UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

BACHELOR’S THESIS no. 95

Multicore Implementation of Crypto-Algorithm Using Network-on-Chip

Mihaela Bakšić

Zagreb, April 2023.

I want to express gratitude to my parents for always being my biggest support and my mentor doc. dr. sc. Daniel Hofman for his advisement.

CONTENTS

1. Introduction

In the light of rapid digitalisation of data and information, the security of these has become more critical and complex than ever. Symmetric encryption plays a significant role in providing data security. One of the main events that has determined the development of symmetric encryption have been continuous breaches of the Data Encryption Standard (DES) algorithm. It has spurred the need for a new standard that would be able to resist future breaking attempts and account for the increasing computing power of the attacker.
In 1997 The National Institute of Standards and Technology (NIST) issued a call for an encryption algorithm to become The Advanced Encryption Standard (AES). The criteria for AES selection were security, cost and algorithm and implementation characteristics. After lenghty assessment of submissions, The Rijndael algorithm was selected as AES, whilst The Serpent algorithm came in second.

Understanding how prevalent encryption and its various applications are, the question that emerges is how to implement encryption algorithms in circumstances that are not tightly coupled with a conventional computer or server. Hardware implements on technologies such as smart cards, microcontrollers and Field Programmable Gate Arrays (FPGA) allow for such conditions.

The Serpent algorithm was presented as "a flexible block cipher with maximum assurance"[8] that adheres to conservative encryption algorithm design principles. Its designers Ross Anderson, Eli Biham and Lars Knudsen were especially confident in the algorithm's theoretical background allowing for efficient hardware implementation, which makes the Serpent algorithm an interesting case.

In order to provide implementation of crypto-algorithms of sufficient performance, multicore execution is inevitable. A novel achitectural approach of Network-on-Chip sets a new trend in developing multicore designs on chip. It introduces well known principles of network design into hardware architectures.

The aim of this thesis is to present the developed hardware implementation of the Serpent algorithm, using FPGA technology and the concept of Network on Chip (NoC). The following chapters thoroughly describe the technologies used and explain the Serpent crypto-algorithm.

2. Network-on-Chip

With the end of Moore's Law, applications can no longer exploit increasing number of transistors placed on a die to gain performance. Instead, the concept of multi-core processing has become a main source of performance gain.

Multi-element processing on a chip has called for a change in the way the elements on a chip are connected and communicate. Classical communication of elements on chip via buses presents a significant bottleneck, as every element has to gain bus control before sending any data and the receiving element has to be ready to obtain the data.

The Network-on-Chip (NoC) concept has provided a refined alternative - a connection-based scheme. Instead of elements being placed on a chip and connected via classical or crossbar bus, the net of switches is running the communication between other elements. Most importantly, this concept is scalable, meaning that even with an increased number of elements on a chip, efficient communication and high performance can be prevailed.

Network-on-Chip heavily relies on the network science theory and it is compound of elements present in almost every type of network.

2.1. Network-on-Chip Architecture

Network-on-Chip consists of two main elements: switches (also referred to as routers) and processing elements (PE). Switches can be interconnected in various ways - defining a network topology. The processing element is connected to a switch by the network interface (NI) and communicates with other elements via the net.[22]

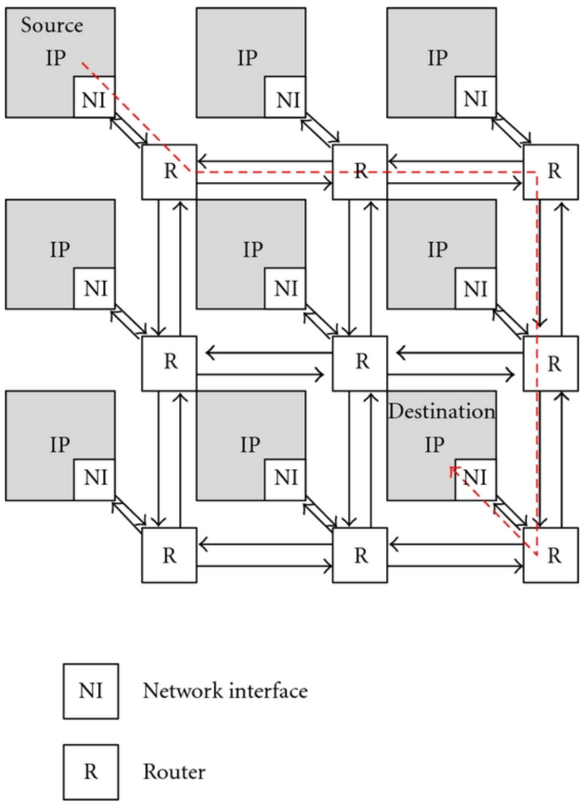


Figure 2.1: NoC structure - routers, network interfaces and processing elements, [22]

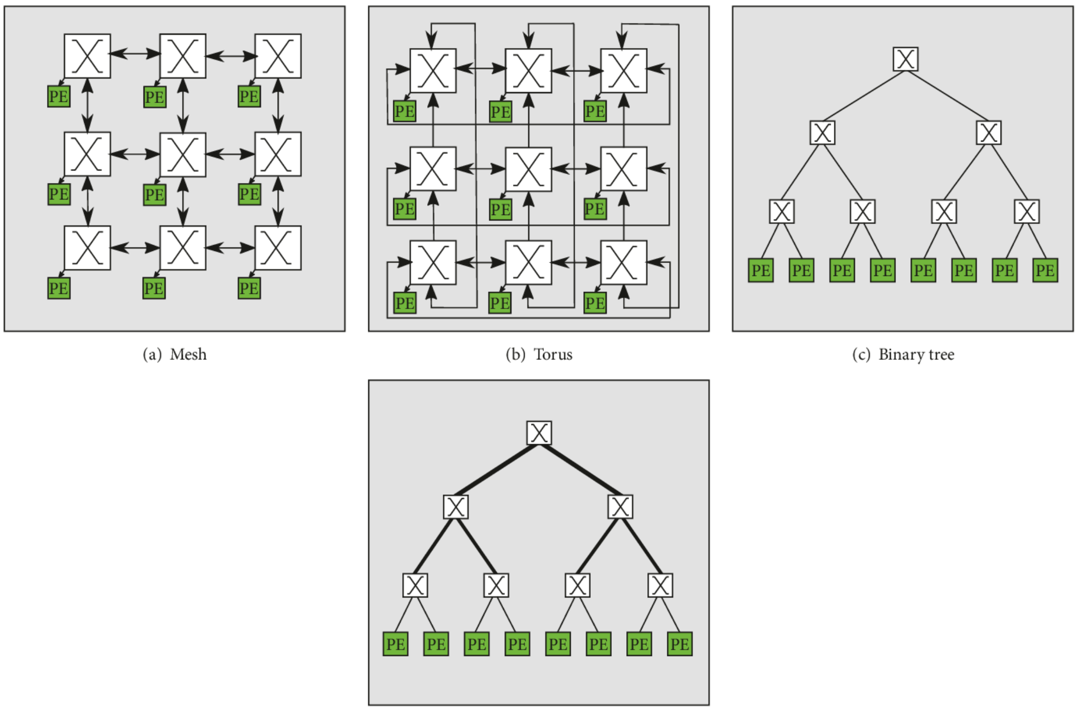
2.1.1. Topology

Topology defines the arrangement of switches (also called nodes) in a net and can significantly impact the performance.

Topologies can be regular and irregular. Regular topologies are simpler to implement, measure and manipulate, while irregular topologies offer more flexibility. Regular topologies are better suited for NoCs with homogeneous processing elements, and irregular topologies for NoCs with heterogeneous processing elements. The majority of NoCs use regular topologies such as mesh, torus, fat tree and octagon.

The property of an NoC topology that affects the performance the most is the bisection bandwidth. Bisection bandwidth is the maximum bandwidth between any two switches in the net and it is equivalent to the bandwidth between the two partitions of a bisectioned net [22]. For example, a mesh topology with n nodes has a bisection bandwidth of $\sqrt{n}$.

Topologies where there is no bottleneck node and no node failure can disrupt proper functioning of the network are preferred. Mesh and torus topologies possess this property and are widely used because of it.



(a) Mesh (b) Torus (c) Binary tree

(d) Fat tree

Figure 2.2: Regular NoC topologies [20]

2.1.2. Routing

Routing in NoC refers to the way in which data packets are distributed throughout the net. Routers/Switches are part of the network that are in charge of receiving the package and resending it to the destination switch. NoCs mostly implement packet routing. No acquisition of the transmission channel is performed and the packets are sent to the network independently.

The common NoC switch has multiple input and output ports and associated control signals. It implements arbitration and the routing algorithm.

Arbitration in switches solves the issue of multiple packages arriving on different input ports of a switch and requesting to be outputted to the same port. Arbitration decides which of the packages will be transmitted by the switch and which is to wait in the queue or some similar control structure.

