VLSI Design, 2002 Vol. 15 (1), pp. 455–468

Taylor & Francis
Taylor & Francis Group

Low-power Implementation of an Encryption/Decryption System with Asynchronous Techniques

NIKOS SKLAVOS*, ALEXANDROS PAPAKONSTANTINOU, SPYROS THEOHARIS and ODYSSEAS KOUFOPAVLOU

Electrical and Computer Engineering Department, VLSI Design Laboratory, University of Patras, Patras, Greece

(Received 19 February 2001; Revised 21 June 2001)

An asynchronous VLSI implementation of the International Data Encryption Algorithm (IDEA) is presented in this paper. In order to evaluate the asynchronous design a synchronous version of the algorithm was also designed. VHDL hardware description language was used in order to describe the algorithm. By using Synopsys commercial available tools the VHDL code was synthesized. After placing and routing both designs were fabricated with 0.6 μm CMOS technology. With a system clock of up to 8 MHz and a power supply of 5 V the two chips were tested and evaluated comparing with the software implementation of the IDEA algorithm. This new approach proves efficiently the lowest power consumption of the asynchronous implementation compared to the existing synchronous. Therefore, the asynchronous chip performs efficiently in Wireless Encryption Protocols and high speed networks.

Keywords: IDEA algorithm; Cryptography; Encryption decryption algorithm; Asynchronous VLSI implementation

INTRODUCTION

Most of the research and development efforts in the area of digital electronics have been oriented towards increasing the speed and the complexity of single chip digital systems. This has resulted in powerful design techniques, which enabled the development of personal workstations, sophisticated computer graphics, and multimedia capabilities. While focusing the attention on speed and area, power consumption has long been ignored. This picture is, however, undergoing some essential changes.

Low-power, yet high-throughput and computationally intensive, circuits are becoming a critical application domain. One driving factor behind this trend is the growing class of personal computing devices (portable desktops, audio- and video-based multimedia products) as well as wireless communications and imaging systems (personal communicators, smart cards) that demand high speed computations, complex functionalities and often real-time processing capabilities combined with low power consumption. Another crucial driving factor is that excessive power consumption is becoming the limiting factor in integrating more transistors on a single chip or on multiple-chip module. Unless power consumption is dramatically reduced, the resulting heat will limit the feasible packing density and performance of VLSI circuits and systems.

Furthermore, circuits with excessive power dissipation are more susceptible to run time failures and present serious reliability problems. Recent micro-processor designs, achieve impressive clocking speeds (up to 1 GHz) at the expense of very large power dissipation (larger than 30 W for CMOS, 100 W for ECL). Until now, this power consumption has not been of great concern, since large packages, cooling fins and fans have been capable of dissipating the generated heat. However, as the density and size of the chips and systems continues to increase, the difficulty in providing adequate cooling might either add significant cost to the system or provide a limit on the amount of functionality that can be provided.

Dealing with power is, therefore, rapidly becoming one of the most important issues in digital system design. This situation is aggravated by the increasing demand for portable systems in the areas of communications, general purpose computing and consumer electronics. Improvements in battery technology are easily offset by the increasing complexity of those applications: it is projected that only a 30% improvement in battery performance will be obtained over the next five years. Thus, to guarantee

*Corresponding author. E-mail: nsklavos@ee.upatras.gr

ISSN 1065-514X print/ISSN 1563-5171 online
DOI: 10.1080/1065514021000012066

a reasonable battery operation time, a dramatic reduction of the power consumption is essential.

The asynchronous CMOS circuits dissipate only when and where active, that is, any subcircuit returns to the standby mode (consuming leakage power only) whenever not in use. In addition, they favor distribution of control, leading to shallow control logic, as well as short status and control wires. We propose the asynchronous architectures since their nature favors low-power consumption. This is because in such architectures no global clock is used and because the functional units are active only when they are used. This means that asynchronous circuits achieve power down of unused units by default.

Power consumption is a parameter that has become important in VLSI designs the last years. It is one of the basic reasons that there has been recently a revival of interest in asynchronous circuits. Among the other possible advantages of asynchronous circuits (as compared to synchronous circuits) is their lower power dissipation, mainly due to the fact that there is no need to drive long clock lines at every cycle. Recently, many asynchronous circuit applications [1–5] have been presented. However, the design of asynchronous circuits is not easier than the design of synchronous ones. Designers of asynchronous systems must take special attention of several problems [6,7], such as:

- *elimination of hazards* (glitches, unexpected signal transitions),
- *race conditions* (the simultaneous change of two or more state variables caused by an internal state change, in a circuit with more than one internal state),
- *metastable states* (an unstable equilibrium (balanced) state in which a circuit can be stuck for an unbounded period of time), and
- *ordering of the system*.

The above problems forced the engineers to develop design methodologies that insure the correctness of the asynchronous circuit designs. One solution is the automatic synthesis of asynchronous logic circuits from a high-level design description. In this asynchronous circuit synthesis, an important issue is the delay model that is being assumed [6,8–15].

Delay models include:

- the *bounded-delay model*, where the delay in all circuit elements and wires is considered to be bounded (fundamental mode Huffman circuits, nonfundamental mode Huffman circuits, burst mode circuits) [1,6,12],
- the *delay-insensitive circuits*, in which the delay in both elements and wires is assumed to be unbounded [6,14],
- the *quasi-delay insensitive circuits*, where the delay-insensitive model with the addition of isochronic forks (forking wires where the delay difference between different destinations is negligible) is considered [6,15],
- the *speed-independent circuits*, that assume unbounded gate delays and negligible wire delays [6,8–10] and
- the *micropipelines*, a form of event-driven elastic pipelines (the amount of data in them may change) with or without internal processing [16].

A very important issue in any structured asynchronous design is the asynchronous interface organization. Asynchronous interface includes control (interconnection) protocol, which is the timing constraint that replaces the global clock, and data paths. The most common control protocols are the 4-phase and the 2-phase transition signaling. In the first, the control signals (Request and Acknowledge) need to return to their initial state after the communication is completed, while in the latter they do not need to. For the passing of data two common methods are the double-rail encoding, which requires two wires per data bit plus an additional acknowledgement wire, and the single-rail encoding (bundled data), which allows a single wire per data bit and requires an extra control wire for each data word.

In general terms, there are two types of key-based encryption algorithms: Symmetric and public key ones. Symmetric algorithms, which are the ones of interest here, are these algorithms in which the encryption key can be calculated from the decryption key and vice versa while in most of them both (encryption and decryption) keys are identical. Symmetric algorithms can be further divided in two other categories: The first category includes the ones that operate on the plaintext a single bit at a time and they are called stream algorithms or stream ciphers. The other category includes these algorithms that operate on the plaintext in groups of bits (usually 64, since this length is considered long enough to preclude analysis and small length to be workable) and the algorithms are called block algorithms or block ciphers [17]. The latter category will be the one of interest in this document.

INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

In general terms, there are two types of key-based encryption algorithms: Symmetric and public key ones. Symmetric algorithms, which are the ones of interest here, are these algorithms in which the encryption key can be calculated from the decryption key and vice versa while in most of them both (encryption and decryption) keys are identical. Symmetric algorithms can be further divided in two other categories: The first category includes the ones that operate on the plaintext a single bit at a time and they are called stream algorithms or stream ciphers. The other category includes these algorithms which operate on the plaintext in groups of bits (usually 64, since this length is considered long enough to preclude analysis and small length to be workable) and the algorithms are called block algorithms or block ciphers. The latter category will be the one of interest in this document.

The block cipher IDEA was first presented by Xuejia Lai and James Massey of the Swiss Federal Institute of Technology in 1990 and was then called PES (Proposed Encryption Standard) [18]. In 1991 after Biham and Shamir presented their results regarding differential cryptanalysis, the authors developed an improved version of the PES algorithm to increase the security against this attack and the new algorithm was called IPES [19] (Improved Proposed Encryption Standard) while finally in 1992 its name was changed officially to IDEA. The algorithm is based on some impressive theoretical foundations and although several academic and military groups have extensively cryptanalyzed IDEA, none of them has announced any successes, with the exception of some progress against reduced round variants. Anyway, the algorithm still seems strong and to most experts opinion [17] it is the best and most secure block algorithm available to the public at this time.

The algorithm most possibly is expected to be used in the future as a replacement of DES and in the meantime is being used in many important applications, such as in r^3TFT (Trusted File Transfer) and PGP (Pretty Good Privacy). It is also registered at the UN/EDIFACT, thus, can be used in the implementation of standard Electronic Data Interchange applications. Regarding the licensing scheme, IDEA is patented in Europe and the United States and the patent is held by Ascom-tech AG. The use of the algorithm obtained in freeware and used in non-commercial environment do not require a license (but even non-commercial applications still require the explicit permission) while commercial applications are subject to a license agreement of the type of a site, product or end-user license.

Overview-algorithm Design Principles

The IDEA is a block oriented encryption algorithm, which operates on a 64-bit plaintext and uses a 128 bit length key. The design philosophy of this algorithm is based on the concept "of mixing operations from different algebraic groups". The substitution boxes and the associated "look-up tables" used in the rest block ciphers available to-date (and among them DES) have been completely dispensed with. The required confusion in this algorithm (dependence of the ciphertext on the plaintext and the key in a complicated and involved way—to be explained better in a later section) is achieved by successively using three different and "incompatible group operations on pairs of 16-bit subblocks and mixing them (in such a way that at no point in the encryption process the same algebraic operation is used contiguously) while the structure of the cipher was carefully chosen to provide the necessary diffusion requirement (influence of each key and plaintext bit on every ciphertext bit-to be better explained in a later section)." These three algebraic operations are the following:

- Bit-by bit XOR (denoted as $\oplus$)
- Addition of integers modulo 2^{16} (denoted as $\otimes$) with inputs and outputs treated as unsigned 16-bit integers
- Multiplication of integers modulo 2^{16} (denoted as $\square$) with inputs and outputs treated as unsigned 16-bit integers (This operation can be also viewed as IDEA's equivalent S-box.)

All these operations (and these are the only operations in the algorithm-there are no bit level permutations as e.g. in DES) operate on 16-bit subblocks. Their use (of these three separate operations) in combination provides for a complex transformation of the input making cryptanalysis (as it will be shown in later sections) much more difficult than with an algorithm such as e.g. DES, which relies solely on the XOR function. Regarding the encryption/decryption procedures, the algorithm's structure has been chosen that, with the exception of the fact that different key subblocks are used, the encryption process consists of eight identical (encryption) steps (known as encryption rounds) followed by an output transformation, while also the decryption process is identical to the encryption procedure once the decryption key subblocks have been computed from the encryption ones. More details will be given in the next section.

Analytical Description of Idea Input/Output Procedure

The 64-bit data block of the input plaintext is divided into four 16-bit sub-blocks: X_1, X_2, X_3 and X_4. These four sub-blocks become the necessary input to the first round (iteration) of the algorithm. As explained before, there are eight rounds in the algorithm in total. Each iteration (round) takes four 16-bit sub-blocks as inputs and produces four 16-bit output blocks. In each round, the four sub-blocks are XORed, added and multiplied with one another and with six 16-bit sub-keys. Between rounds, the second and third sub-blocks are swapped. Finally, the four sub-blocks are combined with four subkeys in an output transformation, producing 16-bit output blocks which are concatenated to form the 64-bit ciphertext.

Iterations Description

The first round of IDEA is depicted in Fig. 1. All iterations have the same structure but with different subkey and plaintext derived inputs. As shown in the figure each round begins with a transformation that combines the four input sub blocks with four subkeys using the (modulo) addition and multiplication operations. This transformation is shown at the upper shaded rectangle of the figure. The four output blocks of this transformation are then combined using the XOR operation to form two 16-bit blocks that are input to the lower dashed rectangle (known also as MA structure), which also takes two subkeys as input and combines these inputs to produce two 16-bit outputs.

Finally, the four output blocks from the upper transformation (the upper shaded rectangle in the figure) are combined with the two output blocks from the MA

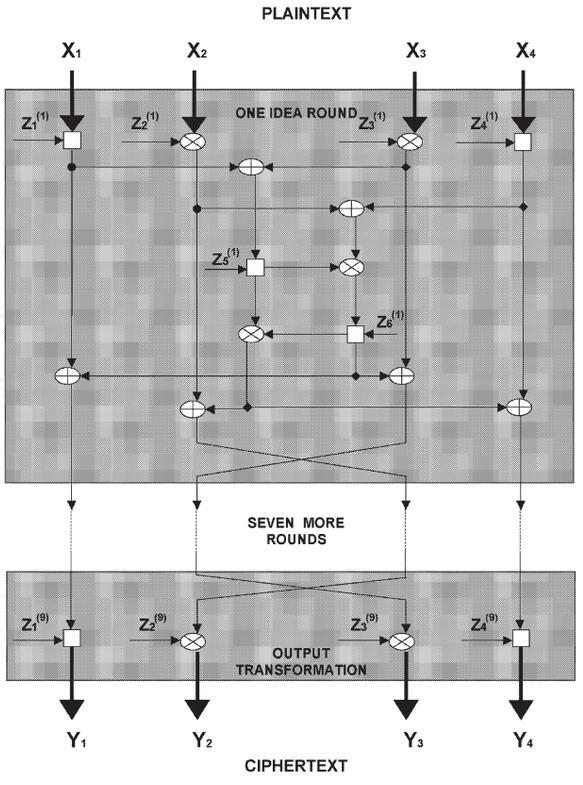


FIGURE 1 IDEA algorithm architecture.

structure (the lower shaded rectangle) using XOR to produce the four output blocks $X_1^{(r+1)}$, $X_2^{(r+1)}$, $X_3^{(r+1)}$, $X_4^{(r+1)}$ of the specific (the rth) round. It should be noted that the two outputs that are partially generated by the second and third inputs ($X_2^{(r)}$ and $X_3^{(r)}$) are interchanged to produce the second and third outputs ($X_2^{(r+1)}$, $X_3^{(r+1)}$) of this round. This interchanging increases the mixing of the bits being processed and makes the algorithm more resistant to (differential) cryptanalysis and it should be mentioned that this simple change was the modification that Lai and Massey performed in IDEA's ancestor which transformed it (i.e. the PES algorithm) to the IDEA algorithm as we know it.

