Laborator 4

Fundamentele Calculatoarelor

Obiective:

- Reprezentarea numerelor negative in binar
  - Semn-Marime (SM)
  - Complement de 1 (C1)
  - Complement de 2 (C2)
- Operatii cu Magistrale in Verilog
- Formatul constantelor in Verilog
- Înțelegerea conceptului de “Negative Test”

Semn Marime (SM)

Formaul Semn Marime foloseste cel mai semnificativ bit (MSB- cel mai din stanga, pe pozitia cea mai mare) pentru a reprezenta semn (+ sau -) astfel:

- 0 == + 0️⃣ ➕
- 1 == - 1️⃣ ➖

Restul bitilor ce il urmeaza, reprezinta modulul numarului, transformat in binar
Inainte de a transforma un numar din zecimal in binar folosind formatul SM, trebuie sa stim pe cati biti sa il reprezentam. Altfel, nu am stii care este cel mai semnificativ bit.

- *Ex: -5 pe 8 biti se reprezinta astfel: 10000101*

Dezavantaje:

In reprezentarea SM operatiile de inmultire si impartire sunt simplu de implementat, dar cele mai frecvente operatii, care sunt adunarea si scaderea, intampina dificultati in traducerea in circuitele masinii de calcul. Facem apel la un principiu fundamental al proiectarii calculatoarelor care este make the common case fast (favorizeaza cazul mai frecvent in detrimentul cazului mai putin frecvent), principiu enuntat de Amdahl.
Pentru cifra „0“ a carei testare in calculator se efectueaza foarte frecvent avem doua reprezentari:

- +0 = 0000 (pe 4 biti)
- -0 = 1000 (pe 4 biti)

Din acest motiv, pe 8 biti, in semn marime se pot reprezenta valori intregi in intervalul [-127, +127].
Pentru n biți: [-($2^{(n-1)}$ - 1), +($2^{(n-1)}$ - 1)]

Complement fata de 1(C1)

Complementul fata de 1 a unui numar binar se obtine prin schimbarea lui 0 in 1 si a lui 1 in 0, dupa cum se prezinta in continuare:

- Numar binar: **1**0110101
- Complementul fata de 1: **1**1001010

!!-> bitul de semn nu se modifica
La fel ca la SM, inainte de a reprezenta un numar in C1, va trebui sa stim pe cati biți sunt necesari.

- *Ex: -5 pe 8 biti se reprezinta astfel in SM: 10000101*
  -5 pe 8 biti se reprezinta astfel in C1: 11111010

Dezavantaje:

Cifra 0 pe 4 biti in C1 este reprezentata ca: 0000 SAU 1111

- *Ex: Sa consideram ca vrem sa adunam +5 si -5. Rezultatul ar trebui sa fie 0 = 0000.*
  +5 = 0101
  -5 = 1101(SM)→ 1010(C1)
  +5 + (-5) = 0101 + 1010 = 1111 = 0 ⁉

Se poate observa ca si in cazul formatului de C1 exista 2 reprezentari a cifrei zero: +0, respectiv -0.
La fel ca si in cazul formatului SM, pe 8 biti se pot reprezenta valori din intervalul [-127, +127].

Complementul fata de 2 (C2)

Pentru a converti un numar in formatul C2 este necesara:

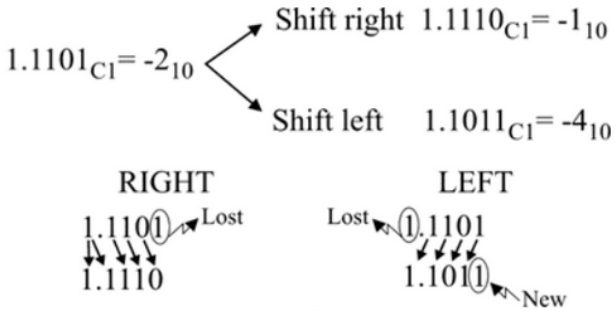
- conversia sa in C1
- La cel mai putin semnificativ bit al rezultatului obtinut anterior se adauga 1

  binar->SM-> C1+1= C2
- *Ex: Sa consideram ca vrem sa adunam +5 si -5. Rezultatul ar trebui sa fie 0 = 0000.*
  +5 = 0101
  -5 = 1101(SM)→ 1010(C1)→1011(C2)
  +5 + (-5) = 0101 + 1011= 0000 = 0 ⁉!!

