# Tema 2: Decryption

Responsabili: Sergiu Dudă
Deadline: Detalii în secţiunea Deadline
Data publicării: 29.11.2020
Data ultimei actualizări: 29.11.2020
Istoric modificări:

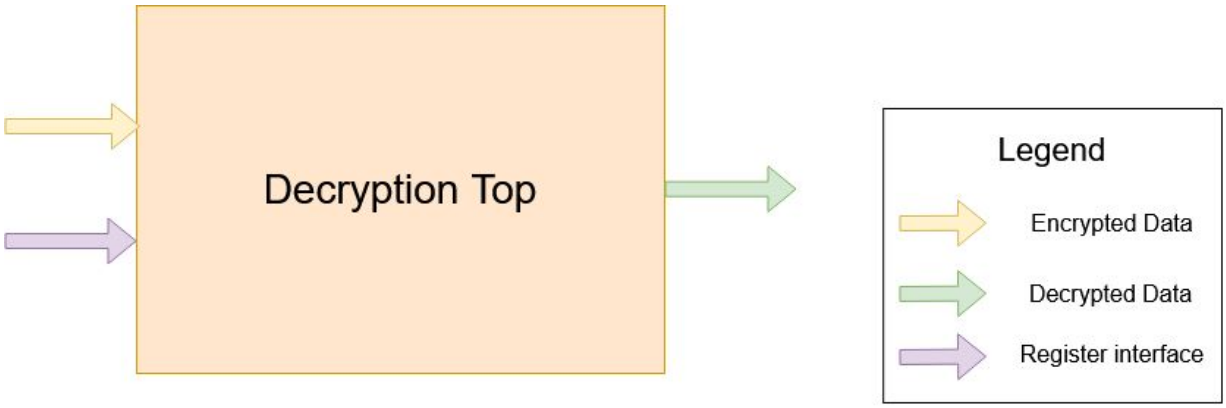
- Versiunea iniţială

## Obiective

Tema are ca scop familiarizarea cu noţiunile limbajului Verilog studiate în cadrul primelor laboratoare: module, construcţii de limbaj, circuite secvențiale, prin:

- Implementarea unor algoritmi de decriptare
- Implementarea unui banc de registre
- Implementarea unui multiplexor/demultiplexor
- Realizarea conexiunilor între module

# Descriere




Implementați un **circuit secvenţial** în Verilog care primește la intrare date criptate și va scoate pe ieşire informaţia decriptată. Modulul va implementa 3 algoritmi de decriptare: Cezar, Scytale și ZigZag. Pe interfaţa de acces la registre se va configura ce algoritm de decriptare va fi folosit și care e cheia de decriptare.

## Cifrul Cezar

Cifrul Cezar[1] este un cifru simplu bazat pe adunarea unei constante fiecărui caracter din propoziție cu o constantă N.

Exemplu pentru N = 3

Textul criptat: DQD DUH PHUH VL SHUH
Textul decriptat: ANA ARE MERE SI PERE

Pentru criptare se execută o adunare cu 3, pentru decriptare se execută o operație de scădere cu 3.

Cheia de decriptare este numărul cu care fiecare caracter a fost shiftat, si anume **N.**

## Cifrul Scytale

În antichitate, pentru decriptarea cifrului Scytale[2] era nevoie de un cilindru în jurul căruia se înfășura o bucată de pergament pe care era scris mesajul. Dacă cilindrul avea dimensiunea potrivită, literele se aliniau și mesajul devenea vizibil.

Acest lucru este echivalent cu a popula o matrice de NxM cu caracterele din mesajul criptat. Matricea este populată pe coloane, apoi mesajul poate fi citit pe linii.

