Tema 1 - Transform the number

- Responsabili: Ionuț P, Ovidiu M (checker)
- Deadline soft (fără penalizări): **04.12.2022**, ora **23:59**
- Deadline hard (cu penalizări): **07.12.2022**, ora **23:59**
- Data publicării: **24.11.2022**
- Data ultimei actualizări: 28.11.2022, 16:02
- Istoric modificări:
  - 24.11.2022, 00:00
    - Publicare temă
  - 24.11.2022, 8:35
    - Modificare nume modul base2_to_base3, precizare pentru testare div_algo
  - 25.11.2022, 9:00
    - Specificație modul div_algo pur combinațional
  - 28.11.2022, 16:02
    - Publicare tester offline

Obiective

Tema are ca scop familiarizarea cu noțiunile de bază ale limbajului Verilog studiate în cadrul primelor laboratoare - module, construcții de limbaj, blocuri always, prin:

- divizarea problemei generale și organizarea ei în module cu o funcționalitate specifică;
- implementarea unui algoritm dat într-o manieră sintetizabilă;
- implementarea unui automat finit după o diagramă dată

Descriere și cerințe

Implementați în Verilog un automat finit care are ca scop transformarea unui număr din baza de numerație 2 în baza de numerație 3.

Circuitul va primi la intrare un număr exprimat în baza 2 împreună cu un semnal de *enable*. Rezultatul va fi reprezentarea numărului în baza 3, însoțit de asemenea de un semnal de validare, *done*.

Întrucât agloritmul de transformare are la bază o împărțire cu cât și rest, executată repetitiv, se preferă implementarea unui automat fitit, care va apela, la nevoie, un modul secundar ce va executa operația de împărțire.

Un exemplu detaliat este prezentat în anexă.

Implementare

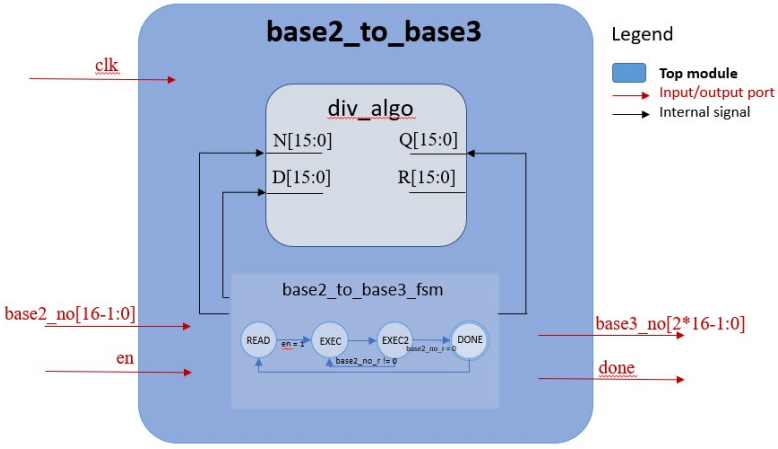


Fig1. Schemă bloc

Pentru rezolvarea temei este necesară împărțirea problemei în 2 module distincte, cu funcționalități specifice. Schema bloc este prezentată în *Fig1. Schemă bloc*. Modulele trebuie să respecte interfețele descrise mai jos, cu următoarele mențiuni:

- ieșirile pot fi declarate de tip registru;
- se pot crea module adiționale ce pot fi instanțiate în modulul principal.

base2_to_base3

Modulul principal, responsabil de implementarea automatului care va executa algoritmul de transformare din baza 2 în baza 3. Înăuntrul acestuia se va instanția modulul secundar, *div_algo*, prin intermediul căruia se va efectua operația de împărțire.

Modulul trebuie să respecte următoarea interfață:

```
module base2_to_base3 (
    output   [31 : 0]  base3_no,
```

```
    output            done,
    input    [15 : 0] base2_no,
    input             en,
    input             clk);
```

Descrierea semnalelor folosite de acest modul este următoarea:

- **base3_no** - valoarea numărului exprimată în baza 3 - fiecare cifra este codată pe 2 biți; urmărește exemplul din Anexă pentru mai multe detalii.
- **done** - semnal ce marchează sfârșitul conversiei; acesta trebuie asertat în momentul în care pe portul de ieșire *base3_no* este prezentă valoarea finală
- **base2_no** - numărul în baza 2 ce trebuie transformat în baza 3; acesta are sens să fie citit doar în momentul în care *en* are valoarea 1;
- **en** - semnal ce marchează faptul că numărul prezent pe portul *base2_no* este valid și poate fi citit
- **clk** - semnal de ceas

Schema propusă pentru implementarea automatului este prezentată în *Fig2. FSM*.

