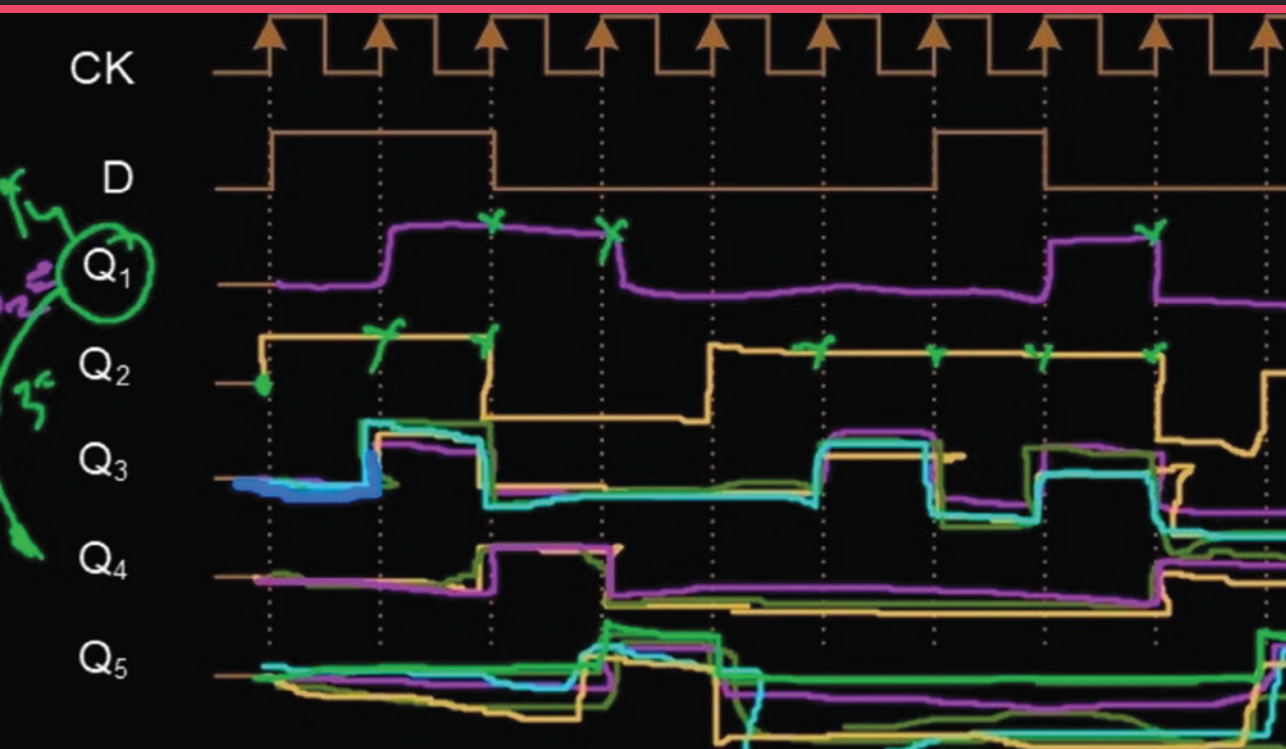


TOMUL PENTRU MEȘTERIT ȘI MUNCIT

DAN NICULA

ELECTRONICĂ DIGITALĂ

CARTE DE ÎNVĂȚĂTURĂ



Dan NICULA

ELECTRONICĂ DIGITALĂ

Carte de învățatură



**EDITURA
UNIVERSITĂȚII
TRANSILVANIA
DIN BRAȘOV**

2025

EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: Str. Iuliu Maniu nr. 41A

500091 Brașov

Tel.: 0268 476 050

Fax: 0268 476 051

E-mail: editura@unitbv.ro

Editură acreditată de CNC SIS, cod 81

ISBN 978-606-19-1794-5 (ebook)

Copyright ©2025 Autorul

Lucrarea a fost avizată de

Consiliul Departamentului de Electronică și Calculatoare din cadrul

Facultății de Inginerie Electrică și Știința Calculatoarelor,

Universitatea Transilvania din Brașov.

Cuprins

Electronică digitală	I
Cuprins	i
Cuvântul cel dintâi către „cetitor”	xi
Tomul pentru zăbovit și gândit	1
Începutul învățaturii „Tomului pentru zăbovit și gândit”	3
1 De la matematică la inginerie	5
1.1 Lumea digitală. Între 0 și 1	7
1.1.1 De ce este nevoie de „Electronică digitală”?	8
1.1.2 Informația. Semnale	12
1.2 Matematică binară	16
1.2.1 Reprezentarea numerelor	17
1.2.2 Conversia numerelor între bazele de numerație 2, 10, 16	22
1.2.3 Reprezentarea numerelor întregi în complement față de 2	28
1.2.4 Adunarea și scăderea numerelor întregi reprezentate în complement față de 2	31
1.2.5 Codul ASCII	33
1.2.6 Codul BCD	35
1.2.7 Algebra Booleană	36
1.3 Inginerie binară	38
1.3.1 Porți logice	38
1.3.2 Justificarea noțiunii de poartă logică	40
1.3.3 Set complet de operatori logici	45
1.3.4 Parametrii porților și semnalelor logice	45
1.3.5 Timp de propagare	51
1.4 Modelarea sistemelor digitale în HDL	54
1.4.1 De ce este nevoie de HDL?	55
1.4.2 Metodologia de proiectare a sistemelor digitale	57

1.4.3	Mediul de simulare HDL	59
1.4.4	Modele HDL	60
1.4.5	Setul de valori	63
1.4.6	Tipuri de date	63
1.4.7	Codați cu stil!	65
1.4.8	Hard-ul nu-i ca soft-ul!	67
2	Circuite logice combinaționale	69
2.1	Reprezentarea circuitelor logice combinaționale	72
2.1.1	Tabele de adevăr	72
2.1.2	Forme standard	74
2.1.3	Diagrame Veitch-Karnaugh	76
2.2	Analiza circuitelor logice combinaționale	79
2.3	Sinteza circuitelor logice combinaționale	84
2.3.1	Minimizarea funcțiilor logice, diagrame V-K	84
2.3.2	Minimizarea funcțiilor logice incomplet definite	86
2.3.3	Minimizarea funcțiilor cu variabile reziduu	87
2.3.4	Minimizarea funcțiilor logice cu ieșiri multiple	88
2.4	CLC particulare	90
2.4.1	Decodificator	91
2.4.2	Codificator	99
2.4.3	Multiplexor	101
2.4.4	Demultiplexor	110
2.4.5	Sumator	117
2.4.6	Comparator	121
2.4.7	Unitate logică și aritmetică	123
2.4.8	Look-Up-Table și memorii ROM	126
2.5	Modelarea circuitelor logice combinaționale în HDL	129
2.5.1	Operatori	129
2.5.2	Operatori aritmetici	131
2.5.3	Operatori relaționali	138
2.5.4	Operatori de egalitate	138
2.5.5	Operatori logici	139
2.5.6	Operatori logici pe vectori de biți	140
2.5.7	Operatori de reducere	141
2.5.8	Operatori de deplasare	143
2.5.9	Operatorul condițional	145
2.5.10	Operatori de concatenare și replicare a biților	145
2.5.11	Precedența operatorilor	146
2.5.12	Modelarea circuitelor logice combinaționale	149
3	Circuite logice secvențiale	157
3.1	Circuite de memorare a stării	159
3.1.1	Latch	163
3.1.2	Bistabile	167

3.1.3	Inițializarea bistabilelor	181
3.2	Automate secvențiale	183
3.3	Reprezentarea automatelor	186
3.3.1	Tabele de tranziții	187
3.3.2	Organigrame	190
3.3.3	Grafuri	191
3.3.4	Analiza circuitelor logice secvențiale	193
3.3.5	Sinteza automatelor	197
3.3.6	Modelarea automatelor în Verilog	200
3.4	Registre și numărătoare	206
3.4.1	Registre cu accesare serie sau paralel	207
3.4.2	Numărătoare	211
3.4.3	Linear-Feedback-Shift-Register	211
3.4.4	Modelarea HDL a registrelor și a numărătoarelor	213
3.5	Memorii	225
3.5.1	Memoria ROM sincronă	225
3.5.2	Memoria RAM	228
3.5.3	Structuri de memorii	237
4	Considerente ingineresti ale sistemelor digitale	245
4.1	Timpul în sistemele digitale	248
4.1.1	Căi de propagare, frecvența semnalului de ceas, slack	248
4.1.2	Parametrii semnalului de ceas: skew, jitter, drift	251
4.1.3	Metastabilitatea bistabilelor	255
4.1.4	Domenii de ceas, circuite de sincronizare	256
4.1.5	Structura pipeline	259
4.2	Protocoale de comunicare sincrone între module digitale	262
4.2.1	Protocolul „request-acknowledge”	262
4.2.2	Protocolul „valid-ready”	266
4.3	Tehnologia circuitelor integrate digitale	270
4.3.1	Metodologia de proiectare a circuitelor integrate digitale	271
4.3.2	Fabricarea circuitelor integrate	274
4.3.3	Tehnologia CMOS	275
4.4	Noțiuni avansate de Verilog	279
4.4.1	Modelarea comportamentală RTL	281
4.4.2	Medii de testare	288
	Tomul pentru zăbovit și gândit - Final	300
	Tomul pentru chibzuit și buchisit	301
	Începutul învățaturii „Tomului pentru chibzuit și buchisit”	303
1	Reprezentarea datelor în sistemele digitale	305
1.1	Pentru seminar	305

1.2	Pentru cei ce vor doar să promoveze examenul	306
1.3	Pentru cei ce vor să învețe	307
1.4	Pentru cei ce vor să devină profesioniști	316
2	Algebră Booleană, porți logice, funcții logice	319
2.1	Pentru seminar	319
2.2	Pentru cei ce vor doar să promoveze examenul	323
2.3	Pentru cei ce vor să învețe	327
2.4	Pentru cei ce vor să devină profesioniști	344
3	Minimizare cu diagrame Veitch-Karnaugh	351
3.1	Pentru seminar	351
3.2	Pentru cei ce vor doar să promoveze examenul	351
3.3	Pentru cei ce vor să învețe	352
3.4	Pentru cei ce vor să devină profesioniști	356
4	Implementare CLC cu porți NAND	363
4.1	Pentru seminar	363
4.2	Pentru cei ce vor doar să promoveze examenul	365
4.3	Pentru cei ce vor să învețe	366
4.4	Pentru cei ce vor să devină profesioniști	371
5	Multiplexoare, demultiplexoare și codificatoare	377
5.1	Pentru seminar	377
5.2	Pentru cei ce vor doar să promoveze examenul	378
5.3	Pentru cei ce vor să învețe	379
5.4	Pentru cei ce vor să devină profesioniști	380
6	Modelarea circuitelor logice combinaționale	393
6.1	Pentru seminar	393
6.2	Pentru cei ce vor doar să promoveze examenul	397
6.3	Pentru cei ce vor să învețe	398
6.4	Pentru cei ce vor să devină profesioniști	399
6.5	Concluzie: Inginer vs. ChatGPT	409
7	Circuite cu bistabile	415
7.1	Pentru seminar	415
7.2	Pentru cei ce vor doar să promoveze examenul	418
7.3	Pentru cei ce vor să învețe	419
7.4	Pentru cei ce vor să devină profesioniști	422
8	Analiza circuitelor logice secvențiale	425
8.1	Pentru seminar	425
8.2	Pentru cei ce vor doar să promoveze examenul	426
8.3	Pentru cei ce vor să învețe	426
8.4	Pentru cei ce vor să devină profesioniști	432