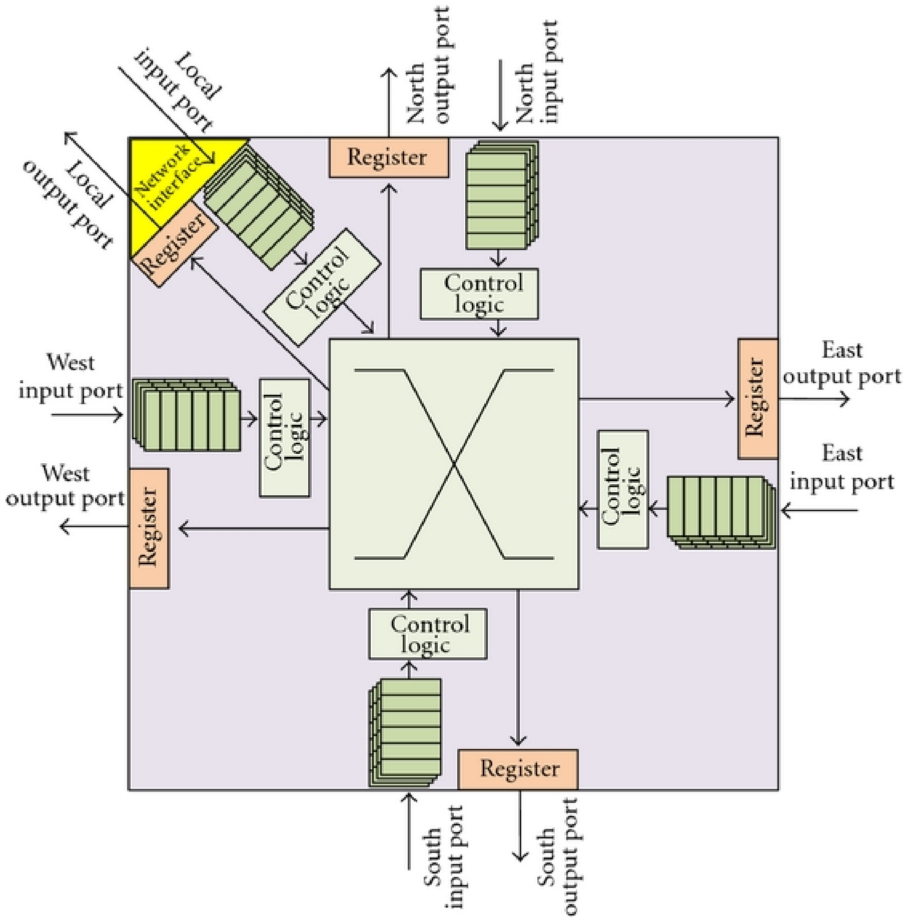


Figure 2.3: Common NoC switch design [22]

Deciding to which switch the data should be routed is done using routing algorithms. There are several routing algorithms. To be considered acceptable, the algorithm has to be resistant to starvation and dead-locking. The two main categories of routing algorithms are static and dynamic algorithms.

Static routing algorithms are deterministic and the paths for packages in the net are predefined. Switches using static routing algorithms, such as flooding and XY-routing, are simple for implementation but do not account for the current state of the network.

Dynamic routing algorithms adapt their strategy based on state-of-the-net parameters. As the state-of-the-net is volatile, there is no predefined path the package will follow. Common state-of-the-net parameters taken into account are the state of the links, congestion occurrences and resource availability. Even though such a strategy allows for a better use of the overall bandwidth, it complicates the switch design and requires the network monitoring system to be implemented.

It is important to bear in mind that choosing the most suitable algorithm for NoC implementation is severely dependant on the topology the NoC uses. For instance, XY-routing is considered the most suitable routing algorithm for torus and mesh topologies.

2.1.3. Packaging

As most of the communication protocols, the NoC often has a set package size. The input data is often not in such a form, which calls for a module to perform packaging. Such a module is often placed at the entrance to the network. Its main purpose is to divide input data into fragments that fit the package size, assign a package number and destination information, the importance of which will be further explained.

After the package reaches the switch, the switch has to decide of where to route the package based on the destination information. Naturally, the package itself has to contain that information. The destination is most commonly placed in the package header. The nature of destination information depends on the net topology, where mesh and torus topologies mark their switches with discrete Cartesian coordinates such as `(0,0)`,`(3,2)`, etc.

Another issue to be considered in packaging is the package order. The network itself cannot guarantee that the package emission order will be the same as the received package order. Noting that many problems require the data to remain in order, the packaging module enumerates the packages and places the number in the packet header, along with the destination information. The packages are reassembled in the right order upon exiting the network.

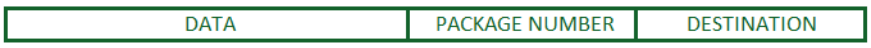


Figure 2.4: Common NoC package structure

2.1.4. Processing Elements

Elements of the Network-on-Chip that perform some useful computation are called Processing Elements (PE). PEs in a NoC can be either homogeneous or heterogeneous, as long as they offer an interface for connecting to the net.

A PE is connected to a switch through a network interface for data exchange. From the perspective of a switch, a PE is just another source and destination of packages, like any other switch it is connected to. A package is passed from the switch to the PE when the destination information of the package matches the marked information of said switch.

Another role of the PE is changing the destination information of the received package. After the computation on the package data is performed, it is reasonable to assume that the package should either exit the network, in the case of homogeneous networks, or should be forwarded to some other PE to perform additional computation, in the case of heterogeneous networks. For exiting the network, the destination information should be set to (0, 0) and for forwarding to other PE-s to the coordinates of said PE.

Even in homogeneous networks, one of the PEs is different and represents an entrance point for data outside of the net. Said PE is responsible for outputting the data from the net as well. Except for its main purposes, all additional preprocessing and postprocessing of the data can be implemented in it.

This work implements a processing element performing the Serpent algorithm encryption and decryption.

2.2. OpenNoc

The Network-on-Chip implementation used for this thesis is OpenNoc. OpenNoc is an open source implementation of NoC for implementation on FPGA technology. It was developed by Reddy, K.S. and Vipin, K.[16]

The topology of OpenNoc is unidirectional torus. Unidirectional torus is a topology in which the nodes are circularly connected along the X-axis and Y-axis. Therefore, each node belongs to two circles, which allows the packages it receives to be transferred to any other switch in the net. The routing algorithm used is *hot-potato* based on X and Y coordinates of switches. The destination information for routing consists of X and Y coordinates. The term *hot-potato* derives from the fact that if the optimal output port from the switch is not available, the packet will be immediately sent to the other available port, being passed as a hot potato.

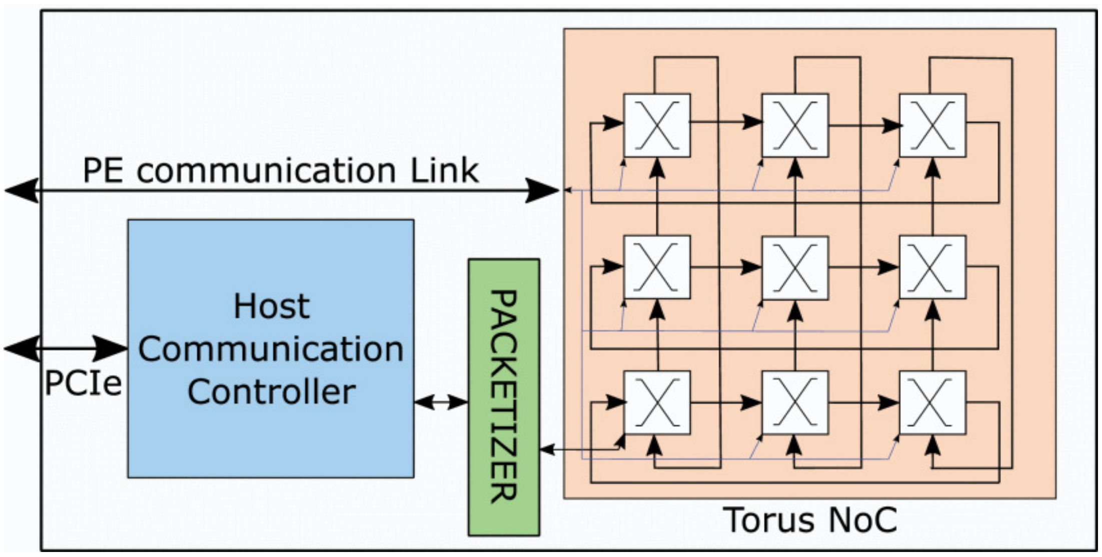


Figure 2.5: OpenNoc architecture [16]

The X coordinate has priority over the Y coordinate, meaning that packets will first move along the X-axis and then along the Y-axis to reach the destination switch. The destionation information in the package consists of X and Y coordinates of the destionation switch. Each switch can accept data from the left (west), the bottom (south) and the PE directions. Packets can be sent to the right (east), the top (north) and the PE directions. The forwarding priority of received packages is determined by the module Arbitrator which generates control signals for selection.

The OpenNoc can be adjusted to the needs of a specific task by providing a customized PE and setting the configurable parameters:

1. `X` - number of switches on X-axis
2. `Y` - number of switches on Y-axis
3. `data_width` - data width of the package
4. `pck_num` - width of the package number field in the package header. It restricts the number of packages in the net to the maximum of 2^{pck_num}

2.2.1. OpenNoc Switch

OpenNoc switch is characterized by its routing algorithm and the priority of inputs. The OpenNoc switch is bufferless, meaning that it cannot queue or store data. Its functionality is purely to forward packets to other switches and PEs. PEs, on the other hand, can queue data, which is further elaborated in the Serpent Algorithm Implementation section.

