**1 – SPECIFICATION ANALYSIS**

* 1. **Introduction**

The goal of the project is to design a simple asymmetric encryption system. Asymmetric encryption is also called public key encryption, but it actually relies on a key pair: two mathematically related keys, one called the public key () and another called the private key or secret key (), are generated to be used together. The private key is never shared, it is kept secret and is used only by its owner. The public key is made available to anyone who wants it. Because of the time and amount of computer processing power required, it is considered “mathematically unfeasible” for anyone to be able to use the public key to re-create the private key, so this form of encryption is considered very secure.

Immagine che contiene testo

Descrizione generata automaticamente

Figura - Function parameters

The asymmetric encryption system has three functions:

1. Generation of the key-pair

The *Key-pair generation* function is described by this formula: . Given a secret key (, generated in a random way and having a value between 0 and 1, it generates the corrisponding public key (.

1. Encryption

The *Encryption* function is described by this formula: . Given each character of the plaintext, the formula above is used to encrypt it.

1. Decryption

The *Decryption* function is described by the formula: Given each character of the chipertext, the formula above is used to decrypt it.

Immagine che contiene testo

Descrizione generata automaticamente

Figura - Scheme of the algorithm

In figure 2 is possible to observe how the algorithm works. We have two entities, each with a key-pair ( and ). To encrypt the message an entity uses the public key of the other entity, whereas for the decryption the entity uses its own secret key.

In our case we have two entities, W and J, that want to communicate. The first step is the key-pair generation, made by each of the entites. Once they obtained the pair they have to exchange the public keys.

At this point W wants to send a message to J. W encrypts each character of the plaintext, in order to obtain the ciphertext, using the public key of J and then sends it in encrypted mode.

J receives W’s message and decrypts it using his secret key. In the case he wants to replay to the message, so J does the same thing as W but he uses the public key of W, and then sends the message. When W receives the message he decyrpts it using his secret key.

1. **– BLOCK DIAGRAM AND DESIGN CHOICES**

**2.1 Design architecture**

**Immagine che contiene testo

Descrizione generata automaticamente**

Figura - Module SAE

* **Input *clk****:* it’s the clock that allows the syncrhonization of the ports;
* **Input *rst\_n****:* it’s the signal for asyncrhonous reset that allows to trigger other signals without waiting for the clock;
* **Input [1:0] *mode****:* allows the selection of the mode that we want to use, that can be encryption, decryption or key generation;
* **Input [7:0] *data\_input****:* represents the input provided by the user, that can be either the plaintext or the ciphertext, depending on the mode selected;
* **Input [7:0] *key\_input****:* represents the input provided by the user, that can be either a secret key or a public key, depending on the mode selected;
* **Input *inputs\_valid****:* indicates when the inputs are valid and can be sampled by the module;
* **Output reg [7:0] *data\_output****:* represents the output of the module, that can either be the plaintext or the ciphertext depending on the selected mode;
* **Output reg *output\_ready****:* indicates when the output is ready;
* **Output *err\_invalid\_ptxt\_char****:* indicates that the plaintext is not valid;
* **Output *err\_invalid\_seckey****:* indicates that the secret key is not valid;
* **Output *err\_invalid­\_ctxt\_char****:* indicates that the ciphertext is not valid.