Exemplu pentru N = 4, M = 4
Textul criptat: ARRPNEEEAMSRAEIR

| | | | |
|---|---|---|---|
| A | N | A | A |
| R | E | M | E |
| R | E | S | I |
| P | E | R | E |

Textul decriptat: ANAAREMERESIPERE

Cheia de decriptare este dimensiunea matricei: **N si M.**

## Cifrul ZigZag

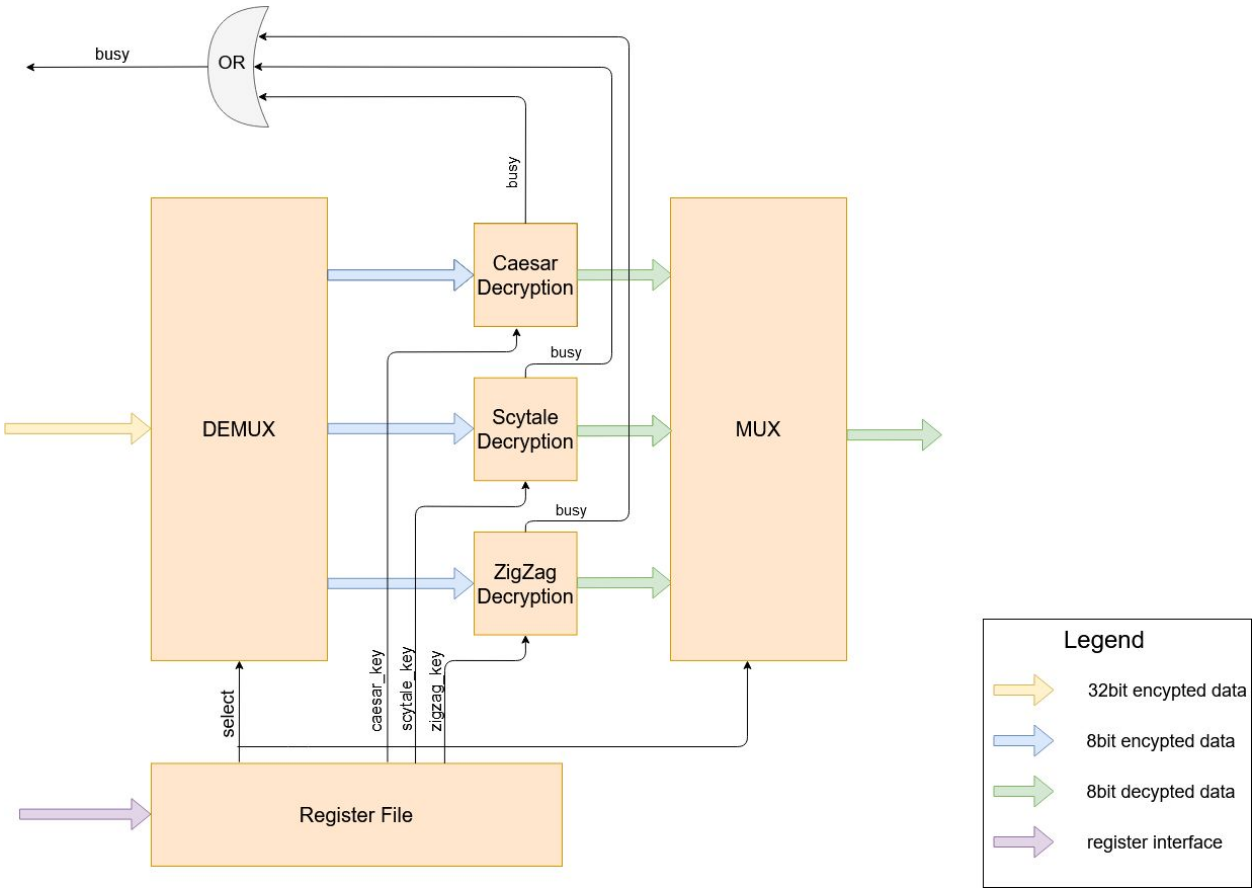
Criptarea cu cifrul ZigZag[3] presupune scrierea mesajului în formă de zig-zag, pe mai multe linii. Textul criptat se obţine citind literele în ordine de pe fiecare linie. Pentru decriptare este necesar să se cunoască numărul de linii pe care a fost criptat mesajul.