Priority of inputs is derived from the fact that switches are bufferless. PE input has the lowest priority since it can store data, whilst switches cannot, meaning that their data has to be forwarded upon the request. If the switch receives input requests from all three inputs, remembering that there are only two outputs available, one of the packages cannot be transmitted. If it were one of the two packages from switches, that package would be lost. If it were the package from the PE, the switch can pressure the package back to the PE queue. That is the main motivation for the PE have the lowest priority.

If two packages arriving from the adjacent switches were to be transmitted to the same output port, one of the packages will be rerouted to the non-optimal output. This increases the total path the package passes in the net.

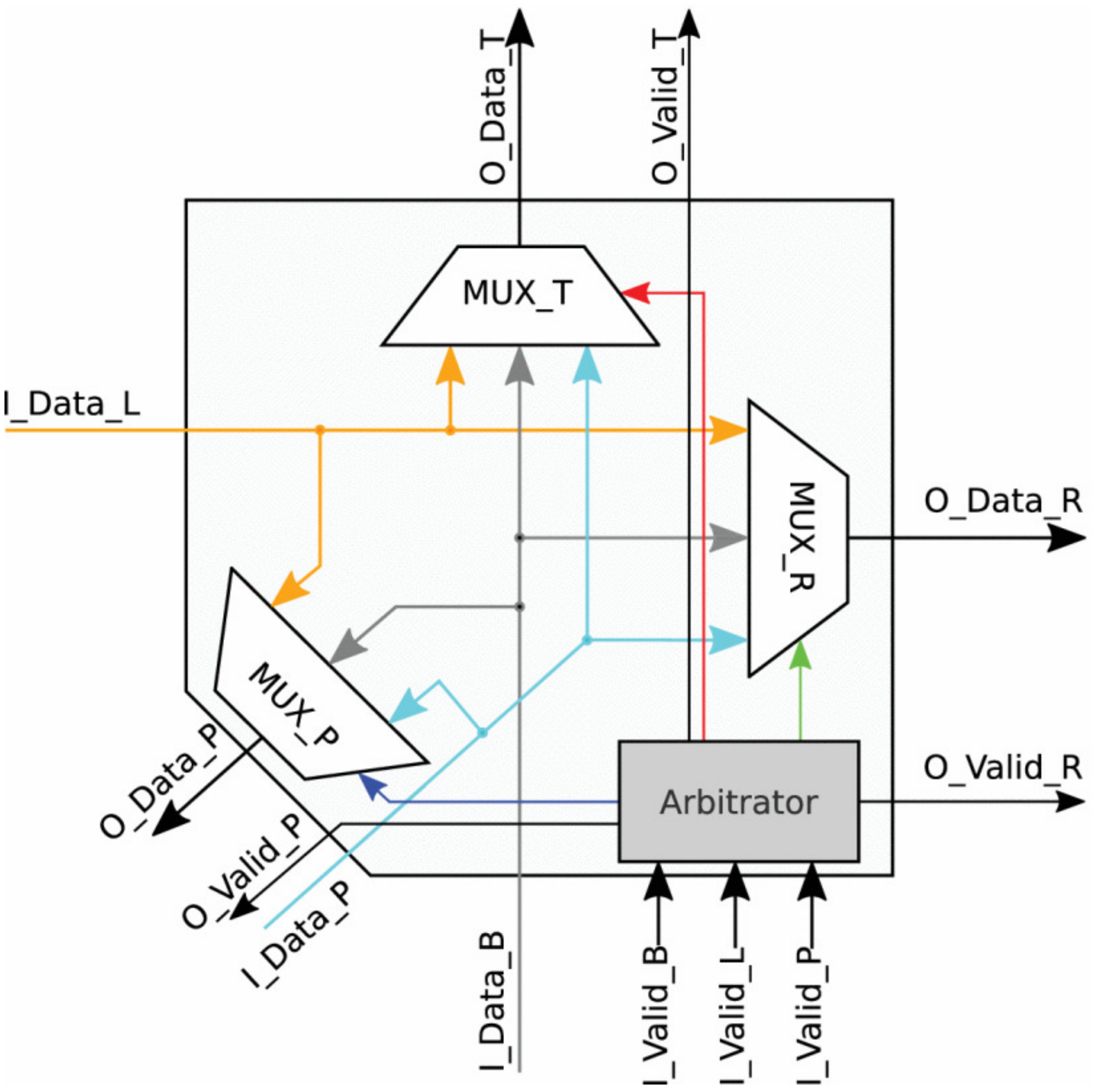


Figure 2.6: OpenNoc Switch [16]

2.2.2. OpenNoc InterfacePE and Scheduler

Module InterfacePE presents the connection of the net with the outer world. It is the entrance point for the data. InterfacePE is responsible for receiving the external data, manipulating it in the way convenient for sending through the net and storing the net outputs externally. As its name suggests, this module is compatible with the interface each PE has to be compatible with. This allows it to be connected to the switch. It is most often placed as the PE coupled with the switch at coordinates `(0,0)`.

InterfacePE consists of two submodules: Scheduler and MyFifo. MyFifo is responsible for outputting the data to the switch at `(0,0)`.

Scheduler has several different roles. Upon receiving the external data (i_data_pci), InterfacePE passes the data to Scheduler. Scheduler sets up the packet number, destination x-coordinate and destination y-coordinate. Packet number has to be set in order to allow the packages to be ordered when received from the net. Then Scheduler passes the data (`o_data`) to MyFifo to be outputted to the net.

Upon receving the data (`i_data_pe`) from switch, the scheduler places it in the predefined memory according to the package number and raises the flag signaling that the package is ready to be outputted. The memory has capacity to store as much packages as can be addressed with `pck_num number` of bits. Scheduler outputs the data its original order. The order in which InterfacePE receives packages from the net is irrelevant. Once the package is received it is placed in the designated memory slot, but will not be outputted until all packages with smaller package number have been outputted. This will result in the packages being ordered after processing.

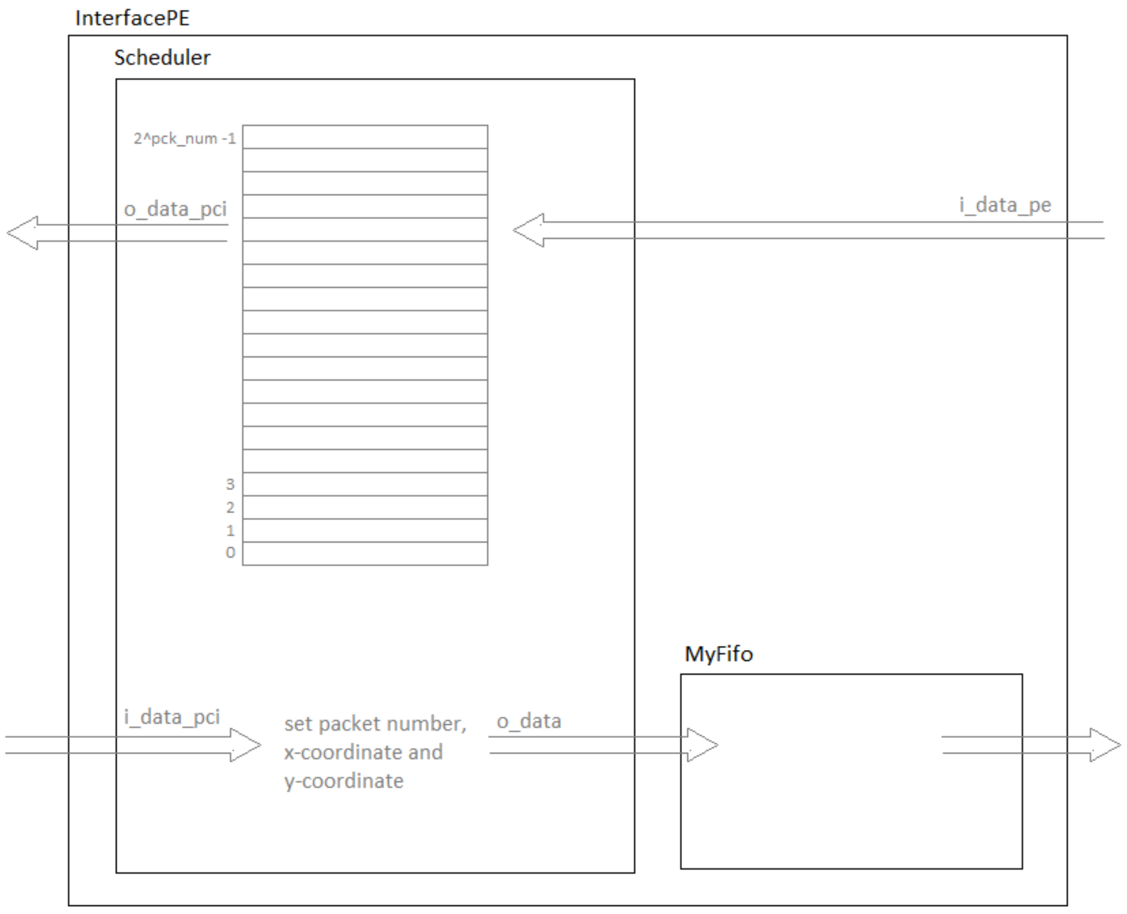


Figure 2.7: OpenNoc Scheduler

2.3. Field Programmable Logic Array (FPGA)

Field Programmable Logic Array (FPGA) is a programmable semiconductor device. What makes this technology so widespread is the fact that it is programmable, meaning that one hardware device can be used for multiple purposes and applications. Instead of creating a specialised hardware for each application, the application is defined using one of the hardware description languages (HDLs) and programmed onto the FPGA. This allows for easier error correction and upgrading, as the board just needs to be reprogrammed [7].