9 Sinteza circuitelor logice secvențiale	439
9.1 Pentru seminar	439
9.2 Pentru cei ce vor doar să promoveze examenul	441
9.3 Pentru cei ce vor să învețe	441
9.4 Pentru cei ce vor să devină profesioniști	449
10 Modelarea automatelor în Verilog	453
10.1 Pentru seminar	455
10.2 Pentru cei ce vor doar să promoveze examenul	457
10.3 Pentru cei ce vor să învețe	457
10.4 Pentru cei ce vor să devină profesioniști	459
11 Registre și numărătoare	469
11.1 Pentru seminar	469
11.2 Pentru cei ce vor doar să promoveze examenul	471
11.3 Pentru cei ce vor să învețe	471
11.4 Pentru cei ce vor să devină profesioniști	474
12 Memorii	485
12.1 Pentru seminar	485
12.2 Pentru cei ce vor doar să promoveze examenul	486
12.3 Pentru cei ce vor să învețe	486
12.4 Pentru cei ce vor să devină profesioniști	488
13 Proiectarea sistemelor digitale	495
13.1 Pentru seminar	495
13.2 Pentru cei ce vor doar să promoveze examenul	496
13.3 Pentru cei ce vor să învețe	496
13.4 Pentru cei ce vor să devină profesioniști	507
14 Probleme de examen	511
14.1 Subiectul 1	511
14.1.1A. Noțiuni generale	511
14.1.2B. Circuite logice combinaționale	516
14.1.3C. Circuite logice secvențiale	518
14.1.4D. Proiectare sistem digital	521
14.2 Subiectul 2	523
14.2.1A. Noțiuni generale	523
14.2.2B. Circuite logice combinaționale	528
14.2.3C. Circuite logice secvențiale	531
14.2.4D. Proiectare sistem digital	532
14.3 Subiectul 3	534
14.3.1A. Noțiuni generale	534
14.3.2B. Circuite logice combinaționale	539
14.3.3C. Circuite logice secvențiale	540
14.3.4D. Proiectare sistem digital	541

14.4	Subiectul 4	542
14.4.1A.	Noțiuni generale	543
14.4.2B.	Circuite logice combinaționale	548
14.4.3C.	Circuite logice secvențiale	550
14.4.4D.	Proiectare sistem digital	552
14.5	Subiectul 5	553
14.5.1A.	Noțiuni generale	553
14.5.2B.	Circuite logice combinaționale	558
14.5.3C.	Circuite logice secvențiale	559
14.5.4D.	Proiectare sistem digital	562
15	Subiecte pentru evaluat ingineri	565
15.1	Ingineri începători	565
15.1.1	Cunoștințe generale	565
15.1.2	Definirea noțiunilor	566
15.1.3	Circuite cu bistabile	575
15.1.4	Proiectare circuite secvențiale pe baza specificațiilor	579
15.1.5	Proiectare circuite secvențiale după model	588
15.2	Ingineri avansați	589
15.2.1	Analiza circuitelor digitale	589
15.2.2	Proiectare sisteme digitale de control	596
	Tomul pentru chibzuit și buchisit - Final	610
	Tomul pentru meșterit și muncit	611
	Începutul învățaturii „Tomului pentru meșterit și muncit”	613
1	Modelarea circuitelor cu porți logice	615
1.1	Desfășurarea lucrării	615
1.2	Temă de casă	618
2	Modelarea circuitelor logice	623
2.1	Desfășurarea lucrării	624
2.2	Temă de casă	626
3	Desenare diagrame de timp, circuite și scheme	629
3.1	Desfășurarea lucrării	631
3.2	Temă de casă	633
4	Modelarea circuitelor digitale în Verilog	635
4.1	Noțiuni introductive	635
4.1.1	Definirea termenilor	635
4.1.2	Modele asociate unui proiect	637
4.1.3	Organizarea fișierelor	638

4.1.4 Metodologia de proiectare a circuitelor integrate digitale	640
4.1.5 Metodologia de simulare HDL cu <i>ModelSim</i>	642
4.2 Desfășurarea lucrării	650
4.3 Temă de casă	651
5 Multiplexoare, demultiplexoare, codificatoare	653
5.1 Desfășurarea lucrării	654
5.2 Temă de casă	656
6 Modelarea circuitelor logice combinaționale în Verilog	659
6.1 Desfășurarea lucrării	660
6.2 Teme de casă	663
7 Modelarea circuitelor cu bistabile	665
7.1 Desfășurarea lucrării	667
7.1.1 Mediu de testare pentru circuite cu bistabile	669
7.2 Teme de casă	672
8 Modelarea circuitelor secvențiale cu bistabile	677
8.1 Desfășurarea lucrării	677
8.2 Temă de casă	680
9 Modelarea FSM	683
9.1 Desfășurarea lucrării	684
9.2 Temă de casă	686
10 Modelarea registrelor și a numărătoarelor	689
10.1 Desfășurarea lucrării	689
10.1.1 Registru cu validare	689
10.1.2 Numărător	690
10.1.3 LFSR	691
10.2 Temă de casă	692
11 Modelarea memoriilor	695
11.1 Noțiuni teoretice	695
11.1.1 Memorii	696
11.1.2 Inițializarea memoriei cu o constantă	698
11.1.3 Inițializarea memoriei	700
11.1.4 Funcții pentru accesul la fișiere de intrare/ieșire	700
11.1.5 Inițializarea memoriei din fișier	701
11.1.6 Salvarea conținutului memoriei în fișier	702
11.2 Desfășurarea lucrării	703
11.2.1 Testarea memoriei single-port	703
11.2.2 Operațiuni cu fișiere	704
11.2.3 Modificări asupra mediului de testare	705
11.3 Temă de casă	706

11.4 Fișiere cu modelul mediului de test	708
11.4.1 Modulul generatorului de stimuli	708
11.4.2 Modulul mediului de testare	709
12 Modelarea sistemelor digitale complexe	713
12.1 Multiplicator secvențial	714
12.2 Circuit watchdog	719
12.3 Circuit pipeline pentru determinarea maximului dintre 3 date	724
13 Subiecte pentru colocviul de laborator	735
Tomul pentru meșterit și muncit - Final	742
Tomul pentru grabă mare	743
Începutul învățăturii „Tomului pentru grabă mare”	745
1 Aritmetică binară	749
1.1 Întrebări	749
1.2 Răspunsuri	758
2 Porți logice	761
2.1 Întrebări	761
2.2 Răspunsuri	772
3 Funcții logice	775
3.1 Întrebări	775
3.2 Răspunsuri	784
4 Circuite logice combinaționale	785
4.1 Întrebări	785
4.2 Răspunsuri	791
5 Circuite logice combinaționale particulare	793
5.1 Întrebări	793
5.2 Răspunsuri	807
6 Elemente de memorare	809
6.1 Întrebări	809
6.2 Răspunsuri	821
7 Automate cu număr finit de stări	823
7.1 Întrebări	823
7.2 Răspunsuri	828
8 Circuite logice secvențiale particulare	829

8.1 Întrebări	829
8.2 Răspunsuri	851
9 Probleme pentru testat inteligența naturală/artificială	853
9.1 Probleme de proiectare	853
Tomul pentru grabă mare - Final	862
Grăirea de încheiere	863
Electronică digitală - Final	866

Cuvântul cel dintâi către „cetitor”

Omul a simțit dintotdeauna dorința să își transmită învățătura către generațiile următoare.

Prima ediție **„Electronică digitală - Carte de învățatură”** a apărut în anul 2012. Subtitlul arhaic își are originea în sentimentul meu că „*învățătura*”¹ devine învechită, fiind înlocuită treptat de „*descurcăreală*”².

După 3 ani, în 2015, am publicat **„Electronică digitală - Carte de învățatură 2.0”**, o ediție revizuită, în care „*m-am adaptat piete*”. Adică, am mai îndulcit tonul, am mai coborât ștacheta și am sortat problemele pentru a realiza o secțiune minimală, dedicată „*celor ce vor doar să promoveze examenul*”.

În 2019, pentru „*cei ce vor să învețe și să devină profesioniști*” am scris volumul **„Verilog - Carte de învățatură”**.

În anul 2020, am simțit nevoia să public un set de chestionare de electronică digitală, cu subtitlul **„Electronică digitală - Carte de învățatură în regim de urgență”**. Problemele au fost adaptate examinării online, prin transformarea problemelor de proiectare și gândire în chestionare potrivite pentru „*metoda ghici*”. Am vrut să marchez astfel perioada nefastă la nivel mondial și să îmi exprim convingerea că nu se pot pregăti profesioniști de electronică *în regim de urgență*.

După 2022, observând schimbarea evidentă a electronicii în ultimii 30 de ani, am actualizat conținutul cursului de Electronică digitală, prin aducerea limbajului de descriere hardware în strânsă conexiune cu circuitele electronice digitale. Am sesizat că studenții sunt mult mai pragmatici, dornici să rezolve repede problema, și mai puțin dornici să înțeleagă justificarea. După încă 2 ani, feed-back-ul primit din partea studenților a fost că „*nu există o bibliografie adaptată cerințelor mele*”.

¹**învățatură** = Procesul de a dobândi cunoștințe. dannicula.ro

²**descurcăreală** = Arta de a rezolva o problemă, fără a o înțelege pe deplin, eventual „*asistat de inteligență artificială*”. dannicula.ro

Înțelegând corectitudinea acestei opinii, m-am decis să rescriu cartea de învățătură în integralitatea ei, sub forma a patru „tomuri”³:

- **Tomul pentru zăbovit și gândit**⁴ suport de curs, noțiuni teoretice alăturate modelării Verilog.
- **Tomul pentru chibzuit și buchisit**^{5 6} suport pentru seminar și probleme de proiectare.
- **Tomul pentru meșterit și muncit**⁷ suport pentru laborator și mici proiecte pentru practică.
- **Tomul pentru grabă mare**⁸ chestionare pentru auto-evaluare și evidențierea noțiunilor ce mai trebuie studiate.

Subtitlurile arhaice sunt intenționate, pentru că tot mai cred că „*a învăța dintr-o carte cu creionul în mână*” este ceva învechit. Mă aștept ca studenții să citească din această carte pe un monitor mare, în care pot vedea două pagini alăturate (similar cu o carte pe hârtie, deschisă), având în față un caiet (sau o tabletă de scris) și un creion (clasic sau „pen”). Din acest motiv, în carte am preferat să pun link-uri de acces rapid în versiunea electronică și să evit să dau detalii despre anumite subiecte cu o relevanță scăzută dar ușor de descoperit pe Internet.