Textul criptat: ARRPNAEEEIEEAMSR

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | | | | R | | | | R | | | | P | | | |
| | N | | A | | E | | E | | E | | I | | E | | E |
| | | A | | | | M | | | | S | | | | R | |

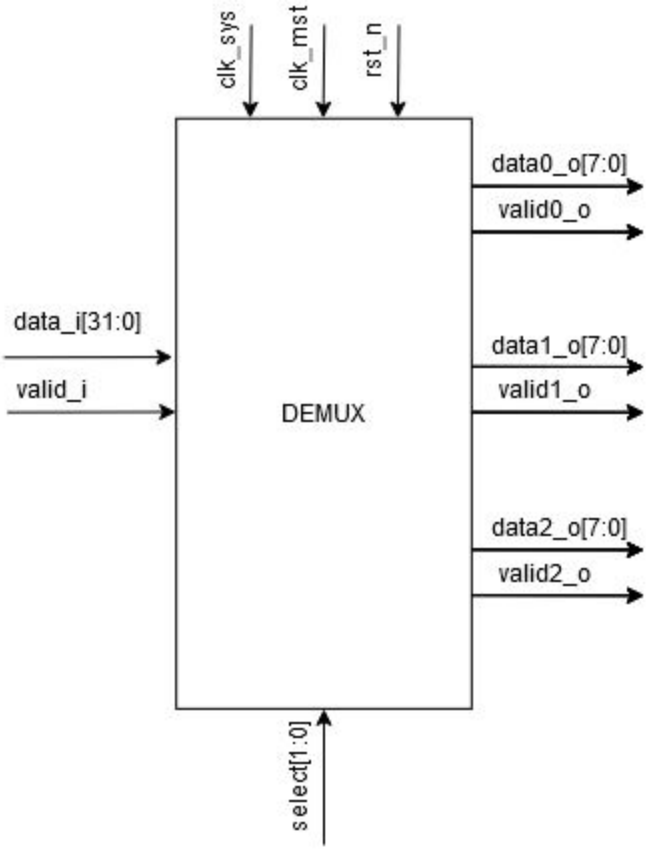
Textul decriptat: ANAAREMERESIPERE.

# Implementare



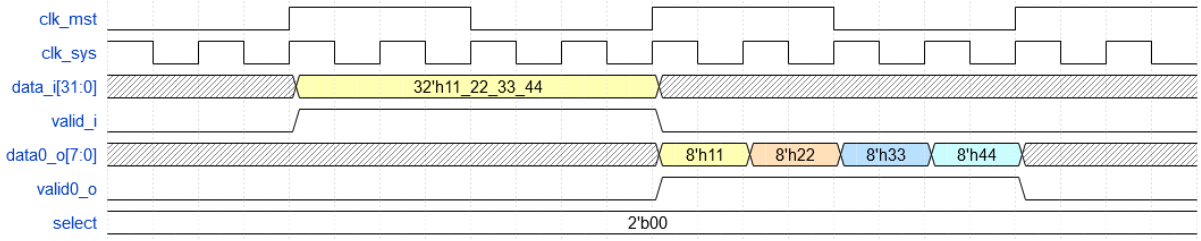

Fiecare algoritm de decriptare va fi implementat într-un modul conform figurii de mai sus. Pentru a ruta datele de intrare către modulul de decriptare dorit se va folosi un **demultiplexor**. Un **multiplexor** va fi folosit pentru a selecta ieşirea corectă. Dacă un modul este ocupat să decripteze datele primite, acesta va ridica semnalul “busy”. Cât timp semnalul busy e pe 1, nu se vor primi date criptate din exterior.

