Decryption

The project consists of the implementation of three decryption modules: Caesar, Scytale and Zigzag, a multiplexer and a demultiplexer, a register bench, as well as making connections between them in a top module, functioning as a system for decrypting input data.

**Task 1 – Implementation of the registry bank**

* The registry bench module receives at the input:
* address of the register from which you want to read or in which you want to store data (addr)
* write signal that has the value of 1 when you want to write in the registry
* read signal that has a value of 1 when you want to read the value in the register
* data to be stored in the register if the address is valid (wdata)
* The module has as outputs:

- the done signal that is put on 1 in the next clock cycle if a value has been stored or read or if an error has been encountered during these operations (if the address received has not been valid)

-error signal which is 1 when a corresponding address has not been received

-data read from the register (rdata)

-decryption keys for each registry

*Note*: The read signal cannot be raised at the same time with the write signal, i.e. only one operation is allowed at a time.

* I checked if the reset signal is activated, if so the registry values are brought to the corresponding reset values.
* If the writing signal is raised then the validity of the address is tested with a case instruction.
* And if read access is flagged, then the value of the address is also tested in this case.
* In both situations, the default case (address is not among the 4 valid) signals an error.

**Task 2 – Deploy decryption modules**

* **Caesar cipher** – uses a key to decrypt a key that subtracts from the value of the input data.

-For this module I used a synchronized sequential block on the positive edge of the clock signal.

-The busy signal is always 0, and the characters are received one at a time, decrypted and sent out in the next clock cycle.

* **Scytale cipher –** its key contains in the first half the number of N lines, and in the second the number of M columns for the matrix formed to be able to display the decrypted data.

-The received data is written on "columns" and after reading on lines for decryption.

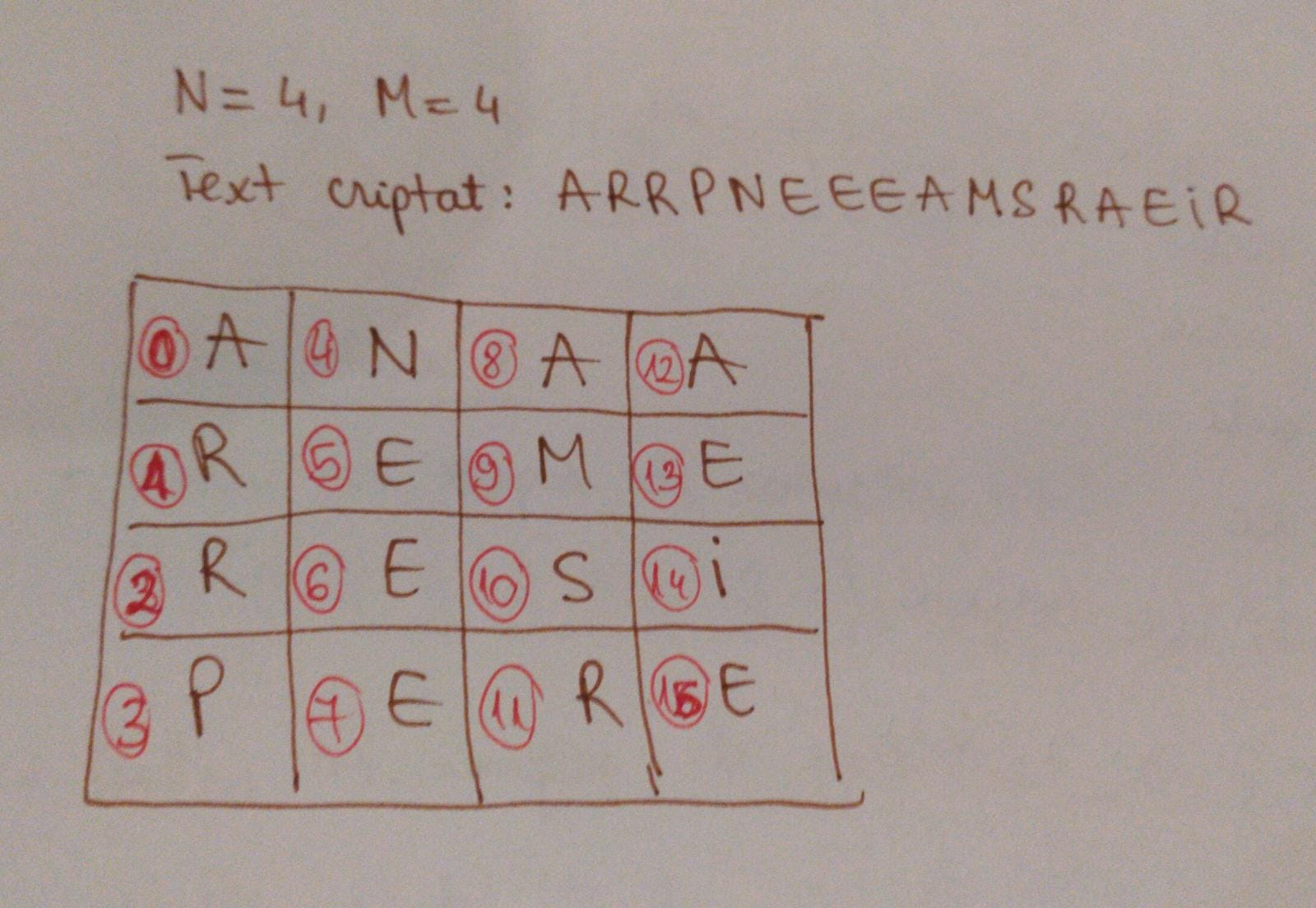
- I've only used a sequential block for this module. When the valid\_i is raised and the busy is not 1, the data is read one character at a time and the values are put in a vector.