Versiunea actuală, 2025, aduce o structurare și o corelare a întregului subiect al electronicii digitale, oferind suport pentru curs, seminar, laborator și studiu individual pentru studenții de la specializările:

- Electronică aplicată
- Telecomunicații
- Calculatoare
- Automatică
- Tehnologia Informației

³**tom** = Fiecare dintre părțile unei lucrări, ale unei opere de proporții mai întinse, formând adesea o unitate independentă și purtând, de cele mai multe ori, un număr de ordine. Volum, carte, lucrare, operă. dexonline.ro

⁴**a zăbovi** = A lucra prea încet, a nu se grăbi. dexonline.ro

⁵**a chibzui** = A reflecta asupra unei situații cumpănind toate eventualitățile. A se gândi. dexonline.ro

⁶**a buchisi** = A citi sau a învăța ceva cu efort și cu mîgală. dexonline.ro

⁷**a meșteri** = A lucra, a munci (cu pricepere) pentru a realiza ceva. dexonline.ro

⁸**grabă mare** = Termen generat de ChatGPT ca un arhaism pentru „stare de urgență”.

Deși subtitlurile sunt arhaice, conținutul este redactat într-o formă ce s-a dovedit acceptată de generația actuală de studenți, obișnuită (și nevoită) să citească mai mult de pe monitor decât de pe hârtie: paragrafe și cuvinte **cheie** evidențiate cu galben, **avertizări** marcate cu roșu, link-uri la pagini web, recomandări ale  ChatGPT și multe încurajări... 😊. ⁹

Asemănările cu aplicațiile de socializare actuale sunt intenționate. 😐

Conceptul cărții este de a permite studenților, la finalul anului al doilea de studii, accesarea stagiilor de pregătire oferite de firme în domeniul hardware.

Fișierele necesare pentru efectuarea lucrărilor se află pe site-ul autorului, dannicula.ro.



Brașov, 2025.

Dan NICULA

»> CONTINUT

⁹Da, m-am gândit și la audio-book. Și la benzi desenate... 😐

Începutul învățaturii „Tomului pentru meșterit și muncit”

Tomul pentru meșterit și muncit¹ este al treilea în serie și conține chestiunile propuse pentru un set de 12 lucrări de laborator.²

Studentii ar trebui să lucreze individual, pe baza informațiilor și cerințelor prezentate în textul lucrării, și să apeleze la suportului profesorului doar la nevoie.

Desigur, pentru ca această activitate să fie abordată simplu, sunt necesare cunoștințele prezentate la curs și seminar, în primele două volume:

Tomul pentru zăbovit și gândit și

Tomul pentru chibzuit și buchisit

Lucrările de laborator sunt concepute astfel încât să poate fi începute chiar înaintea primelor cursuri și seminarii, pe baza cunoștințelor de la cursurile anterioare de matematică.

- Primele două lucrări se bazează pe desenarea și simularea circuitelor logice cu aplicația web  [CircuitVerse](#).

Studentii au ocazia să facă trecerea de la funcțiile logice, introduse la matematică, la circuitele electronice, care implementează funcții logice. Utilizând noțiunile de minimizare cu diagrame V-K, studenții au ocazia să înțeleagă, prin propria experiență, de ce un circuit mai mic este de preferat.

- A treia lucrare prezintă trei aplicații web pentru documentare, utilizate pentru descrierea specificațiilor în electronică digitală:

- [WaveDrom](#)
- [TimeGen](#)
- [draw.io](#)

¹ **a meșteri** = A lucra, a munci (cu pricepere) pentru a realiza ceva. [dexonline.ro](#)

² **laborare** = A lucra (latină)

- Următoarele lucrări se bazează pe utilizarea simulatorului HDL *ModelSim* pentru simularea modelelor de circuite digitale combinaționale și secvențiale.
- Ultima lucrare propune modelarea unor sisteme digitale de complexitate mai mare:
 - **Multiplicator secvențial**, modelat pe baza arhitecturii RTL ³ proiectate de un alt inginer, cu interfață *request-acknowledge*.
 - **Circuit watchdog**, a cărui funcționare este prezentată prin forme de undă ale semnalelor generate.
 - **Circuit pipeline pentru determinarea maximului dintre trei date**, modelat pe baza unor specificații sumare, cu interfețe *valid-ready*.

Unele lucrări de laborator sunt însoțite de teme de casă.

Pentru studenți, fiecare din subiectele descrise în carte necesită mai mult decât două ore de lucru. Intenția este de a furniza studenților interesați subiecte pentru dezvoltarea **pasiunii** ⁴ 😊 pentru proiectarea sistemelor digitale, prin modelare Verilog și simulare.

Este de așteptat ca, la finalul lucrărilor de laborator, studenții să poată descrie în Verilog un sistem digital de complexitate mică și să utilizeze aplicația *ModelSim* pentru simularea unui mediu de testare HDL. Și, datorită acestor calități, să devină atractiv pentru firme să le ofere stagii de pregătire în domeniul hardware, la finalul anului al doilea de studii.

Fișierele necesare pentru efectuarea lucrărilor se află pe site-ul autorului, dannicula.ro.



Brașov, 2025.

Dan NICULA

³RTL = *Register Transfer Level* = nivel de detaliere care evidențiază transferurile de date între registre, engl.

⁴**pasiune** = Înclinație vie, însoțită de plăcere pentru obiectul studiat sau pentru profesia exercitată. dexonline.ro

Lucrarea 1

Modelarea circuitelor cu porți logice

Prima lucrare de laborator poate fi planificată chiar înaintea primului curs. Din acest motiv, subiectul lucrării se bazează pe cunoștințele dobândite la cursurile anterioare de matematică sau pe noțiuni introduse în timpul orei de laborator.

De la cursurile anterioare sunteți deja familiarizați cu noțiunile algebrei Booleene, variabilele și funcțiile logice elementare (NOT, AND, OR).

Noțiunile de inginerie dobândite constau în asocierea funcțiilor logice (matematice, abstracte) cu noțiunea de **poartă logică**, care desemnează un *circuit electronic digital* cu o funcție de transfer descrisă ca o funcție logică.

Porțile logice asociate operatorilor elementari (NOT, AND, OR) sau compuși (NAND, NOR, XOR, XNOR) au asociate simboluri grafice pentru a permite reprezentarea și proiectarea unor circuite electronice digitale cu funcții mai complexe.

Tabelul 1.1 prezintă tabelele de adevăr ale operatorii logici și simbolurile grafice asociate operatorilor.

1.1 Desfășurarea lucrării

1. Utilizați contul instituțional de student pentru a vă deschide un cont gratuit pe site-ul  CircuitVerse (figura 1.1).

Stabiliți-vă o parolă unică (pe care nu o mai aveți la alte site-uri), de o complexitate rezonabilă. Verificați calitatea (siguranța) parolei pe

A	B	NOT  $\bar{A}$	AND  $A \cdot B$	NAND  $\overline{A \cdot B}$	OR  $A + B$	NOR  $\overline{A + B}$	XOR  $A \oplus B$	XNOR  $\overline{A \oplus B}$
0	0	1	0	1	0	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	0	1	1	0	1	0
1	1	0	1	0	1	0	0	1

Tabelul 1.1

Operatori logici: tabele de adevăr și simboluri grafice.

site-uri specializate.¹

Rețineți parola.

Asociați-vă la grupul de studenți în circuitverse.org.

2. Citiți documentația circuitverse.org.

Familiarizați-vă cu elementele de circuit și simulatorul circuitverse.org.

Descoperiți secțiunea de elemente de circuit: *Intrări, Ieșiri, Porți, Decodificatoare și multiplexoare, Elemente secvențiale, Adnotări, Diverse* (figura 1.2).

Descoperiți secțiunea de proprietăți ale circuitului.

Editați circuitul prin includerea simbolurilor de porturi de intrare și ieșire, porți logice, indicatoare (*flag*, engl.) și elemente de interconectare a porturilor elementelor de circuit.

3. Proiectați cu porți logice un circuit de vot majoritar cu 3 intrări. Desenați circuitul cu porți logice. Simulați funcționarea și verificați diagramele de timp. Funcția logică a circuitului de vot majoritar cu 3 intrări este:

$$VOT = (V1 \cdot V2) + (V1 \cdot V3) + (V2 \cdot V3)$$

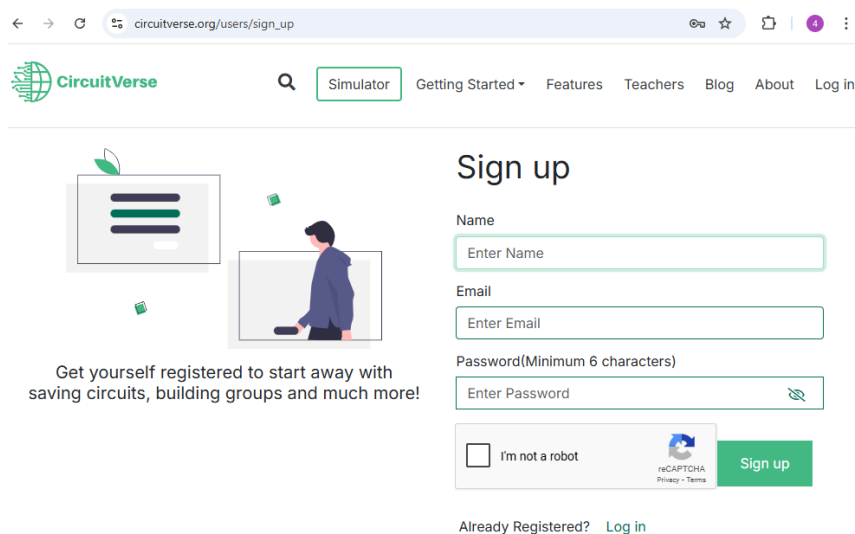
În Verilog, descrierea acestui circuit este:

```
assign VOT = (V1 & V2) | (V1 & V3) | (V2 & V3);
```

4. Proiectați un circuit care acceptă la intrare un număr pozitiv în domeniul 0 ... 7, reprezentat în binar pe 3 biți și activează ieșirea dacă numărul de la intrare este divizibil cu 3.

- Tabel de adevăr.

¹Recomandare: <https://www.security.org/how-secure-is-my-password/>



← → ↻ circuitverse.org/users/sign_up

CircuitVerse 🔍 [Simulator](#) [Getting Started](#) [Features](#) [Teachers](#) [Blog](#) [About](#) [Log in](#)

Sign up

Get yourself registered to start away with saving circuits, building groups and much more!

Name
Enter Name

Email
Enter Email

Password (Minimum 6 characters)
Enter Password

☐ I'm not a robot  [Privacy](#) [Terms](#) [Sign up](#)

Already Registered? [Log in](#)

Figura 1.1 Pagina de deschidere cont pe circuitverse.org.