# Demultiplexor

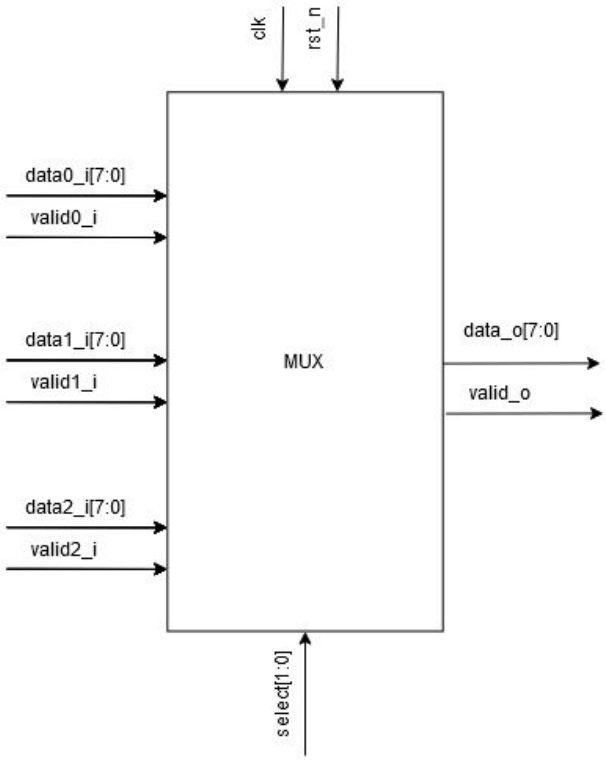



Demultiplexorul va ruta datele primite de pe interfaţa de intrare către interfaţa de ieşire corespunzătoare în funcţie de valoarea semnalului *select***.** Masterul care trimite datele criptate funcţionează pe un ceas de 4 ori mai lent (clk_mst) aşa că va trimite informaţia în grupuri de câte patru caractere (4x8 biti). Sistemul nostru va serializa datele şi le va trimite pe interfaţa corespunzătoare în 4 cicli de ceas consecutivi conform figurii de mai jos. Datele sunt considerate valide şi vor fi rutate către ieşirea corespunzătoare **doar dacă** semnalul *valid* este 1.

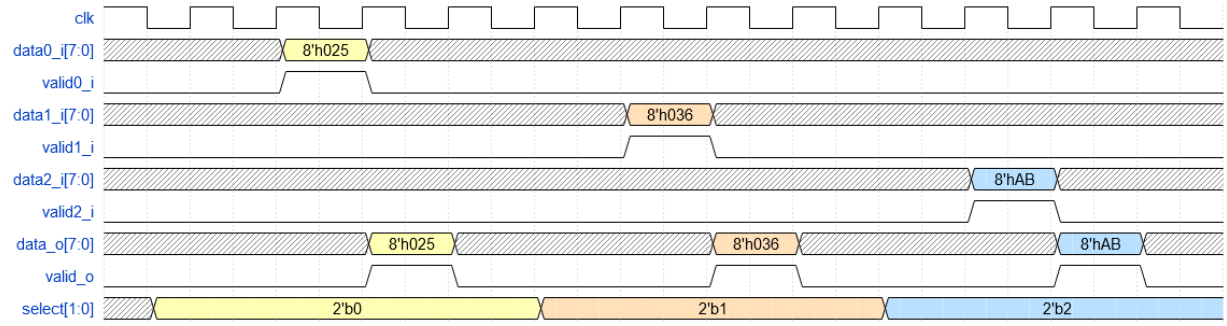
Observație! Ceasurile clk_mst şi clk_sys sunt sincronizate.



# Multiplexor



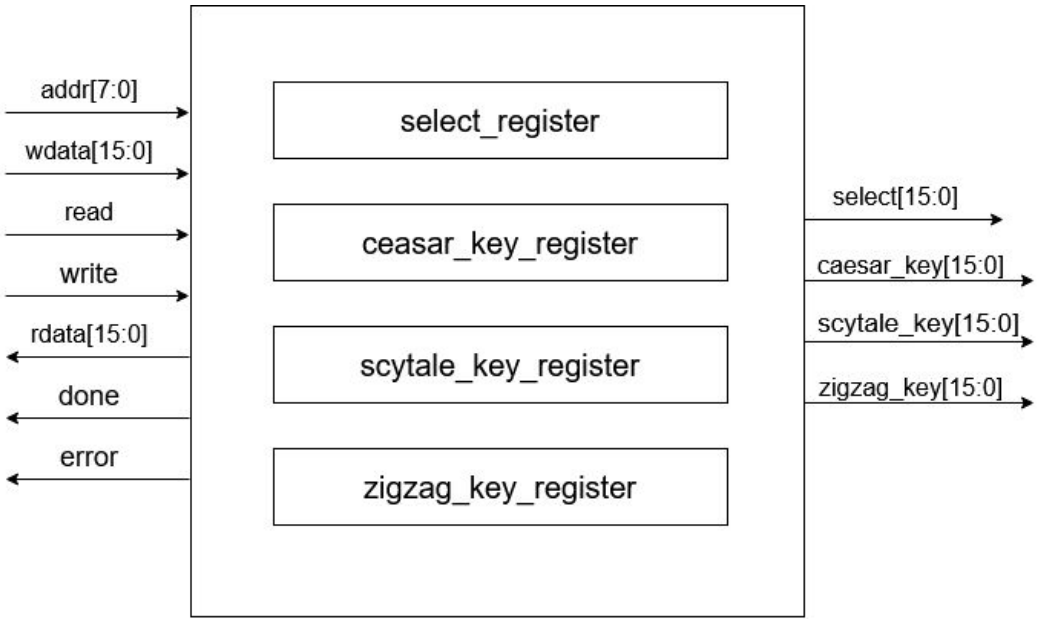

Multiplexorul va primi datele de la fiecare modul de criptare şi în funcţie de semnalul *select*, va ruta informaţiile către ieşire. Multiplexorul va funcționa doar pe ceasul sistemului (clk_sys).