-When it receives the special character of beginning decryption (0xFA), busy becomes 1, following that the valid\_o to be raised in the next clock cycle.

-I used a counter for vector storage, one for lines, one for columns and one for displaying data on the output.

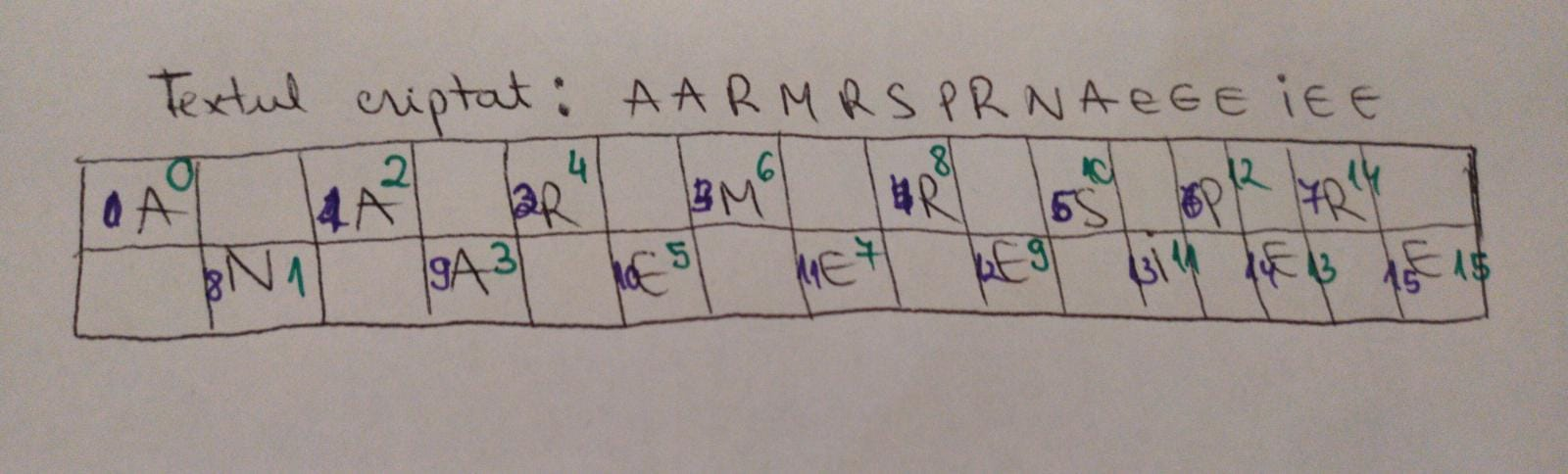
-Decryption works in the following mode: Display the first character read, then increment the display counter with the number of lines, to move to the second character on the first line (characters being stored on columns) and continue increment until the number of columns has been exceeded, that is, until the first line has been displayed in its entirety, then proceed to line 2 using the same principle of increment , except that the initial index on line 2 is 1, and on line 3 2 and so on. Repeat until all lines are completed.