Bitul de paritate isi pastreaza proprietatea.
Intervalul pe 8 biti: [-128, 127].
Pentru n biți: [-($2^{(n-1)}$ - 1), +($2^{(n-1)}$ - 1)].

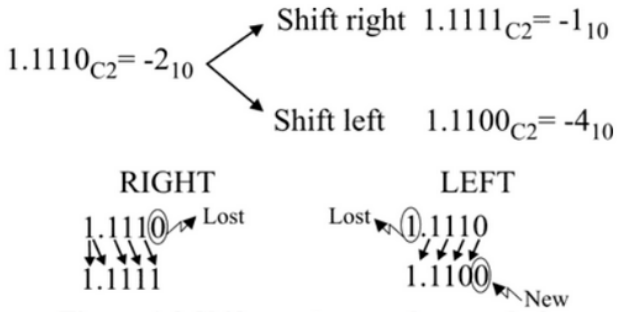
Shiftarea in formatul C1

La C1 avem shiftarea la stânga/dreapta din Figura de mai jos, unde se explică şi modul de shiftare: bitul care vine pe poziţia lăsată liberă este identic cu cel de dinainte de shiftare (shiftarea la dreapta – right), sau este ‘1’ (shiftarea la stânga).



Shiftarea in formatul C2

La C2, Figura de mai jos ne prezintă maniera în care se shiftează la stânga/dreapta: bitul de pe poziţia rămasă liberă va fi identic cu cel de dinainte de shiftare (right shift), sau are valoarea ‘0’ (left shift).



Operatii cu Magistrale in Verilog

📝**Reamintim:**

In Verilog, o magistrala mai este denumita si vector. Aceasta poate fi reprezentata folosind sintaxa:

- input [**MSb**:**LSb**] nume_variabila
- output [**MSb**:**LSb**] nume_variabila
  !! daca folosim always adaugam reg output reg[**MSb**:**LSb**] nume_variabila
- reg [**MSb**:**LSb**] nume_variabila
- wire [**MSb**:**LSb**] nume_variabila

Selectarea unui bit dintr-o magistrala

Selectia/accesarea unui bit dintr-o magistrala este echivalenta cu accesarea unui element dintr-un vector in C, si anume:

Sintaxa:
var[pos] → selecteaza bit-ul de pe pozitia **pos** din magistrala **var.**
!! Numaratoarea pozitiilor incepe de la dreapta (bit-ul 0) la stanga.
!! Se pot folosi variabile in locul lui **pos** pentru aceasta selectie.
!! Operatia poate aparea in stanga egalului
Exemplu de program, pentru numararea bitilor de 1 din var(8 biți), putem folosi urmatorul cod:

```
module count1s(input[7:0] var, output reg[3:0] k);  /* var poate avea maxim
8 biți de 1 --> k trebuie sa fie pe 4 biți*/
      /* Modulul numara cati biți de 1 sunt in intrarea var si returneaza
numărul prin k */
      reg[3:0] i;
      always @ (var) begin
          k = 0;  // initializam counter-ul
            for(i = 0; i < 8; i = i + 1) begin
                  if(var[i] == 1) begin
                        k = k + 1;
                  end
            end
      end
endmodule

module count1s_tb();
      reg[7:0] var_tb;
      wire[3:0] k_tb;

      count1s dut (.var(var_tb), .k(k_tb));
      initial begin
```

```
	for(var_tb = 0; var_tb < 255; var_tb=var_tb+1) begin
		#1;
		$display("var = %d (binar: %b) --> k = %d", var_tb, var_tb,
k_tb);
	end
	#1;
	$display("var = %d (binar: %b) --> k = %d", var_tb, var_tb, k_tb);
	end
endmodule
```

Exemplu:
Consideram **var** = 4'b0101, atunci var[0] = 1, var[1] = 0, var[2] = 1, var[3] = 0;

Selectarea unei secvente de biti dintr-o magistrala prin specificarea bitilor de start si stop (Part-Select)

Selectia mai multor biți dintr-o magistrala se face asemanator cu selectia unui singur bit, cu exceptia faptului, ca de data asta definim inceputul si sfarsitul secventei.

