Implementarea sistemelor secvenţiale

Variante tehnologice (1)

- ASIC
  - *Application-Specific Integrated Circuits*
  - proiectarea - realizată în mod specific pentru un anumit circuit sau clasă de circuite
  - avantaj - optimizare pentru rezolvarea problemei abordate → performanţe superioare
  - tipuri
    - full-custom
    - semi-custom

Variante tehnologice (2)

- circuite programabile
  - permit implementarea a diferite automate
    - în funcţie de problema abordată
    - pot fi reconfigurate după necesităţi
  - avantaje
    - flexibilitate în proiectare şi depanare
    - preţ redus - datorită producţiei de serie
  - dezavantaj
    - nu pot asigura obţinerea de performanţe maxime

Circuite programabile (1)

- arii de porţi (*Gate Arrays*)
  - număr mare de celule
  - fiecare celulă - sumă (OR) de termeni produs (AND)
    - de obicei se folosesc doar o parte din intrări (câte sunt necesare)
  - corespunde cu modul de proiectare al circuitelor combinaţionale
  - celulele sunt interconectate - circuite complexe

Circuite programabile (2)

- CPLD
  - *Complex Programmable Logic Devices*
  - fiecare celulă include şi un bistabil
    - la ieşirea părţii combinaţionale
    - poate fi folosit sau nu (comportament pur combinaţional)
    - poate exista şi reacţie înapoi - de la ieşirea bistabilului spre partea combinaţională
  - modelează comportamentul secvenţial

Circuite programabile (3)

- FPGA
  - *Field Programmable Gate Arrays*
  - celulele - de obicei mai simple decât la CPLD
  - partea combinaţională - LUT (*Look-Up Table*)
  - implementare LUT - variante
    - multiplexor
    - memorie ROM
    - memorie RAM - mai uşor de reconfigurat

Implementare (1)

Probleme

1. proiectarea logică a circuitului
2. implementarea fizică
   - configurarea şi interconectarea celulelor
   - utilizarea circuitelor disponibile în cadrul celulelor
     - structura fizică poate diferi de cea logică

- dificil de gestionat

Implementare (2)

Soluţia

- compilatoare de hardware
  - realizează implementarea fizică
  - ţin cont de structura hardware concretă pe care se face implementarea
- se folosesc limbaje dedicate de descriere a hardware-ului
  - VHDL, Verilog etc.

Limbajul Verilog

Organizare

- circuitele - implementate ca module
- moduri de descriere a circuitelor
  - structurală
    - nivel jos
    - indicarea componentelor folosite
  - comportamentală
    - nivel înalt
    - se descrie comportamentul dorit al circuitului
  - pot fi combinate

Module

- nume modul
- lista parametrilor (semnale de intrare-ieşire)
- declararea tipului parametrilor
  - input
  - output
  - inout (mai rar)
- descrierea modulului
  - diverse moduri

Exemplu

- proiectarea unui multiplexor $2 \rightarrow 1$
  - intrări
    - de date: I0, I1
    - de selecţie: Sel
  - ieşire: E
- pot fi folosite ambele moduri de descriere, inclusiv combinaţii

Varianta 1

```
module MUX(Sel,I0,I1,E);
input Sel,I0,I1;
output E;
assign E=(I1&Sel)|(I0&~Sel);
endmodule
```

- `assign` - asignare continuă
  - orice schimbare la intrare duce la schimbarea ieşirii (combinaţional)

Varianta 2

```
module MUX(Sel,I0,I1,E);
input Sel,I0,I1;
output E;
wire x,y,z;
not n0(z,Sel);
and a1(x,I1,Sel);
and a0(y,I0,z);
or o0(E,x,y);
endmodule
```

Varianta 2 (continuare)

- `wire` - declară semnale interne
- `not`, `and`, `or` - funcţii (module) predefinite
  - primul parametru - ieşirea
  - număr variabil de intrări
- se pot utiliza şi alte module definite de utilizator