# Bancul de registre

Bancul de registre conține 4 registre de 16 biți conform tabelului de mai jos

| Nume registru | Adresă | Valoare de reset | Funcționalitate |
|---|---|---|---|
| select_register | 0x0 | 0x0 | Sunt folosiţi doar biţii [1:0]<br>Orice scriere a biților [15:2] este ignorată. |
| caesar_key_register | 0x10 | 0x0 | Conţine key de decriptare pentru cifrul Cezar |
| scytale_key_register | 0x12 | 0xFFFF | biții [15:8] conţin numărul de linii N<br>biții [7:0] conţin numărul de coloane M |
| zigzag_key_register | 0x14 | 0x2 | Conţine numărul de linii necesare pentru decriptarea cifrului ZigZag |




Valoarea stocată în fiecare registru este assignată asincron către ieşirile select, caesar_key, scytale_key şi zigzag_key.

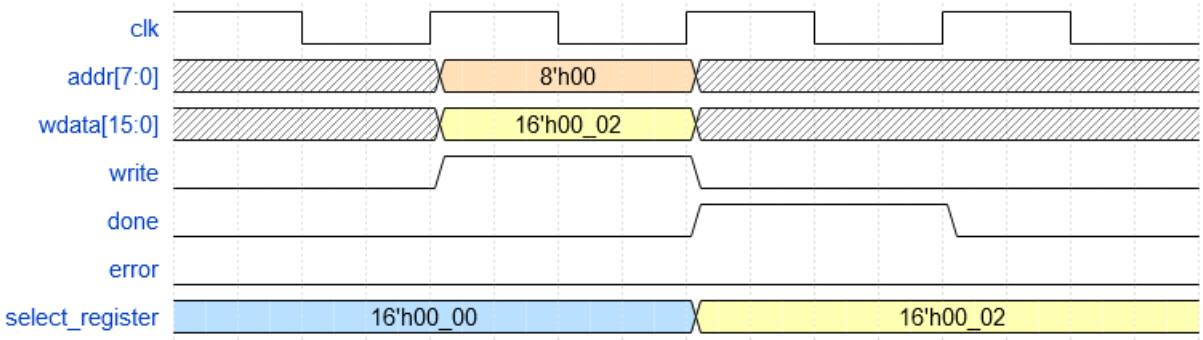
## Descrierea semnalelor

| Semnal | Descriere |
|---|---|
| addr[7:0] | Transmite adresa la care va avea loc accesul de citire/scriere |
| read | Validează adresa pentru citire. Nu poate să fie ridicat odată cu semnalul *write* |
| write | Validează adresa pentru scriere, validează datele de pe wdata[15:0]. Nu poate să fie ridicat odată cu semnalul *read.* |
| wdata[15:0] | Conţine datele ce vor fi scrise în registrul (validat de semnalul write) |
| rdata[15:0] | Conţine datele ce au fost citite din registru (validat de semnalul done) |
| done | Marchează încheierea tranzacţiei:<br>● Pentru write că registrul a fost scris<br>● Pentru read validează datele de pe rdata<br><br>**Semnalul done trebuie pus pe 1 în ciclul de ceas următor!** |
| error | În cazul în care accesul se face la un registru inexistent, semnalul error se asertează. Este validat de semnalul done. |

Exemple de accese:

