Laborator 02
Extinderea proiectului curent: afișorul pe 7 segmente

1. Obiective

Descrierea, implementarea și testarea:

- **Afișorul pe 7 segmente**
- **Unitate Aritmetico-Logică simplă (Arithmetic Logic Unit - ALU)**

Aprofundarea cunoștințelor legate de:

- Vivado Webpack
- Xilinx Vivado Design Suite User Guide
- Digilent Development Boards **(DDB)**
  - **Digilent Basys Board – Reference Manual**
- Artix 7 FPGA

2. Afișorul pe 7 segmente cu 4 cifre

Placa Basys vine echipată cu un afișor pe 7 segmente cu 4 cifre (Seven Segment Display - SSD). Pe scurt, această interfață folosește șapte leduri pentru fiecare cifră; fiecare cifră este activată de un semnal de anod. Toate semnalele interfeței SSD (7 semnale comune de catod și 4 semnale distincte de anod) sunt active pe 0. Semnalele de catod controlează ledurile care se aprind de pe acele cifre care au semnalul de anod activ (de exemplu dacă se activează toate 4 anodurile, atunci se va afișa aceeași cifră pe cele 4 poziții).

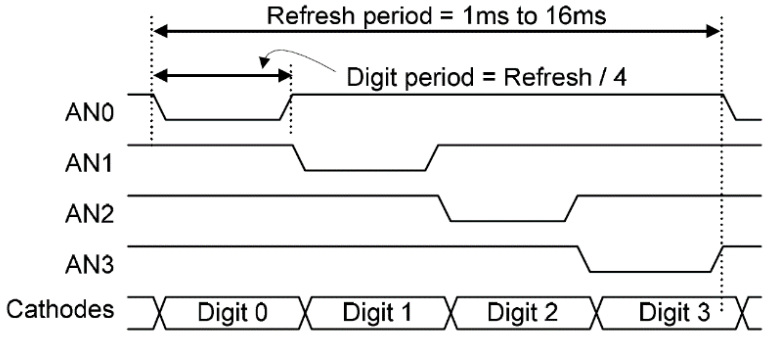


Figura 1: Diagrama de timp pentru SSD [1]

Pentru a afișa 4 cifre diferite pe SSD, este necesară implementarea unui circuit care trimite cifrele pe semnalele de catod ale SSD în concordanță cu diagrama de timp din Figura 1. Perioada maximă de reîmprospătare (refresh) este astfel calculată

încât ochiul uman să nu perceapă aprinderea și stingerea succesiva a fiecărei cifre de pe SSD (16 ms <=> 60 Hz). Se realizează astfel o afișare ciclică a cifrelor (la un moment dat doar o cifră este afișată, dar ochiul nu percepe asta). Deschideți manualul de referință Basys și citiți secțiunea legată de afișorul pe 7 segmente.

În figura de mai jos se prezintă o posibilă implementare a circuitului de afișare pe SSD. Intrările sunt 4 semnale de 4 biți (cifrele de afișat) și semnalul de ceas al plăcii; ieșirile sunt reprezentate de semnalele de anod (an) și semnalele de catod (cat), toate active pe zero.

În acest circuit există un automat cu stări finite, cu o implementare particulară. Care componentă îl reprezintă, și care sunt semnalele lui de control (ieșirile) ?

