Țae Andreea-Cristina, 332AB

TEMA2 – README

Task 1 – decryption_regfile

Atât pentru scriere, cât și pentru citire m-am folosit de un case în funcție de adresă în care am citit din registru sau am scris în registru din wdata și de o variabilă auxiliară done_aux pentru a trece variabila done în 1 pe ciclul de ceas următor. La fel, pentru ambele cazuri (citire și scriere), am avut un caz default în instrucțiunea case în care dacă se face citirea/ scrierea la o adresă inexistentă, semnalele de error și done devin 1.

Pentru a respecta condiția ca semnalele read și write să nu fie ridicate simultan, pentru scriere am impus ca doar semnalul write să fie ridicat, iar pentru citire, doar semnalul read.

Un alt lucru de menționat: pentru scrierea asincronă în registre m-am folosit de o variabilă auxiliară căreia i-am dat valoare în blocul always secvențial în cadrul instrucțiunii case, iar apoi într-un bloc always combinațional am atribuit valoarea variabilei auxiliare registrului respectiv.

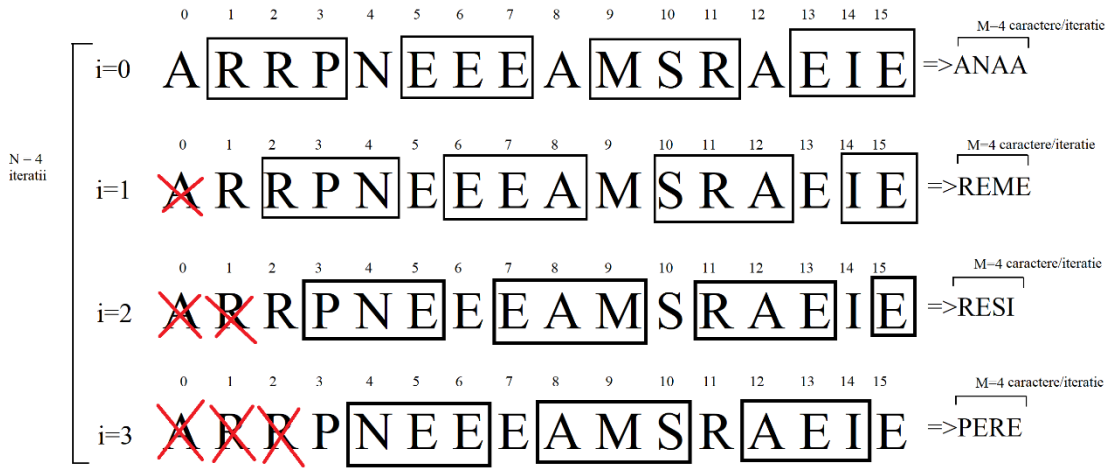
Task 2 – caesar_decryption, scytale_decryption, zigzag_decryption

Pentru **cifrul caesar** am folosit o variabilă auxiliară data_o_aux pentru a scrie informația decriptată pe următorul ciclu de ceas: în blocul secvențial am atribuit variabilei auxiliare informația decriptată dacă valid_i este ridicat și am trecut pe 1 și semnalul de valid_o, iar în blocul combinațional am dat valoarea variabilei auxiliare lui data_o.

Pentru **cifrul scytale** am stocat informația primită pe data_i într-o variabilă auxiliară vect_stoc, un vector de dimensiune 50 x D_WIDTH. Pe lângă aceasta, am mai declarat în plus 3 variabile auxiliare: 2 indici i și j pentru navigarea în vectorul vect_stoc și size_vect pentru memorarea dimensiunii secvenței de date primite la intrare.

În vect_stoc am memorat secvența primită la intrare atâta timp cât valid_i este ridicat și printre caracterele memorate nu se află 0xFA. Atâta timp cât se primesc date la intrare pentru memorat, size_vect crește.

Când se primește de la intrare caracterul 0xFA, semnalul busy se trece în 1, se începe decriptarea și semnalul de valid_o este ridicat. În continuare, decriptarea se execută doar cât semnalul busy este 1, iar la finalul acesteia, el este trecut în 0.



În schemă am ilustrat algoritmul dedus pentru scytale, cu j se va naviga prin vectorul de caractere ARRPNEEEAMSRAEIE (datele primite la intrare), iar cu i se va contoriza numărul de iterații. Formula după care sunt introduse caracterele în ordinea corectă spre ieșire după decodificare este **i + key_N*j**, unde i = 0..key_N-1, iar j = 0..key_M-1.

Pentru a rezolva exemplul anterior, în data_o caracterele trebuie puse în ordinea 0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, adică:

- **iterația i = 0**, j = 1..3
  - 0 + 4*0 = **0**
  - 0 + 4*1 = **4**
  - 0 + 4*2 = **8**
  - 0 + 4*3 = **12**
- **iterația i = 1**, j = 1..3
  - 1 + 4*0 = **1**
  - 1 + 4*1 = **5**
  - 1 + 4*2 = **9**
  - 1 + 4*3 = **13**
- **iterația i = 2**, j = 1..3
  - 2 + 4*0 = **2**
  - 2 + 4*1 = **6**
  - 2 + 4*2 = **10**
  - 2 + 4*3 = **14**
- **iterația i = 3**, j = 1..3
  - 3 + 4*0 = **3**
  - 3 + 4*1 = **7**
  - 3 + 4*2 = **11**
  - 3 + 4*3 = **15**

Algoritmul se termină și indicii, semnalul de busy, de valid_o , size_vect și data_o se resetează atunci când se ajunge la ultima iterație a lui i, adică key_N-1.

Task 3 – decryption_top, mux, demux

În implementarea modulului **mux** am considerat încă 4 variabile auxiliare, data_o_aux0, data_o_aux1, data_o_aux2 și data_stoc. Variabilele auxiliare de tipul data_o_aux sunt ridicate dacă valid_i respectiv pentru fiecare semnal este ridicat (ex: se ridică valid0_i, se ridică și data_o_aux0), iar în același timp, este atribuită informația primită la intrare pe data_i lui data_stoc.

În final, într-un case în funcție de select se verifică dacă a fost ridicat data_o_aux respectiv și dacă da, se atribuie lui data_o valoarea din data_stoc, se ridică semnalul valid_o și se resetează data_o_aux și valid_o (ultimul semnal se resetează înafara case-ului dacă a fost ridicat înainte).

În modulul **demux** s-au încercat mai multe metode de rezolvare, fiecare având diverse probleme. Varianta prezentă în arhivă se folosește de 2 blocuri secvențiale, unul pentru clk_sys și altul pentru clk_mst. În blocul secvențial pentru clk_mst se citește informația primită la intrare într-un reg de 32 biți vect_stoc, dacă semnalul valid_i este ridicat. În blocul secvențial în funcție de clk_sys, se numără într-o variabilă count numărul de bătăi de ceas după ce se ridică valid_i, iar dacă acesta trece de 4 (numărul necesar pentru trecerea unui ciclu de clk_mst), se începe scrierea informației primite la intrare în data_o din vect_stoc. În continuare se naviga în vect_stoc cu ajutorul unei variabile i care se incrementa la începerea scrierii, până la 3, când aceasta se termină.

În cele mai multe variante de implementare a modulului demux, în scrierea pe data_o, pe ultimele 2 poziții din fiecare secvență se pune un caracter dublat. Nu s-a găsit o rezolvare pentru

această problemă.