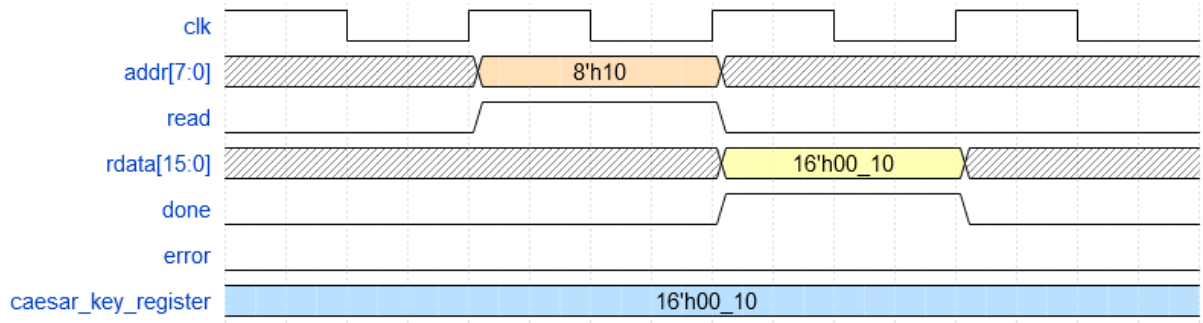
1. Acces de scriere

Se pune pe magistrală adresa dorită: 0x0 (select_register, datele pe care le scriem: 0x2 şi se pune semnalul write pe 1. După ce registrul a fost scris, semnalul *done* stă pe 1 un ciclu de ceas.



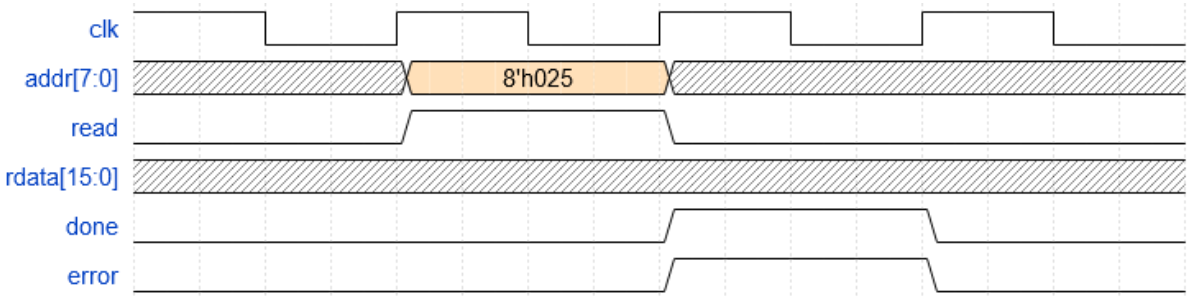

Semnalul done poate fi pus pe 1 oricând după ce a fost primit accesul de scriere.

2. Acces de citire



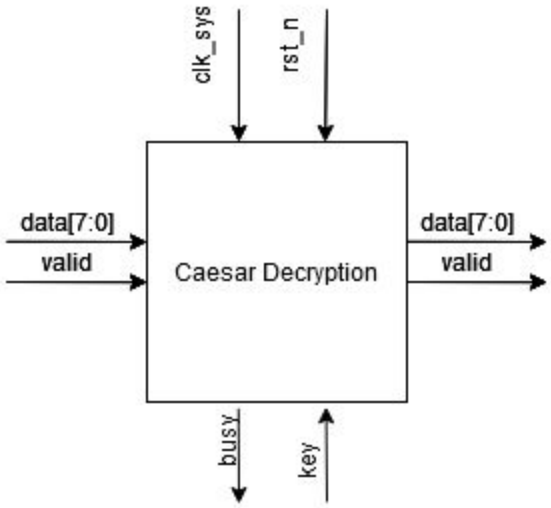

În acest exemplu, a fost citit registrul caesar_key_register, care a returnat valoarea 0x10.

3. Acces la o adresă inexistentă

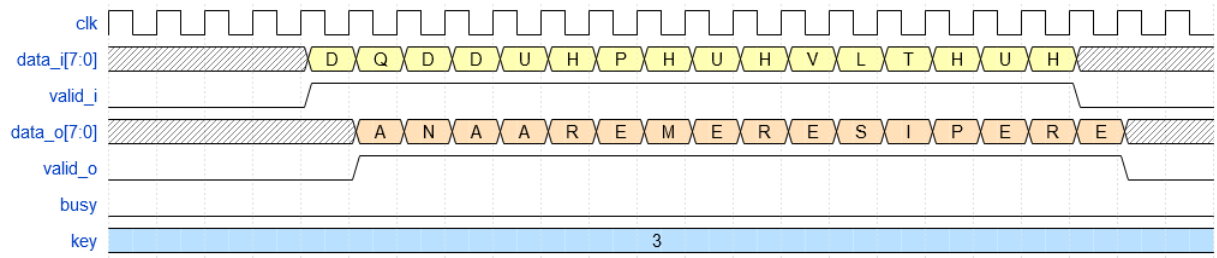



În acest caz este citită adresa 0x25, care nu există. Odată cu done se ridică şi semnalul error. Datele de pe magistrala rdata nu contează.