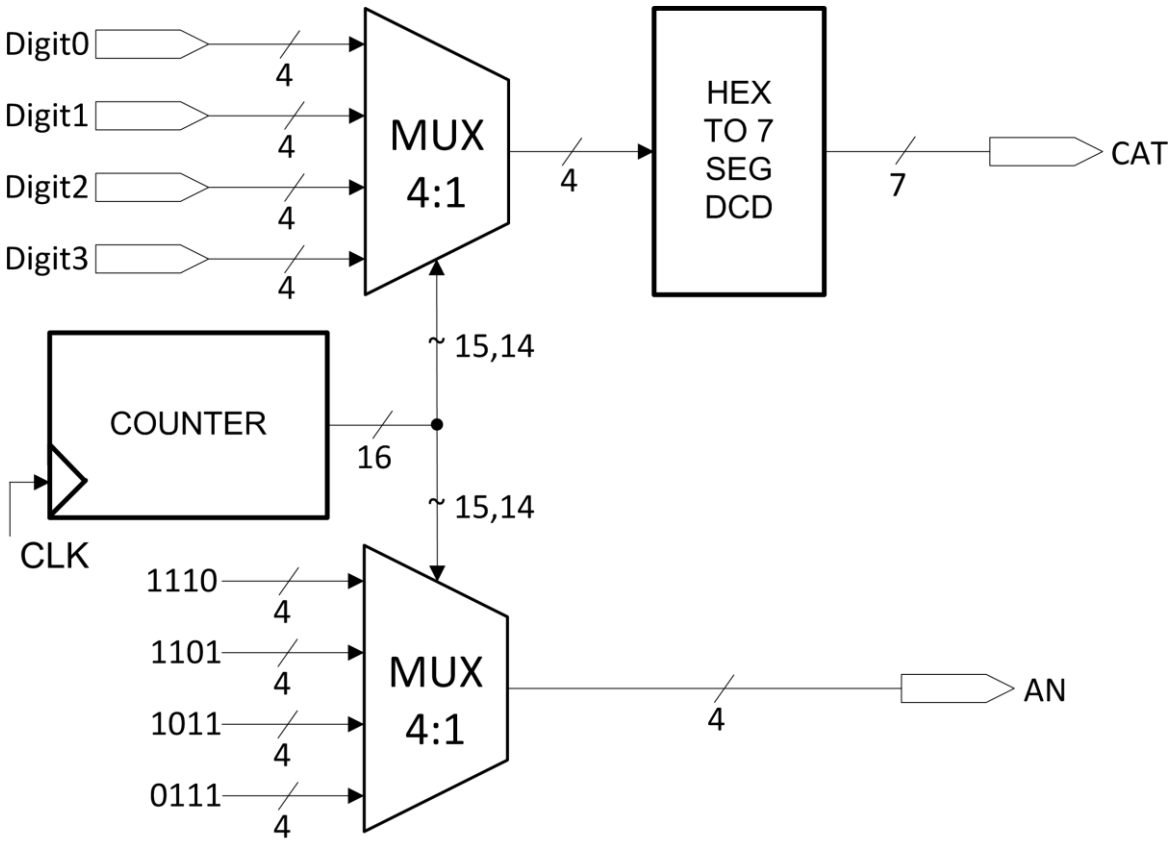


Figura 2: Schema circuitului de afișare SSD

3. Structura generală a proiectelor pentru placa Basys (pentru acest semestru)

După ce veți termina activitatea 5.1 din acest laborator, toate proiectele viitoare vor avea o structură generală ca în figura de mai jos. Această structură oferă interfața necesară pentru lucrul cu placa Basys. Descrierea circuitelor specifice fiecărui laborator (comportamental și / sau prin instanțierea unor componente) se va face în nor.

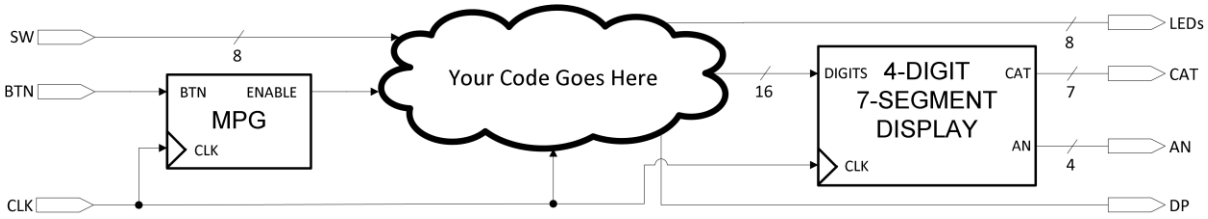


Figura 3: Schema generală a viitoarelor proiecte

4. Operații tipice pentru o ALU

Fiecare operație care urmează este descrisă din punct de vedere teoretic, după care urmează indicii privind descrierea în VHDL (mult simplificată față de descrierea teoretică, ajungând o linie de cod VHDL).

