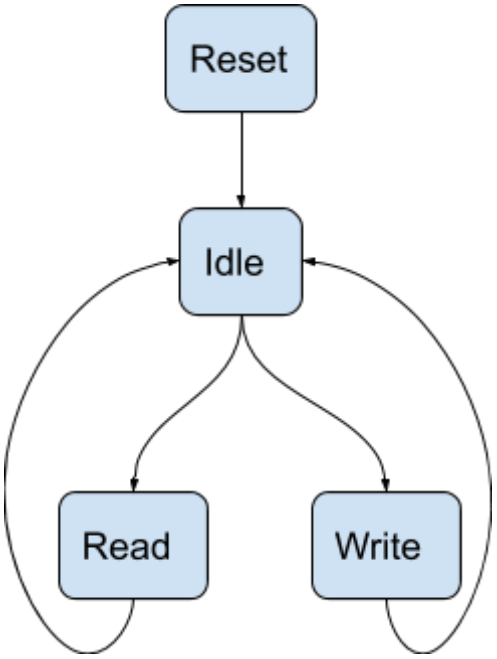
# TASK 1: Implementarea bancului de regiştri

Codificare stări:

- RESET 0
- IDLE 10
- READ 20
- WRITE 30




1. Logica secvenţială:
   Trecem în starea următoare şi latch-uim adresa şi datele ce trebuie scrise. Dacă semnalul de reset este activ (rst_n este 0) trecem în starea de RESET.

2. Logica combinationala:
   Avem un FSM cu 4 stări, din starea de RESET mergem în starea de IDLE, stare în care aşteptam sa se ridice semnalul write sau read pentru a trece în starea corespunzătoare acestora.
   În starea READ citim datele de la adresa specificata de addr şi le punem pe magistrala (în registrul rdata), în stare WRITE copiam datele puse pe magistrala în registrul specificat de addr, în ambele cazuri daca adresa este invalida ridicăm semnalul de eroare. Din aceste 2 stări ne întoarcem în starea de IDLE şi aşteptam operaţia următoare.

   **PS: Din orice stare putem sa trecem în starea de RESET dacă semnalul rst_n devine activ**

# TASK 2: Implementarea modulelor de decriptare

## Modulul Caesar Decryption

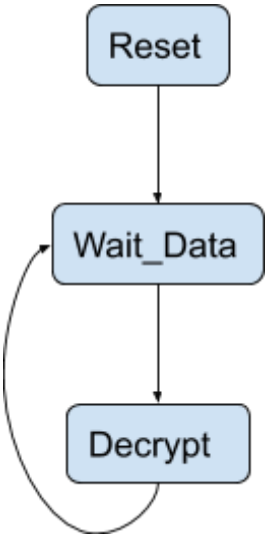
Verificăm dacă semnalul de intrare validă este activ şi dacă da, latch-uim datele de la intrare pe ieşire după ce scoatem cheia primită din acestea, dacă intrarea nu mai este validă atunci latch-uim pe ieşire şi pe validarea ieşirii 0

Dacă semnalul de reset este activ, setam valorile ieşirilor

## Modulul Scytale Decryption

Codificare stări:

- RESET 0
- WAIT_DATA 10
- DECRYPT 20




1. Logica secvenţială

   Trecem în starea următoare şi facem latch pe intraţi şi pe variabilele folosite ca şi “contori” în logica combinationala. Dacă semnalul de reset este activ (rst_n este 0) trecem în starea de RESET.

2. Logica combinationala

   Aici avem 3 stări:

   a. RESET: Starea în care resetam toate semnalele importante din cadrul modulului, atat valorile puse pe ieşire cat şi semnale registre intermediare. După care mergem în starea de aşteptare a datelor

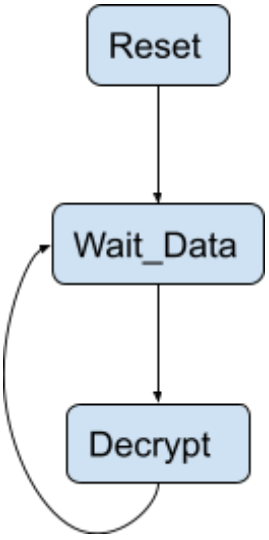
b. WAIT_DATA: Starea de stocare a datelor, aici stocăm datele ce vin pe intrare dacă semnalul valid_i este pe 1 şi caracterul venit la intrare nu are valoarea 0xFA în hexa. Dacă are valoarea 0xFA atunci o sa trecem în starea de decriptare şi punem semnalul busy pe 1.
c. DECRYPT: Starea în care punem datele pe ieşire, la fiecare pas calculăm poziţia in vector a elementului corespunzător din forma matriceala cu formula Key_N * coloana curenta + rând curent. Cand terminam de parcurs toate coloanele avansam pe randul urmator, la fiecare pas scădem numărul total de elemente şi cand ajungem la 0 revenim in starea de aşteptare a datelor.