- Diagramă VK.
- Minimizare.
- Deducere funcție logică.
- Desenare structură de porți logice.
- Simulare.
- Codificarea numerelor zecimale în cod BCD:

Zecimal	Binary Code Decimal (BCD)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

5. Proiectați circuitul cu porți logice care implementează funcția:

$$F = A \cdot \overline{B} + C \oplus D$$

Desenați și simulați circuitul cu circuitverse.org.

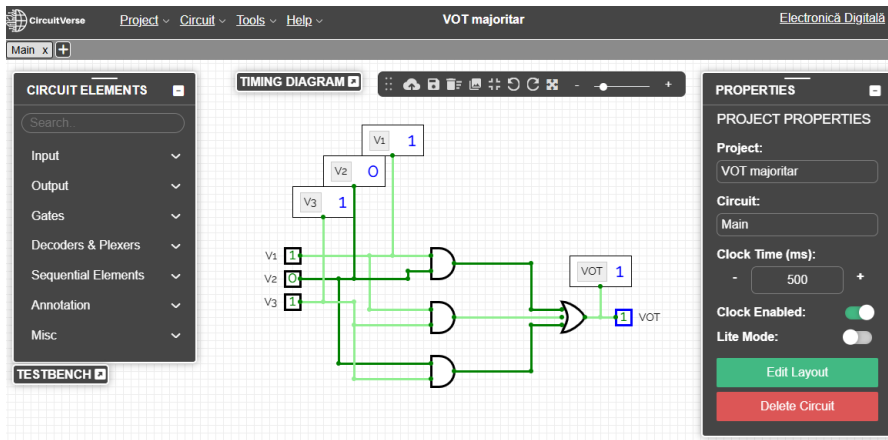


Figura 1.2 Pagina de editare de circuit și simulare în circuitverse.org.

1.2 Temă de casă

Proiectați un circuit convertor BCD (Binary Code Decimal) - 7 segmente. În figura 1.3 sunt prezentate modurile de aprindere a segmentelor pentru a descrie cifrele zecimale. Utilizați elementul **SevenSegDisplay** pentru a verifica circuitul.

Etape de proiectare

- Tabel de adevăr.
- Diagramă VK.
- Minimizare.
- Deducere funcție logică.
- Desenare structură de porți logice.
- Simulare.

Cerințe

- Circuit implementat cu porți logice în circuitverse.org.
- Dacă la intrare se introduce o combinație binară între 10 și 15, afișajul va prezenta litera „E” (eroare).

Descrierea Verilog a interfeței modulului este prezentat în continuare. Păstrați aceleași nume pentru porturi.

```
module bcd_7seg(
input      [3:0] bcd, // domeniul 0 ... 9
output reg  a, b, c, d, e, f, g
);
```

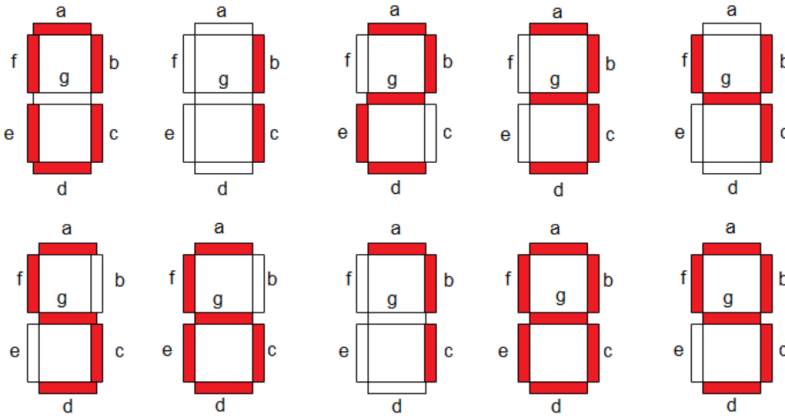


Figura 1.3 Afișarea cifrelor zecimale pe afișajul cu 7 segmente.

```
// codul aici

endmodule // bcd_7seg
```

Livrare:

Link la proiectul circuitverse.org inclus în mod text și/sau obiect inclus.

După ce desenați circuitul, salvați proiectul și părăsiți modulul simulator pentru a reveni în meniul principal „Dashboard” (meniul „Project” → „Save Online”). În secțiunea „Dashboard” vi se prezintă toate proiectele existente sub forma unor casete, ca în figura 1.4.

Apelați la opțiunea  pentru a accesa secțiunea de proprietăți ale proiectului. Apelați la opțiunea  pentru a edita tipul de acces la proiect. Câmpul „Project access type” poate avea 3 valori: „Public”, „Private” și „Limited access”. Modificați tipul de acces pentru a fi „Limited Access”. Astfel, alți utilizatori vor putea rula simularea circuitului dar nu îl vor putea modifica. Actualizați câmpurile „Name” și „Description” cu informații potrivite. Salvați modificările .

Apoi faceți click pe  și apoi pe  în fereastra „Embed Your Circuit”.

În fereastră va apare o secțiune de cod HTML utilizată pentru referirea obiectului *circuitverse*, similar cu figura 1.5. Copiați textul în fereastra de text HTML a temei, în aplicația Moodle.

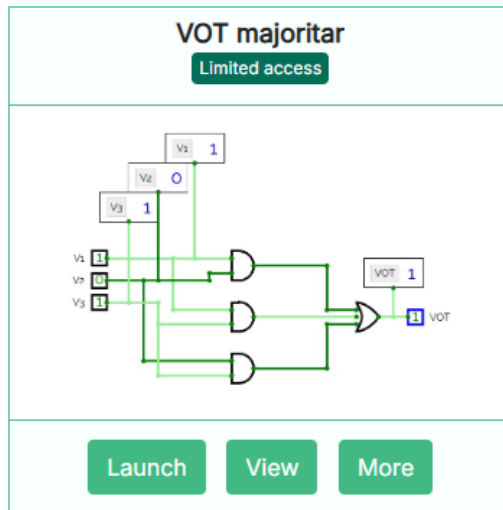


Figura 1.4 Proiectul „VOT majoritar” în Dashboard circuitverse.org.

Imaginea obiectului cu circuitul proiectat va arăta ca în figura 1.6. Circuitul va putea fi simulat direct în pagina HTML care va referi obiectul.

Atenție! Dacă în loc de circuit vedeți un dreptunghi **roșu**, schimbați editorul HTML și eliminați atributul „*sandbox*” din sintaxa „*iframe*”.

Obținerea unui link la un proiect *circuitverse* se face prin opțiunea  și copierea textului din caseta URL, așa ca în figura 1.7.

Copiați textul din caseta URL în fereastra de text HTML a temei Moodle.
2

Verificați accesibilitatea link-ului în altă fereastră de browser.

Copiați link-ul în fereastra text HTML de pe Moodle.

Desigur, puteți ascunde link-ul sub un text sau imagine, pentru a arăta ca în figura 1.6. Însă nu va fi obiect inclus în pagină HTML ci va fi un link spre o pagină în care va apare obiectul *circuitverse*.

²Va fi ceva similar cu:

<https://circuitverse.org/simulator/embed/vot-majoritar-bfe33267-9704-4ab1-adeb-102671b083ff>

Publish

```
<iframe src="https://circuitverse.org/simulator/embed/vot-majoritar-  
bfe33267-9704-4ab1-adeb-102671b083ff?  
theme=&display_title=false&clock_time=true&fullscreen=true&zoom_in_out=tr  
ue" style="border-width:; border-style:; border-color:;" name="myiframe"  
id="projectPreview" scrolling="no" frameborder="1" marginheight="0px"  
marginwidth="0px" height="500" width="500" allowFullScreen></iframe>
```

Copied to Clipboard!

Figura 1.5 Secțiune de cod HTML care descrie un obiect circuitverse pentru a fi inclus într-o pagină de web cu tag-ul „*iframe*”.

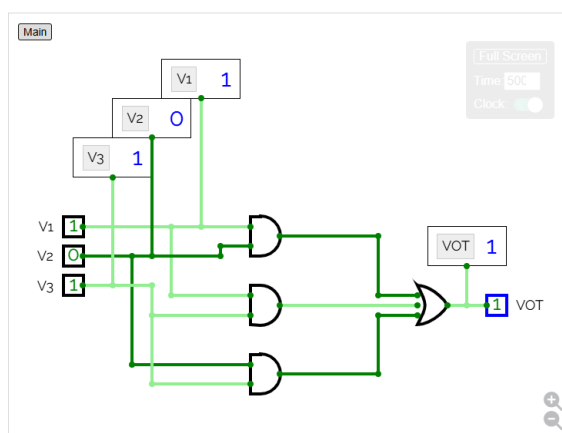


Figura 1.6 Obiect *circuitverse* referit într-o pagină de web.

Embed Your Circuit ×

Height

500

Width

500

URL

<https://circuitverse.org/simulator/embed/vot->

Figura 1.7 Obținerea unui URL pentru obiectul *circuitverse*.

Lucrarea 2

Modelarea circuitelor logice

A doua lucrare de laborator are ca scop fixarea cunoștințelor dobândite de la cursurile anterioare de matematică și justificarea inginerescă a motivelor din spatele unor teorii matematice.

De exemplu, se justifică necesitatea minimizărilor funcțiilor logice prin faptul că implementările funcțiilor au un cost. Costul pentru o funcție logică este dat de suprafața ocupată pe siliciu pentru realizarea funcției pe un circuit integrat. Optimizarea circuitelor pentru funcționare la frecvență mare este un alt criteriu de care nu se menționează la matematică.

Codul Gray este introdus la matematică, simultan cu prezentarea diagramelor V-K. Această lucrare își propune să justifice de ce diagramele V-K au căsuțele ordonate după codul Gray și de ce este necesară minimizarea funcțiilor. Datorită faptului că este o lucrare de început și studenții nu au primit încă informațiile necesare referitoare la tehnologia de realizare a circuitelor integrate, se propune să se justifice necesitatea minimizării funcțiilor prin munca necesară a desena structuri de porți logice (mai simple sau mai complexe) dar care realizează aceleași funcții logice.

Desigur, este de preferat de muncit mai puțin, dacă efectul este similar cu cel al unei munci mai de durată.

Cazul concret îl consituie desenarea unor circuite în *circuitverse* pentru anumite funcții logice pe baza unor expresii de definire a acestora și, ulterior, desenarea unor circuite care implementează aceleași funcții dar pe baza unor expresii logice minimizate. Se preconizează ca studenții să observe că efortul de minimizare este justificat prin economia de muncă pentru desinare. Ulterior, va fi intuitiv că un număr mai mic de porți logice ocupă o suprafață mai mică pe chip și are un timp de propagare mai scurt, în comparație cu un circuit care implementează aceeași funcție dar pe baza expresiei logice neminimizate. Cele două circuite propuse sunt matematic echivalente, dar, ingineresc, se pot diferenția prin caracteristici

legate de arie și frecvență de operare.