Output Transformation

The process described above for one algorithm iteration is repeated in all eight rounds of IDEA using different 16-bit key for each iteration. Following the eighth round there is a final output transformation stage. This has the same structure as the upper shaded rectangle of Fig. 1. The only difference lies in the fact that the second and third inputs (i.e. $X_2^{(8)}$ and $X_3^{(8)}$) are interchanged (once again) before being applied to the (modulo) addition and multiplication operational units. This has the effect of undoing the interchange at the end of the eighth iteration. This extra interchange gives to the decryption procedure the same structure as the encryption as it can be proven [18,19].

Sub Key Generation-decryption

The 52 subkeys required for the whole execution of the algorithm (six subkeys for each of the eight rounds and four more for the output transformation) can be easily generated by just a series of (simple) shifts on the original 128-bit key. More specifically, the 128-bit (original) key is divided into eight 16-bit subkeys. These are the first eight subkeys for the algorithm—the six of them used in the first round and the rest two in the second one. Then the key is rotated 25 bits to the left and again divided into eight subkeys the first four of which are used in round 2 and the last four in round 3 and so on until the end of algorithm's execution.

Regarding the decryption procedure, this is exactly the same, except for the fact that the subkeys are reversed and are slightly different. More specifically, the decryption subkeys are generated from either the additive or multiplicative inverses of the encryption subkeys.(for more details see Refs. [17,19]). The calculation of them needs certainly some effort but this has only to be done once for each decryption key.

SYSTEM ARCHITECTURE

Two different ASIC designs were fabricated in this work in order to implement the IDEA algorithm in hardware. These two designs are similar in their functionality but different clocking techniques were used in each of them. The first design is synchronous and the second asynchronous.

IDEA Encryption/Decryption Board

The system is a powerful encryption/decryption board, utilizing a microprocessor, memory, communication resources and the IDEA ASIC. This architecture defines most of the interface characteristics of the IDEA ASIC. However, as the ASICs will be used in other systems beyond this certain board, this interface is more generic, to cover many possible system configurations.

The interfacing of the IDEA ASICs with the system which host them is a typical slave-device interface. Some more specific interfacing characteristics are:

- The two ASICs have completely identical interfaces in terms of pinout, timing, power supply, clocking etc. Since it is possible that some of the interface pins for the synchronous and asynchronous chips will differ, the package have the superset of both pinouts for the two variants. The replacement of a synchronous chip with an asynchronous one not requires any PCB modifications or software changes. Only small modifications that can be realized through the appropriate setting of a DIP switch is required.
- The interface is realized entirely as a PI-bus compliant slave. All control and data signals to the IDEA core is

passed as commands through this interface. The chips connect directly on the bus through ADDRESS, DATA, OPCODE, READ/WRITE and ACKNOWLEDGE SIGNALS.
- No other connection between the ASICs and the system is realized, i.e. there is no FIFOs, no control, no parallel buses, no interrupt sources etc.). This ensure the reusability of the ASIC, a very important feature as the chips can be used in various system configurations.

In Fig. 2, the encryption/decryption system is illustrated. The various elements of the system are:

- The mircoprocessor. A fast processor is required to handle all system-level operations at speeds up to 25 Mbits/s. For this reason, we have decided to use a 486-class controller (NS486SXF).
- The IDEA ASIC, which includes the IDEA core and the bus, interface functions.
- The FLASH memory, which contains the program code (firmware). This memory is located on the processor's local bus.
- The shared memory. A RAM unit that is linked to the bus through a programmable MMU. This RAM hosts all plaintext and ciphertext data.
- A Serial I/O port, which will operate at standard 9600 bps. This port is used to interface the system with a PC. The protocol for this interface is based on simple packets of data that consist of header information and a variable length payload and is realized in firmware.
- The socket for bus extension. This socket is used as a high-speed interface to the system. It is able to transfer data at speeds up to 64 Mbits/s at a 4 MHz clock. External systems must contain bus interface logic to exploit this speed.
- The bus. It is a variant of the PI-bus. The BCU, and the two bus interface units (for the processor and the serial port) is implemented in a programmable logic device (FPGA).

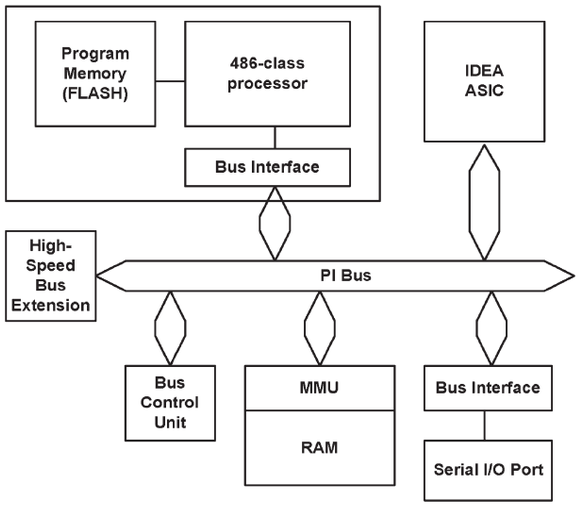


FIGURE 2 System architecture.

IDEA ASIC Architecture

The ASIC is a slave chip in a encryption/decryption system of a common bus architecture. A master device, such as microprocessor is required for the operation of this slave circuit. The microprocessor can take the leadership of the shared bus function and can initialize the communication with the slave circuit (IDEA ASIC). The processor can also send request signals to the device it has decided to communicate. In our case the IDEA ASIC can take requests and data from the microprocessor such as:

- Microprocessor commands
- Subblock Key
- Subblock of plaintext/ciphertext data
- Request for the operation mode of ASIC
- Read write request to the device registers

The initialization procedure of certain signals with predefined time events is called bus communication protocol. The handshake protocol for our system is showed in Fig. 3. The diagram of this figure reports a read and a write operation after the initialization procedure of the ASIC.

Such the read as the write operation needs two cycles of the system clock. The first clock's cycle is called address cycle and the second data cycle. The communication protocol demands some certain signals events both from the microprocessor that sends the requests, and the interface, that performs to the processor's commands.

The IDEA ASICs both have the architecture shown in Fig. 4. The ASICs include:

- The Interface Unit. A standard slave interface for the PI-bus. It can serve read (and write) requests from (and to) the internal chip registers.
- The IDEA Core Unit. This unit is different in the Asynchronous and the Synchronous versions of the ASIC. It implements the IDEA core algorithm.

Interface Unit

The Interface Unit (Fig. 5) is used in order to transfer all the data and control commands between the Core Unit and the microprocessor. The communication interface between the Interface Unit and the Core Unit (number and type of signals) is almost the same for the two versions (synchronous and asynchronous).