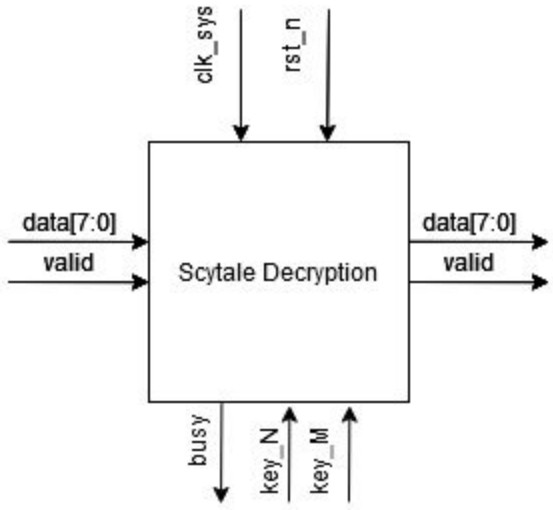
# Cifrul Cezar




Odată primit, fiecare caracter va fi scos în următorul ciclu de ceas decriptat conform cheii primite. Semnalul busy va fi mereu 0.



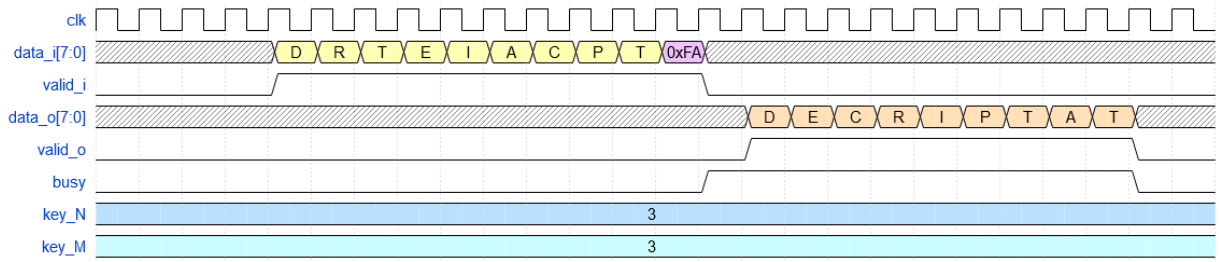
# Cifrul Scytale



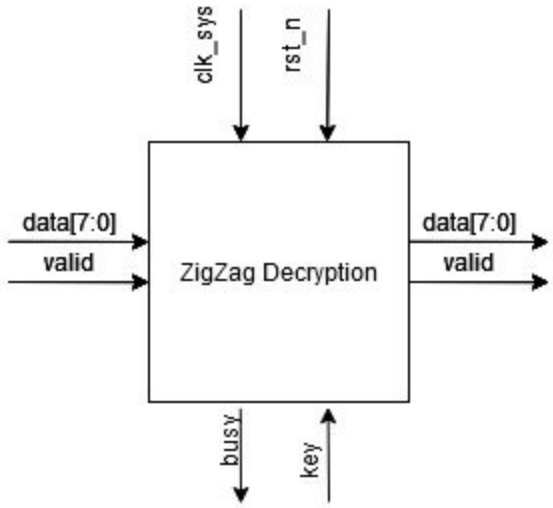

Spre deosebire de Cifrul Cezar, pentru Scytale este nevoie să avem toate caracterele pentru a putea începe decriptarea. Pentru acest lucru, adaugăm un caracter special de cod 0xFA, care va marca momentul când s-a terminat un transfer de date şi decriptarea poate să înceapă. După acest moment, semnalul busy este ridicat şi ţinut pe 1 până când toate caracterele decriptate sunt scoase pe ieşire.

Observație!