FPGAs are frequently used in the process of design prototyping and troubleshooting before the design goes into production. For instance, if a new core were in the process of development and the testing were performed on the specialised hardware, with every change in the core design, the hardware would need to be produced again, which is both costly and inefficient. Using FPGA only requires that for every design change, the board should be reprogrammed. When the final design is ready, it is put into production as the dedicated hardware, mostly a microcontroller or application-specific integrated circuit (ASIC).

The concept of reprogrammability FPGAs offer has become so alluring that producers of modern processors started placing FPGA-like reprogrammable components on processor chips.

The price for great flexibility that comes with reprogrammability is mostly paid in optimisation of the design. The same functionality will always be better optimized on its dedicated hardware than on the FPGA. Sometimes this is a price the designers and users are willing to pay, and sometimes not.

Most of the FPGA architectures consist of three main elements:

1. Configurable Logic Blocks (CLBs)

2. Configurable I/O Blocks

3. Programmable Interconnect

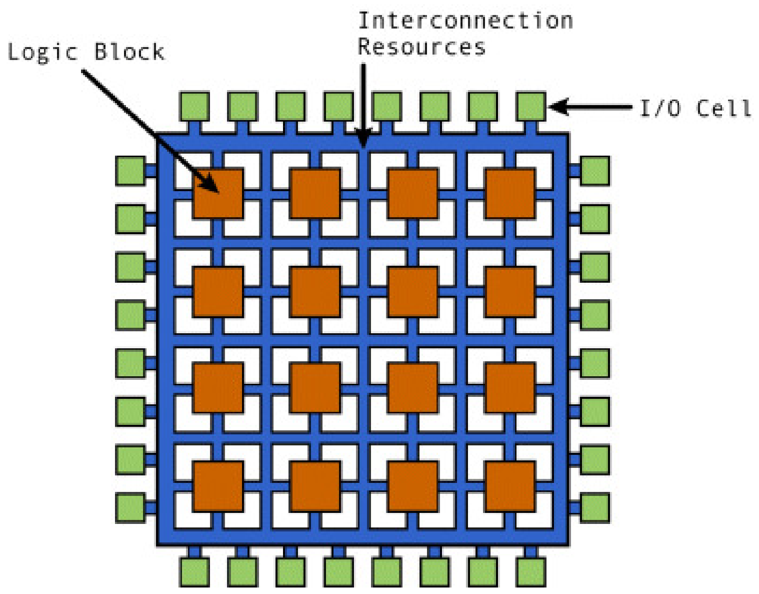


Figure 2.8: FPGA Architecture [5]

CLBs contain the logic for FPGA, which is used for creating a small state machine. Connecting multiple CLBs together enables even the most complex functionalities to be implemented. CLBs are constituted of several look-up-tables (LUT), flip-flops, multiplexers and carry logic [1]. LUTs provide a fixed output for a given input, which speeds up processing. LUTs mostly consist of six inputs and 2 outputs, whereby the inputs are independent. Independent inputs allow the LUT to be programmed in several ways, thus providing logical functions with a distinct number of input variables [3].

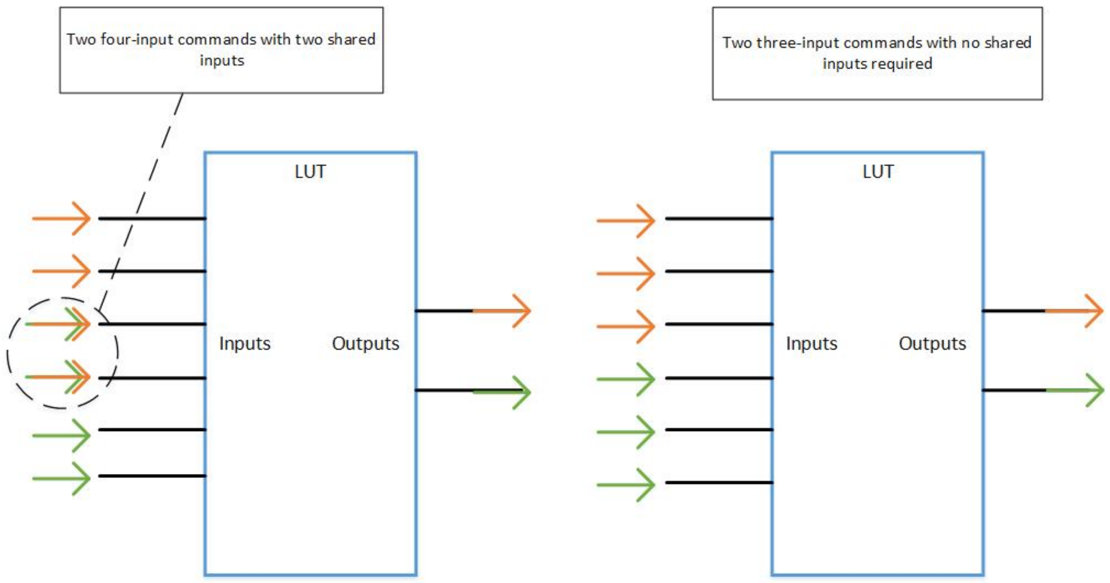

Two four-input commands with two shared inputs
LUT
Inputs
Outputs
Two three-input commands with no shared inputs required
LUT
Inputs
Outputs

Figure 2.9: FPGA look-up-table [3]

Configurable I/O Blocks are used for the communication of the chip with peripheral devices. I/O Blocks can be used both as input and output. They can be configured to fit the specifications of a wide range of peripherals.

Programmable Interconnect connects multiple CLBs and enables the creation of more complex combinational and sequential logic than a single CLB can fit. The programmable part of the Interconnect are Programmable Switch Matrices (PSMs), which define which of the CLBs are connected.

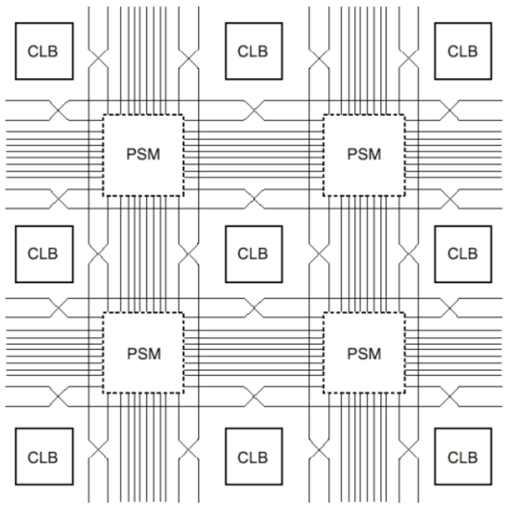


Figure 2.10: FPGA Programmable Interconnect [13]

The programming of the FPGA is performed using the design bitstream. During production, the bitstream is placed in the non-volatile memory (NVM) of the FPGA and it is programmed to load the bitstream and program the device on powerup. During the development phase, the bitstream is transfered to the FPGA board via cable. The genesis of bitstream is further explained in the next chapter on Hardware Description Languages.

2.4. Verilog Hardware Description Language

The language used for description of OpenNoC and the Serpent algorithm is Verilog. Verilog is a hardware description language. It can model various digital systems such as memory, flip-flops, all the way to modules implementing complex algorithms. It is imporant to note that the description of the system is fully independent of the technology used for its implementation.

The central term in hardware description is module. A module is a block of code implementing certain functionality [19]. Modules can be combined and nested into higher-level modules, defining a module hierarchy. The module at the top of the hierarchy is referred to as a top level module.

The process of setting the Verilog code onto the FPGA board consists of synthesis and implementation. Synthesis converts the HDL modules into a list of instance objects called a net list. A net list is specific to the device used. The implementation process maps the net list to device's physical resources and generates bitstream that defines the configuration of the device. Lastly, the device is programmed using the generated bitstream [2].

Design verification, can be done in different phases of implementation by running various types of simulations.

It is importat to note that Verilog, like any other programming language, can be used to define practically any functionality. All functionalities that can be implemented can be used in simulation. On the other hand, physical FPGA boards come with many constraints, namely number of CLBs, pins and interconnects. Considering that implementation is done in hardware, number of functionalities in Verilog that can be implemented are reduced. Apart from quantity constraints, this is due to the fact that simulation allows for certain states of the execution to remain undefined, while the hardware does not.

Several FPGA manufacturers offer different tools for HDL development and for this purpose Xilinx Vivado tool was used for design, simulation and synthesis.

3. Serpent Crypto-Algorithm

The Serpent crypto-algorithm is one of the finalists of the NIST competition for AES. It is a symmetric key block cipher that operates on 32-bit words. The algorithm is available both in standard and bitslice mode. Its design allows for maximum parallelism, as all operations are applied on all 32 bits of word in parallel.

The creators of this cipher, E. Biham, L. Knudsen and R. Anderson, have paid special attention to the fitness of the algorithm for hardware implementation. This has incited this work to choose the Serpent algorithm for accelerated hardware implementation using NoC.

The details of the cipher design and underlying mathematical background are further elaborated in the following chapters.

