1, $\overline{x}_0$), NOR($\overline{x}_2$, x_3)))

$t_7 \Leftarrow$ OR(MUX(AND($\overline{x}_0$, $\overline{x}_2$), x_0,XOR(NOR(OR($\overline{x}_1$, $\overline{x}_0$), x_2),MUX($\overline{x}_3$, x_3, $\overline{x}_2$))), NOR(XNOR(x_1, $\overline{x}_0$),MUX($\overline{x}_3$, x_3, $\overline{x}_2$)))

$t_8 \Leftarrow$ XNOR(MUX(NOR($\overline{x}_3$, $\overline{x}_1$), $\overline{x}_3$, x_2),NAND($\overline{x}_0$,NAND($\overline{x}_1$, x_2)))

$t_9 \Leftarrow$ XNOR(MUX(x_2, $\overline{x}_1$, x_3),NAND(x_0,OR($\overline{x}_1$, x_3)))

$t_{10} \Leftarrow$ NAND(NAND(NOR($\overline{x}_2$,OR($\overline{x}_1$, x_3)), $\overline{x}_0$),MUX(MUX(NOR($\overline{x}_1$, x_3), $\overline{x}_1$, NOR($\overline{x}_0$, $\overline{x}_3$)),OR($\overline{x}_0$,NAND($\overline{x}_1$, x_2)), x_2))

$t_{11} \Leftarrow$ MUX(x_3,MUX($\overline{x}_0$, $\overline{x}_3$, x_2),OR(NOR($\overline{x}_0$,NAND($\overline{x}_1$, x_2)), x_1))

$t_{12} \Leftarrow$ XOR(MUX($\overline{x}_2$,AND($\overline{x}_0$, x_1), x_3),NAND($\overline{x}_1$, x_0))

$t_{13} \Leftarrow$ MUX(OR(NOR($\overline{x}_1$, x_3),XNOR(NAND($\overline{x}_1$, x_0),MUX(x_0, NOR($\overline{x}_1$, x_3), $\overline{x}_2$))), x_3,XNOR($\overline{x}_2$,AND($\overline{x}_0$, x_1)))

$t_{14} \Leftarrow$ XOR(NOR($\overline{x}_1$,XNOR($\overline{x}_2$,AND($\overline{x}_0$, x_1))),XOR($\overline{x}_3$,MUX(x_0,NOR($\overline{x}_1$, x_3), $\overline{x}_2$)))

$t_{15} \Leftarrow$ XOR(NOR(NOR($\overline{x}_1$, x_3),NOR($\overline{x}_3$,NOR(NOR($\overline{x}_1$, x_3),XNOR(NAND($\overline{x}_1$, x_0), MUX(x_0,NOR($\overline{x}_1$, x_3), $\overline{x}_2$))))),NOR($\overline{x}_2$, x_0))

$y_0 \Leftarrow$ MUX(MUX(t_0, t_4, x_4),MUX(t_9, t_{13}, x_4), x_5)

$y_1 \Leftarrow$ MUX(MUX(t_1, t_5, x_4),MUX(t_{10}, t_{14}, x_4), x_5)

$y_2 \Leftarrow$ MUX(MUX(t_2, t_6, x_4),MUX(t_{11}, t_{15}, x_4), x_5)
$y_3 \Leftarrow$ MUX(MUX(t_3, t_7, x_4),MUX(t_{12}, t_{16}, x_4), x_5)