4.1. Sumatoare (Adders)

Un sumator este un circuit digital care efectuează adunarea între numere. În calculatoarele moderne, sumatoarele sunt plasate in unitatea aritmetico-logica (ALU), împreună cu alte operații. Ecuațiile booleane pentru un sumator complet pe 1 bit sunt:

$$Sum = A\ xor\ B\ xor\ Cin$$
$$Cout = (A\ and\ B) or\ (A\ and\ Cin)\ or\ (B\ and\ Cin)$$

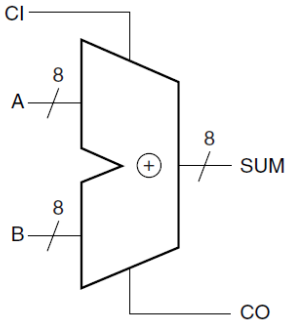


Figura 4: Sumator complet pe 8-Biți (cu Carry in și Carry out)

VHDL: …suma <= termen1 + termen2

- în acest laborator ignoram problema transportului

4.2. Scăzătoare (Subtractors)

Un scăzător este un circuit digital care efectuează operația de scădere. În hardware, pentru numere reprezentate în complement față de 2, scăderea se realizează prin adunarea cu negativul scăzătorului.

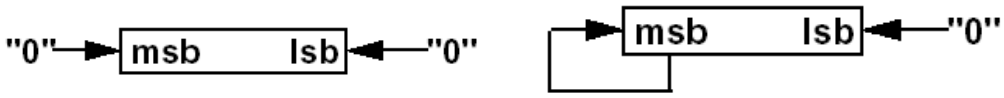
$$A - B = A + \overline{B} + 1$$

VHDL: …diferenta <= termen1 - termen2

4.3. Circuite de deplasare (Shifters)

Un circuit de deplasare este un circuit digital care translatează un cuvânt, conținând un sir de biți, cu un număr specificat de poziții. Există două tipuri de circuite de deplasare:

1. Deplasare logică – pe pozițiile rămase goale se pune 0.
2. Deplasare aritmetică – la deplasare spre dreapta se extinde bitul de semn.



Deplasare logică Deplasare aritmetică

Figura 5: Tipuri de deplasare

În general, pentru ca Xilinx să recunoască explicit un circuit ca fiind de deplasare, circuitul trebuie sa fie combinațional, cu două intrări și o ieșire, și să se folosească doar operațiile predefinite de deplasare (sll, srl, etc.) sau operatorul de concatenare &.

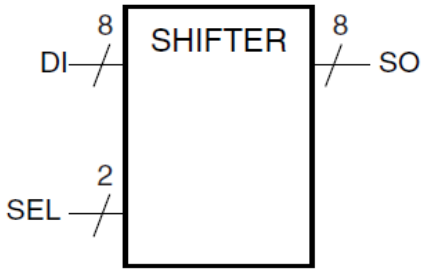


Figura 6: Circuit de deplasare

VHDL: …exemplu pe 32 de biți, deplasare stanga cu 5 poziții

iesire(31 downto 0) <= intrare(26 downto 0) & ”00000”

Ca lectură opțională, în anexa 1 din acest document, se prezintă un circuit de deplasare combinațional realizat prin cascadarea mai multor niveluri de multiplexoare, care poate efectua deplasare variabilă (de la 0 până la maxim 7 poziții).

4.4. Detector de zero (Zero Detector)

Detectorul de zero pe n biți este implementat de obicei printr-o poartă NOR, cu intrarea pe n biți, ieșirea de 1 bit. Acest circuit este folosit în special în unitățile ALU pentru a evalua condiția de salt (de exemplu pentru instrucțiunea Branch on Equal la procesorul MIPS).

VHDL: …atribuire concurentă cu *when/else*

4.5. Extindere cu semn/zero (Sign/Zero Extender)

Circuitul de extindere cu semn / zero din procesorul MIPS este folosit în operații aritmetico-logice pentru care unul de operanzi este un imediat pe 16 biți. Extensia este necesară deoarece ALU lucrează cu operanzi pe 32 de biți.