3.1. Design

Symmetrical encryption algorithms require only one key both for encryption and decryption. The starting data for encryption is referred to as the plaintext, whilst the product of encryption is called the ciphertext. All operations are applied to blocks of data, making it a block cipher. As per AES requirements, it has a block size of 128 bits and supports key sizes of 128, 192 and 256 bits.

The algorithm consists of 32 rounds or simpler, 32 iterations. Even though 16 rounds of the algorithm were deemed sufficient against the known attacks, the designers opted for a very conservative approach using 32 rounds to secure it against the possible future attacks. Each round consists of three main layers:

1. key-mixing
2. substitution using substitution boxes (S-boxes)
3. linear transformation

The final, 32nd round is the only round that differs, as its third layer performs key-mixing instead of a linear transformation. The output data of every layer presents the input data for the following. Such structure of rounds places Serpent in the family of the so-called substitution-permutation networks. Substitution refers to the substitution layer, whilst permutation refers to the linear transformation layer.

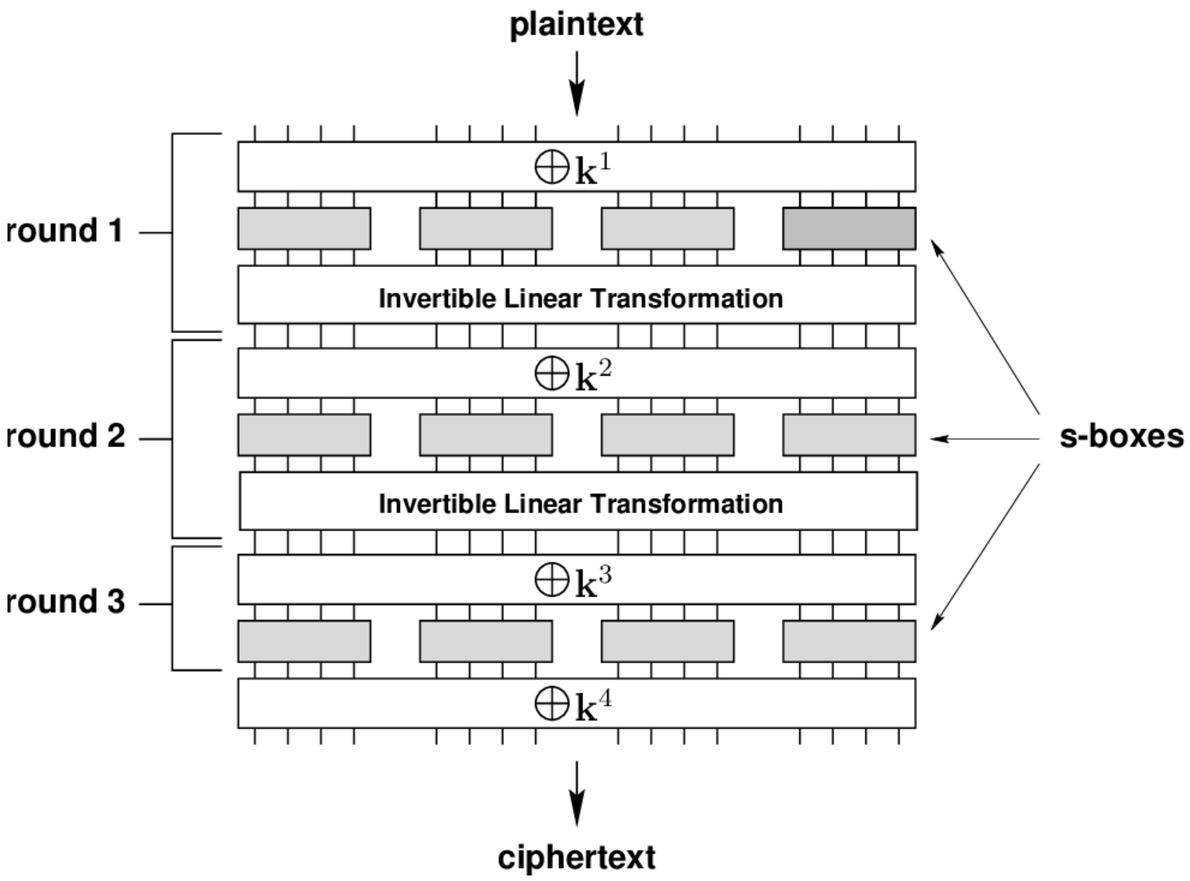


Figure 3.1: Substitution-permutation network for three rounds [14]

3.1.1. Key scheduling

Having closely observed the substitution-permutation network of Serpent, it is easy to conclude that the numbers don't add up. The algorithm requires 33 keys of 128 bits each, one for each of the 31 identical rounds and 2 for the final 32nd round, adding up to the total of 4224 bits of key material. Still, the algorithm is provided with either 128, 192 or 256 bits of key material.

This difference is solved by the process of key scheduling (also called key expansion). Key scheduling performs several operations on the provided key material, in order to expand it and transform it to 33 round keys, each consisting of 128 bits. Each round key is used in one round's key-mixing layer. Key-mixing is performed using the bitwise exclusive-OR operation on round data and round key.

3.1.2. S-boxes

Substitution boxes represent the only non-linear part of the algorithm. Their design is of utmost importance for the security of the algorithm.

One S-box is designed as a lookup table of 16 slots per 4 bits. Such an S-box takes four bits as an input, uses them to reference the table slot and outputs four bits stored in the slot. Serpent has eight S-boxes circulating through the rounds. The S-box used for a particular round is chosen by the following equation:

```
sbox_number = (round) mod 8
```

Even though some ciphers generate S-boxes dynamically from the key, Serpent uses fixed S-boxes, which presents a more conservative approach.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12

Figure 3.2: Serpent S-box 0

Considering that one S-box takes 4 bits, outputs 4 bits and the block is 128 bit long, the pass through the same S-box is performed 32 times in parallel. The algorithm views the 128-bit block as four 32-bit words $w_0 w_1 w_2 w_3$. Four bits presenting the input to the first of the 32 parallel S-box replications are first bits of each word, second bits for the second S-box replication and so on. The output follows the same logic.

This significantly contributes to the execution speed of the algorithm.

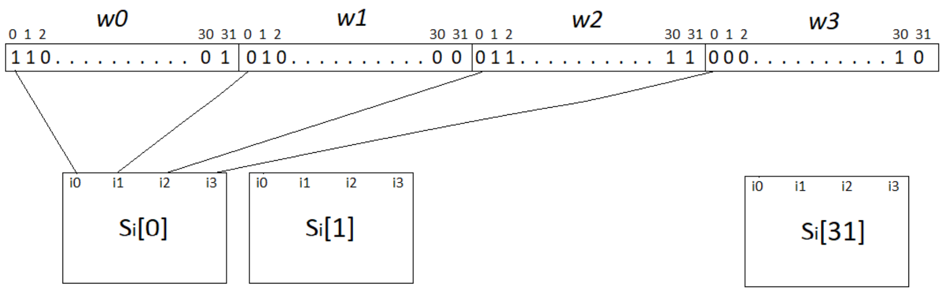


Figure 3.3: S-box inputs

Properties

The S-box design relies on choosing the best differential and linear properties of the 4-bit permutations. Serpent initially used DES S-boxes with an estimate 2^{100} of the known plaintexts/ciphertexts required to perform the attack. Soon the designers discovered the alternative approach for generating S-boxes that allowed that estimate to rise from 2^{100} to 2^{256}, making the algorithm much more secure.

The properties of the S-boxes observed for security are differential characteristics and linear characteristics. The differential characteristics define the input and output differences and the differences between the round data. The better the differential characteristics, the less likely is differential cryptanalysis to yield meaningful results.

On the other hand, bad linear characteristics are mostly exploited in the process of linear cryptanalysis. The aim of linear cryptanalysis is to find linear dependencies between bits of plaintext and bits of ciphertext. Presence of one linear dependency decreases the effective key size by one bit.

The properties of Serpent S-boxes aimed at preventing differential and linear cryptanalysis from yielding useful results are predominantly of statistical manner. The key property is the nonlinear order of output bits. The nonlinear order of output bits describes how many bits of the output have changed, given that the input has changed in one bit. The goal of the S-box design is to maximize the nonlinear order of output bits and ensure that a one-bit difference of inputs never results in a one-bit difference in outputs. The Serpent proposal for AES argues that the non-linear order of output bits is maximized, having the value of three. However, the later study by B. Singh, L. Alexander and S. Burman has pointed out that several output bits have the nonlinear order of 2, which pointed out the need for revisiting the security of the Serpent cipher [17].

3.1.3. Linear transformation

The third layer of each round is linear transformation. The transformation consists of several rotations, shifting and exclusive or operations on 32-bit words.

The chosen linear transformation aims to maximize the avalanche effect. The avalanche effect is the property of crypto-algorithms that ensures that for a small change in the input, the change in the output is significant. It is said that the propagation of change is big. Additionally, over three rounds, every plaintext bit affects all round bits after three rounds. This is also true for every round key.

The complete procedure of a single round goes as it follows:

```
X0, X1, X2, X3 := Si(Bi xor Ki)
               X0 := X0 <<< 13
               X2 := X2 <<< 3
               X1 := X1 xor X0 xor X2
               X3 := X3 xor X2 xor (X0 << 3)
               X1 := X1 <<< 1
               X3 := X3 <<< 7
               X0 := X0 xor X1 xor X3
               X2 := X2 xor X3 xor (X1 << 7)
               X0 := X0 <<< 5
               X2 := X2 <<< 22
               Bi+1 := X0, X1, X2, X3
```

The <<< denotes left rotation and << denotes left shift.