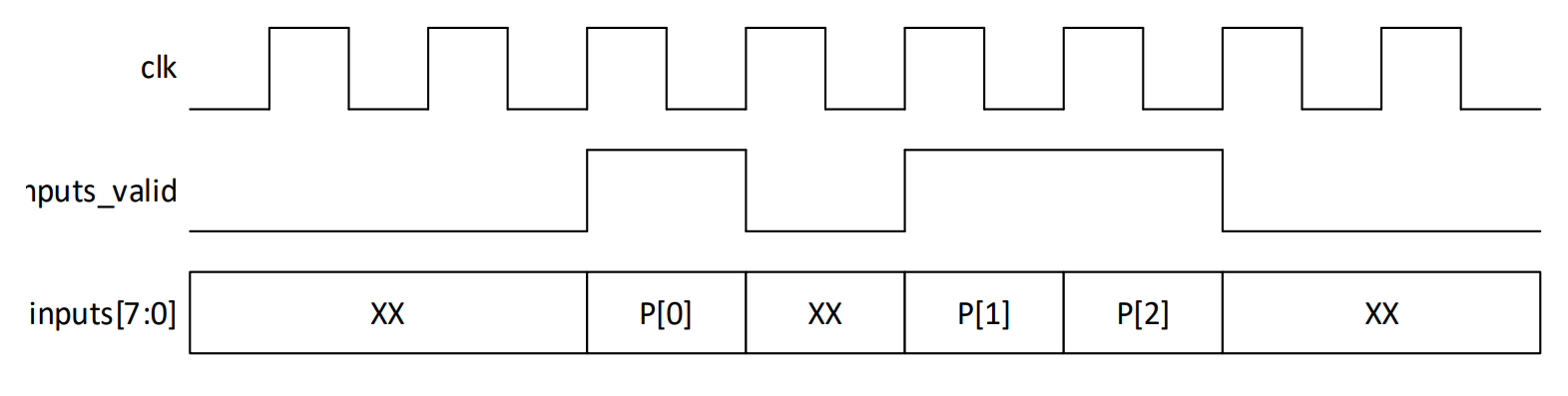
****

Figura - Valid input

The system has an input port that is asserted when the inputs are valid, and in the figure 4 is represented the expected waveform.

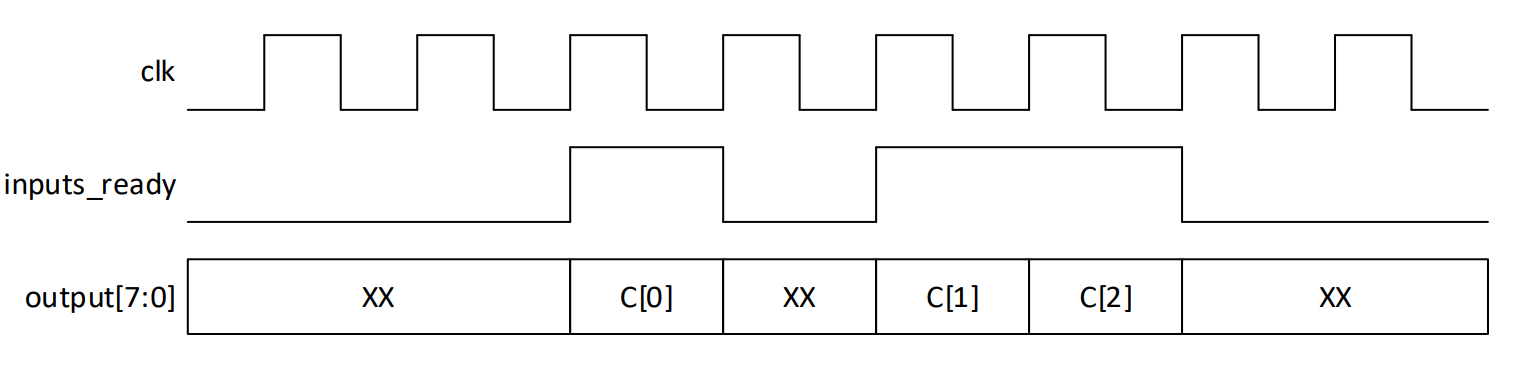
****

Figura - Valid output

The system has an output that is asserted when the output is available and is correct, and in the figure 5 is represented the expected waveform.

