Laboratorul 5 - Limbajul Verilog: Circuite secvențiale - Partea a II-a

În laboratorul anterior a fost introdus și studiat conceptul de “circuit secvențial”. Totodată au fost prezentate atribuirile non-blocante în contextul blocurilor always edge-triggered și vi s-a oferit o perspectivă generală asupra expresiilor regulate și asupra automatelor de stări. În laboratorul curent se vor parcurge următoarele noțiuni:

- Sintaxă Verilog utilă în lucrul cu circuite secvențiale
- Tipuri și implementări de automate de stări
- Depozitarea informației digitale: Registrul
- Memorie cu acces aleatoriu (RAM)

Macro: `define

Un macro este un nume căruia i se poate asocia o valoare înainte de compilarea codului. Macro-urile sunt utile pe post de aliasuri, fără a utiliza resursele compilatorului. Acestea nu sunt variabile, prin urmare nu pot fi atribuite valori unui macro în timpul simulării. Majoritatea limbajelor de programare, inclusiv Verilog suportă definirea de macrouri.

În Verilog, un macro este specificat cu ajutorul directivei de compilator **`define**. Aceasta înlocuiește textul definit cu o valoare specifică. Un nume de tip `define este un macro global, însemnând că dacă este declarat într-un modul, va rămâne declarat și la ieșirea din modul. După ce macroul este declarat, poate fi apelat în cod cu ajutorul caracterului ` (back-tic). Indiferent dacă sunt declarate în interiorul sau în afara unui modul, compilatorul le tratează la fel.

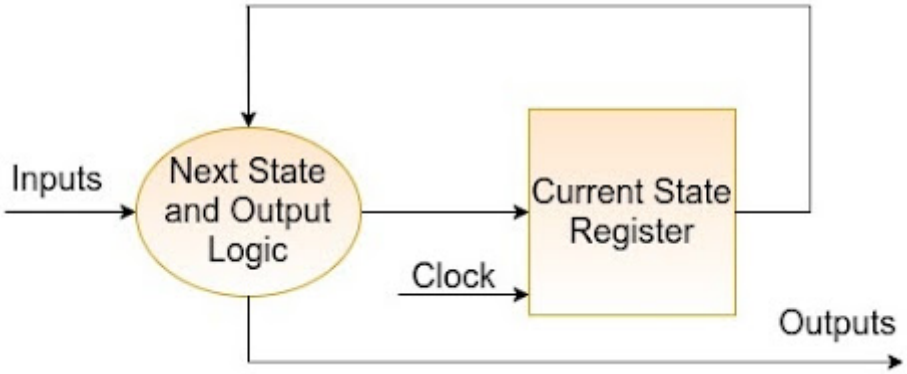
```
`define MY_NUMBER 5
`define MY_STRING “Hello world!”
`define ADD2PLUS2 2 + 2
```

Tipuri de automate de stări

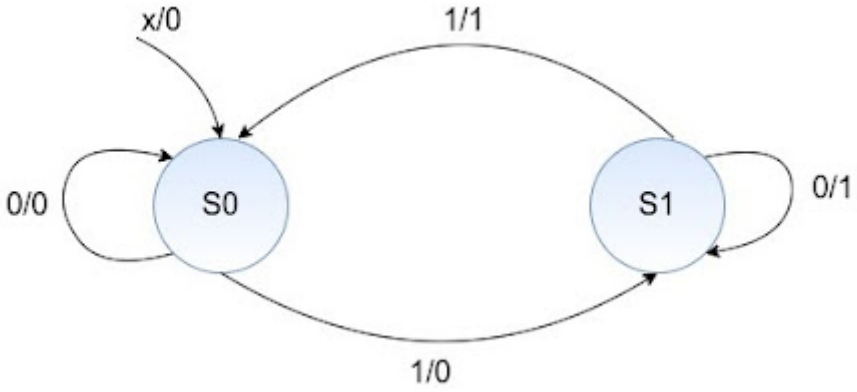
Automatele de stări sunt absolut necesare în implementarea oricărui circuit digital. Există două tipuri de automate de stări, clasificate după tipurile de ieșiri generate de fiecare. Primul tip este **Mașina Mealy,** caracterizată de faptul că una sau mai multe ieșiri depind atât de starea curentă, cât și de una sau mai multe intrări, iar al doilea este **Mașina Moore**, ale cărei ieșiri sunt doar o funcție care depinde doar de starea curentă.

Mașina Mealy

Un model general al unei mașini secvențiale Mealy este format dintr-o rețea combinațională care generează ieșirile, starea următoare și o stare “Current State Register” reprezentând starea curentă, ca în figura de mai jos. Starea “Current State Register” este modelată utilizând bistabili D și este sensibilă la semnalul de ceas (Clock). Atât ieșirea, cât și determinarea stării următoare depind de intrare și de starea curentă.



