Stefan Andreea-Constanta

Grupa 332AB

Task 1

Pentru scrierea sau citirea intr-un/ dintr-un registru am facut un case in functie de adresa. Pentru fiecare adresa se verifica daca write sau read sunt activate. Daca write e 1, se realizeaza o scriere in registrul corespondent adresei alese si variabila count se face 1; daca read e 1, se realizeaza o citire din registrul corespondent adresei alese, se pune in rdata valoarea gasita in registru si count se face 1. Daca niciunul din semnalele read si write nu este 1, inseamna ca nu se poate realiza nici citire, nici scriere si se ajunge in cazul de eroare, in care variabila count_error se face 1. La final, verific daca variabila count este 1, daca da, atunci done este 1 (s-a facut o scriere/ citire) si se reseteaza la 0 count, daca nu, verific daca variabila count_error este 1 si in caz afirmativ error si done devin 1 si se reseteaza la 0 count_error.

Task 2

Decriptarea caesar

Pentru a decripta se scade valoarea cheii din valoarea introdusa. Cand valid_i este 1, datele care se introduc sunt valide si trebuie decriptate, altfel pe iesire nu se pune nimic si valid_o e 0.

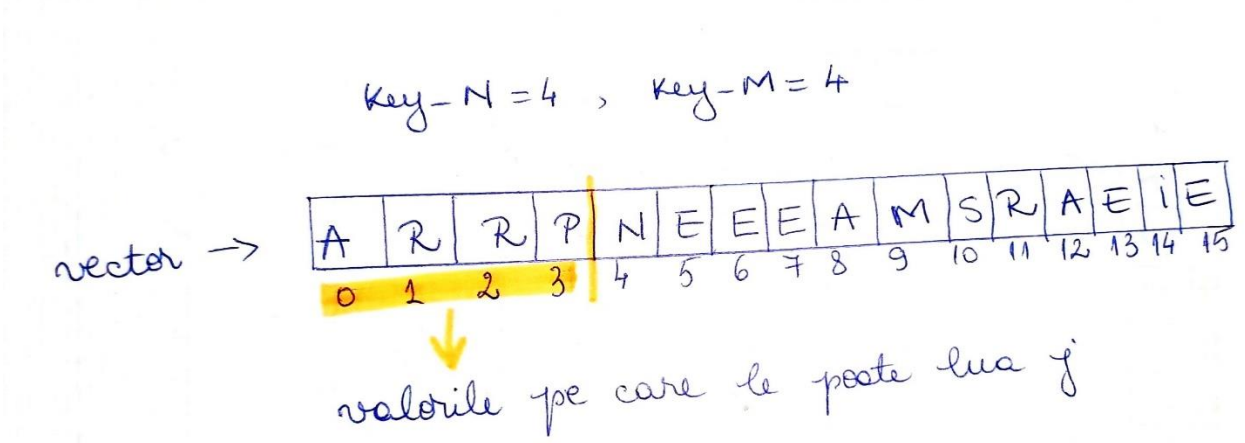
Decriptarea scytale

Pentru decriptarea scytale am facut un automat care are starile reset, fetch, decode, idle.

- Starea reset: se reseteaza la 0 datele de iesire si variabilele pe care le folosesc pentru pozitionarea in vector si se trece in starea fetch.

- Starea fetch: cat timp nu am parcurs toate datele necesare pentru o decriptare (key_N*key_M) si cat timp valid_i este 1, introduc in vector datele. Cand a fost introdus caracterul 8'hFA, se trece in starea de decriptare (decode) si busy se face 1.

- Starea decode:



Variabila j reprezinta pozitia din care se incepe parcurgerea vectorului:

j=0 => parcurgerea incepe din pozitia 0;

j=1 => parcurgerea incepe din pozitia 1;

..... pana la j=key_N - 1.

In exemplul de mai sus, key_N este 4 => vectorul se parcurge cu pasul 4.

Pentru j=0 => se afiseaza valorile de pe pozitiile 0,4,8,12 (A N A A);

Pentru j=1 => se afiseaza valorile de pe pozitiile 1,5,9,13 (R E M E);

Pentru j=2 => se afiseaza valorile de pe pozitiile 2,6,10,14 (R E S I);

Pentru j=3 => se afiseaza valorile de pe pozitiile 3,7,11,15 (P E R E).

Cand j se face key_N se trece la starea idle.

- Starea idle: s-a terminat decriptarea, semnalele valid_o si busy se fac 0 si se trece din nou in starea de reset.

Decriptarea zig-zag

Implementarea este realizata tot sub forma unui automat, asemanator cu cel de la scytale.