In the following example in the image it is noted that the correct display order for the first line is 0, 4, 8, 12, following that on line 2 to start from index 1 with the same increment.



* **Zigzag cipher –** its key specifies the number of lines on which characters are stored, which read in zigzag shows the deciphered text.
* For **Key 2**

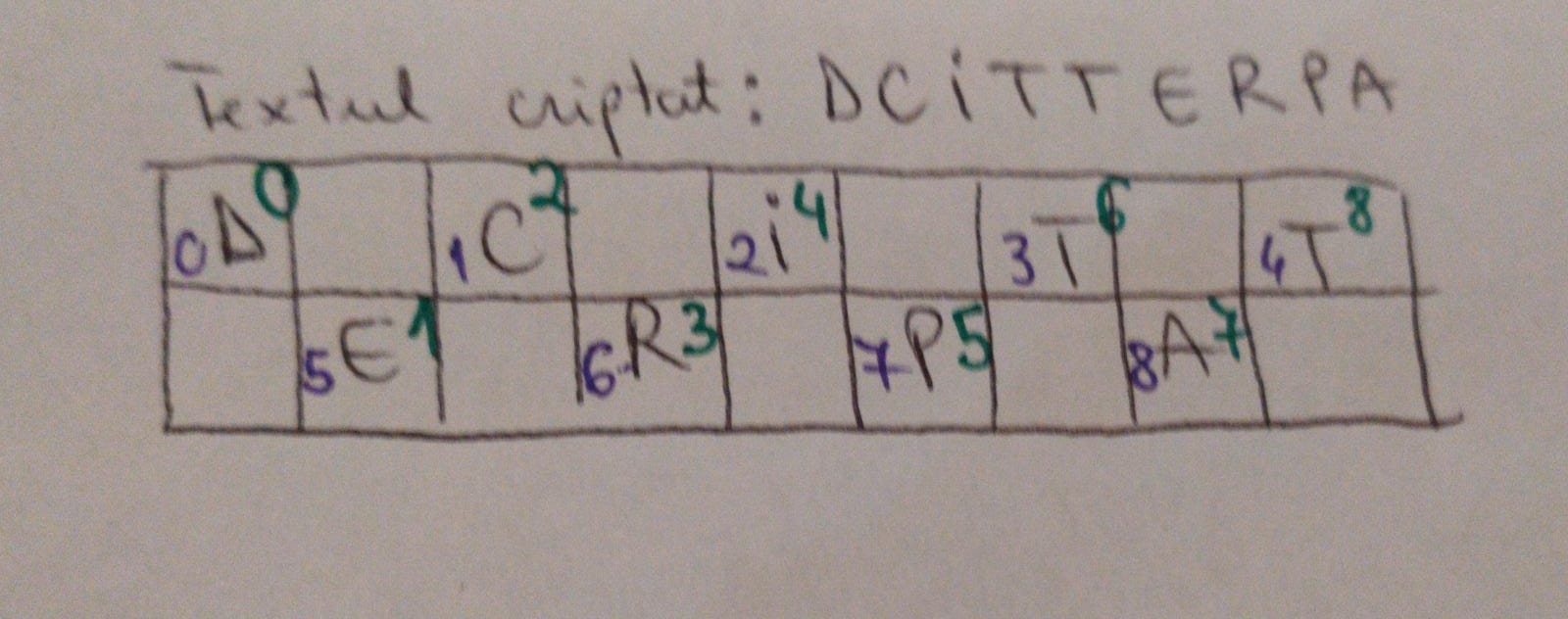
-For **even number of characters**



In the image above, it is noted that to display the characters in the order corresponding to the decryption is taken a character on the first line, after which the index increments by half the number of characters (number of characters shifted to the right by 1 bit) to move to the next character that is on line 2, after which it returns to line 1 , with index 1 and increments again to pass on line 2 and so on until all characters have been displayed.

