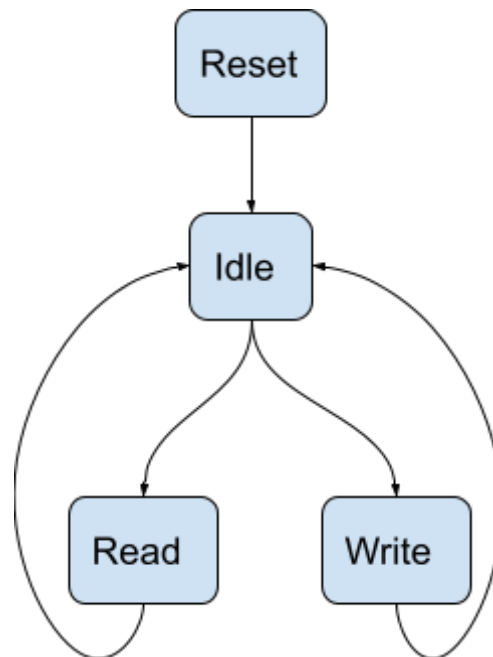


TASK 1: Implementarea bancului de regiștri

Codificare stări:

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



1. Logica secvențială:

Trecem în starea următoare și latch-uim adresa și datele ce trebuie scrise. Dacă semnalul de reset este activ (`rst_n` este 0) trecem în starea de RESET.

2. Logica combinatională:

Avem un FSM cu 4 stări, din starea de RESET mergem în starea de IDLE, stare în care așteptăm să se ridice semnalul write sau read pentru a trece în starea corespunzătoare acestora.

În starea READ citim datele de la adresa specificată de `addr` și le punem pe magistrala (în registrul `rdata`), în stare WRITE copiam datele puse pe magistrala în registrul specificat de `addr`, în ambele cazuri dacă adresa este invalidă ridicăm semnalul de eroare. Din aceste 2 stări ne întoarcem în starea de IDLE și așteptăm operația următoare.

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

TASK 2: Implementarea modulelor de decriptare

Modulul Caesar Decryption

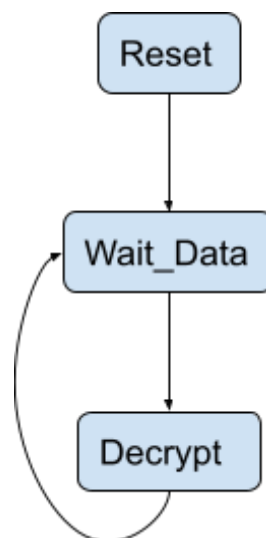
Verificăm dacă semnalul de intrare validă este activ și dacă da, latch-uim datele de la intrare pe ieșire după ce scoatem cheia primită din acestea, dacă intrarea nu mai este validă atunci latch-uim pe ieșire și pe validarea ieșirii 0

Dacă semnalul de reset este activ, setăm valorile ieșirilor

Modulul Scytale Decryption

Codificare stări:

- RESET 0
- WAIT_DATA 10
- DECRYPT 20



1. Logica secvențială

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

2. Logica combinatională

Aici avem 3 stări:

- a. RESET: Starea în care resetăm toate semnalele importante din cadrul modului, atât valorile puse pe ieșire cât și semnale registre intermediare. După care mergem în starea de așteptare a datelor

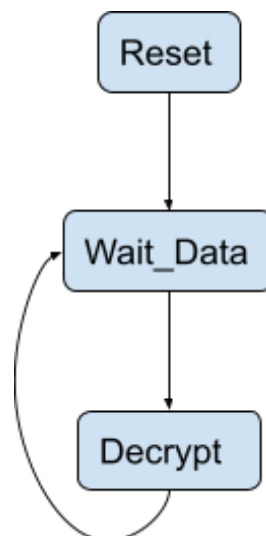
- b. WAIT_DATA: Starea de stocare a datelor, aici stocăm datele ce vin pe intrare dacă semnalul valid_i este pe 1 și caracterul venit la intrare nu are valoarea 0xFA în hexa. Dacă are valoarea 0xFA atunci o sa trecem în starea de decriptare și punem semnalul busy pe 1.
- c. DECRYPT: Starea în care punem datele pe ieșire, la fiecare pas calculăm poziția în vector a elementului corespunzător din forma matriceala cu formula $\text{Key}_N * \text{coloana curenta} + \text{rând curent}$. Când terminăm de parcurs toate coloanele avansăm pe rândul următor, la fiecare pas scădem numărul total de elemente și când ajungem la 0 revenim în starea de așteptare 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 ieșire. 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 și catul împărțirii la 2 a nr de elemente, prin extragerea banilor din numărul inițial, 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 ieșire folosim o variabila line ce desemnează linia curentă, $line = k - 1 \Rightarrow$ linia curentă este linia k.

Dacă suntem pe prima linie a matricei atunci nu trebuie sa adunam deplasamentul calculat mai devreme, punem datele la ieșire, modificăm valoarea liniei (din 1 în 0 sau invers, depinde de caz), dacă suntem pe linia 2 atunci o sa adunăm deplasamentul la indexare și punem pe ieșire caracterul corespunzător și 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 și cand ajungem la 0 ca și în cazul modulului precedent revenim in starea de așteptare 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 împărțirea la 4 pe același 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 operație are ca și 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 și de pe ultima linie.