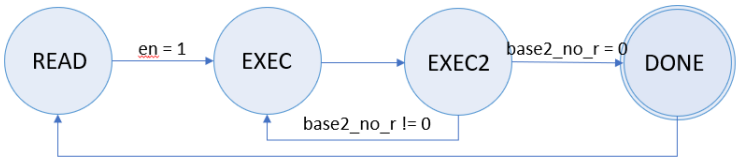


Fig2. FSM

Descrierea stărilor folosite este următoarea:

- **READ** - starea în care se citește valoarea de intrare și se salvează într-un registru auxiliar *base2_no_r*; trecerea la starea următoare se va face după ce numărul este citit;
- **EXEC** - starea în care se execută operația de împărțire; în această stare sunt date valorile corespunzătoare către modulul *div_algo*;
- **EXEC2**- starea în care se citește rezultatul împărțirii; în urma acestuia se alcătuiește numărul în baza 3 și se decide tranziția următoare, în funcție de valoarea lui *base2_no_r*;
- **DONE** - starea finală care marchează finalul execuției; în această stare se asertează semnalul *done*

Este permisă implementarea unui automat diferit față de cel propus, respectând efectuarea unei singure operații de împărțire pe ciclu de ceas!

div_algo

Modulul are rolul de a efectua operația de împărțire cu cât și rest a două numere naturale reprezentate pe 16 biți (Resurse), cu datele provenite de la modulul *base2_to_base3*. Modulul implementat trebuie să fie pur **combinațional**.

Modulul trebuie să respecte următoarea interfață:

```
module div_algo (
   output   [15 : 0]   Q,
   output   [15 : 0]   R,
   input    [15 : 0]   N,
   input    [15 : 0]   D);
```

Descrierea semnalelor folosite de acest modul este următoarea:

- **Q** - câtul
- **R** - restul
- **N** - deîmpărțitul
- **D** - împărțitorul

Operațiile folosind operatorii / și % nu sunt permise în rezolvarea modulului div_algo.

Notare

- +6 pct: implementarea corectă a modulului **div_algo**; se vor testa toate combinațiile de numere pe 8 biți și combinații aleatorii de numere pe 16 biți
- +4 pct: implementarea corectă a modulului **base2_to_base3**;
- +1 pct: fiecare bug găsit în implementarea de referință - cea din tester - (se acordă primei persoane care-l semnalează);

- -10 pct: folosirea construcțiilor nesintetizabile;
- -10 pct: folosirea construcțiilor cu număr variabil de iterații (ex: while x != 0);
- -6 pct: folosirea operatorilor *, / , % (excepție înmulțirea / împărțirea la puteri ale lui 2. ex: x * 2, x / 8, etc);
- -1 pct: lipsa fișierului README;
- -1 pct: **indentare haotică** (incluzând **spațiere inutilă**);
- -0.5 pct: pentru fiecare zi de întârziere; tema poate fi trimisă cu maxim 3 zile întârziere față de termenul specificat în enunț (față de deadline-ul soft);
- -0.5 pct: folosirea incorectă a atribuirilor continue (assign), blocante (=) și non-blocante (⇐);
- -0.2 pct: diverse alte probleme constatate în implementare (per problemă)
- -0.1 pct: comentarii inutile

Punctajul inițial al checker-ului nu reprezintă punctajul final al temei. Acesta va fi acordat de către asistent, în urma analizei individuale a fiecărei implementări.

Dacă tema primește 0 puncte pe platforma vmchecker, se pot acorda maxim 2 pct pentru ideea de implementare, la latitudinea asistentului. Ideea și motivele pentru care nu funcționează trebuie **documentate** temeinic în README și/sau comentarii. Temele care au **erori** de compilare vor fi notate cu 0 puncte.