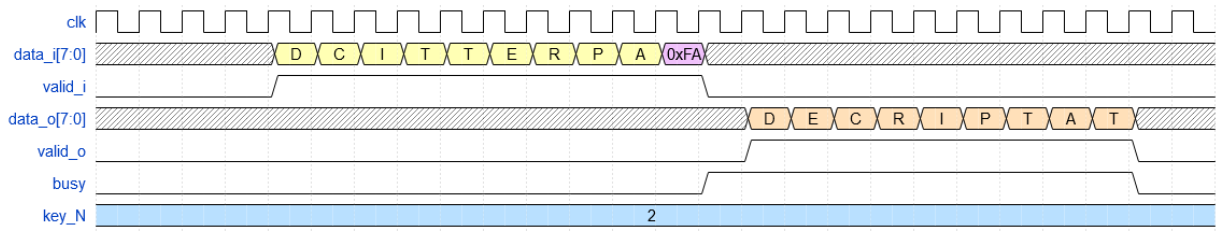
- Datele care vin pe intrare cât timp busy este ridicat sunt ignorate.
- Se garantează că nu se trimit niciodată mai mult de 50 de caractere consecutive.



# Cifrul ZigZag




Similar cu ZigZag, decriptarea se va face numai după ce este primit caracterul special 0xFA. **Se cere decriptarea doar pentru valorile 2 şi 3 ale cheii de decriptare.**




Observație!

- Datele care vin pe intrare cât timp busy este ridicat sunt ignorate.
- Se garantează că nu se trimit niciodată mai mult de 50 de caractere consecutive.

# Deadline

Tema “Decryption” va avea trei parti:

- Task 1: Implementarea bancului de registre
  - Deadline soft: 6.12.2020, 23:59
  - [1]Deadline hard: 20.12.2020, 23:59
- Task 2: Implementarea modulelor de decriptare Cezar, Scytale si ZigZag.
  - Deadline soft: 13.12.2020, 23:59
  - [2]Deadline hard: 20.12.2020, 23:59
- Task 3: Modulele mux/demux si decryption_top
  - Deadline soft: 20.12.2020, 23:59
  - Deadline hard: 20.12.2020, 23:59

[1] Fiecare zi de intarziere se depuncteaza cu 0.5 / 10  (echivalent cu 0.1 din nota finală)
[2] Fiecare zi de intarziere se depuncteaza cu 0.5 / 10 (echivalent cu 0.3 din nota finală)

# Notare

- +2 pct: implementarea modulului register_file
- +2 pct implementarea modului caesar_decryption
    - -1p pentru stocare a mai mult de 1 caracter
- +2 pct implementarea modului scytale_decryption
    - -1p pentru stocare în formă matricială
- +2 pct implementarea modului zigzag_decryption
    - -1p pentru stocare în formă matricială
- +2 pct implementarea modulelor mux/demux şi conexiunile în modulul decryption_top
- +2 pct bonus: Implementarea Cifrului ZigZag pentru valoarile 4 si 5 ale cheii de decriptare.
- -12 pct: folosirea construcţiilor nesintetizabile din Verilog (while, repeat, for cu număr variabil de iteraţii, operatorii / şi %, instrucţiuni de întârziere etc.);
- -1 pct: lipsa fişierului README;
- -0.5 pct: pentru fiecare zi de întârziere; tema poate fi trimisă pana la deadline-ul hard specificat în enunţ (faţă de deadline-ul soft).
- -0.5 pct: folosirea incorectă a atribuirilor continue (assign), blocante (=) şi non-blocante (<=).
- -0.5 pct: indentare haotică
- -0.2 pct: lipsa comentariilor utile
- -0.1 pct: comentarii inutile (ex. wire x; // semnalul x)
- -0.2 pct: diverse alte probleme constatate în implementare (per problemă)

Dacă tema primeşte 0 puncte pe platforma vmchecker, se pot acorda maxim 2 pct pe ideea implementării, la latitudinea asistentului. Ideea şi motivele pentru care nu funcţionează trebuie documentate temeinic în README şi/sau comentarii. Temele care au erori de compilare vor fi notate cu 0 pct.

# Referințe

[1] https://en.wikipedia.org/wiki/Caesar_cipher
[2] https://en.wikipedia.org/wiki/Scytale
[3] https://en.wikipedia.org/wiki/Rail_fence_cipher