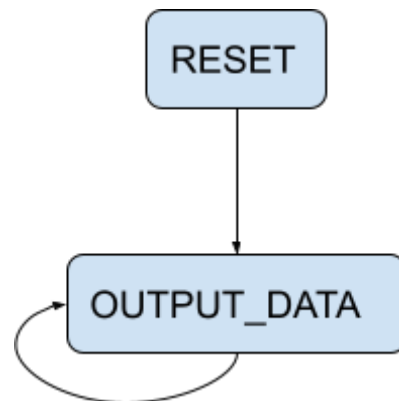
PS: Mda, era suficient ca aici sa mă folosesc de faptul ca numărul de cicluri complete este $nr_elemente/4$, dar dacă tot am avut alta idee am zis sa o las așa, mai ales ca este și corecta

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

PPS: Detalierea implementării bonusului se regaseste în secțiunea BONUS de la finalul documentului

TASK 3: Implementarea modulelor DEMUX/MUX și TOP

Modulul DEMUX



Codificare stări:

- RESET 100
- OUTPUT_DATA 200

Macrodefinitii:

- caesar 0
- scytale 1
- zigzag 2

1. Logica secvențială

- a. Logica pe sys_clk: în acest caz singura operație pe care o facem este să punem în *i*, valoarea stocată în *i_next* în cadrul logicii combinatoriale
- b. Logica pe mst_clk: facem latch pe datele de la intrare, trecem în starea următoare și verificăm dacă este cazul să trecem în starea de reset

2. Logica combinatorială:

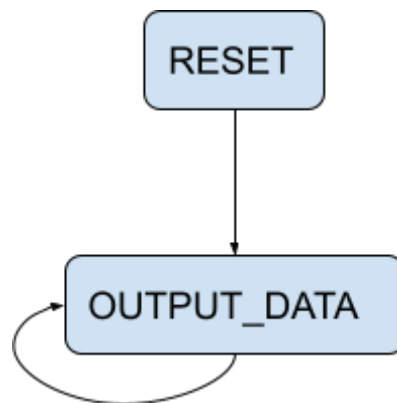
- a. La fiecare pas reinitializăm toate semnalele de ieșire cu 0 și verificăm în ce stare ne aflăm.
- b. În reset reinitializăm toate semnalele de ieșire și toate semnalele interne relevante, de asemenea trecem în starea în care punem datele pe ieșire.
- c. Starea de punere pe ieșire a datelor verifică dacă semnalul de valid de la master (cel căruia i-am făcut un latch) este activ și dacă este activ verificăm către ce modul de decriptare trebuie să rutăm datele. De fiecare dată când punem date pe o ieșire decrementăm *i_next*. Atunci când *i* ajunge să fie 0, resetăm *i_next* pentru că pe următorul ciclu de sys_clk dacă este nevoie să putem pune date pe o ieșire

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 secvențială:
 - a. Verificam daca trebuie sa trecem în starea de reset, altfel mergem în starea următoare
 - b. În variabila data_reg stocăm în funcție de combinația semnalelor de valid_i datele corespunzătoare intrării pe care trebuie s-o rutam la ieșire
 - c. În valid_i_sys stocăm 1 dacă vreunul din semnalele valid_i este activ
2. Logica combinatională:
 - a. În starea de reset resetam datele puse pe ieșire și trecem în starea de punere a datelor pe ieșire
 - b. În starea de punere a datelor pe ieșire verificăm dacă semnalul valid_i_sys este activ și în acest caz punem datele pe ieșire, 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 funcționarii modulelor

BONUS: Implementarea decriptării cu cheia 4 și 5 pentru ZigZag

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

Pentru determinarea numărului de cicluri pentru cheia 4 și 5 am utilizat modulul division (instantiat să facă împărțirea pe 8 biți) implementat la tema anterioară. Numărul de elemente de pe linii a fost determinat folosind nr de cicluri compleți și restul.

A						E						S	
	N				R						E		I
		A		A				M		R			
									E				

Pe exemplul de mai sus observăm că avem 2 cicluri complete și restul 2, deci trebuie să adăugăm câte un element la numărul de elemente de pe fiecare linie, astfel numărul de elemente de pe prima linie devine egal cu numărul de cicluri + 1, iar numărul de elemente de pe a doua linie egal cu $2 * \text{numărul de cicluri complete} + 1$.

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 = $2 * \text{număr de cicluri}$, pentru orice altă linie

Efectul restului

Prima dată o să împărțim restul în 2 bucăți, al căror efect însumat o să fie efectul total:

1. $\text{rest_bucata_1} = \text{rest} \% (\text{key} + 1)$, valori între 0 și key
 - a. dacă restul calculat cu formula de mai sus are cel puțin valoarea k, k între 0 și key atunci adăugăm câte 1 la numărul elementelor de pe liniile de indice mai mic sau egal cu k
2. $\text{rest_bucata_2} = \max(\text{rest} - \text{rest_bucata_1}, 0)$, cu valori între 0 și key - 3
 - a. dacă restul calculat cu formula de mai sus are valoarea k, atunci adăugăm câte 1 la numărul de elemente de pe liniile cu indexul în mulțimea $\{\text{key} - 1, \dots, \text{key} - k\}$

PS: Formulele și operațiile utilizate mai sus au doar scop demonstrativ, ele nu au legătură cu codul verilog al temei, și doar fac înțelegerea operațiilor mai ușoară