The interface unit of the ASIC serves read (and write) requests from (and to) the internal chip registers. It also has the responsibility for the data to be transported without errors, between the IDEA core unit and the microprocessor. In other words the interface is the intermediate

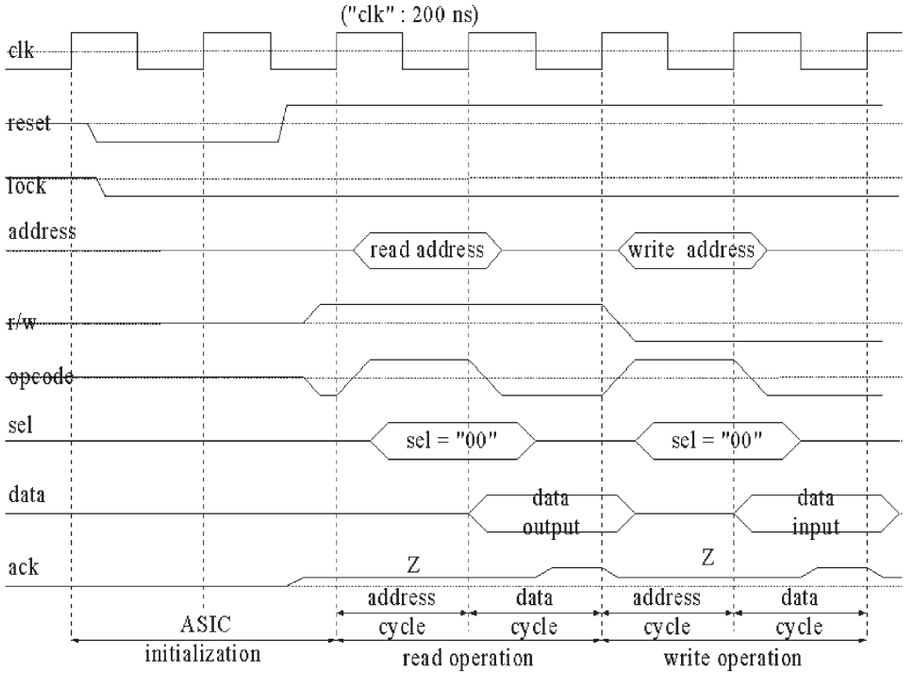


FIGURE 3 Handshake protocol.

buffer unit between the microprocessor and the IDEA core. In the asynchronous version of the system the interface performs an extra function. It is an intermediate stage between the synchronous way that the bus operates and the asynchronous way that the IDEA core operates.

The interface Unit is divided in two major parts: The Bus Slave part and the IDEA-Buffer part. The Bus Slave part communicates with the system bus (PI Bus) and includes registers which store the:

- Encryption/decryption keys
- Data (from the microprocessor or the IDEA Core)
- Status Registers
- Command Registers

It operates based on the PI Bus system clock. So, both the synchronous and asynchronous versions of the Bus Slave are operate synchronous.

The IDEA Buffer part communicates with the IDEA Core. It is used for temporary store of the encrypted/decrypted data. It comprises two FIFOs for the data store, one for each direction. The implementation of the IDEA Buffer has major differences between the two IDEA ASIC versions because (1) this part does not use clock in the asynchronous version and (2) the communication protocol

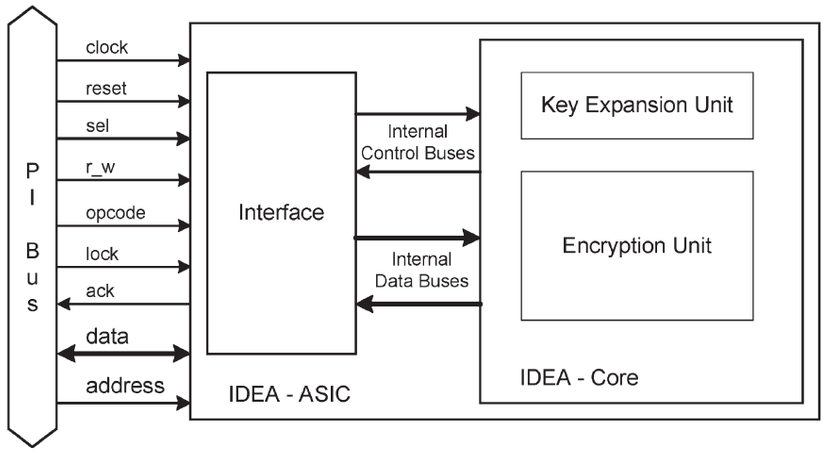


FIGURE 4 Block structure of the IDEA ICs.

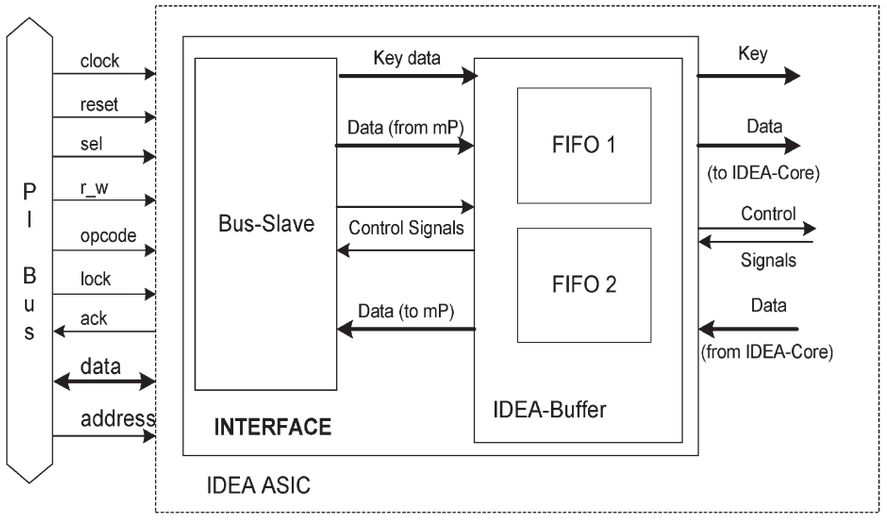


FIGURE 5 Interface unit block diagram.

between the Interface Unit and the IDEA Core is different in the two versions.

IDEA Core

The IDEA core is the most important unit in the IDEA ASICs. It is implemented in two versions (one that uses clock-synchronized processing and one that is based on state-of-the-art asynchronous techniques for reduced power consumption), resulting in two variants of the IDEA chip: Synchronous (Fig. 6) and Asynchronous (Fig. 7).

The architecture consists of the five rounds (four normal IDEA rounds and the output transformation) and a feedback loop. The IDEA rounds all use encryption subkeys generated from the user-defined 128-bit key.

ASYNCHRONOUS VLSI DESIGN

Two implementations of the encryption/decryption system have been designed. Although these two versions performs the same operation there are basic differences between them. This is due to the fact that the asynchronous version does not based on the system clock operation. Some extra elements have been designed especially for this version, in order to avoid errors in the data transport and to control the read/write requests from and to the IDEA Core.

Asynchronous Control Units

In the asynchronous version special control units have been designed in order to replace control operations based on clock, which the synchronous implementation uses. The main demand of the asynchronous control units is to avoid their non-deterministic behavior. In order to achieve this, asynchronous circuits must be hazard-free under circuit delay model. Certain forms of MIC behavior can be tolerated. The most general signal concurrency must be controlled by arbitration in order to avoid circuit hazards. For example, if some circuit is to react one way if it sees a transition on signal A and react differently for a transition on signal B, then some guarantee must be provided differently for a transition on inputs A and B. Nondeterministic behavior will occur if this guarantee cannot be provided. Such mutually exclusive signal conditioning is usually provided by arbitration [20].