3.2. Decryption

Since Serpent is a symmetric encryption algorithm, the key used for encryption is also used for decryption. This makes the key expansion process for encryption and decryption identical.

Decryption is done in similar steps as encryption, but in reversed order. It consists of 32 rounds, whereby round layers are constituted in a reverse order:

1. Inverse linear transformation
2. Decryption S-box pass
3. Key mixing

3.3. Security

In cryptography, the main goal is achieving the so-called, perfect security. For a cipher to have perfect security, as defined by Claude Shannon, it should disclose no information of the plaintext, even if the attacker has unlimited resources. This makes the probability of deciphering the cipher text equal to the probability of achieving so by guessing the key.

However, the perfect security is impractical to implement, so multiple theoretical approaches have been developed to provide a secure cipher that is also practical to implement. Still, there is no definite proof of security for any widespread cipher. Another important concept to follow in cipher security is avoiding security by obscurity. That is, the cipher should be secure even if all information except for the key are available. This is also one of the reasons crypto-algorithms are unclassified.

Further chapters analyse the security of the Serpent crypto-algorithm and some known attacks.

3.3.1. Attacks

The main conceptual attacks in cryptography are known plaintext, chosen plaintext, known ciphertext, brute-force and dictionary attacks.

In a known plaintext attack the attacker has access to the plaintext and its corresponding ciphertext. Given several pairs of plaintext-ciphertext, the attacker can try to partially or completely break the cipher. In a chosen plaintext attack, the attacker can request the encryption of the plaintext of his choice. The fact that the attacker can choose the plaintext gives him bigger posibilities to break the cipher.

A known ciphertext attack consists of the attacker possessing only the ciphertext and performing intensive computations to get the knowledge of the plaintext.

Brute-force and dictionary attacks, on the other hand, focus on trying to guess the key. During a brute-force attack, the attacker tries all key combinations until the one that produces the correct plaintext is found. With the current computational power available, the 60-bit keys can be broken in this manner in under a day, and therefore, much longer keys like the 256-bit ones should be in use. A dictionary attack is a type of brute-force attack that, instead of performing an exhaustive search on the complete key space, performs it on its subset that contains popular expressions, common words and some frequent modifications of words.

3.3.2. Attacks on Serpent

The initial Serpent submission claimed that the cipher could not be broken in any way other than by an exhaustive search of the key space. The number of texts required for performing linear or differential cryptanalysis against any key is 2^{256}. The designers claim that, for breaking the cipher, a new theoretical breakthrough in the field of cryptanalysis needs to be made.

A few attacks were performed on algorithm versions with a reduced number of rounds. In 2000 Kohno et al. presented two attacks against 6 and 9 of the 32 rounds of the algorithm. Another linear cryptanalysis attack, presented by E. Biham et al., broke 10 of 32 rounds of Serpent-128 with 2^{118} known plaintexts and 2^{89} steps.

Even though no Serpent cipher has been broken so far, which would make the cipher inherently unsecure, the paper arguing that the several output bits of S-boxes have a nonlinear order less than 3, contradictory to the designers claims, has called for a revision of the algorithm's security.

4. Serpent Algorithm Implementation

The implementation of the Serpent crypto-algorithm was carried out using Verilog, a commonly used HDL. To provide a full functionality of both encryption and decryption, two main modules Encryptor and Decryptor were developed. These modules represent the Processing Elements for the OpenNoc and are connected as such.

PEs are modeled to perform the algorithm on the 128b input data (the Serpent data block). Upon restart, the PEs perform key expansion over a given user key. Upon receiving the input data, the algorithm is performed and the output data is calculated. The output of the algorithm is queued for exiting the PE into OpenNoc.

It is importatnt to note that the algorithm is implemented in bitslice mode with 32-bit words. Bitslice mode defines the size of the words the main 128b block is divided into and how the operations are applied to said words. The algorithm design allows for operations to be carried out on words of size as small as a single bit, to match the targeted processor architecture, bus width and word size. This implementation breaks 128b input data into four 32b words, using operators as it is elaborated in the algorithm specification.

The implementation details of modules are further explained in subsequent chapters.

4.1. Encryption Module

The aim of encryption module is to encrypt the input plaintext of 128 bits and produce the 128-bit ciphertext.

The module interface is given below.

```
module Encryptor #(parameter total_width=0, x_size=0, y_size=0,
    pck_num=0) (
        input wire clk,
        input wire rst,
        input wire [total_width-1:0] i_data,
        input wire i_valid,
        output wire [total_width-1:0] o_data,
        output wire o_valid,
        input wire i_ready,
        output wire o_ready
    );
```

Listing 4.1: Encryptor.v module declaration

As per declaration, the Encryptor module can be parametrized. The `total_width`, `x_size`, `y_size` and `pck_num` parameters should be set upon instantiation. The parameters define the structure of the received packet and it is important to match the packet parameters decided on and defined in the OpenNoc. Otherwise, some bits might be dropped or wrong bits of the packet might be extracted as the input data, eventually resulting in faulty algorithm functionality.

Apart from the conventional wires like `clk` (clock) and `rst` (reset), the interface consists of several flags, data input and data output. A careful reader has already noted that the interface is compatible with the OpenNoc switch presented in Figure 2.6. The flags are used for signaling between the switch and the PE. The switch is interested whether the PE can accept the data or has data to send, whilst the PE is interested in whether the switch can accept the data or has data to send. Input data and output data wires are used for package transmission when both parties are ready.

The main course of actions in the Encryptor module when package is received is:

1. if this is the first package received after startup, wait until the key expansion has finished

2. extract the data from the package

3. perform encryption

4. construct the output package consisting of unmodified packet number, the encrypted data and the destination information set to `(0,0)`

5. send the output package to the switch.

Part of the logic inside the Encryptor module is fragmented into smaller submodules implementing specific parts of the logic, such as key expansion and S-boxes.

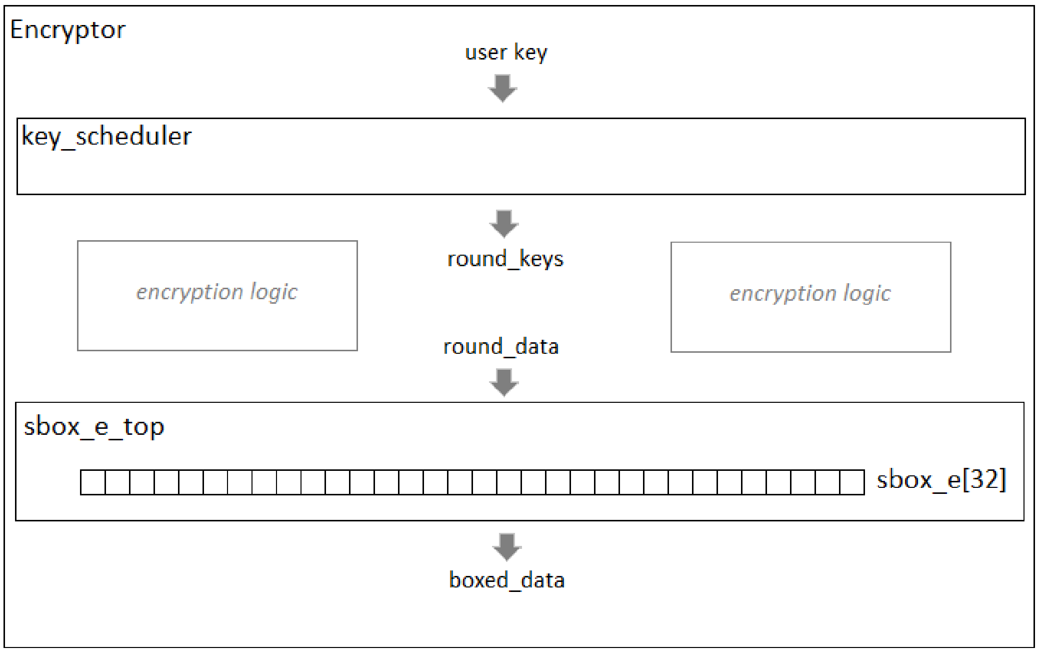


Figure 4.1: Encryptor module hierarchy

Module key_scheduler performs key expansion, deriving 33 128b round keys from the 256b user key.

Module sbox_e_top is the module that instatiates 32 S-boxes used during encryption and properly wires the bits of the 128b data onto the S-box inputs and outputs. The rest of the encryption logic is implemented in the Encryptor module.

4.1.1. Intellectual Property MyFifo

After defining the key expansion, S-boxes and encryption logic, one thing that remains unsolved for the Encryptor processing element is what if the switch isn't ready to accept the output data once the data is ready. If the switch gets signal requests to obtain data from multiple entities, i.e. the left and bottom adjacent switches and the PE and decides to accept the data from the left and bottom switch, the outgoing data from the PE should be conveniently stored.

For that purpose, the functionality of outputting the data is delegated to the specific Intellectual Property (IP) module MyFifo. Intellectual property refers to a previously designed module with some common functionality that is mostly appropriated to reuse in other user modules.