**2.2 Block diagram**

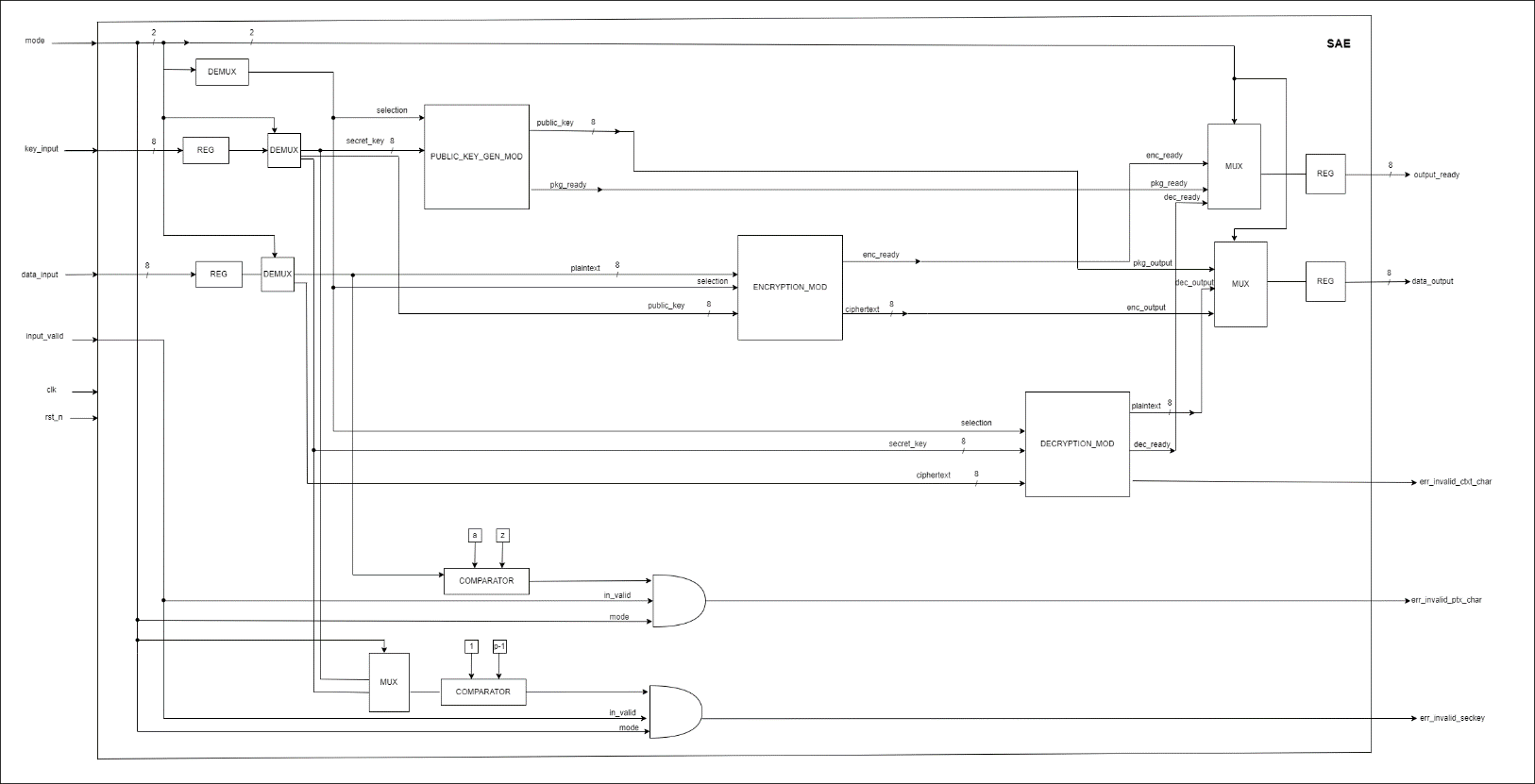
****

Figura - Block diagram of SAE

The input signals of the SAE are:

* ***Mode***: used to select which of the three blocks to activate and also used as the selection signal for the mux/demux;
* ***Key\_input****:* is the input that can be used as secret key or as public key, according to the block that is selected. This is also used as the input for the comparator that manages the *err\_invalid\_seckey*;
* ***Data\_input****:* is the input that can be used as plaintext or as chipertext, according to the block that is selected. This is also used as the input for the comparator that manages the *err\_invalid\_ptxt\_char*;
* ***Input\_valid****:* used to manage the errors.

The output signals of the SAE are:

* ***Output\_ready:*** used to show that the operation of the sub-modules is terminated in a correct way;
* ***Data\_output:*** gives the results of the sub-modules, that are public\_key or the chipertext or plaintext, according to the module that is executed;
* ***Err\_invalid\_ctxt\_char:*** indicates that there is an error during the decryption module;
* ***Err\_invalid\_ptxt\_char:*** indicates that there is an error in the plaintext (such as characters that aren’t in a specific range);
* ***Err\_invalid\_seckey:*** indicates that the secret key is not in a specific range.