C Element

C Element is a component that can be used for arbitration. The C Element, showed in Fig. 8a, is a commonly used asynchronous circuit component. The use of this element prevents another pending request from passing through the arbiter until after the active request cycle has cleared. C Element is a simple combinational logic implementation of a 2-input circuit. The component produces an output of 1 only after all of its inputs are 1, and produces an output 0 only when all of its inputs are 0. These values showed in details in Fig. 8b,c. For all other inputs, the component holds its input value. A C Element serves as a 4-cycle protocol preserving rendezvous.

VHDL Implementation of the C Element

Digital Electronics systems are increasing exponentially in their complexity over time. This fact, coupled with decreasing product lifetimes and increasing reliability requirements, has forced designers to dramatically increase their productivity and the quality of their designs.

VHDL was developed in response to these trends. Borrowing the complexity management and error detection techniques from the software engineering world. VHDL was developed to eliminate irrelevant detail, allow technology-independent description, catch errors earlier

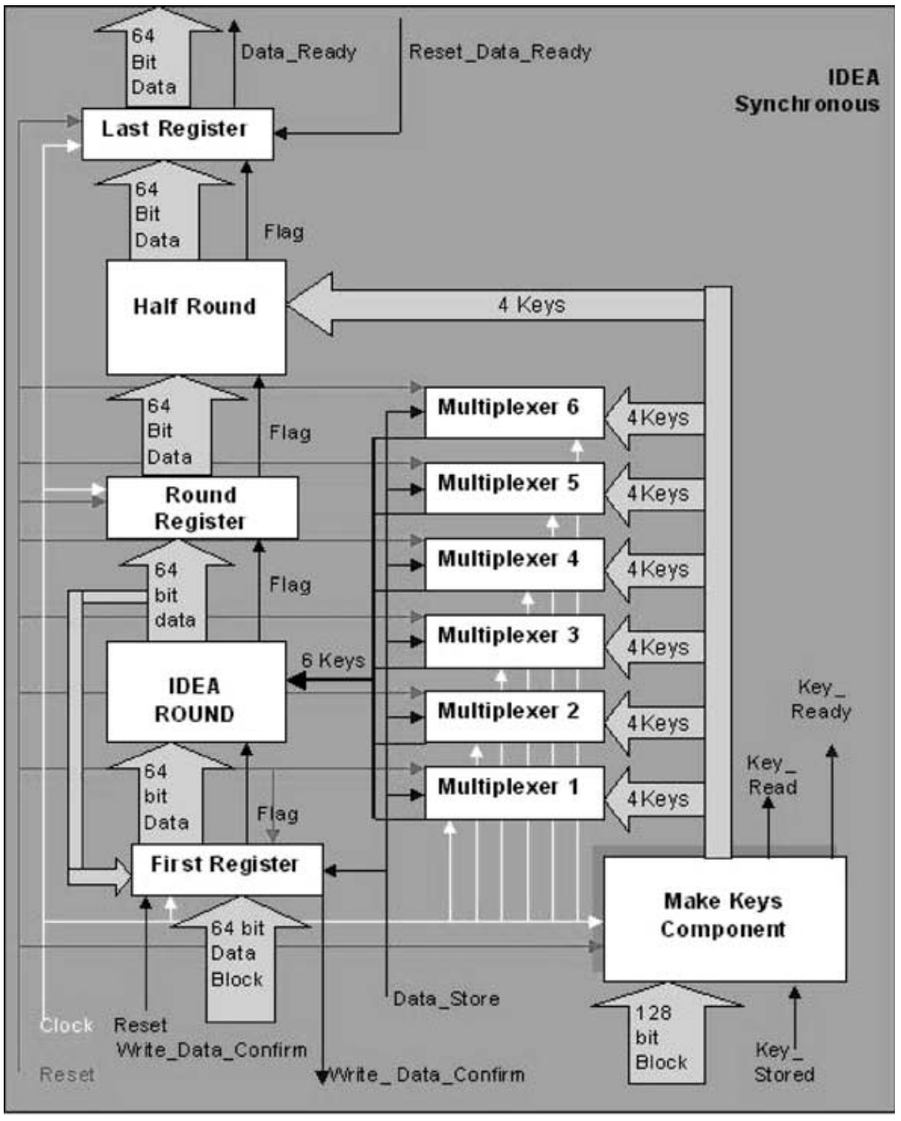


FIGURE 6 IDEA core block diagram (synchronous version).

and promote portable and interoperable models from the gate to the system level.

An alternative way of describing the implementation of an entity is to specify how is composed of subsystems. We can give a structural description of the entity of the C Element with the VHDL code that follows, with declarations and architecture bodies for the subsystems.

VHDL Code for C Element:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY c_element IS PORT
   (reset: IN std_ulogic;
   C_element_input_A: IN std_ulogic;
   C_element_input_B: IN std_ulogic;
   C_element_output_C: INOUT std_ulogic);
END c_element;

ARCHITECTURE structural OF C_element IS
COMPONENT and2
PORT (and2_in1, and_in2:IN std_ulogic;
        and2_out:OUT std_ulogic);
END COMPONENT;

COMPONENT or3
PORT (or3_in1, or3_in2, or3_in3:IN std_ulogic;
Or3_out:OUT std_ulogic);
END COMPONENT;

SIGNAL s1, s2, s3,s4: std_ulogic;