**PS: Din orice stare putem sa trecem în starea de RESET dacă semnalul rst_n devine activ**

## Modulul ZigZag Decryption

Codificare stări:

- RESET 0
- WAIT_DATA 10
- DECRYPT 20




Singurele diferente fata de modulul Scytale în logica utilizata sunt reprezentate de modul de decriptare, adică de cum parcurgem vectorul în care am stocat datele de la intrare pentru a le scoate la ieşire. Cum avem mai multe chei ce pot sa fie utilizate o sa le luăm pe rând:

1. Cheia de criptare = 2

În acest caz după ce am stocat toate valorile primite la intrare o sa calculăm restul si catul împărţirii la 2 a nr de elemente, prin extragerea banilor din numărul iniţial, dacă restul este 1 atunci la

cat o sa adunam 1 pentru ca acesta sa fie egal cu nr de elemente de pe prima linie a "matricei".

În etapa de punere a datelor pe ieşire folosim o variabila line ce desemnează linia curentă, line = k - 1 => linia curentă este linia k.

Dacă suntem pe prima linie a matricei atunci nu trebuie sa adunam deplasamentul calculat mai devreme, punem datele la ieşire, modificăm valoarea liniei (din 1 în 0 sau invers, depinde de caz), dacă suntem pe linia 2 atunci o sa adunăm deplasamentul la indexare şi punem pe ieşire caracterul corespunzător şi trecem pe linia 1.

De fiecare data cand ajungem pe linia 0 incrementam valoarea lui i, care ne spune ce element de pe linia curentă a matricei afisam. La fiecare pas scădem 1 din numărul de elemente pe care trebuie sa le mai afisam şi cand ajungem la 0 ca şi în cazul modulului precedent revenim in starea de aşteptare a datelor.

2. Cheia de criptare = 3

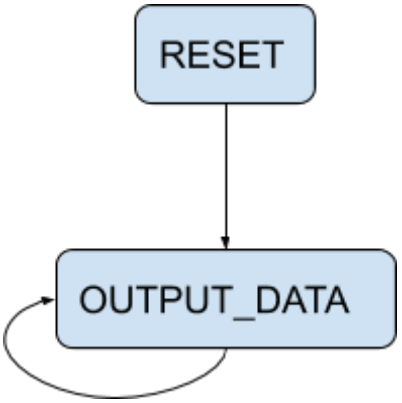
Acest caz este similar cu cel de mai sus, doar ca de aceasta data avem mai multe "linii". O sa mă rezum în a explica cum am calculat numărul de elemente de pe fiecare linie în parte. Numărul de elemente de pe prima linie l-am calculat prin pseudo împărţirea la 4 pe acelaşi principiu ca mai sus, dacă restul este nenul atunci o sa adaugam 1. Pentru numărul de elemente de pe ultima linie am procedat similar decat ca am scăzut 2 din nr de elemente, aceasta operaţie are ca şi efect mutarea tuturor caracterelor de pe ultima linie pe prima linie, iar numărul de elemente de pe a doua linie a fost calculat prin scaderea din numărul total de caractere a numărului de caractere de pe prima şi de pe ultima linie.

PS: Mda, era suficient ca aici sa mă folosesc de faptul ca numărul de cicluri complete este nr_elemente/4, dar dacă tot am avut alta idee am zis sa o las aşa, mai ales ca este şi corecta

**PS: Din orice stare putem sa trecem în starea de RESET dacă semnalul rst_n devine activ**
**PPS: Detalierea implementării bonusului se regaseste în secţiunea BONUS de la finalul documentului**