Diagramele Veitch-Karnaugh (V-K) sunt o modalitate de reprezentare grafică a funcțiilor logice. Pentru o funcție de N variabile, diagrama core-spunzătoare are 2^N căsuțe. Dispunerea căsuțelor se face astfel încât căsuțele vecine să aibă coduri care diferă printr-un singur bit. Din acest motiv, codurile căsuțelor, pe orizontală și verticală, sunt în cod Gray, nu în cod binar.

În figura 2.1 sunt prezentate diagramele V-K pentru 2, 3, 4, 5 și 6 variabile de intrare. Se observă că indexul căsuțelor (atât pe verticală cât și pe orizontală) este în ordinea codului Gray. Fiecare căsuță are un număr de căsuțe vecine egale cu numărul variabilelor de intrare. Două căsuțe se consideră „vecine” dacă au asociate coduri care diferă printr-un singur bit. În imagine au fost marcate cu simbol negru căsuțele vecine căsuțelor marcate cu simbol alb. Se observă faptul că proprietatea de vecinătate determină ca vecine nu doar căsuțele adiacente ci și căsuțe ce devin adiacente prin rotirea sau plierea diagramei V-K, conform figurii 2.1.

Codul Gray are următoarele proprietăți:

- două coduri binare succesive diferă printr-un singur bit și
- este un cod ciclic (ultimul cod diferă tot printr-un singur bit de primul cod).

Funcțiile logice trebuie minimizate înainte de implementare cu porți logice pentru a se obține un circuit cât mai mic și cât mai rapid. Implementarea pe suprafața de siliciu trebuie să fie cât mai mică. Un număr mai mic de porți logice (cu mai puține nivele logice înseriate) va determina și un timp de propagare mai scurt pentru semnale, obținând astfel o frecvență mai mare de operare a sistemului digital.

2.1 Desfășurarea lucrării

1. Implementați funcția $F = \sum(5, 7, 13, 14, 15)$ ca sumă de mintermi, cu un nivel de porți AND și o poartă OR (eventual inversoare).

Minimizați funcția.

Implementați expresia minimă a funcției cu două nivele de porți: AND și OR.

Pe baza teoremei lui De Morgan, implementați expresia minimă a funcției cu două nivele de porți: NAND și NAND.

Justificați efortul minimizării expresiei logice a funcției prin aprecierea resurselor (porți logice) necesare implementării circuitului.

2. Proiectați un circuit logic care determină dacă numărul pozitiv repre-

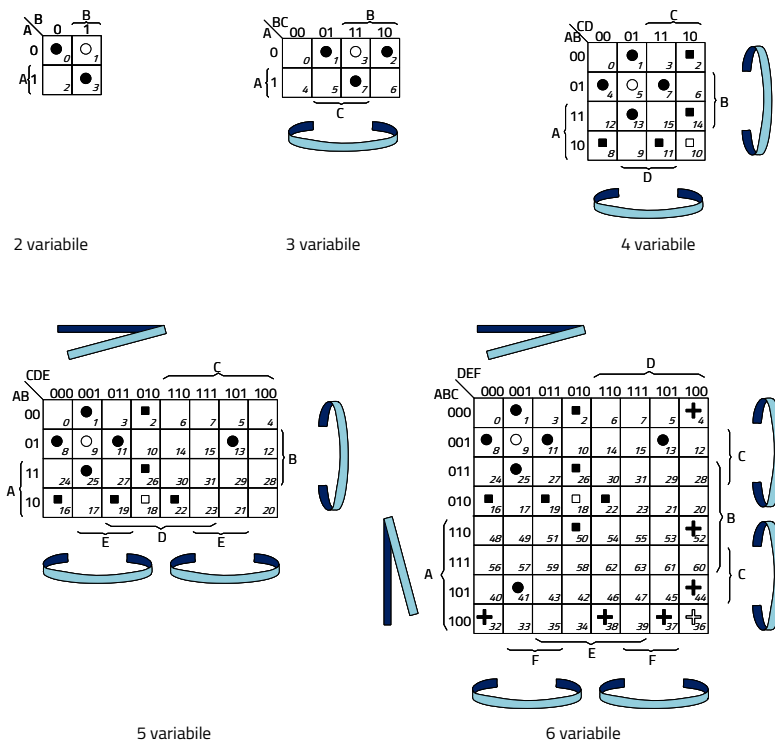


Figura 2.1 Diagrame V-K cu 2, 3, 4, 5 și 6 variabile.

zentat pe 4 biți (în domeniul 0 ... 15) este număr prim.

Implementați circuitul în două moduri:

- fără minimizare, cu porți AND și OR,
- după minimizare, cu porți NAND și NAND.

Se justifică efortul minimizării funcției?

Modulul Verilog are următoarea interfață:

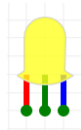
```
module no_prim(
input    [3:0] nr    , // domeniu 0 ... 15
output reg      prim
);

// codul aici

endmodule
```

2.2 Temă de casă

Proiectați un circuit de comparare a două numere pozitive reprezentate pe câte 2 biți, A și B (în domeniul $0 \dots 3$). Ieșirea este conectată la un LED în 3 culori (elementul **RGBLED**).¹



- dacă $A < B$ led-ul se aprinde **roșu**
- dacă $A = B$ led-ul se aprinde **galben**
- dacă $A > B$ led-ul se aprinde **verde**

Sugestii

Circuitul are 3 ieșiri care depind de aceleași 4 intrări.

Pentru a se aprinde galben led-ul, trebuie ca intrările RGB ale acestuia să fie 110.

Deoarece **roșu** + **verde** = **galben**.

Adăugați o facilitare: dacă $A = B = 0$, led-ul se aprinde **albastru**.

Cerințe

Circuit implementat cu porți logice în *circuitverse*.

Link la proiectul *circuitverse* inclus în mod text sau obiect inclus.

Descrierea Verilog a interfeței modulului.

```
module comparator(
input      [1:0] A , // domeniu 0 ... 3
input      [2:0] B , // domeniu 0 ... 3
output reg  [2:0] rgb // 100 = roșu   (A < B)
                        // 110 = galben (A = B)
                        // 010 = verde  (A > B)
);

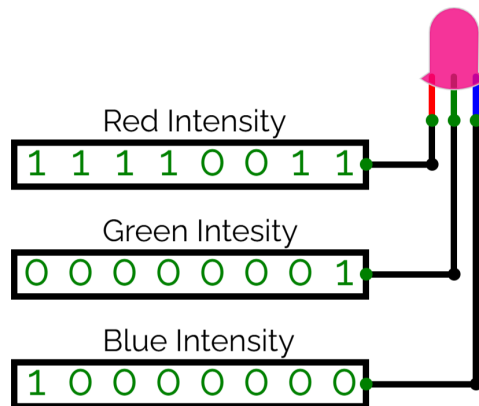
// codul aici

endmodule
```

¹RGB - Red, Green, Blue (engl.)

Propunere

Elementul RGBLED acceptă practic câte 8 biți pentru fiecare plan de culoare, putând astfel afișa 2^{24} culori distincte.



Utilizați „color picker”² pentru a găsi codurile RGB ale culorilor.

Converțiți codurile în binar pe 8 biți³ și realizați circuite cu porți logice care aprind led-ul în diverse nuanțe, pentru diferite condiții.

²<https://htmlcolorcodes.com/color-picker/>

³<https://codebeautify.org/decimal-binary-converter>

Lucrarea 3

Desenare diagrame de timp, circuite și scheme

Pentru depanarea circuitelor logice se utilizează foarte frecvent vizualizarea formelor de undă pe simulator HDL (Hardware Description Language). Formele de undă reprezintă variații ale valorilor logice ale semnalelor în timp. Se consideră că:

- pe orizontală este reprezentată axa timpului (fără a fi desenată) iar
- pe verticală sunt reprezentate valorile logice ale semnalelor (0=jos și 1=sus).

Depanarea unui sistem digital prin vizualizarea formelor de undă constă în compararea **formelor de undă simulate** (prezentate de programele software de simulare) cu **forme de undă așteptate** (gândite de către proiectant).

Pentru a documenta protocolul semnalelor și **forme de undă** așteptate, acestea trebuie incluse de către proiectant în documentul de specificații. Deci, inginerul proiectant trebuie să deseneze forme de undă așteptate a fi generate de sistemul digital proiectat.

Funcționarea circuitelor logice secvențiale poate fi descrisă prin **grafuri** de tranziții. Grafurile sunt reprezentări în care stările sunt asociate cu nodurile (reprezentate ca cercuri) iar tranzițiile între stări sunt reprezentate prin arce (săgeți) orientate, de la starea prezentă la starea viitoare. Un graf de tranziții este o formă foarte frecventă de a documenta un automat secvențial cu număr finit de stări.

Sistemele complexe sunt de multe ori partajate în blocuri mai mici. Schema arhitecturii sistemului digital prezintă **blocuri** (reprezentate ca dreptunghiuri) interconectate, alături de alte simboluri grafice: linii, săgeți, simboluri de cilindri (pentru fișiere), etc.

Algoritmii implementați de sisteme digitale sunt frecvent descriși cu simboluri similare **schemelor logice**: dreptunghiuri (stări, acțiuni), ovale (început și sfârșit), romburi (decizii).

Comportamentul automatelor secvențiale, cu un număr finit de stări, poate fi reprezentat prin **organigramă**, într-o manieră similară cu schemele logice.

Desigur, există aplicații profesionale pentru realizarea desenelor. Această lucrare prezintă însă soluții gratis, la îndemâna studenților, pentru desenarea chestiunilor specifice electronicii digitale:

- Forme de undă,
- Grafuri de tranziții,
- Organigrame, scheme logice, scheme bloc (arhitecturi).



WaveDrom

WaveDrom (wavedrom.com) este o aplicație online pentru desenarea formelor de undă descrise textual. Aplicația conține un limbaj de descriere propriu și un editor specific care funcționează în browser sau poate fi instalat local.

Desenele pot fi salvate în fișiere text.



TimeGen (xfusionsoftware.com) este o aplicație de desenare a formelor de undă. Versiunea **TimeGen Lite** se poate instala local, gratis pentru studenți.

Desenele pot fi salvate în fișiere cu formate specifice.



draw.io (draw.io) este o aplicație care rulează în browser pe site-ul draw.io sau se instalează local, utilizată pentru realizarea de desene schematice pe baza unei multitudini de simboluri grafice.

Desenele pot fi salvate în fișiere cu formate specifice.

Concluzie: Specificațiile de proiectare elaborate conțin desene necesare documentării sistemelor digitale proiectate. Scopul lucrării este de convinge studenții că proiectantul trebuie să realizeze desene rapid pentru a putea alocă cea mai mare parte a timpului pentru proiectarea propriu-zisă. Iar desenele trebuie să fie clare, pentru a putea fi înțelese de alții (colegi, clienți).

3.1 Desfășurarea lucrării

1. Accesați în browser sau instalați local aplicațiile software utilizate pentru desenare:
 - *WaveDrom*
 - *TimeGen*
 - *draw.io*
2. Desenați cu *WaveDrom* formele de undă prezentate în figurile 3.1 și 3.2.