**2.2.1 SAE inputs description**

The input *mode* has different roles. It is used to select the different type of function that can be executed and it passes throw the demux taking the name of *selection.* It is also the selection input of the mux and demux that are used to generate the outputs of the SAE: *output\_read* and *data\_output.*

*Key\_input* is stored in the register in order to be available during all the phases of the execution. This value is passed in the demux and according to which function is activated, based on the value of *mode*, it can select as input: the *secret\_key* if is selected the **public\_key\_gen\_mod** or the *public\_key* if is selected the **encryption\_mod**.

*Data\_input* is stored in the register, in order to be available during all the phases of the execution. This value is passed in the demux and according to which function is activated, based on the value of *mode*, it can select as input: the *plaintext* if is selected the **encryption\_mod** or the *ciphertext* if is selected the **decryption\_mod**.

*Input\_valid* is used to verify if the inputs are valid and comply with the correctness checks.

**2.2.2 SAE internal blocks description**

The SAE is internally composed by three blocks that are: **public\_key\_gen\_mod**, **encryption\_mod** and **decryption\_mod**. In the following section we are going to analyze them in detail.

Immagine che contiene testo

Descrizione generata automaticamente

Figura - public\_key\_gen\_mod scheme

The **public\_key\_gen\_mod** has two inputs: *selection* and *secret\_key*. The *secret\_key* is a variable composed of 8 bits that represents the secret key of the user, and it’s generated by a python script using a pseudo-random algorithm. This function produces two outputs that are: *public\_key* that is the key generated for the user composed of 8 bits, and *pkg\_ready* that, when asserted, indicates that the operation has terminated in a successfull way.

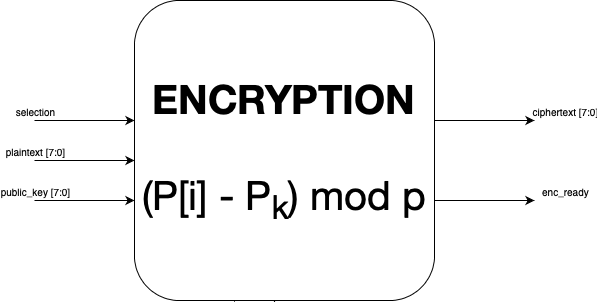


Figura - encryption\_mod scheme

The **encryption\_mod** has three inputs: *selection, public\_key* and *plaintext.* The *public\_key* input represents the public key of the receiver whereas the *plaintext* is a variable composed of 8 bits. This function produces two outputs that are: *ciphertext* and *enc\_ready* that, when asserted, indicates that the operation has terminated in a successfull way.

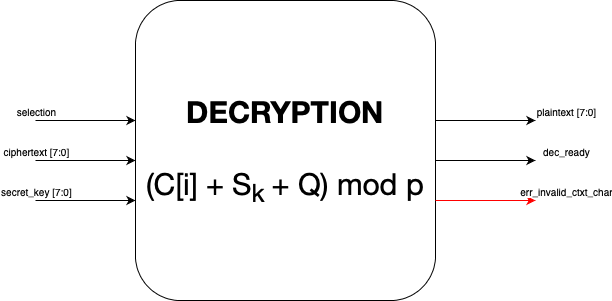


Figura - decryption\_mod

The **decryption\_mod** has three inputs: *selection, ciphertext* and *secret\_key.* The *secret\_key* input represents the private key of the user and is a variable composed of 8 bits and the *ciphertext* is a variabile composed of 8 bits and represents the text in an encrypted form. This function produces three outputs that are: *plaintext, dec\_ready* that if asserted indicates that the operation has terminated in a successfull way, and *err\_invalid\_ctxt\_char* that when asserted indicates that there has been error during the execution of the function.