Specifically, in the example above the display order is 0, 8, 1, 9... 7, 15. A number of sets of 2 characters is formed equal to the half of the number of characters, i.e. 8 sets, the number of catacts being 16.

-For **odd number of characters** the same principle is used, only that the index will be incremented by the half of the number of characters +1 and will be a number of sets of 2 characters equal to the amount of the division of the number of characters to 2 and will remain separated the middle character, it will be displayed last.



In the example above are 9 characters, so increment will be done with (9/2) +1 = 4+1=5

The display order will be 0, 5, 1, 6, 2, 7, 3, 8, and the character with index 4, i.e. the 5th, the one in the migame, is last displayed.

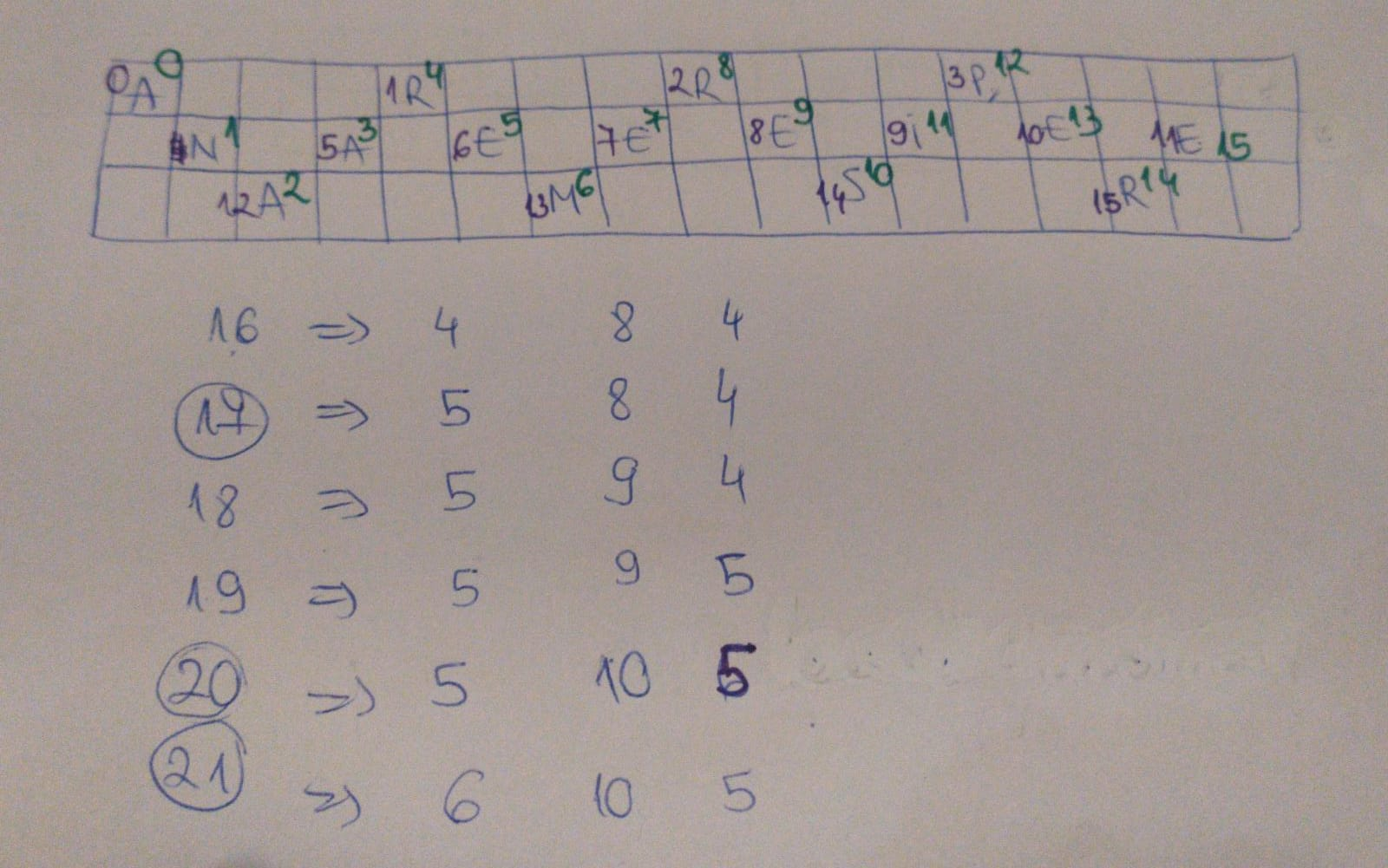
* For **key 3**, we found 4 distinct casesdepending on the divisibility of the number of carats with 4: case 4k characters, 4k+1, 4k+2, 4k+3.