Atenție: veți avea nevoie de descrierea acestui circuit în laboratoarele viitoare.

VHDL: …în funcție de valoarea unui bit de selecție (cu semn / zero) folosiți operatorul de concatenare **&** pentru a construi semnalul extins de ieșire (pentru extensie cu semn, se testează bitul de semn pentru a determina daca se concatenează cu biți de 0 sau de 1).

4.6. Comparatoare

Un comparator este un circuit digital care compară două numere în formă binară și generează un semnal de 1 bit, zero sau unu, care arată dacă numerele comparate sunt egale sau nu. Într-o ALU, comparația a două numere se face efectuând operația de scădere (fără scrierea rezultatului) după care se verifică dacă rezultatul este zero (vezi detectorul de zero).

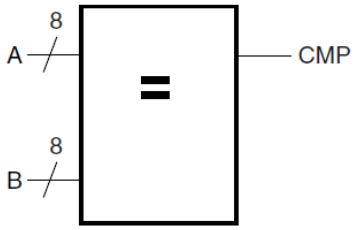


Figura 7: Comparator de numere pe 8-biți

5. Activități practice

5.1. Implementarea circuitului de afișare pentru SSD, 4 cifre

Se continuă lucrul în proiectul din laboratorul anterior (*test_env*, cel puțin din activitatea 3.3, cu MPG inclus și funcțional). Descrieți în VHDL o componentă (entitate separată, SSD, adăugați sursă nouă la proiect) care să implementeze circuitul de afișare pe 4 cifre, conform schemei din Figura 2. **În noua entitate creată, folosiți doar semnale declarate în arhitectură, procese pentru a descrie numărătorul și cele două multiplexoare (cu case), și folosiți Language Templates pentru a implementa "HEX TO 7 SEG DCD" (componenta din afișor care transformă cifra pe 4 biți în combinația de 7 leduri aprinse/stinse).**

Declarați noua componentă în arhitectura entității *test_env* și instanțiati-o. Conectați un numărător pe 16 biți la cele 4 intrări ale afișorului SSD. Numărătorul se va incrementa doar la apăsarea unui buton (folosiți componenta MPG pentru a valida incrementarea numărătorului). Acum, structura proiectului trebuie să semene cu structura descrisă în Figura 3.

5.2. O unitate aritmetico-logică simplă - ALU

Folosind proiectul anterior (care conține MPG și SSD), implementați o simplă ALU cu următoarele funcții: ADD, SUB, SHIFT LEFT 2, SHIFT RIGHT 2. Vezi figura următoare.

Rezultatul pe 16 biți al operațiilor din ALU sunt trimise (afișate) pe interfața SSD (Digit3...Digit0). Folosiți un numărător pe 2 biți (controlat prin buton, via MPG) pentru a selecta operația ALU dorită.

Operanzi de intrare ai ALU sunt comutatoarele (switches) plăcii Basys. Schema ALU este prezentată mai jos. Descrierea VHDL a acestei ALU se va face strict (fără entitate nouă) în arhitectura *test_env*, folosind doar semnale interne, procese și atribuiri concurente.