**2.2.3 SAE output description**

*Output\_ready* is the result of a mux that has as inputs *enc\_ready, pkg ready* and *dec\_ready* and has *mode* as selection input. The output of the mux is passed to the register and is propagated in output as a variable, composed of 8 bits, that indicates if the executed function required by the user is terminated in a successfull way.

D*ata\_output* is the result of a mux that has as inputs *pkg\_output, dec\_output* and *enc­\_output* and has *mode* as selection input. The output of the mux is passed to the register and is propagated in output as a variable, composed of 8 bits, that indicates the output of the function executed that can either be the plaintext, the ciphertext or the public key generated.

E*rr\_invalid\_ptxt\_char* is obtained by putting *input\_valid* in an AND port with the output of a comparator. This comparator verifies if the characters that compose the plaintext are in a correct range.

*Err\_invalid\_seckey* that is obtained by putting *in\_valid* (that represent input\_valid) in an AND port with *mode* and with the output of a comparator. This comparator verifies if the value of the key is in the range [1, p-1]. The key is selected using a mux that can choose the secret\_key used for decryption\_mod or the secret\_key user for the public\_key\_gen\_mod.

1. **– EXPECTED WAVEFORMS**

In this section there are several waveforms, one for each step of the algorithm.

(mettere le onde della fase di generazione chiave, enc. e dec. e poi alla fine quella generale per far vedere il tutto)

(descrizione delle varie forme d’onda)