The case in which it falls is checked by dividing the number of characters by 4 (shifted to the right by 2 bits) and after multiplied by 4 (shifted to the left by 2 bits), if the result is equal to the number of characters then the number of characters is multiple of 4, i.e. falls in the case of 4k characters.

For other cases the number of characters is equal to the number divided by 4 and multiplied by 4 added by 1, 2 or 3.

-For this case with key 3, after all data is stored in the vector, it is placed in good order in another auxiliary vector, in a combination block, after which the display is made in order of the auxiliary vector.

I noticed that on the first line the index for decrypted data is incremented by 4, starting with index 0, on line 2 indecsii are incremented with 2, starting with 1, and on line 3 are incremented with 4, starting with 2. The number of steps for each line depends on where the number of characters falls. (can be seen under the conditions of each instruction for each case)



Under the table in the image above, I took a few cases of number of characters and I wrote how many characters are on the first line, on the 2nd and on the 3rd.

More detailed explanations on how the algorithm works are written in the comments in the Zigzag module code.

**Task 3 – Implementation of mux, demux and decryption\_top modules**

**Demux**module:

-Receives input data when valid\_i is 1 and depending on its select value, gives the data to the output interface corresponding to the module specified by the select: 0 for Caesar, 1 for Scytale and 2 for Zigzag.

- The master who sends the encrypted data works on a clock 4 times slower (clk\_mst), so it will send the information in groups of four characters (4x8 bits), sent in 4 consecutive clock cycles.

-Each input value has 32 bits, so they will be sent in 4 clock cycles in the following order: first part [31:24], then [23:16], [15:8] and then [7:0].

-In the sequential block synchronized on the clk\_mst values are given to the valid outputs out, and in the other output data.

**Mux module:**

-It selects what value the data will take out and valid lysting out depending on its select value. Specifically, if select is 0 will be given to the output of Caesar mode, if it is 1 those of the Scytale module, and for the value 2 those of the Zigzag module.

**Module Decryption\_top:**

-In this module are instantiated all other component modules and connections are made.

-The busy signal is 1 when any of the busy signals of the decryption modules is 1.