Exemplu de automat de stări Mealy:

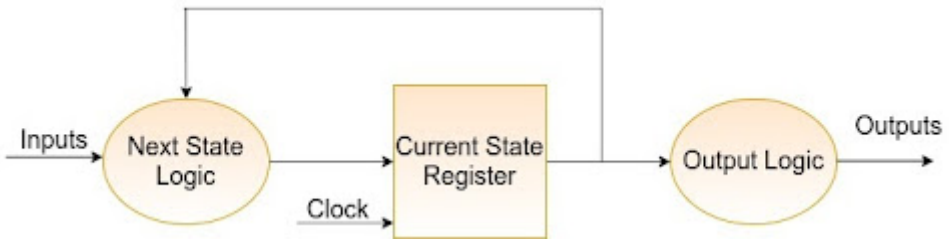



```
module mealy_fsm(
    output reg parity,
    input clk,
    input reset,
    input x);

    reg state, next_state;
    parameter S0=0;
    parameter S1=1;

    // Partea secvențială
    always @(posedge clk or negedge reset)
        if (!reset)
            state <= S0;
        else
```

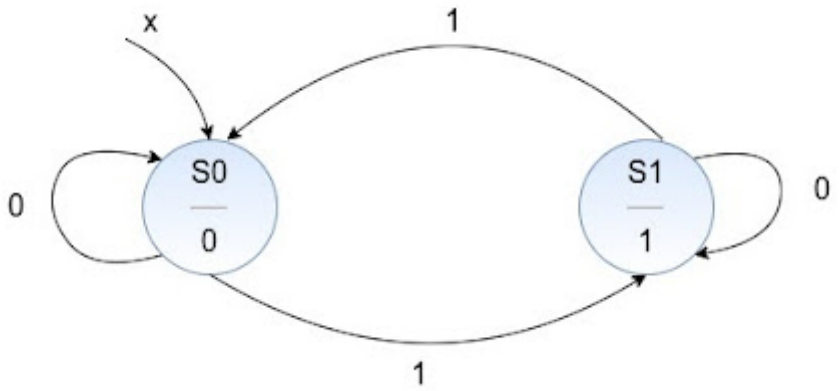
```
            state <= next_state;

    // Partea combinațională
    always @(*) begin
        parity = 1'b0;
        case(state)
            S0:
                if(x)
                    next_state = S1;
                else
                    next_state = S0;
            S1:
                if(x) begin
                    parity = 1;
                    next_state = S0;
                end
                else begin
                    parity = 1;
                    next_state = S1;
                end
            default:
                next_state = S0;
        endcase
    end
endmodule
```

Mașina Moore

Un model general al unei mașini secvențiale Moore este prezentat mai jos. Ieșirea sa este dependentă doar de blocul stării curente, iar starea următoare este determinată pe baza intrării și a stării curente. În schema de mai jos, starea “Current State Register” este modelată utilizând bistabili D. Mașinile Moore obișnuite sunt descrise prin intermediul a trei blocuri, dintre care unul conține logică secvențială, iar celelalte două conțin logică de tip combinațională.



Exemplu de automat de stări Moore:




```
module moore_fsm(
    output reg parity,
    input clk,
    input reset,
    input x);

    reg state, next_state;
    parameter S0=0;
    parameter S1=1;

     // Partea secvențială
    always @(posedge clk or negedge reset)
        if (!reset)
            state <= S0;
        else
            state <= next_state;

    always @(*) begin
        case(state)
            S0: begin
                parity = 0;
                if (x)
                    next_state = S1;
                else
                    next_state = S0;
            end
            S1: begin
                parity = 1;
                if(!x)
                    next_state = S1;
                else
                    next_state = S0;
            end
        endcase
    end
endmodule
```

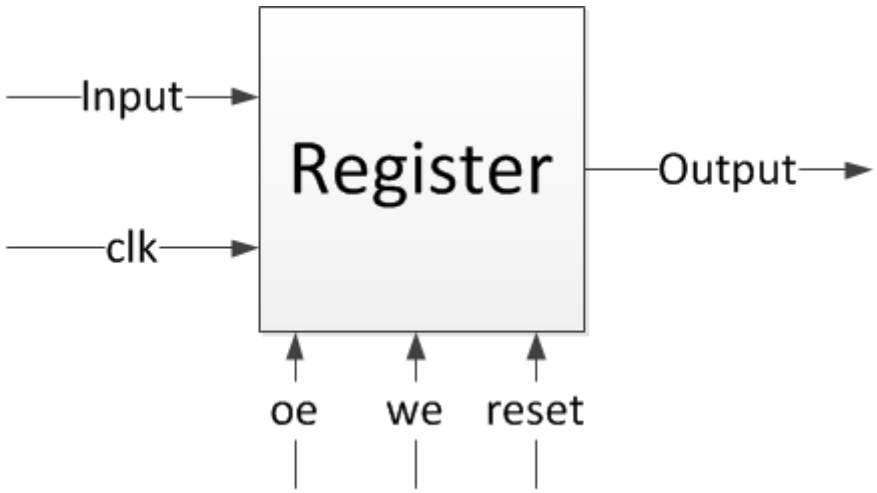
Depozitarea informației digitale: Registrul