Sintaxa:
var[start:stop] → selecteaza secventa de biți incepand cu start pana la stop (ambele capete sunt incluse).
‼ start >= stop.
‼ start si stop nu pot fi variabile! (var[i:j] va genera eroare). start si stop trebuie sa fie **doar** numere
‼ Operatia poate aparea in stanga egalului

Exemplu:
Consideram var = 8'b11001010, atunci var[4:0] = 5'b01010, var[7:4] = 4'b1100, var[2:3] **EROARE**‼, var[a:b] **EROARE**‼

Exemplu de program ce imparte o intrare pe 8 biți in 2 parti, fiecare pe 4 biti:

```
module split_byte_to_nibble(input[7:0] var, output[3:0] msn, lsn);
	/* Modulul primeste o intrare pe 8 biti si o imparte in cei mai
semnificativi 4 biti (msn) si cei mai putin semnificativi 4 biti (lsn) */
	assign msn = var[7:4];  // 7 - 3 + 1 = 4 biti
	assign lsn = var[3:0];
endmodule
```

```
module split_byte_to_nibble_tb();
	reg[7:0] var_tb;
	wire[3:0] msn_tb, lsn_tb;

	split_byte_to_nibble dut (.var(var_tb), .msn(msn_tb), .lsn(lsn_tb));
	initial begin
	for(var_tb = 0; var_tb < 255; var_tb=var_tb+1) begin
		#1;
		$display("var = 0b%b --> msn = 0b%b, lsn=0b%b", var_tb, msn_tb,
lsn_tb);
	end
	#1;
	$display("var = 0b%b --> msn = 0b%b, lsn=0b%b", var_tb, msn_tb,
lsn_tb);
	end
endmodule
```

Selectarea unei secvente de biti dintr-o magistrala prin selectarea bitului de start si a lungimii (indexed Part-Select)

Aceasta operatie este asemanatoare cu Part-Select, cu distinctia ca nu specificam ambele valori de start si de stop. Aceste valori vor fi calculate de catre verilog
Sintaxa:
var[pos +: lungime] → Selectia bitilor pornind de la pos si selectand un numar de biți din stanga lui pos egal cu lungime (se include si bitul de pos). Se poate traduce in **var[pos+lungime-1:pos]**
var[pos -: lungime] → Asemanator cu functionalitatea de mai sus, insa selecteaza bitii de la dreapta lui pos. Se poate traduce in: **var[pos:pos-lungime+1]**
‼ **pos** poate fi variabila, dar **lungime** trebuie sa fie numar
‼ Operatia poate aparea in stanga egalului

Exemplu:

```
reg [31: 0] a_vect;
reg [31:0] sel;
reg [63: 0] dword;

a_vect[ 0 +: 8] // == a_vect[ 7 : 0]
a_vect[15 -: 8] // == a_vect[15 : 8]
```

```
dword[8*sel +: 8] // variable part-select with fixed width
```

Concatenarea magistralelor

Este inversul lui Part-Select. Cum ii spune si numele, operatia de concatenare alipeste magistrale/biti in ordinea data

Sintaxa:
{variabila1, **variabila2**, **variabila3**, … **variabilan}**
!! Operatia poate aparea in stanga egalului

Exemplu de program ce imparte o intrare pe 8 biți in 2 parti, fiecare pe 4 biți (aceeasi problema rezolvata mai sus cu part-select):

```
module split_byte_to_nibble(input[7:0] var, output[3:0] msn, lsn);
	/* Modulul primeste o intrare pe 8 biti si o imparte in cei mai semnificativi 4 biti (msn) si cei mai putin semnificativi 4 biti (lsn) */
	assign {msn, lsn} = var;
endmodule

module split_byte_to_nibble_tb();
	reg[7:0] var_tb;
	wire[3:0] msn_tb, lsn_tb;
	split_byte_to_nibble dut (.var(var_tb), .msn(msn_tb), .lsn(lsn_tb));
	initial begin
	for(var_tb = 0; var_tb < 255; var_tb=var_tb+1) begin
	#1;
	$display("var = 0b%b --> msn = 0b%b, lsn=0b%b", var_tb, msn_tb, lsn_tb);
	end
	#1;
	$display("var = 0b%b --> msn = 0b%b, lsn=0b%b", var_tb, msn_tb, lsn_tb);
	end
endmodule
```