BEGIN
```

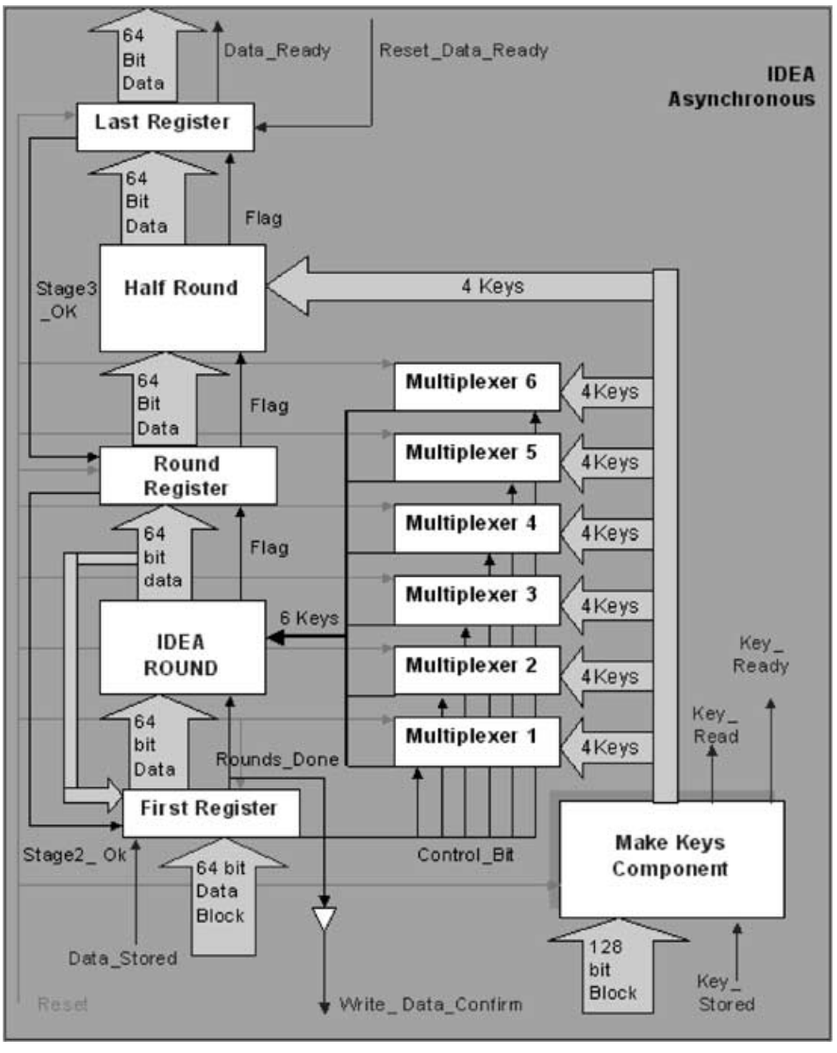


FIGURE 7 IDEA core block diagram (asynchronous version).

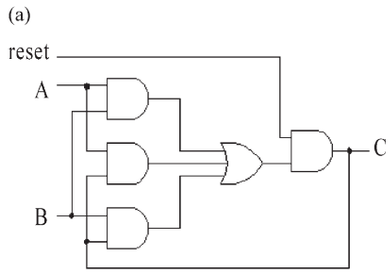
And2_1: and2 port map (C_element_input_A,C_element_input_B,s1);
And2_2: and2 port map (C_element_input_A,C_element_output_C,s2);
And2_3: and2 port map (C_element_input_B,C_element_output_C,s3);
And2_4: and2 port map (s4,reset,C_element_output_C);
Or3_1: or3 port map (s1, s2, s3, s4);
END structural;

The VHDL architecture body declaration describes the structure shown in Fig. 8. The signal declaration, before the keyword begin, defines internal signals of the architecture. Within the architecture body the ports of the entity are also treated as signals. In the other part of the architecture body, a number of component instances are created, representing the subsystems from which the C Element entity is composed. Its component instance is a copy of the entity representing the subsystem, using the corresponding basic architecture body. The port map specifies the connection of the ports of each component instance to signals within the enclosing architecture body.

Delay Elements

Except the C Element, very important units for the asynchronous implementation are the delays elements. The function of such an element is the attribution of a signal value after a predefined time delay. The schematic of such a delay unit is showed in Fig. 9.

A buffer or an inverter has a certain delay for the transport operation of its input signal value to the output. This delay time depends on the technology that is used. If we combined a certain number of inverters or buffers in serial architecture we would archive a longer delay period



(b)

reset	A	B	C
0	X	X	0
1	0	0	0
1	0	1	stable
1	1	0	stable
1	1	1	1

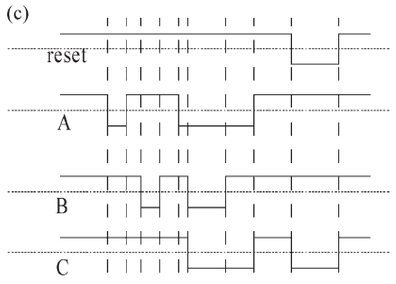


FIGURE 8 (a) A two input C Element. (b) Truth table of the C Element. (c) Waveform of C Element's operation.

for the total unit. The delay time of its buffer (T_d), that is used, can be added and so we can compute the total delay time of the unit. In this way we design units with certain delay time.

$$\text{Total}_{\text{Delay Time}} = T_{d1} + \cdots + T_{dn}$$

CHIP CHARACTERISTICS

The characteristics of the two ASIC chips are shown in Table I.

The two designs were fabricated with 0.6 μm CMOS technology. The system clock is up to 8 MHz and the power supply is 4.5–5.5 V. Although there is a difference in the effective area of the two implementations the type package is the same. A basic aim was to keep the same interface for both implementations. This is due to the fact that we did not want the replacement of a synchronous chip with a asynchronous one to require any PCB modifications or software changes. In order to achieve our goal, we use the same package type.

The layouts of the two chips are shown in Figs. 10 and 11. The DIL 40 (40 pins) package type was used. Five of these 40 pins are dedicated for the power supply and five more for the ground. The databus uses 16 pins, while 6 pins are used for the address bus. One pin is needed for every control signal of the encryption/decryption system. These signals are: Reset, Lock, Ack, Opcode, R_W and Clock. Of course in the asynchronous version, as it has been mentioned above, the clock signal is used only for a small part of the ASIC interface and not in the IDEA Core. The last two pins of the package are used for the select signals. From all these pins only the 16 pins of the databus are bi-directional, while the rest of them are used to send requests and microprocessor's commands to the ASIC (uni-directional).

Both implementations operate with 5 MHz frequency. This value produces a period clock of 200 ns. The four rounds and the last transformation of IDEA's implementation need $5(4+1)\text{rounds} \times 200\,\text{ns} = 1000$ ns/data block. In other words, the ASIC produces 64 bits every 1000 ns, so the throughput is 64 Mbits/s.

CHIP TESTING

During the testing procedure, a specific configuration was used which is presented in this section. More specifically, the ASIC samples under examination were placed in a test board, which is shown in Fig. 12. This board includes an ALTERA FPGA (FPGA Type: Flex EPF8636ALE), which is positioned between the IDEA ASIC and the rest board

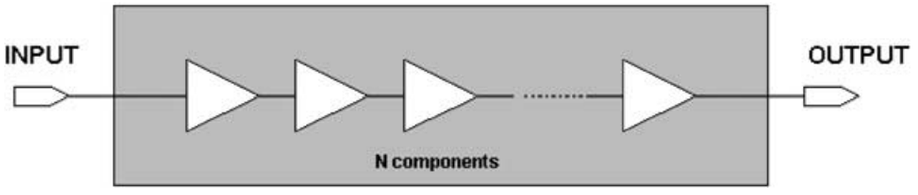


FIGURE 9 Delay unit architecture.

TABLE I Chip characteristics of the IDEA synchronous/asynchronous chips

Code	Parameter	Value (synchronous/asynchronous)	Unit
N I/O	Number of I/O pins	40	Pins
Package	ASIC Package	DIL40	Package type
X	Length	8920/8431	um
Y	Height	5990/5932	um
Area	Chip effective area	53,43/50,01	mm^2
Ngates	Equivalent gates	47555/44708	2 input NAND
Ntran	Transistor Count	190223/178835	Transistors
VDD	Power supply	4.5–5.5	Volt
F	Operation frequency	Up to 8	MHz

components (μP, memories etc.), the role of which is through the use of a host PC to be programmed appropriately, so as to be able to create real operation conditions of the system. This is achieved through the help of the MAX + plus programming environment of ALTERA, which creates the required VHDL, code and a description of the whole structure, using an executable graphics file to be analyzed below. The board has furthermore a number of pins, which are connected through an appropriate connector to the parallel port of the host PC through which the programming data (FPGA's bitstream) are downloaded onto the board for programming accordingly the ALTERA IC.