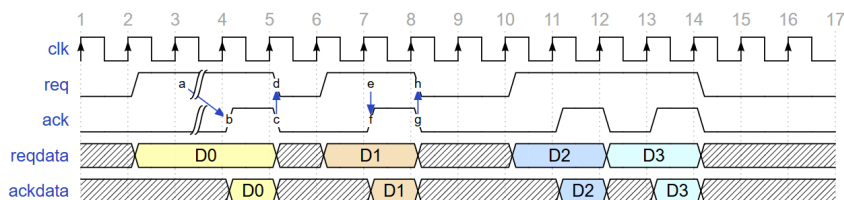


Figura 3.1 Definirea protocolului „request-acknowledge”.

3. Desenați cu *TimeGen* formele de undă prezentate în figurile 3.3 și 3.4.
4. Desenați cu *draw.io* graful de tranziții prezentat în figura 3.5. Salvați local datele pentru re-utilizarea și modificarea ulterioară a schemei.
5. Se consideră circuitul cu porți logice prezentat în figura 3.6. Circuitul implementează funcțiile logice $A \cdot B$ și $A + B$. Formele de undă obținute din simulare sunt prezentate în aceeași figură 3.6.

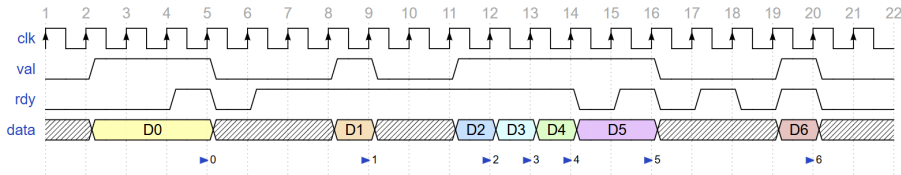


Figura 3.2 Definirea protocolului „valid-ready”.

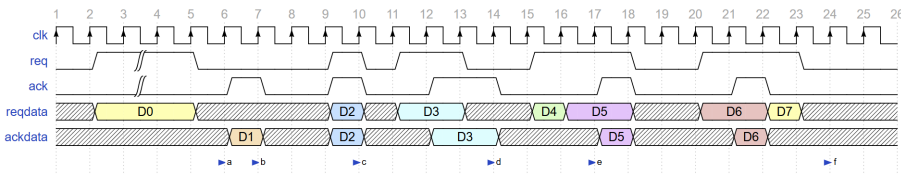


Figura 3.3 Erori ale protocolului „request-acknowledge”.

Găsiți 7 greșeli în formele de undă prezentate în figura 3.6, provenite de la simularea circuitului.

Desenați formele de undă așteptate cu **WaveDrom**.

Adăugați formele de undă greșite (figura 3.6) și marcați cu săgeți momentele unde formele de undă contravin așteptărilor.

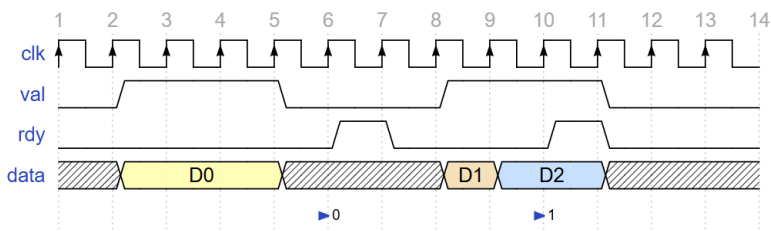
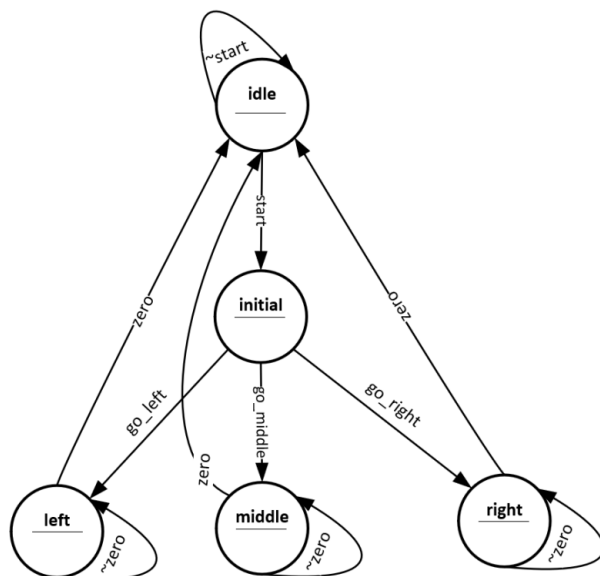


Figura 3.4 Erori ale protocolului „valid-ready”.

**Figura 3.5** Graf de tranziții.

3.2 Temă de casă

1. Desenați cu *WaveDrom* formele de undă pentru circuitul din figura 3.7. Presupuneți la intrare toate combinațiile posibile.

Salvați local descrierea textuală pentru modificări ulterioare.

Livrați un fișier text cu descrierea formelor de undă în format *WaveDrom*.

2. Desenați cu *TimeGen* diagramele de timp care caracterizează circuitul prezentat în figura 3.8. Presupuneți la intrare toate combinațiile posibile.

Salvați local descrierea formelor de undă în formatul specific *TimeGen*.

Livrați o imagine cu forme de undă desenate cu *TimeGen*.

3. Desenați cu *draw.io* schema logică a algoritmului de înmulțire prin adunări repetate.

Livrați o imagine cu desenul algoritmului, cu simboluri de schemă logică.

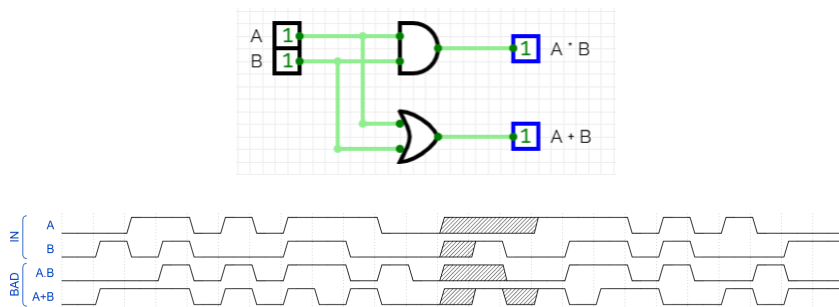


Figura 3.6 Circuit cu porți logice.

Forme de undă simulate pentru circuitul prezentat în figura 3.6.

Atenție! Formele de undă conțin 7 greșeli.

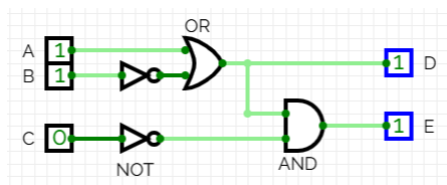


Figura 3.7 Circuit cu porți logice.

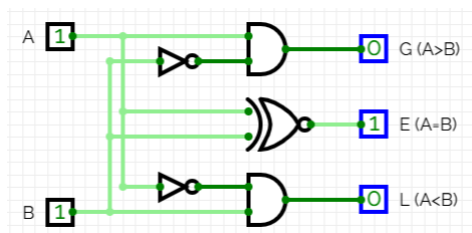


Figura 3.8 Circuit cu porți logice.

Lucrarea 4

Modelarea circuitelor digitale în Verilog

Începând cu acest laborator studenții vor utiliza pentru simularea modelelor descrise în limbajul de descriere hardware Verilog aplicația



Se presupune că *ModelSim* este instalat în sistem de operare Windows. Pentru Linux unele comenzi și noțiuni sunt diferite.

4.1 Noțiuni introductive

4.1.1 Definirea termenilor

Folder de proiect este folderul din sistemul de fișiere unde se vor găsi toate fișierele și bibliotecile asociate proiectului. Utilizatorul trebuie să aibă drept de scriere în acest folder. În continuare, se va considera utilizarea folderului `C:\home` ca fiind folder de proiect. Nu utilizați *Desktop* ca folder de proiect.

Fișierele cu cod sursă sunt fișiere text care conțin modele pentru module Verilog. Se recomandă ca fiecare modul Verilog să fie stocat într-un fișier separat. Extensia fișierelor ce conțin cod Verilog este `.v`. Pentru ușurință, numele fișierului ar trebui să fie identic cu numele modului conținut. De exemplu, modulul **comparator** va fi inclus în fișierul **comparator.v**. Editarea fișierelor sursă se va face cu editoare de text simple (nu Word): *Notepad*, *Notepad++*, *EditPlus+*. Este de preferat

ca editorul să aibă facilitatea de evidențiere a cuvintelor cheie asociate sintaxei ¹ limbajului Verilog. Nu utilizați spații, nici caractere de punctuație, nici caractere diacritice pentru nume de fișiere. Nu editați cod sursă cu caractere TAB, utilizați spații pentru alinierea codului. Nu utilizați caractere diacritice în fișierele de cod Verilog, nici în cadrul comentariilor. Salvați fișierele de cod în folderul de proiect, nu pe *Desktop*.

Compilarea reprezintă conversia codului sursă într-un format specific simulatorului. Compilarea necesită ca parametrii unul sau mai multe nume de fișiere cu cod sursă. La simulatorul *ModelSim*, comanda de compilare se numește **vlog**.

Biblioteca de lucru este un folder unde se vor plasa rezultatele compilării, în formatul specific simulatorului. *ModelSim* este un simulator de cod Verilog/VHDL compilat. În comanda de compilare este necesară specificarea unei biblioteci unde se va plasa codul compilat. Pentru bibliotecă, se recomandă folosirea denumirii implicite **work**. Rezultatul compilării se depune în biblioteca specificată (implicit **work**).

Simularea reprezintă procesul de creare a unui model abstract pentru sistemul simulat. La simulatorul *ModelSim* comanda de simulare se numește **vsim**. Comanda de simulare necesită ca parametru un modul Verilog. Totdeauna modelul ce se pune în simulare nu are porturi. În modulul pus în simulare se instanțiază modelul circuitului de testat **DUT = Device Under Test** ² și module sau specificații adiționale: generatoarele de stimuli **TB = Test Bench** ³, monitoarele de semnale, module care verifică aserții. Toate aceste module sunt instanțiate în modulul de nivel înalt, fără porturi, care se lansează în simulare.

Rularea modelului reprezintă procesul de simulare a funcționării sistemului digital. La simulatorul *ModelSim*, comanda de rulare se numește **run**. Ea are ca parametru timpul pentru care se va rula modelul. Există opțiunea de a se rula modelul la infinit, cu comanda **run -all**.

Modelul RTL ⁴ reprezintă un model HDL pentru un circuit ce urmează a fi implementat în hardware. Datorită procesării ulterioare automate, sunt acceptate doar anumite specificații Verilog, posibil de convertit într-o structură hardware. Modelul structural rezultat, denumit **netlist** ⁵ este un model HDL ce conține instanțieri de componente

¹ *syntax highlight*, engl.

² *DUT = Device Under Test* = Modulul de testat, engl.