Exemplu de program ce primeste 2 intrari pe 4 biți si le concateneaza la iesirea pe 8 biți:

```
module merge_nibble_to_byte(input[3:0] msn, lsn, output[7:0] var);
	assign var = {msn, lsn};
endmodule

module merge_nibble_to_byte_tb();
	reg[3:0] msn_tb, lsn_tb;
	wire[7:0] var_tb;
	merge_nibble_to_byte dut (.msn(msn_tb), .lsn(lsn_tb), .var(var_tb));
	initial begin
```

```
        for(msn_tb = 0; msn_tb < 15; msn_tb=msn_tb+1)
        for(lsn_tb = 0; lsn_tb < 15; lsn_tb = lsn_tb+1) begin
                #1;
                $display("msn = 0b%b, lsn=0b%b --> var = 0b%b", msn_tb, lsn_tb, var_tb);
        end
        end
endmodule
```

Repetarea

Este asemanatoare cu concatenarea. Se poate “traduce” prin concatenarea repetata a unei singure magistrale sau unui singur bit.

Sintaxa:
{nr{var}} → Concateneaza **var** de **nr** ori
!!nr trebuie sa fie un numar

Exemplu de program ce primeste un numar la intrare, pe 32 de biți, reprezentat in Complement de 2 si il extinde pe 64 de biți:

```
module int32_to_int64(input[31:0] nr_32, output[63:0] nr_64);
        assign nr_64 = { {32{nr_32[31]}}, nr_32};  /* extinderea unui numar se face prin
repetarea bitului de semn la stanga */
endmodule

module int32_to_int64_tb();
        reg signed [31:0] nr_32_tb;
        wire signed [63:0] nr_64_tb;
        int32_to_int64 dut (.nr_32(nr_32_tb), .nr_64(nr_64_tb));
        initial begin
        nr_32_tb = -10;
        repeat (20) begin
        #1;  /* asteptam sa se proceseze iesirea */
        $display("nr_32=%d (hex: %X) --> nr_64 = %d (hex: %X)", nr_32_tb, nr_32_tb,
nr_64_tb, nr_64_tb);
        nr_32_tb = nr_32_tb + 1;
        end
        end
endmodule
```

Formatul constantelor

Acest format este:
<numar_biti>'**<specificator_baza>**<valoare>, unde:
<numar_biti> e un numar zecimal intreg, reprezentand numarul de biti pe care sa fie reprezentata constanta. Desi optional, este un "good practice" sa se specifice numarul de biti al unei constante.
<specificator_baza> poate fi b pentru binar, o pentru octal, d pentru decimal si h pentru hexazecimal. Este optional, de asemenea, cu baza zecimala fiind implicita.
<valoare> este valoare efectiva, scrisa in baza definita anterior.

Exemple:
32'haabbccdd
32'hAABBCCDD
8'o143
8'b11000011
16'd865
'd312 // numărul minim de biți necesari vor fi folositi
'haa // numărul minim de biți necesari vor fi folositi
'hAb
321

Negative Test

Testarea negativa se traduce prin testarea unei componente / unui sistem / unei functii / etc. intr-un mod in care nu a fost menit a fi folosit. Ideea testarii negative este identificarea reactiilor in sistem sau software la stimuli invalizi sau date de intrare corupte: sistemul sau software-ul ar trebui sa nu fie afectate negativ de acesti stimuli (sa nu genereze erori de logica, sa nu se inchida, sa nu arunce exceptii, etc.). Testarea negativa se observa cel mai bine in interfetele web, mai ales formulare, unde un tester ar putea:

- Introducerea literelor in locuri nepermise (ex. Anul nasterii)
- Necompletarea unui camp obligatoriu
- Adaugarea de caractere speciale in locuri unde este evident ca nu ar trebui sa apara (ex. Nume de famili / CNP / Serie Buletin)
- s.a.m.d

In toate aceste cazuri, aplicatia web ar trebui sa functioneze in continuare, cu semnalarea greselilor. Aplicatia web nu ar trebui sa accepte aceste valori si ar trebui sa le trateze ca invalide, neinregistrand datele.

Un alt caz de astfel de test se poate observa si la parole, cand, de exemplu, un site obliga utilizatorul sa indeplineasca anumite criterii pentru parola (ex. Mai mult de 10 caractere; cel putin o litera mare, o litera mica, un caracter special; sa nu contina numele de utilizator; sa nu fie in lista celor mai usor de spart parole: password, parola, etc). Un tester ar putea incerca neindiplinirea fiecarei conditii in parte si sa observe reactia aplicatiei, care nu va trebui sa accepte aceste valori si sa creeze un utilizator.