1. **– TESTBENCH**
2. **– IMPLEMENTATION OF RTL DESIGN ON FPGA AND RESULTS**

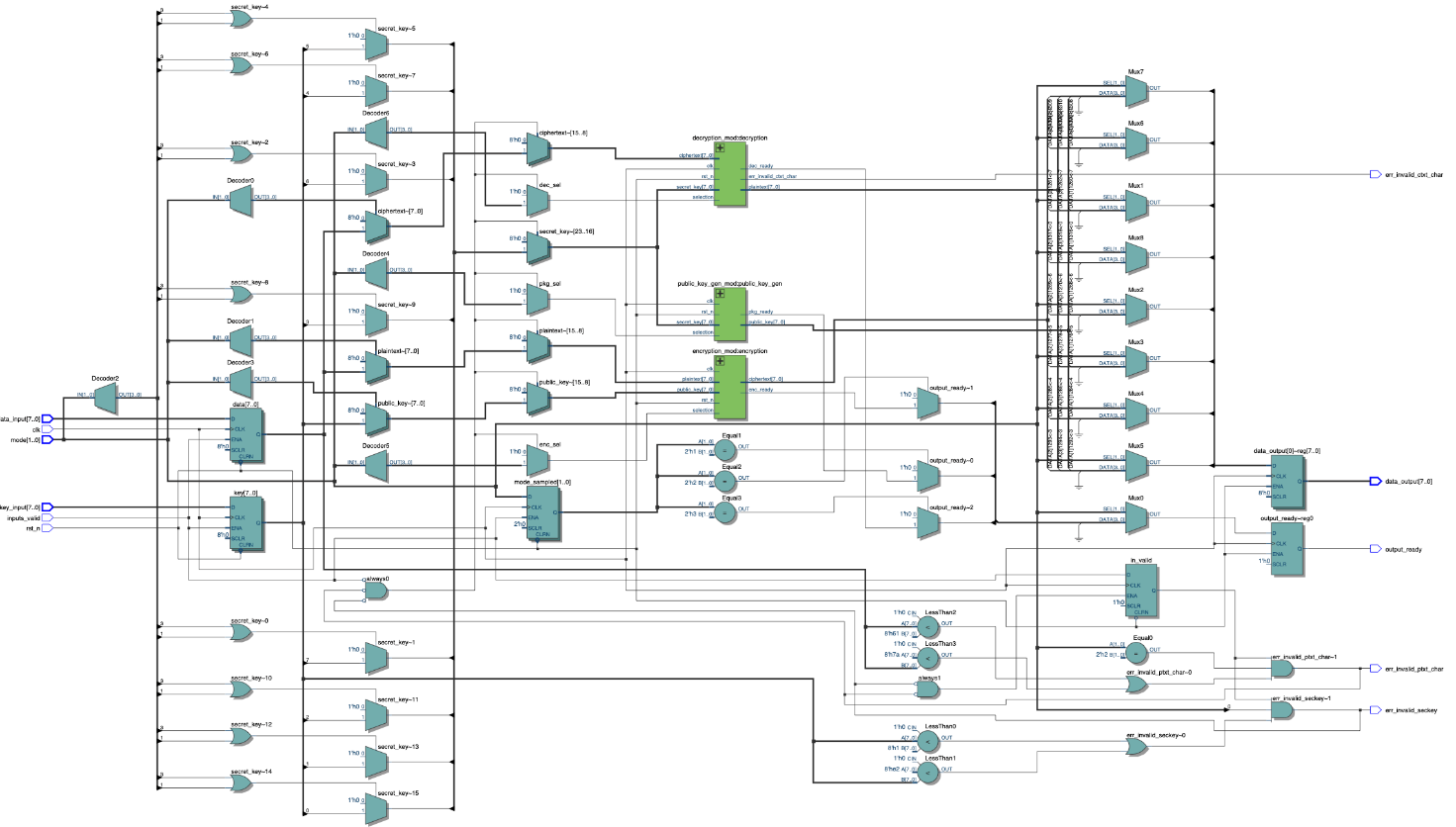
****

Figura – RTL schema

The figure represent the RTL (Register Transfer Level) design of the simple asymmetric encryption algorithm that allows us to view the digital components in an abstract way.

This design is defined by HDL (Hardware Description Language) and in particular with Verilog.

Inserire snap dei summary su quartus e commentarli

1. **– STATISTIC TIMING ANALYSIS (STA)**

Static Timing Analysis is a very important step of the design flow because it allows to check if the implemented module behaves in the correct way, respecting some additional requirements and constraints. The time constraints related to input and output signals of the implemented module, have been written on *constr.sdc* file.

* 1. **Synopsys Design Constraints File**

STA divides the design into timing paths and calculates the signal propagation delay. The aim is to understand if the constraints applied to input and output are violated or not.

On each path we have a *starting point* where the data are launched by a clock edge or where there is the imposition of specific requirements, and for this reasons there is a register clock, or an input port. Then there is a *combinational logic network* that represents the elements without internal states or memories. The last component of the path is the *endpoint* that rapresents the point in which a data should be available at a specific time or a clock edge that captures the data and it has to be an output port or a pin.

Immagine che contiene testo

Descrizione generata automaticamente

Figura - constr.sdc

INSERIRE SCREEN FATTO DA QUARTUS AL POSTO DI FIG.11

In the figure 11 we can see the *constr.sdc* file that contains the time constraints developed for our analysis.

In the first line of the code there is the command ***create\_clock*** that has name *clk* and has the role of creator for the clock object, that has a period of 10 ns, and it’s linked to the input port of the module that concerns the clock signal.

In the third line of the code there is the command ***set\_false\_path*** which specifies that the reset signal doesn’t have to be restricted with the time constraints because it is an asynchrnous signal and it’s indipendent from the clock signal during the activation.

In the last lines there are two commands: ***set\_input\_delay*** and ***set\_output\_delay.*** These commands represent specific constraints on the input and on the output ports, because those can have delays. For both the input and the output ports are specified minimum and maximum delay: the minimum is set to be 10% of the clock period whereas the maximum is set to be 20% of the clock period.

In our specific case, if we leave this parameters we have bad results because INSERIRE PERCHE’ NON POSSIAMO LASCIARE I PARAMETRI CHE CI SONO IN FIGURA…. , for this reason we have to change the clock period, from 10 to 30. Doing so the parameter of min/max change into 3 and 6 because the minimum has to be the 10% of the clock period whereas the maximum has to be the 20% of the clock period.

* 1. **Virtual Pins**

PRENDERE FIGURA DA QUARTUS RELATIVA AI VIRTUAL PINS

The figure shows us that both the input and the output ports have been specified as virtuals, whereas the clock signal is not specified as virtual because it’s a physical signal that is received from the outside.

* 1. **Analysis of frequencies**

INSERIRE IMMAGINI DELLE FREQUENZE MASSIME, PRESE DA QUARTUS & COMMENTARLE