Bistabilul este o celulă de memorie având capacitate de un bit, care poate fi utilizată pentru a stoca date digitale. Pentru a extinde capacitatea de stocare în ceea ce privește numărul de biți, se va utiliza un grup de bistabili, cunoscut și sub termenul de **registru**. Registrul de **n** biți este alcătuit din **n** bistabili și are capacitate de stocare de **n** biți.

Un registru, ca orice alt circuit secvențial este sensibil la schimbarea de front a semnalelor *clk* și *reset*. De asemenea, pentru a putea fi conectat la o magistrală, acesta are nevoie să execute două operații simple:

- citire - informația deja existentă în registru este preluată și eliberată pe ieșirea registrului
- scriere - informația aflată pe magistrală la un moment de timp se scrie în registru

Pentru o vizualizare mai clară a specificațiilor unui astfel de circuit digital secvențial, studiați interfața acestuia:



Memorie cu acces aleatoriu (RAM)

Memoria poate fi asociată cu creierul uman, fiind folosită pentru a stoca date și instrucțiuni. Memoria unui calculator este spațiul de stocare din calculator unde sunt păstrate datele care urmează să fie procesate și instrucțiunile necesare pentru procesare. Aceasta se împarte în mai multe elemente cu caracteristici similare, numite celule. Fiecare celulă are o adresă unică numerotată de la 0 la N-1, unde N este dimensiunea blocului de memorie (numărul de celule din memorie).

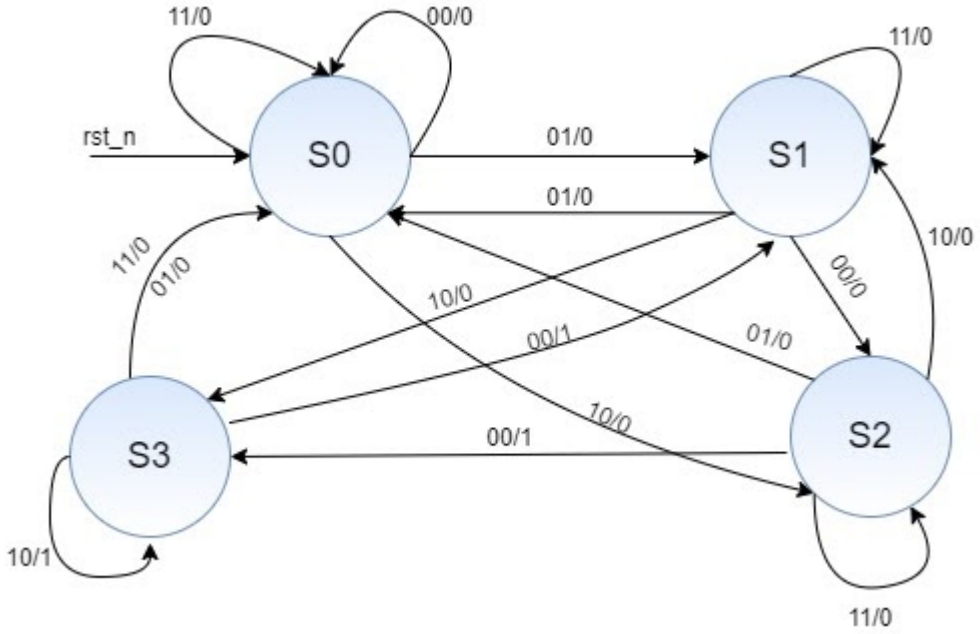
Componenta hardware de tip memorie a unui calculator unde sunt stocate sistemul de operare, programele de bază și datele utilizate la momentul curent, pentru a fi accesate cu ușurință de procesor se numește **RAM** (Random Access Memory). RAM-ul este o memorie volatilă, însemnând că toate informațiile stocate în acesta vor fi pierdute la deconectarea calculatorului de la sursa electrică, urmând să fie recuperate la repornirea sistemului de pe HDD/SSD. RAM-ul este mic, atât ca dimensiune fizică, cât și din punct de vedere al capacității de stocare de date.