In starea fetch, cand introduc in vector datele de intrare, cu ajutorul variabilei nr_char numar cate valori sunt introduse. Cand se introduce valoarea 8'hFA, verific daca nr_char este par sau impar, adica daca ultimul bit al numarului este 0 sau 1 (am observat ulterior ca am facut gresit verificarea ultimului bit). Daca este par, variabila x primeste jumatatea numarului (shiftare la dreapta cu 1 = impartire la 2), daca este impar, variabila x primeste jumatate + 1.

Starea decode_key2:

- parcurg pozitiile vectorului pana la jumatate;

- daca k < nr total de caractere, se afiseaza valoarea de pe pozitia k si k se incrementeaza cu x (jumatate);

- daca k > nr total de caractere, daca nr de caractere este par se afiseaza valoarea de pe pozitia (k-nr_char+1) si k devine pozitia de pe care s-a afisat + jumatate; daca nr de caractere este impar se afiseaza valoarea de pe pozitia (k-nr_char) si k devine pozitia de pe care s-a afisat + jumatate.

Exemplu pentru numar par de caractere:

Key = 2

vector

A	A	R	M	R	S	P	R	N	A	E	E	E	i	E	E
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

=> 16 caractere => nr. par de caractere

x = 8

i = 0 < 8

k = 0 < 16 data_0 = vector[0] = A

k = 0 + 8 = 8

k = 8 < 16 data_0 = vector[8] = N

k = 8 + 8 = 16

k = 16 < 16 FALS

par == 1 data_0 = vector[16 - 16 + 1]

= vector[1] = A

k = 16 - 16 + 1 + 8 = 9

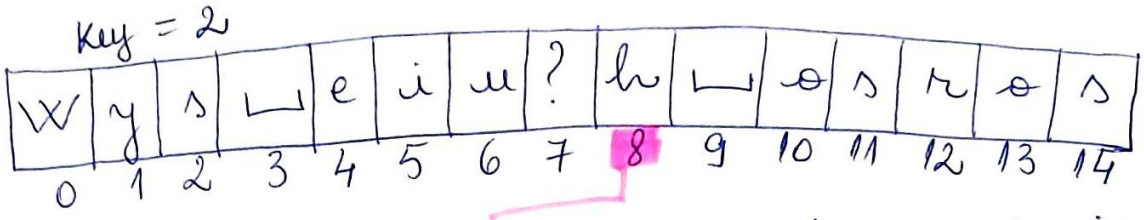
i = 0 + 1 = 1

i = 1 < 8 k = 9 < 16 data_0 = vector[9] = A

k = 9 + 8 = 17

- - - - -

Exemplu pentru numar impar de caractere:



=> 15 caractere => nr. impar

15 / 2 = 7

nr_char = impar ≠> x = 7 + 1 = 8

i = 0 < 8

k = 0 < 15 data_o = vector[0] = W

k = 0 + 8 = 8

k = 8 < 15 data_o = vector[8] = h

k = 8 + 8 = 16

k = 16 < 15 FALS

par == 0 data_o = vector[16 - 15]

= vector[1] = y

k = 16 - 15 + 8 = 9

i = 0 + 1

i = 1 < 8

k = 9 < 15 data_o = vector[9] = ␣

k = 9 + 8 = 17

- - - - -

Task 3

Modulul demux

Folosesc variabila count pentru a numara ciclurile de ceas master. Dupa un ciclu de ceas master se incepe afisarea datelor pe iesire. In variabila valid_i_reg se salveaza valoarea lui valid_i, dar semnalul de pe grafic al variabilei auxiliare este intarziat cu un ciclu de ceas fata de valid_i, deci acesta se va termina exact cand se termina scrierea pe iesire a datelor. Deci cand valid_i_reg este 0, se fac 0 iesirile.

Modulul mux

In functie de semnalul select, se alege care dintre intrari sa fie afisata pe iesire.

Decriptari

Decriptarea caesar este cea de la taskul 2.

Pentru decriptarea scytale am pastrat algoritmul, dar am implementat codul fara automat deoarece apareau niste intarzieri ale semnalelor pe grafic din cauza unor stari.

Decriptarea pentru zig-zag:

- parcurg vectorul de date de la 0 la numarul de caractere-1;

- daca pozitia din vector curenta este para, pe data_o se va pune valoarea de pe (pozitia curenta impartita la 2);

- daca pozitia din vector curenta este impara, pe data_o se va pune valoarea de pe (pozitia curenta impartita la 2) + jumatate;

- daca s-a parcurs tot vectorul, iesirile se fac 0 si celelalte variabile se reseteaza la 0.