Board Programming Procedure

Initially the required code in VHDL was created that would permit emulation of the IDEA ASIC operation. In order to be able to read/write data to/from the IDEA ASIC, (through the data bus) it was required to use the tristate behavior of the corresponding input/output data of the test port which was achieved by the relevant VHDL code used for simulating the ASIC's environment. For this reason, a graphical executable file was created. The values of the input/output signals were monitored through the help of a logic analyzer, which was connected to the whole board structure.

Test Scenarios

In this section the test scenarios that were applied to check the ASIC's correct functionality is analyzed. More specifically, the test procedure followed can be described in the following steps:

- After the initialisation phase (activation of the "active low" reset) a byte is written at the address position 16 (the *ID_reg* register) followed by a read from the same address.
- The following step is to write 8 words of 16 bits at the addresses 01-08 and the command "000" is given to the *status*_1 register ("0001" is written at the address 15 (Hex)
- The *status0* register is read [at the address 14 (Hex)] the contents of which should be zero
- 4 16-bits words with plaintext are written at the addresses 0C-09 (at the plaintext registers) and then the command "0002" is given to the *status1* register

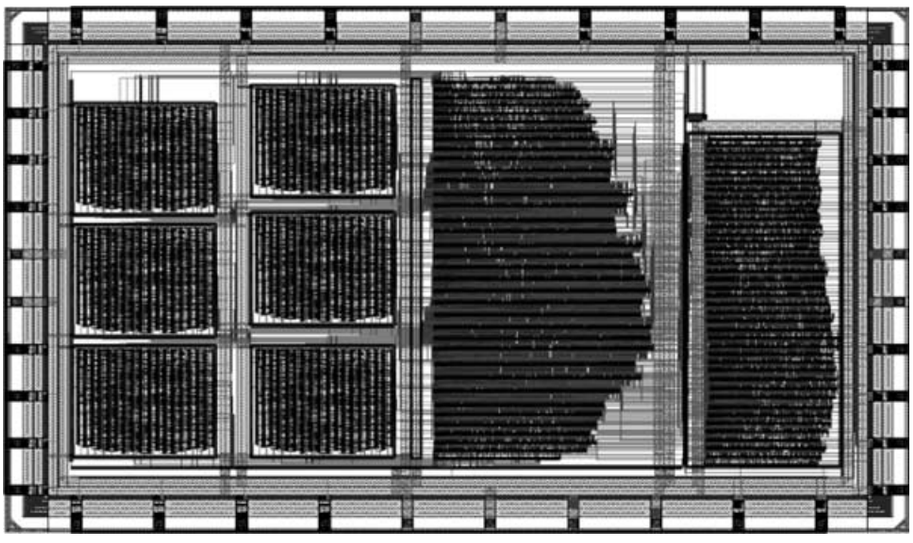


FIGURE 10 Synchronous IDEA chip layout.

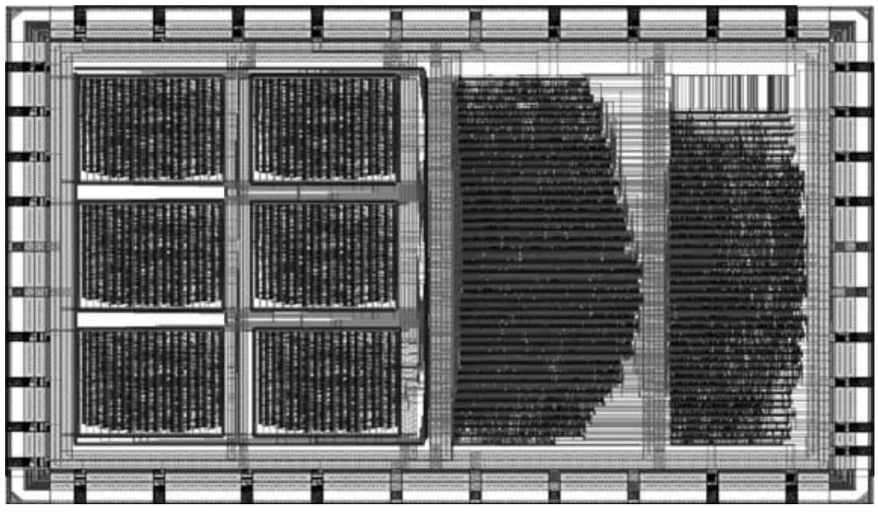


FIGURE 11 Asynchronous IDEA chip layout.

- The previous two steps could be repeated three times more in order to enter (into the ASICs) four plaintext blocks in total
- When the content of the *status0* register is no zero anymore, then 4 16-bit words (corresponding to the produced ciphertext) can be read from the addresses 10-0D (i.e. from the ciphertext registers) and then the command "0004" is given to the status1 register (address 15 in Hex) so as to inform it that the first ciphertext block was read

The previous step could be repeated until the FIFO containing the produced ciphertext data empties. This can be achieved by checking the content of the *status0* register.

Test Vectors

During the test procedure a number of test vectors were used to verify the correct operation of the received synchronous and asynchronous ASIC samples. These test vectors were mostly selected in a random way, but there have also been included some special test vectors (like "FFFF" and "0000") to ensure maximum test coverage.

Power Consumption

The most important difference between the two implementations is the lower power consumption of the asynchronous one. The testing procedure gives us results of this consumption. The values of the total power dissipation showed in the Table II. While there are three different scenarios in the power measurement operation the conclusion is still the same for all of them.

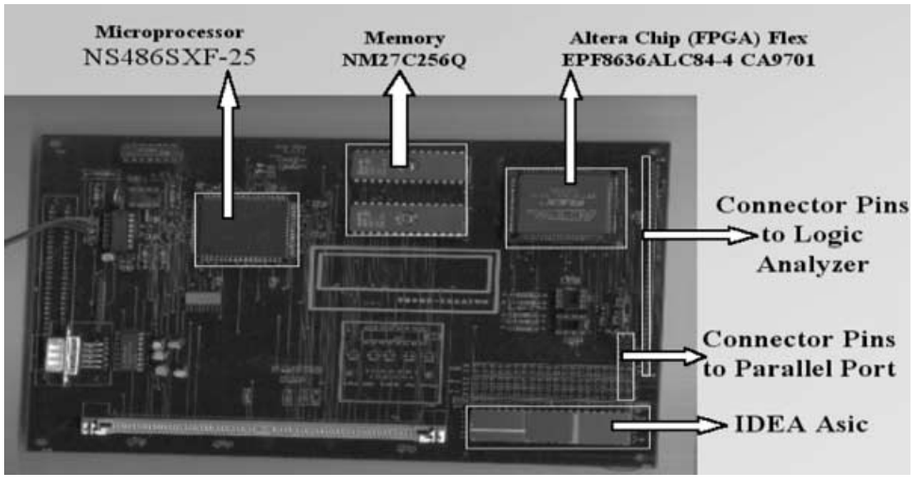


FIGURE 12 PCB for the IDEA chips testing.

TABLE II Power Consumption of the IDEA synchronous/asynchronous Chip

Scenario	Supply curent (mA) (synchronous/asynchronous)	Total power disspation (mW) (synchronous/asynchronous)
Standby	7.51/4.25	37.55/21.25
One plain/cipher continuously	10.40/8.40	52.00/42.00
Three plain/cipher continuously	11.60/8.25	58.00/41.25

Asynchronous implementation has the lower power consumption for every operation mode of the ASIC. The difference of the values is in percentage units about 20–40 %. This proves that the asynchronous version is more useful and performs better in terms of power consumption than the synchronous.

CONCLUSIONS

A very high speed, low power VLSI block encryption system has been designed. The choice of IDEA as the encryption/decryption algorithm ensures the strength of the data encryption operation. Two implementations of the system have been presented. The only know fast single chip implementation is [21]. This chip has a power consumption of the 1.25 W, while our synchronous version has a power consumption of 58 mW and our asynchronous 41.25 mW in the worst cases of operation. Although our synchronous design has a low power dissipation, the asynchronous one has significantly very low power consumption. With the second implementation (asynchronous) of the encryption/decryption system the total power dissipation is decreased at about 20–40% in percentage units. This integrated circuit can be applied as a very fast and low power encryption/decryption device in high speed networks.

Acknowledgements

Supported by the ECC under ESPRIT PROJECT 25249.

References

[1] van Berkel, K., Burgess, R., Kessels, J.L.W., Peters, A., Roncken, M. and Schalij, F. (1994) "A fully asynchronous low-power error corrector for the DCC player", *IEEE Journal of Solid-State Circuits* **29**(12), 1429–1439.
[2] Bruvard, E., Furber, S., Nanya, T., eds, (1996) IEE Proceedings—Computers and Digital Techniques **143**, (5).