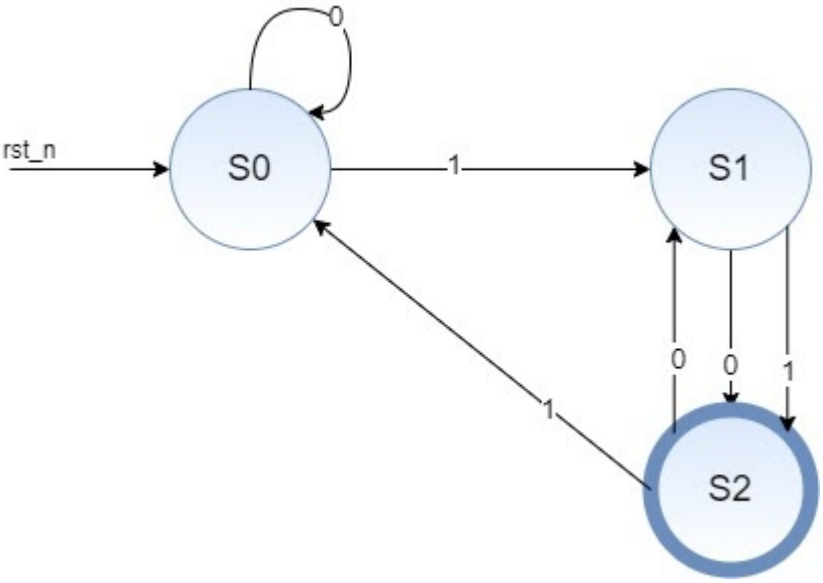
În comparație cu registrele, memoria de tip RAM este mai greu de accesat de către procesor. Fiind un circuit secvențial complex sunt necesari mai mulți cicli de ceas pentru a citi/scrie informația necesară.

Totodată, oferă o capacitate mult mai mare de stocare, de care registrele nu dispun. Prin urmare, pentru implementarea eficientă a unui circuit digital, este foarte importantă gestionarea resurselor între memorie și registre, astfel încât să se permită stocarea și accesul la toate informațiile necesare într-un timp cât mai scurt.

Exerciții

1. (Opțional - de implementat acasă) Implementați automatul de stări din figura de mai jos și identificați tipul acestuia:

1. 

2. 

1. Implementați modulul register pornind de la declarația din fișierul *register.v*. Semnalele oe și we reprezintă Output Enable, respectiv Write Enable.
   - oe controlează ieșirea registrului. Când oe este high ieșirea este activă având valoarea memorată de registru. Când oe este low ieșirea va fi 0. Acest semnal trebuie să fie asincron: modificarea lui va avea efect imediat asupra ieșirii și nu se va aștepta tranziția semnalului de ceas.
   - we controlează scrierea în registru. Când we este high registrul va memora valoarea aflată în semnalul de intrare. Când we este low valoarea registrului nu se va modifica, ignorând practic

semnalul de intrare. Acest semnal trebuie să fie sincron: modificarea valorii memorate de registru se face doar în momentul tranziției semnalului de ceas.
   - Semnalul *disp_out* este folosit pentru afișare/debugging pe display, iar valoarea acestuia trebuie să fie cea memorată de registru în momentul curent. În mod normal acest semnal nu este prezent într-un calculator. Acest semnal nu trebuie să fie afectat de oe, valoarea disponibilă pe disp_out fiind în orice moment egală cu valoarea memorată de registru.
   - Semnalul de reset rst_n este activ pe low (0).
   - *Hint*: Puteți folosi operatorul condiţional.
2. Parametrizați modulul **register** astfel încât data de intrare și ieșire din registru să aibă o dimensiune configurabilă.
   - *Hint*: Utilizați construcția de limbaj **parameter**
3. Pornind de la interfața modulului **sequential_multiplier** din scheletul de cod, implementați un automat de stări care să folosească instanțe parametrizate ale modulului **register** pentru a îndeplini următoarele funcționalități:
   - La activarea semnalului *write* să se scrie pe câte un registru (parametrizat corespunzător) valorile semnalelor a și b
   - La activarea semnalului *multiply* să fie extrase valorile din cele două registre, să se înmulțească și să se adauge pe un al treilea registru.
   - La activarea semnalului *display*, semnalul *out* să primească valoarea aflată pe cel de-al treilea registru.
   - Prioritatea celor trei semnale este dată de ordinea în care au fost descrise (e.g. dacă *write* este activ, se ignoră semnalele *multiply* și *display*; dacă *multiply* este activ, se ignoră semnalul *display*)
2. Vi se pune la dispoziție un RAM de tip Block Memory Generator instanțiat în modulul **ram_reader**. Completați modulul astfel încât să puteți gestiona citirea din memorie de la o adresă *am_out* în momentul în care semnalul *read* este activ (1).

Resurse

- Schelet de cod
- PDF laborator
- Soluție laborator