MyFifo implements the functionality of the First-In-First-Out (FIFO) structure, essentially a queue. Once the PE has finished computation and wants to output data, the data is queued in the FIFO structure. The FIFO structure communicates with the switch and outputs the data from the queue when the switch is ready to accept it and the data is present. Even though the structure is limited and in theory can be overflowed, resulting in lost packages, such occurrence has essentially minimal possibility during runtime.

Seeing that the inverse situation is also possible, that the switch accepts data from PE, and denies data from the left adjacent switch, the left switch should be able to store the outgoing data as well. However, the OpenNoc does not provide this functionality in order to reduce transmission latency and keep the switch design and implementation simple. Instead, the switch pressures the PE packet back into the PE until the switch becomes available again.

4.2. Decryption Module

Module Decryptor, used for decryption, has the same interface as the Encryptor module, seeing that they provide the inverse functionality and are connected to the switch in the same manner.

The Decryptor module performs key expansion in the same way as the Encryptor, thus it uses the key_scheduler module in its hierarchy. Where the two essentially differ is the encryption logic and the used S-boxes. Module sbox_inv_d model the inverse S-boxes used for decryption and the module sbox_d_top instantiates 32 inverse S-box modules and wires the them to the 128b input and output. MyFifo is used for the already elaborated purposes.

4.3. S-boxes

Seeing how crucial S-boxes are for the Serpent algorithm and that their design is something of a trademark of the Serpent algorithm, this section will further explain the hardware implementation of S-boxes.

There are eight distinct S-boxes for encryption and 8 distinct inverse S-boxes for decryption. Each S-box is fundamentally a lookup table with 16 slots per 4 bits, so it can be observed as a form of miniature memory. Output bits are defined by the 3b box number and 4b input bits.

The implementation models each S-box by a 64b packed array register. The array of S-boxes is implemented as an unpacked array.

New `out_bits` value is assigned every time `box_num` or `in_bits` inputs change, as it is defined in the procedural always block.

```
module sbox_e(
        input wire [3:0] in_bits,
        output reg [3:0] out_bits,
        input [2:0] box_num
    );

    reg [63:0] s_boxes [0:7];

    initial begin // defining s-boxes
        s_boxes[0] <= 64'b1100_0111_0101_1110_1111_0010...;
        s_boxes[1] <= 64'b1111_1000_1001_0110_0100_0111...;
        s_boxes[2] <= 64'b0001_1011_1100_0000_1110_0111...;
        s_boxes[3] <= 64'b0000_1011_0011_0101_1101_0100...;
        s_boxes[4] <= 64'b1000_0100_0011_1001_0001_0010...;
        s_boxes[5] <= 64'b1111_0000_0010_1011_0100_0111...;
        s_boxes[6] <= 64'b1110_0111_0001_1011_0011_1000...;
        s_boxes[7] <= 64'b1000_1110_0111_1001_1111_0011...;
    end

    always @(in_bits or box_num)
    begin
        // address the s-box and assign the out_bits based on in_bits
        out_bits = s_boxes[box_num][63-in_bits*4 -: 4];
    end

```

```
endmodule
```

Listing 4.2: sbox_e module

4.4. Simulation Sources

In order to simulate the behaviour of the net, all the modules need to be instantiated and wired by the so called top module. Processing elements (Encryptors) are instantiated by the ProcTop module. Switches are instantiated and connected by the OpenNocTop module, with exception of the switch at position `(0,0)`, where InterfacePE is placed. Parameters of ProcTop and OpenNocTop define the number of switches and PEs, and provide wire interfaces for connecting PEs to switches.

Module EncryptorTB is the top module, it instantiates ProcTop and OpenNocTop and connects their interfaces. This module is responsible for retrieving input data and storing output data. For simulation purposes, the data is stored in files on the disc. Data size must be known for loading and storing.

This altogether defines the design of OpenNoc performing encryption using Serpent crypto-algorithm.

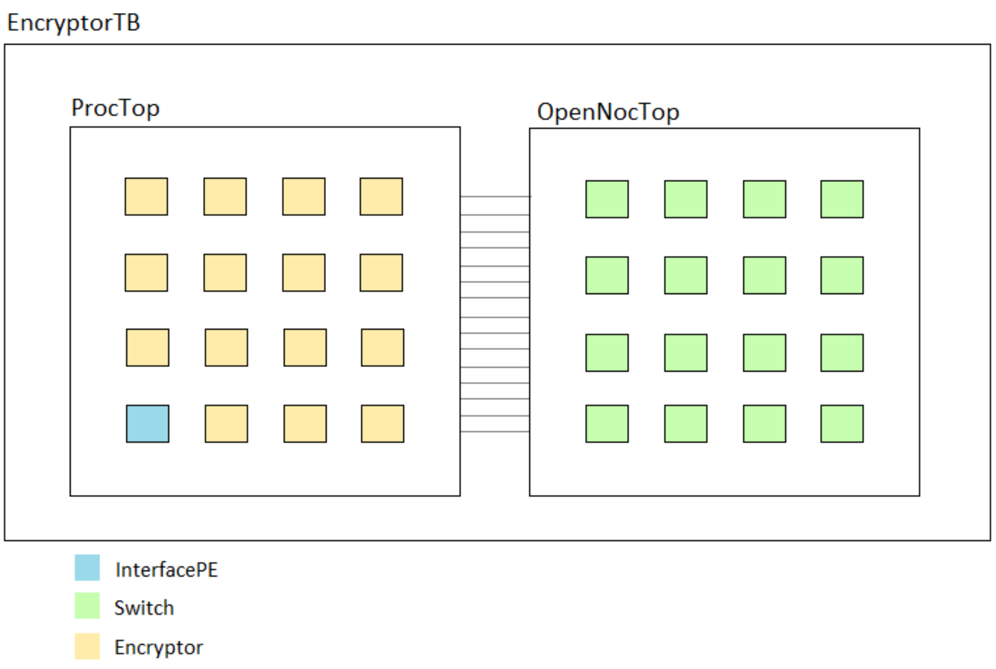


Figure 4.2: Instantiation hierarchy

5. Verification and performance assessment

Performance assessment was conducted in two stages, first being the qualitative assessment of modules Encryptor and Decryptor, second being quantitative assessment of the OpenNoC encryption as a whole.

5.1. Testing

5.1.1. Verification of Encryptor and Decryptor modules

Verification of Encryptor and Decryptor modules was performed using the referece implementation of the algorithm in Python by Frank Stajano. Test data was retrieved from Eli Biham's Serpent page. Test vectors are in the NESSIE (New European Schemes for Signature, Integrity, and Encryption) printout form and therefore were first converted into the Serpent format. Each test vector is constituted of the 256b key, 128-bit plaintext and 128b ciphertext.

The verification was performed by entering both Verilog and Python implementation in the debug mode, stepping through and comparing intermediate results of the two implementations.

Verification of the key expansion module key_scheduler was conducted in the same manner.

5.1.2. Verification of OpenNoC

OpenNoc encryption verification of the whole process of encryption was conducted simulating the top module EncryptionTB. EncryptorTB reads the data from the file, sends it to the InterfacePE element. At the end of executon EncryptorTB stores the encrypted data to the file.

The initial test aims at sending one package through the net. The data is read from a file, sent into the net, and the received packages are disassembled and the result of encryption is stored to the output file.

Afterwards, the test was repeated with multiple data examples of various size.

5.1.3. Simulation

Simulation was used to provide execution time estimates and several useful conclusions were established based on provided data. The standalone modules simulated were key_scheduler, Encryptor and Decryptor. Modules EncryptorTB and Decryptor simulate the working of the whole OpenNoc.

Considering the simulation statistics provided in Table 5.1, one can see that key scheduling process takes up a significant amount of time during encryption and decryption, namely around 40%. That is why it is crucial for performance that the key scheduling is performed only when the provided key is changed, not every time encryption or decryption are performed. Furthermore, encryption and decryption take up similar amount of time, which was expected considering that they are inverse.

The NoC should provide hardware acceleration, which was performed successfully. According to the measurement results, encrypting 10 blocks (1280b) takes up only about two times the time needed to perform encryption of a single block of data (128b). The time of encryption can be estimated as:

$$t_{encryption} = t_{encryptor} - t_{key} = 1036 - 400 = 636 \text{ ns} \tag{5.1}$$

The expected time for sequentially encrypting n blocks of data is

$$t_{seq} = t_{key} + n * t_{encryption} \tag{5.2}$$

Module	Simulation time (ns)
key_scheduler	400
Encryptor	1036
Decryptor	1072
EncryptorTB (16 cores - 10 packets)	2208
DecryptorTB (16 cores - 10 packets)	2200

Table 5.1: Simulation statistics

5.1.4. Multicore execution

The idea of multicore execution is to distribute the work to multiple cores and execute it in parallel.

In order to properly evaluate the multicore execution of the net, test data had to be substantially bigger. The data was generated using Python pseudorandom number generator.

Encryption has been repeated for 4, 16 and 32 PEs constituting the NoC.

The data is scheduled onto the PEs and sent into the net. On the other hand, the net was faulty when the number of packages exceeded the number of cores. After the PEs encrypted the first block of data and returned it, they remained ready to receive new packages. The new packages, however, were never sent to their dedicated PEs. This is a sign of possible congestion and deadlocks in the network. It is also reasonable to assume that the lengthy execution has hindered some functionality of the network.