[3] Jacobs, G.M. and Brodersen, R.W. (1990) "A fully asynchronous digital signal processor using self-timed circuits", *IEEE Journal of Solid-State Circuits* **25**(6), 1526–1537.
[4] Nowick, S.M. (1996) "Design of a low-latency asynchronous adder using speculative completion", *IEE Proceedings—Computers and Digital Techniques* **143**(5), 301–307.
[5] M.J. Wiener. "Efficient DES key Search" presented at Crypto'93, August 1993.
[6] Hauck, S. (1995) "Asynchronous design methodologies: an overview", *Proceedings of the IEEE* **83**, 69–93.
[7] Unger,, S.H. (1995) "Hazards, critical races, and metastability", *IEEE Transactions on Computers* **44**(6), 754–768.
[8] Beerel, P.A. and Meng, T.H.-H. (1992) "Automatic gate-level synthesis of speed-independent circuits", Proceedings of the International Conference on Computer Aided Design (ICCAD), pp 581–586.
[9] Beerel, P.A., Hsieh, C.T. and Wadekar, S. (1996) "Estimation of energy consumption in speed-independent control circuits", *IEEE Transactions on CAD of Integrated Circuits and Systems* **15**(6), 672–680.
[10] Jung, S.T. and Jhon, C.S. (1994) "Direct synthesis of efficient speed-independent circuits from deterministic signal transition graphs", Proceedings of the International Symposium on Circuits and Systems (ISCAS), pp 307–310.
[11] Jung, S.T., Park, E.S., Kim, J.S. and Jhon, C.S. (1995) "Automatic synthesis of speed-independent circuits from signal transition graphs", Proceedings of the International Symposium on Circuits and Systems (ISCAS), pp 1211–1214.
[12] Kudva, P., Gopalakrishnan, G. and Jacobson, H. (1996) "A technique for synthesizing distributed burst-mode circuits", Proceedings of the Design Automation Conference (DAC).
[13] Lavagno, L., Keutzer, K. and Sangiovanni-Vincentelli, L. (1995) "Synthesis of hazard-free asynchronous circuits with bounded wire delays", *IEEE Transactions on CAD of Integrated Circuits and Systems* **14**(1), 61–86.
[14] Leung, S.C. and Li, H.F. (1995) "On the realizability and synthesis of delay-insensitive behaviors", *IEEE Transactions on Integrated Circuits and Systems* **14**(7), 833–848.
[15] Molina, P.A., Cheung, P.Y.K. and Bormann, D.S. (1996) "Quasi delay-insensitive bus for fully asynchronous systems", Proceedings of the International Symposium on Circuits and Systems (ISCAS), pp 189–192.
[16] Sutherland, I.E. (1989) "Micropipelines", *Commun. ACM* **32**(6), 720–738, Turing Award Lecture.
[17] "Applied Cryptography", Bruce Schneier, 1996, Wiley editions.
[18] Lai X. and Massey J.L. "A proposal for a new Block Encryption Standard". *Proceedings of Eurocrypt'90*, Aarhus, Denmark, May 21–24 1990, pp. 389–404.
[19] Lai, X., Massey, J.L. and Murphy, S. (1991) "Marcov Ciphers and differential Cryptanalysis", *Advances in Cryptology-Eurocrypt*, 17–38.
[20] Davis, Al. and Nowick, Steven M. (1995) "Asynchronous circuit design: movitation, background, and methods", Asynchronous Digital Circuit Design (Springer, Berlin), Workshops in Computing, pp 1–49.
[21] Zimmermann, R., Curiger, A., Bonneberg, H., Kaeslin, H., Felber, N. and Fichtner, W. (1994) "A 177 Mb/s VLSI implementation of the international data encryption algorithm", *IEEE Journal of Solid States Circuits* **29**(3).

Nikos Sklavos received the Diploma of Electrical and Computer Engineering from the University of Patras, Greece, in 2000. He is currently pursuing the Ph.D. degree at the department of Electrical and Computer Engineering, University of Patras, Greece. His research includes VLSI and Low Power Design, cryptography implementations for wireless communications in circuit and architecture level and reconfigurable computing architectures. He has published technical papers in the sectors of his research.

Alexandros Papakonstantinou received the Diploma of Electrical and Computer Engineering from the University of Patras, Greece, in 1999, and the MSc degree in Analogue and Digital Integrated Circuit Design from

Imperial College, London, in 2000. He became a Ph.D. student at Patras University, in 2000 where he is currently carrying out his research on Microprocessor Design. He has been also actively cooperating with Infineon Technologies on the design of Floating Point Units and datapath blocks for embedded microprocessors. His interests include computer arithmetic, computer architecture and VLSI design.

Spyros Theoharis received his Diploma in Computer Engineering and Informatics from the University of Patras, Greece, in 1994. He received the Ph.D. degree at department of Electrical and Computer Engineering, University of Patras, in 2000. His research interests include low power design, multilevel logic synthesis, parallel architectures, and power estimation. Dr Theoharis has published many technical papers in the sectors of his research.

Odysseas Koufopavlou received the Diploma of Electrical Engineering in 1983 and the Ph.D. degree in Electrical Engineering in 1990, both from University of Patras, Greece. From 1990 to 1994 he was at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, University of Patras, Greece. His research interests include VLSI, low power design, and high performance communication subsystems architecture and implementation. Dr Koufopavlou has published more than 65 technical papers and received patents and inventions in these areas.