Precizări

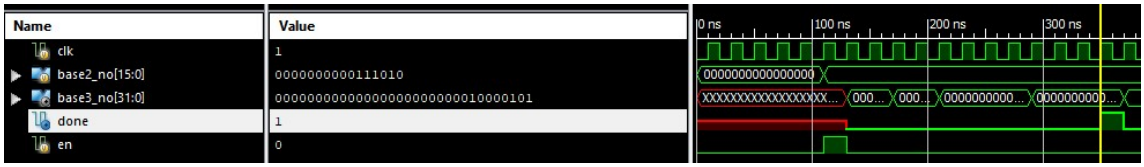
- Arhiva temei (de tip **zip**) trebuie să cuprindă în rădăcina sa (*fără alte directoare*) **doar**:
  - fișierele sursă (extensia .v);
  - fișierul README.
- Arhiva **nu** trebuie să conțină fișiere de test, fișiere specifice proiectelor etc.
- Fișierului README va conține minim:
  - numele și grupa;
  - prezentarea generală a soluției alese (ex: descrierea de nivel înalt a algoritmului folosit);
  - explicarea porțiunilor complexe ale implementării (poate fi făcută și în comentarii);
  - alte detalii relevante.
- Vmchecker ne permite să revenim la orice soluție încărcată de voi; cereți revenirea la cea mai convenabilă soluție trimisă (punctaj teste automate + depunctare întârziere) printr-un mail titularului de laborator.
- Tema trebuie realizată individual; folosirea de porțiuni de cod de la alți colegi sau de pe Internet (cu excepția site-ului de curs și a resurselor puse la dispoziție în conținutul temei) poate fi considerată **copiere** și va fi penalizată conform regulamentului.
- Aduceți-vă aminte de **recomandările** prezentate în Tema 0.
- În cadrul tester-ului există module de test pentru fiecare bloc pentru a-i verifica separat funcționalitatea.

Resurse

- **Tester** - Tester offline
- **Wikipedia** - Division algorithms [https://en.wikipedia.org/wiki/Division_algorithm]

- Verilog 2000 Standard [https://sutherland-hdl.com/papers/2001-SNUG-presentation_Verilog-2000_standard.pdf]
- Ghidul studentului la AC [https://ocw.cs.pub.ro/courses/ac-is/studentguide]
- Utilizarea vmchecker [https://ocw.cs.pub.ro/courses/ac-is/tutoriale/6-vmchecker-utilizare]
- Debugging folosind Xilinx ISE [https://ocw.cs.pub.ro/courses/ac-is/tutoriale/3-ise-debug]

Anexă



Pentru exemplificarea temei vom alege intrarea base2_no = 58. Algoritmul de transformare în baza 3 este reprezentat de împărțirea repetată și salvarea restului, până când numărul devine 0.

```
base2_no = 16'd58 (16'b0000_0000_0011_1010) -> base3_no = 32'b0000_0000_0000_0000_0000_0000_1000_0101;

58 > 0 ? DA -> 58 : 3 = 19 rest 1 ( coeficient pentru 3**0 )
19 > 0 ? DA -> 19 : 3 =  6 rest 1 ( coeficient pentru 3**1 )
 6 > 0 ? DA ->  6 : 3 =  2 rest 0 ( coeficient pentru 3**2 )
 2 > 0 ? DA ->  2 : 3 =  0 rest 2 ( coeficient pentru 3**3 )
 0 > 0 ? NU -> **STOP**

deci 58 = 2 * 3**3 + 0 * 3**2 + 1 * 3**1 + 1 * 3**0;

Valorile pe care trebuie să le scriem în base3_no sunt reprezentate de valorile coeficienților (2, 0, 1, 1).
Întrucât standardul VERILOG 2000 nu permite decât tablouri unidimensionale în interfața modulului, acestea sunt concatenate, fiecare valoare fiind reprezentată pe 2 biți.
    1 -> 'b01 salvat pe pozițiile [1:0]
    1 -> 'b01 salvat pe pozițiile [3:2]
    0 -> 'b00 salvat pe pozițiile [5:4]
    2 -> 'b10 salvat pe pozițiile [7:6]
Deci rezultatul final va fi {'b10, 'b00, 'b01, 'b01} -> 'b1000_0101
```

Pentru selecția unei secțiuni dinamice de 8 biți a unei variabile folosită într-o buclă iterativă, construcția aux[i+7:i] nu este permisă. Este necesară folosirea operatorului +: sau -:, astfel: aux[i+7-:8] sau aux[i+:8]. În acest fel sunt selectați biții din intervalul i, i+7. Mai multe detalii în Standardul Verilog 2000, Indexed Vector Part Selects, din resurse