³ *TB = Test Bench* = Modulul generator de date utilizate pentru testare, engl.

⁴ *RTL = Register Transfer Level* = nivel de detaliere care evidențiază transferurile de date între registre, engl.

⁵ *netlist = NETWORK LISTing* = listă de componente, engl.

specifice unei biblioteci de tehnologie, interconectate într-o rețea. Modelul RTL detaliază comportamentul sistemului până la nivelul transferului de date între elementele de memorare (registre).

Sinteza RTL reprezintă procesul de conversie automată a codului RTL într-un netlist, cu ajutorul unor programe software specializate EDA⁶, pe baza unei biblioteci de tehnologie.

4.1.2 Modele asociate unui proiect

Simularea HDL necesită 4 tipuri de module, fiecare cu particularitățile lor specifice.

Modulul de testat este modelul pentru circuitul digital ce se va implementa în hardware. Acesta este referit cu abrevierea *DUT*. Scopul simulării este verificarea comportamentului acestui modul prin comparație cu specificațiile de proiectare. Modulul de testat trebuie descris în HDL doar cu specificații acceptate de sintetizatoare RTL. În cazul unor circuite complexe, modulul poate fi structurat în module de complexitate mai mică. În cazul unor circuite simple, modelarea este comportamentală, la nivel RTL (fără a instanția porți logice predefinite în limbaj sau modele de tranzistoare).

Generatorul de stimuli *TB* este modelul pentru sistemul exterior modulului de testat. În cazul unor stimuli simpli, acest modul poate lipsi, fiind înlocuit cu specificații de atribuire de semnale prezente în modulul de test. Pentru modelarea generatoarelor de stimuli pot fi folosite orice specificații, deoarece aceste module au rol exclusiv pentru verificarea *DUT* și nu se sintetizează în hardware.

Modulul de test este modelul pentru sistemul format din modulul de testat *DUT* și generatorul de stimuli *TB*. Acest modul conține instanța modelului *DUT* care se verifică. Alături de aceasta se pot instanția unul sau mai multe generatoare de stimuli sau module de monitorizare a datelor. Tot aici pot apărea specificații pentru verificarea fluxului de date (aserti). Se remarcă faptul că modulul de test are întotdeauna interfața vidă (nu are porturi).

Monitoarele de date sunt modele optionale care încapsulează reguli de verificare (aserti) asupra datelor. Acestea au exclusiv rol în verificarea *DUT*.

Figura 4.1 prezintă grafic relațiile dintre diversele modele HDL implicate în simulare. Scopul simulării este de a se certifica faptul că modulul *DUT*

⁶EDA = *Electronic Design Automation* = suita de aplicații software utilizate pentru proiectare în electronică, engl.

se comportă conform specificațiilor, prin expunerea acestuia la diverse scenarii de verificare.

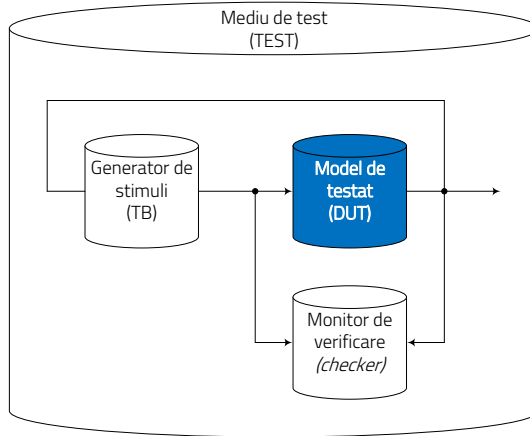


Figura 4.1 Reprezentarea grafică a modelelor implicate în simulare: DUT, generator de stimuli TB, monitoare, modul de test TEST.

De remarcat că, în simulatorul *ModelSim*, modulele implicate în simulare sunt prezentate textual, în fereastra *sim* .

Ierarhia modulelor (modul de includere a unora în altele) este prezentată prin alinierea (indentarea) numelor modulelor.

4.1.3 Organizarea fișierelor

În cadrul unui proiect de simulare HDL sunt implicate multe fișiere, cu roluri diferite:

- fișiere RTL care modelează hardware și vor fi utilizate în continuare pentru sinteză,
- fișiere care modelează mediul de verificare și sunt utilizate exclusiv pentru verificare,
- fișiere ce conțin scripturi pentru automatizarea simulării,
- fișiere generate de simulator pentru stocarea diferitelor date,
- fișiere de date citite de către modelele HDL în cursul simulării, sau de date scrise în timpul simulării (cu scop de depanare și informare).

Într-un proiect mare, numărul acestor fișiere impune o disciplină a gestionării acestora. De aceea, se impune structurarea riguroasă a fișierelor în foldere specifice. O propunere de structurare a folderelor este prezentată în continuare. Pentru exemplificare, se consideră ca folder de proiect `C:\home`.

- Folderul `C:\home\doc` conține fișiere cu specificații pentru descrierea comportamentului sau structurii modelului RTL.
- Folderul `C:\home\hdl` conține fișiere RTL care modelează hardware și vor fi utilizate în continuare pentru sinteză.
- Folderul `C:\home\include` conține fișiere incluse de către modulele RTL care modelează aserții și specificații utilizate pentru depanarea modului RTL (comenzi de acces la fișiere în scopul extragerii de date pentru depanare). Aceste fișiere sunt incluse în simulare dar nu sunt utilizate pentru sinteza RTL.
- Folderul `C:\home\debug` conține două foldere cu fișiere utilizate exclusiv pentru depanare.
 - Folderul `C:\home\debug\hdl` conține fișiere utilizate pentru verificare: generatoare de stimuli, medii de testare, monitoare.
 - Folderul `C:\home\debug\sim` este folderul utilizat ca folder de proiect pentru simulator. Aici se stochează fișiere utilizate pentru automatizarea simulării *sim.do*, *wave.do*. Tot aici se va crea, prin intermediul *ModelSim*, folderul asociat bibliotecii *work* în care simulatorul va stoca, într-un format intern, modulele simulate. Acest folder se poate structura, după necesități, prin crearea unor foldere specifice funcțiilor. De exemplu:
 - * Folderul `C:\home\debug\sim\input_data` pentru fișiere de date citite de generatoarele de stimuli sau de monitoare (fișiere cu datele așteptate).
 - * Folderul `C:\home\debug\sim\output_data` pentru fișiere scrise în cursul simulării cu date eșantionate în anumite puncte din circuit, la anumite momente de timp.
- În folderul `C:\home` pot exista fișiere batch (cu extensia *.bat*) utilizate pentru operații asupra fișierelor și folderelor. De exemplu:
 - Fișierul `cleanup.bat` care include comenzi pentru eliminarea fișierelor temporare în vederea arhivării proiectului, prin păstrarea exclusiv a fișierelor sursă necesare.

- Fișierul **build.bat** care colectează exclusiv fișierele RTL în vederea transmiterii acestora spre verificare sau sinteză.

Pentru exemplificare, în continuare se prezintă structura de foldere asociată acestei lecții.

```
Directory of C:\Home\lab4
<DIR>          doc
<DIR>          debug
<DIR>          hdl
<DIR>          include
cleanup.bat

Directory of C:\Home\lab4\debug
<DIR>          hdl
<DIR>          sim

Directory of C:\Home\lab4\debug\hdl
gen_stimul.v
vot3_test.v
vot3_stim_test.v

Directory of C:\Home\lab4\debug\sim
sim.do
wave.do

Directory of C:\Home\lab4\debug\sim
<DIR>          input_data
<DIR>          output_data

Directory of C:\Home\lab4\hdl
vot3.v
```

4.1.4 Metodologia de proiectare a circuitelor integrate digitale

Metodologia de proiectare a circuitelor integrate digitale, prezentată grafic în figura 4.2, include următoarele 3 etape majore:

Modelarea reprezintă scrierea de către proiectant a unui cod HDL RTL. Prin simulare se verifică faptul că acesta se comportă conform specificațiilor de proiectare.

Sinteza reprezintă conversia automată a modelului RTL într-un netlist cu componente din biblioteca de tehnologie. Simularea netlist-ului este

necesară deoarece există posibilitatea ca sintetizatorul să nu producă structura hardware așteptată, îndeosebi datorită modelării incorecte de către proiectant, ne-observate în timpul simulării RTL.

Implementarea reprezintă conversia netlist-ului într-un model la nivel fizic, prin plasarea și rutarea componentelor din biblioteca de tehnologie prezente în acest model. La finalul acestei etape se obține un netlist ce conține informații temporale asociate semnalelor. Aceste informații permit simularea modelului la frecvențe reale și realizarea diferitelor statistici și măsurători: banda de date necesare, puterea consumată în regim normal, etc.

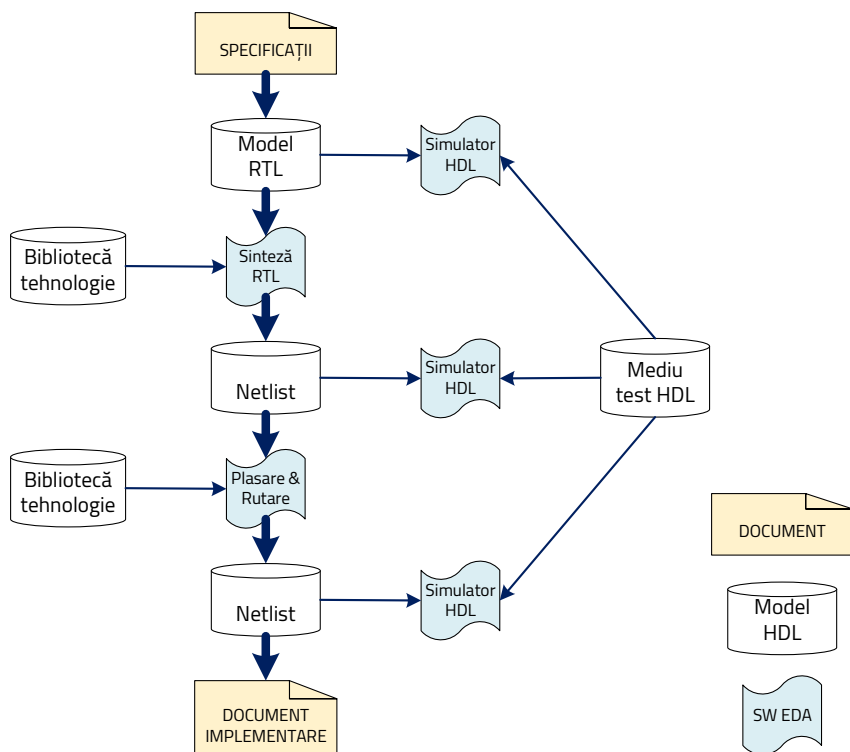


Figura 4.2 Metodologia de proiectare a circuitelor integrate digitale. Se observă multiple simulări în cadrul metodologiei: simulare RTL, simulare netlist, simulare model temporal.