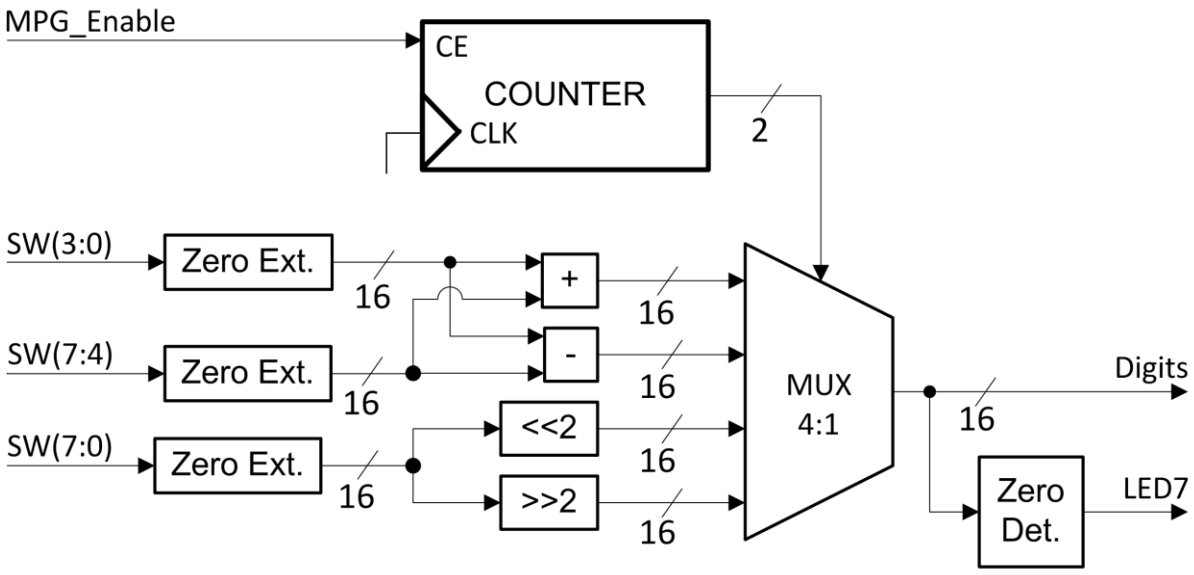


Figura 8: Schema unei ALU simple.

6. Referințe

[1] Manual de referinta pentru placa Basys 3 (Artix 7), disponibil pe site la Xilinx
[2] Xilinx Vivado WebPACK – aici.
[3] Help online pentru VHDL, http://vhdl.renerta.com/
[4] Vivado Design Suite User Guide, Appendix C: HDL Coding Techniques

Anexa 1 – Combinational Shifter Implementation

A shifter can also be implemented as a sequence of multiplexers. In such an implementation the output of one MUX is connected to the input of the next MUX in a way that depends on the shift distance. The number of multiplexers required for an n-bit word is $n * \log_2 n$.

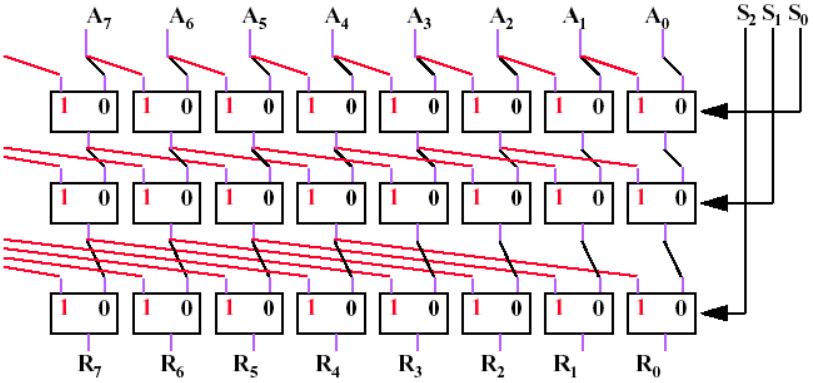


Figura 9: Multi-level (logarithmic) 8-bit right shifter

```
process(sw)
begin
   if sw(5) = '1' then    -- shift with 1 position
      if sw(7) = '0' then
         shift1 <= sw(3 downto 0) & '0';          -- shift left
      else
         shift1 <= sw(4) & sw(4 downto 1);        -- shift right arithmetic
      end if;
   else
       shift1 <= sw(4 downto 0);
   end if;
end process;

process(sw, shift1)
begin
   if sw(6) = '1' then    -- shift with 2 position
      if sw(7) = '0' then
         shift2 <= shift1(2 downto 0) & "00";                    -- shift left
      else
         shift2 <= shift1(4) & shift1(4) & shift1(4 downto 2);   -- shift right arithmetic
      end if;
   else
       shift2 <= shift1;
   end if;
end process;
led <= shift2;
```