Therefore, revision of the OpenNoc implementation is needed, with special attention paid to deadlocks and managing timely PE processing.

5.1.5. Synthesis

Some extended information about the design can be obtained from the netlist that is generated in the process of synthesis. Netlist is device specific and the device used for synthesis is xc7k70tfbv676-1 from the Xilinx Kintex-7 family of FPGAs.

Netlist is a collection of information about nets. Net is an abstract representation of a physical wire. It is mapped to wire upon routing and therefore provides useful information on design complexity and resource occupation. Number of LUTs occupied is also a representative measurement of design complexity. The measures are presented in the table 5.2.

Module	LUT number	Nets generated	Leaf Cells
key_scheduler	1507	6243	5987
MyFifo	51	632	2
Encryptor	51	25	16
Decryptor	51	25	16

Table 5.2: Synthesis statistics

5.2. Other hardware implementations

Apart from this FPGA implementation of Serpent algorithm, several other implementations in hardware, such as smart cards and application specific integrated circuits (ASIC), are available.

The smart card implementation is very common for different cryptographic purposes. A smart card shoud be convenient for user and perform high speed computation, whilst keeping RAM acquisition minimal. The Serpent creators claim that Serpent can be implemented to use up about of 80 bytes of RAM and yield minimal code size. This is thanks to the bitslice mode and design of the S-boxes as lookup tables. One of the main memory and speed bottlenecks of the Serpent algorithm is establishing round keys from the user key material. Establishing round keys is computationally as costly as performing a single encryption. Still, Serpent remains a cipher suited for smart card execution.

Apart from low memory and high speed requirements, smart cards present a challenge from the standpoint of security as well. Except for the already mentioned power consumption side channel attacks, smart cards have been challenged by microprobing attacks [21]. Microprobing is an attack that aims at reading internal secrets (i.e. keys) by attaching microscopic needles to the chip.

Considering that FPGA implementation is in various aspects similar to the smart card implementation, further improvements of FPGA should be guided by the research on smart cards implementation.

5.3. Improvements and future work

The developed system performing Serpent algorithm encryption and decryption can be used in simulation and can be implemented on a FPGA chip. Depending on the usage, substantial improvements of the design can be made in order to increase performance and provide additional flexibility.

Currently, this NoC implementation can be used for encryption and decryption on chip, but not at the same time. Only a single module can be connected to the net as a PE. This can be quite inconvenient for a system that requires both functionalities of encryption and decryption, as it would require two chips, meaning increased space and power consumption. This disadvantage could be overcome by adding the decryption and encryption mode and altering the PE module to be able to perform both encryption and decryption, depending on the mode. It is reasonable to expect that the module with dual mode would take up more memory than each of the single mode modules (Encryptor and Decryptor), but this would allow a single chip to perform encryption and decryption, without the need to reprogram it in between.

In regards to the power consumption, the general goal for embedded and FPGA system design is to minimise it. Bearing in mind that this design performs encryption and decryption, secondary power consumption requirements need to be established. These requirements mostly regard protection of the cipher from the so called side channel attacks. Side channel attacks on ciphers aim at determining information about the plaintext or the key by measuring physical parameters. This work is mostly concerned by power consumption side channel attacks that measure and analyse power consumption. Some attacks of that nature have proven to be successful. Therefore, power consumption analysis of this implementation and determining the possibility of power consumption side channel attack is called for.

Even though some systems use fixed keys for encryption, some prefer the flexibility of using multiple different keys on the same chip. Current implementation does not allow that without reprogramming, seeing that the keys are stored in the Encryption and Decryption modules. One way to acheve this is to load the key alongside the data, distribute the key to all PEs and then continue using the net as described in the previous chapters. This does increase the overall net traffic, but it is a small price for the flexibility it offers.

Alongside with the already mentioned possible improvements, the revision of OpenNoc, Encryption and Decryption modules is Bakcalled for, as it is further elaborated in the section 5.1.4.

6. Conclusion

This work has presented the novel concept of Network-on-Chip and the improvements it brings for the intrachip communication. The main virtue of the Network-on-Chip architecture is scalability and acceleration, which are one of the central problems of modern architectures and computation. Adding more elements to the chip has an insignificant impact on the NoC performance. The NoC implementation used in this work is OpenNoc.

This work accelerated the Serpent crypto-algorithm encryption and decryption algorithms. Serpent is a substitution-permutation network operating on 128-bit blocks. Its design is highly befitting for the hardware implementation. Serpent encryption and decryption algorithms are implemented as Verilog modules and placed into the OpenNoc as processing elements. Number of processing elements can be altered for the specific purpose and expected data size.

This construct can perform encryption and decryption of data of various length. The architectural approach of NoC has proven to be superior to sequential execution, given that data is big. Seeing that plaintexts for encryption are mostly big, encryption processing elements fully exploit the potential of NoC.

BIBLIOGRAPHY

[1] Fpga fundamentals. `https://www.ni.com/en-rs/innovations/white-papers/08/fpga-fundamentals.html`.

[2] Fpga implementation. `https://digitalsystemdesign.in/fpga-implementation-step-by-step/`.

[3] Fpga – configurable logic block. `https://blog.digilentinc.com/fpga-configurable-logic-block/`.

[4] Module instantiation in verilog. `http://www.vlsifacts.com/module-instantiation-verilog/`.

[5] Programmable logic puts mcu-based designs on the fast track. `https://www.mouser.in/applications/programmable-logic-fast-track/`.

[6] Serpent. `https://cryptography.fandom.com/wiki/Serpent`.

[7] What is an fpga. `https://blog.digilentinc.com/what-is-an-fpga/`.

[8] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A flexible block cipher with maximum assurance.

[9] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard.

[10] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent and smartcards.

[11] Ross Anderson, Eli Biham, and Lars Knudsen. The case for serpent. 2000.

[12] Eli Biham. Serpent nessie test vectors. `http://www.cs.technion.ac.il/~biham/Reports/Serpent/Serpent-256-128.verified.test-vectors`.

[13] Anantha Chandrakasan. Reconfigurable logic architectures - lecture notes. http://web.mit.edu/6.111/www/s2008/LECTURES/l12.pdf.

[14] H. Meijer, S. Tavares, and Liam Keliher. Linear cryptanalysis of substitution-permutation networks. 2004.

[15] Kuladeep Sai Reddy and Kizheppatt Vipin. Opennoc source code. https://github.com/kuladeepsaireddy/OpenNoc.

[16] Kuladeep Sai Reddy and Kizheppatt Vipin. Opennoc: An open-source noc infrastructure for fpga-based hardware acceleration. *IEEE Embedded Systems Letters*, 11(4):123–126, 2019.

[17] Bhupendra Singh, Lexy Alexander, and Sanjay Burman. On algebraic relations of serpent s-boxes.

[18] Frank Stajano. Serpent reference implementation. https://www.cl.cam.ac.uk/~fms27/serpent/, 1998.

[19] Synario Design Automation. *VHDL Reference Manual*, 1997. An optional note.

[20] Kizheppatt Vipin. Asyncbtree: Revisiting binary tree topology for efficient fpga-based noc implementation. *International Journal of Reconfigurable Computing*, 2019.

[21] Michael Weiner. Invasive attacks. https://www.ei.tum.de/en/sec/research/invasive-attacks/.

[22] Jiang Xu, Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen. Networks on chips: Structure and design methodologies. *Journal of Electrical and Computer Engineering*, 2012.

Višejezgrena implementacija kriptiranja korištenjem mreže na čipu

Sažetak

Ovaj rad opisuje nov pristup rješavanju problema izvođenja resursno intenzivnih izračuna u *hardware*-u - mreži na čipu. Mreža na čipu omogućuje višejezgreno izvođenje izračuna nad velikom količinom podataka. Korištena je javno dostupna implementacija mreže na čipu pod imenom OpenNoc. OpenNoc je dizajniran za implementaciju na tehnologiji FPGA. Kriptoalgoritmi su jedni od algoritama koji su predominantno konkurentni te su stoga prikladni za *hardware*-sku akceleraciju. Izrađena je implementacija procesirajućeg elementa koji izvodi enkripciju i dekripciju podataka kriptoalgoritmom Serpent. Procjena performansi obavljena je simuliranjem izvođenja na različitom broju jezgri u mreži na čipu. Također, predložena su poboljšanja koja se odnose na fleksibilnost implementacije, sigurnost te potrošnju memorijskog prostora i energije.

Ključne riječi: mreža na čipu, OpenNoc, procesni elementi, Serpent kriptoalgoritam, S-box, FPGA, Verilog, višejezgrenost

Multicore Implementation of Crypto-Algorithm Using Network-on-Chip

Abstract

This work describes a refined architectural approach to solving intensive parallel computation in hardware - a Network-on-Chip. Network-on-Chip allows for multicore processing of vast amounts of data in hardware. The implementation of NoC used is an open source named OpenNoc, designed for the FPGA technology. Crypto-algorithms perform predominantly concurrent computation and therefore benefit of hardware acceleration. The work provided implementation of custom processing elements performing the Serpent crypto-algorithm encryption and decryption. Performance was assessed in simulation of the NoC with various numbers of cores. Several improvements regarding the implementation flexibility, security, power and memory consumption are suggested.

Keywords: Network-on-Chip, OpenNoc, Processing Element, The Serpent Crypto-Algorithm, S-box, FPGA, Verilog, Multicore