Varianta 3

```
module MUX(Sel,I0,I1,E);
input Sel,I0,I1;
output E;
reg E;
always @(Sel,I0,I1)
  E=(I1&Sel)|(I0&~Sel);
endmodule
```

Varianta 3 (continuare)

- `reg` - variabila poate reţine o valoare între două asignări
  - de obicei secvenţial (dar nu neapărat)
- `always` - buclă infinită
  - se execută de fiecare dată când se modifică una din intrările din lista asociată
  - asignarea pentru variabila E nu este continuă
    - valoarea variabilei E se schimbă doar când se execută instrucţiunea respectivă

Varianta 4

```
module MUX(Sel,I0,I1,E);
input Sel,I0,I1;
output E; reg E;
always @(Sel,I0,I1)
begin
  if(Sel==0) E=I0;
  else E=I1;
end
endmodule
```

Elemente ale limbajului Verilog

Vectori (1)

- folosiţi pentru semnalele care constau din mai mulţi biţi
  - numere, adrese etc.
  - domeniul din care fac parte indicii - flexibil

```
input [7:0]x;
wire [0:7]y;
reg [10:3]z;
```

  - toate definesc semnale pe 8 biţi

Vectori (2)

- pot fi accesaţi
  - integral (toţi biţii simultan)
  - bit cu bit
  - o parte din biţi

```
z=x;
z[6]=y[4];
z[5:3]=x[6:4];
```

Tablouri (1)

- similare celor din limbajele software
- la un moment dat poate fi accesat un singur element al tabloului

```
wire x[3:0]; //tablou cu 4
  elemente pe 1 bit fiecare
a=x[0]; //corect
b=x; //eroare
c=x[2:0]; //eroare
```

Tablouri (2)

- elementele unui tablou pot fi vectori
- pot fi accesate direct şi la nivel de bit

```
reg [7:0]y[15:0]; // tablou de
  16 elemente pe 8 biti fiecare
reg [7:0]z;
z=y[5];
```

Atribuiri (1)

- utilizate în descrierile comportamentale
  - nu şi pentru `assign`
- două tipuri
  1. blocante (=)
  - atribuirea curentă va începe doar după terminarea celei anterioare
  2. neblocante (<=)
  - atribuirea curentă poate începe înainte de terminarea celei anterioare

Atribuiri (2)

- `a` are iniţial valoarea 7

```
a=5;
b=a+3;
```

  - `b` va primi valoarea 8

```
a<=5;
b<=a+3;
```

  - `b` va primi valoarea 10

Întârzieri (1)

- instrucţiunile din exemplul anterior sunt raportate ca executându-se la acelaşi moment
  - indiferent de tipul de atribuire utilizat
- trebuie să specificăm faptul că o instrucţiune se execută mai târziu decât cea dinaintea sa
  - şi după cât timp

Întârzieri (2)

```
      a=5;
#5    b=a+3;
```

- a doua instrucţiune se execută la 5 unităţi de timp după prima
- deci nu mai putem avea execuţie în paralel
  - chiar dacă se folosesc atribuiri neblocante
- dacă nu specificăm o întârziere - similar cu #0

Baze de numeraţie

- valori exprimate în diferite baze de numeraţie
  - baza 2: `7'b0101110`
  - baza 8: `8'o247`
  - baza 10: `8'd25` sau `25` (implicit)
  - baza 16: `12'hA0F`
- numărul dinaintea semnului `'` exprimă întotdeauna numărul de biţi al reprezentării

Înaltă impedanţă

- valorile pe care le poate lua un semnal
  - 0
  - 1
  - `x` (nedeterminat)
  - `z` (înaltă impedanţă)
- exemplu de utilizare
  - `a <= 8'bzzzzzzzz;`
  - valoarea `z` nu poate fi testată

Concatenări de semnale

- sintaxa: **{***semnal***,***semnal***,...}**
- exemplu
  - `reg [3:0]a,b;`
  - `reg [7:0]c,d;`
  - `{a[2],c} <= {d[6:1],b[2:0]};`