# TASK 3: Implementarea modulelor DEMUX/MUX şi TOP

## Modulul DEMUX




Codificare stări:
- RESET 100
- OUTPUT_DATA 200

Macrodefinitii:
- caesar 0
- scytale 1
- zigzag 2

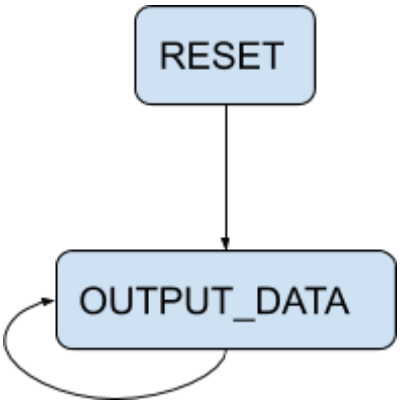
1. Logica secvenţială
    a. Logica pe sys_clk: în acest caz singura operatie pe care o facem este sa punem în i, valoarea stocată în i_next în cadrul logicii combinationale
    b. Logica pe mst_clk: facem latch pe datele de la intrare, trecem în starea următoare şi verificăm dacă este cazul sa trecem în starea de reset
2. Logica combinationala:
    a. La fiecare pas reinitializam toate semnalele de ieşire cu 0 şi verificăm în ce stare ne aflăm.
    b. În reset reinitializam toate semnalele de ieşire şi toate semnalele interne relevante, de asemenea trecem în starea în care punem datele pe ieşire.
    c. Starea de punere pe ieşire a datelor verifica dacă semnalul de valid de la master(cel căruia i-am făcut un latch) este activ şi dacă este activ verificam către ce modul de decriptare trebuie sa rutam datele. De fiecare data cand punem date pe o ieşire decrementam i_next. Atunci cand i ajunge sa fie 0, resetam i_next pentru ca pe următorul ciclu de sys_clk dacă este nevoie să putem pune date pe o iesire

**PS: Din orice stare putem sa trecem în starea de RESET dacă semnalul rst_n devine activ**

## Modulul MUX

Codificare stări:

- RESET 0
- OUTPUT_DATA 2




1. Logica secvenţială:
    a. Verificam daca trebuie sa trecem în starea de reset, altfel mergem în starea următoare
    b. În variabila data_reg stocăm în funcţie de combinaţia semnalelor de valid_i datele corespunzătoare intrării pe care trebuie s-o rutam la ieşire
    c. În valid_i_sys stocăm 1 dacă vreunul din semnalele valid_i este activ
2. Logica combinationala:
    a. În starea de reset resetam datele puse pe ieşire şi trecem în starea de punere a datelor pe ieşire
    b. În starea de punere a datelor pe ieşire verificăm dacă semnalul valid_i_sys este activ şi în acest caz punem datele pe ieşire, altfel pe punem pe 0

**PS: Din orice stare putem sa trecem în starea de RESET dacă semnalul rst_n devine activ**

## Modulul DECRYPTION_TOP

### Realizează conexiunile necesare funcţionarii modulelor

## BONUS: Implementarea decriptarii cu cheia 4 şi 5 pentru ZigZag

Au fost adaugata logica combinationala pentru determinarea numărului de elemente de pe fiecare linie în cazul cheii 4 sau 5.

Pentru determinarea numărului de ciclii pentru cheia 4 şi 5 am utilizat modulul division(instantiat sa faca impartirea pe 8 biti) implementat la tema anterioara. Numărul de elemente de pe linii a fost determinat folosind nr de ciclii compleţi şi restul.

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

Pe exemplul de mai sus observam ca avem 2 cicluri complete şi restul 2, deci trebuie sa adaugam cate un element la numărul de elemente de pe fiecare linie, astfel numărul elemente de pe prima linie devine egal cu numărul de cicluri + 1, iar numărul de elemente de pe a doua linie egal cu 2 * numărul de cicluri complete + 1.

Pentru determinarea numărului de elemente de pe o linie în cazul unui număr întreg de cicluri, avem 2 cazuri:

- număr de elemente = număr de cicluri, pentru prima, respectiv ultima linie
- număr de elemente = 2 * număr de cicluri, pentru orice alta linie

***Efectul restului***

Prima data o sa împărţim restul în 2 bucăţi, al căror efect insumat o sa fie efectul total:

1. rest_bucata_1 = rest % (key + 1), valori între 0 şi key
    a. dacă restul calculat cu formula de mai sus are cel puţin valoarea k, k între 0 şi key atunci adaugam cate 1 la numărul elementelor de pe liniile de indice mai mic sau egal cu k
2. rest_bucata_2 = max(rest - rest_bucata_1, 0), cu valori între 0 şi key - 3
    a. dacă restul calculat cu formula de mai sus are valoarea k, atunci adaugam cate 1 la numărul de elemente de pe liniile cu indexul în mulţimea {key - 1 , …, key - k}

**PS: Formulele şi operaţiile utilizate mai sus au doar scop demonstrativ, ele nu au legatura cu codul verilog al temei, şi doar fac înţelegerea operaţiilor mai uşoară**