Simularea modelelor este necesară în cadrul fiecărei din cele 3 etape deoarece procesările intermediare realizate cu software EDA pot schimba

comportamentul modelului RTL inițial. Fiecare model al sistemului proiectat (RTL, netlist, model temporal) ar trebui simulat în același mediu de test. Mediul de testare reprezintă un model pentru sistemul proiectat (*DUT*) conectat în mediul exterior.

4.1.5 Metodologia de simulare HDL cu *ModelSim*

Această secțiune prezintă în detaliu simularea unui proiect în *ModelSim*.

Simularea cu *ModelSim* presupune existența fișierelor cu codul sursă, în folderul **C:\Home\lab4**:

- Fișier cu modelul RTL al *DUT*, vot majoritar (*vot3.v*).
- Generator de stimuli pentru circuitul cu 4 intrări (*gen_stimul.v*).
- Fișiere cu modele pentru generatoare de stimuli *TB*, (*vot3_stim_test.v*).
- Fișierul cu mediul de testare *TEST*, (*vot3_test.v*).

Etapile simulării sunt:

- Lansarea simulatorului;
- Inițierea unui proiect și configurarea acestuia;
- Compilarea fișierelor cu cod sursă;
- Simularea modelului de test;
- Rularea simulării;
- Diagnoza comportamentului sistemului de testat, prin utilizarea resurselor simulatorului;
- Automatizarea procesului de simulare.

În secțiunile următoare se vor detalia aceste etape, cu exemplificare pentru cazul simulării unui **circuit pentru vot majoritar cu 3 intrări**.

Lansarea în execuție și configurarea *ModelSim* prin meniuri și comenzi

ModelSim se poate lansa în execuție cu două interfețe:

- *Interfață grafică*, recomandată pentru depanarea unui modul, permite vizualizarea formelor de undă și utilizarea unor comenzi prin meniuri grafice.

- *Interfață text*, recomandată pentru rulări multiple, ne-interactive, pentru verificarea unor multiple scenarii, după ce s-a realizat depanarea cazurilor frecvente.

În continuarea lecției se presupune rularea *ModelSim* în mediu grafic.

ModelSim se lansează în execuție în mod grafic prin dublu-clic pe icon-ul prezent pe desktop  sau din meniul de programe. Se va deschide fereastra principală a *ModelSim*, cu ferestrele *Library* (descrierea textuală a bibliotecilor de componente existente) și *Transcript* (fereastra de comenzi textuale).

Din meniul principal al *ModelSim*, se stabilește folderul de simulare ca folder curent:

File > Change Directory... selectați *C:\Home\lab4\debug\sim*.

Comanda asociată acestei acțiuni este:

```
ModelSim> cd C:/Home/lab4/debug/sim
```

Înainte de a compila un fișier cu cod Verilog, trebuie creată o bibliotecă de proiect și asociată cu un nume generic. Se recomandă păstrarea numelui implicit **work** pentru bibliotecă.

Crearea bibliotecii se poate face din meniul grafic astfel:

File > New > Library...

În caseta de dialog *Create a New Library* (figura 4.3) se selectează:

Create > a new library and a logical mapping to it.

Se recomandă păstrarea câmpurilor *Library Name* și *Library Physical Name* cu valorile completate implicit **work**. Selectați *OK*.

Această acțiune va determina crearea unui sub-folder numit **work**, biblioteca de proiect, în folderul curent. Acest sub-folder conține un fișier particular, denumit **_info**.

Ca alternativă, se poate crea biblioteca de lucru cu o comandă specifică lansată în fereastra *Transcript* sau din fișiere script:

```
ModelSim> vlib work
```

Ulterior, biblioteca de lucru se asociază cu un nume generic prin comanda:

```
ModelSim> vmap work work
```

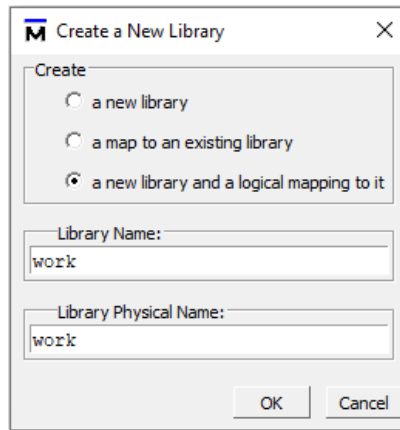


Figura 4.3 Caseta de dialog *Create a New Library*.

Compilarea și simularea unui proiect

Compilarea unui fișier reprezintă analiza codului Verilog din acel fișier și realizarea unui model de simulare într-un format propriu simulatorului. Modelul de simulare este stocat într-o bibliotecă de simulare materializată sub forma unui folder și a unor informații stocate în fișiere gestionate de către simulator. Având biblioteca creată și asociată cu un nume, se poate trece la etapa de compilare a fișierelor sursă. Ordinea de compilare a modulelor Verilog este indiferentă. *ModelSim* face verificarea existenței tuturor modulelor referite în proiect numai în momentul simulării modulului în care se instanțiază acestea (nu în momentul compilării).

Compilarea fișierelor sursă Verilog se face prin acționarea butonului

Compile  sau cu comanda textuală:

ModelSim> vlog <nume fișier>

Fișierele se compilează unul câte unul, sau mai multe deodată, prin selecția multiplă a acestora. Se vor compila atât fișierele RTL (*DUT*) cât și fișierele cu generatoarele de stimuli (*TB*) și mediul de testare (*TEST*).

ModelSim> vlog <nume fișier>

La apăsarea butonului *Compile*  se va deschide caseta de dialog *Compile Source Files* (figura 4.4). În această casetă se permite selecția multiplă a fișierelor cu cod Verilog, urmată de apăsarea tastei *Compile*. Urmare a compilării, în fereastra *Transcript* apar posibilele mesaje de eroare sau

de atenționare. Se închide caseta de dialog prin apăsarea butonului *Done*.

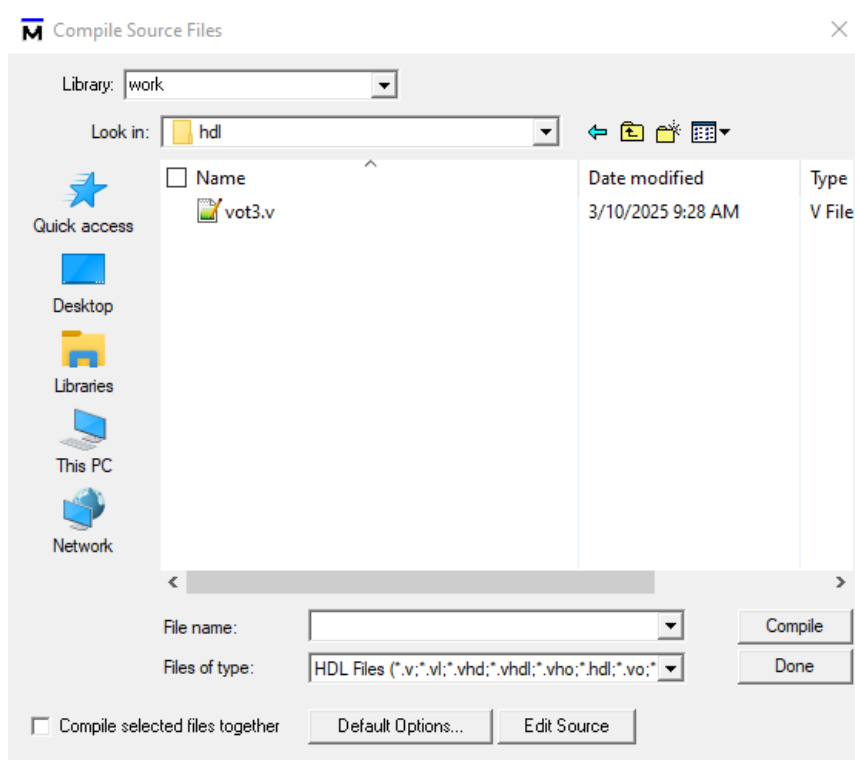


Figura 4.4 Caseta de dialog *Compile HDL Source Files*.

După finalizarea compilării, modulele se găsesc în biblioteca de simulare *work*, într-un format propriu simulatorului. Acest lucru se poate verifica în fereastra *Library*, după selectarea bibliotecii *work* (figura 4.5).

Simularea se face întotdeauna cu precizarea modulului cu mediul de test (cel cu interfață vidă în care este instanțiat modulul de testat *DUT*).

Simularea se lansează selectând butonul *Simulate*  sau la promptul ferestrei *Transcript*:

ModelSim> vsim <modul de top>

Va apărea caseta de dialog *Start Simulation*, similară celei prezentate în figura 4.6.

Comanda alternativă în fereastra *Transcript* este:

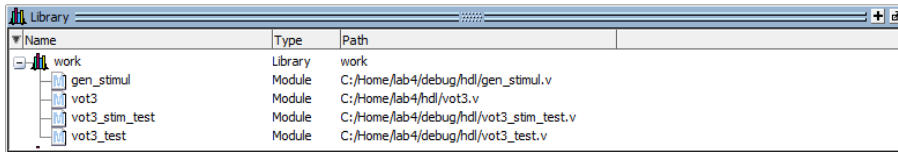


Figura 4.5 Fereastra *Library* cu modulele conținute de biblioteca *work*.

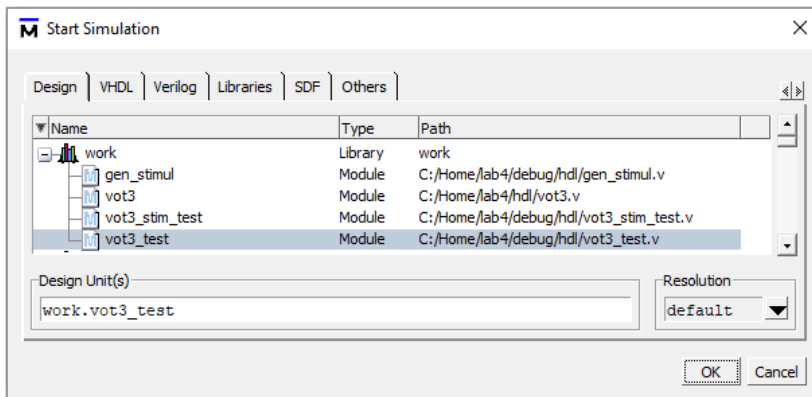


Figura 4.6 Caseta de dialog *Start Simulation*.

ModelSim> vsim -gui work.vot3_test

Ca efect, în fereastra *Transcript* apar mesajele:

```
vsim -gui work.vot3_test
Start time: 10:48:25 on Mar 10, 2025
Loading work.vot3_test
Loading work.vot3
```

După finalizarea simulării, în partea stânga-jos a ferestrei principale apare timpul de simulare: **Now: 0 ps.**

Pornirea simulării se face prin apăsarea butonul .

Această acțiune va determina rularea simulării pentru un timp selectat (implicit 100 ns).

Se poate da o comandă de rulare și din fereastra *Transcript*:

VSIM> run

Se poate rula simularea la infinit (sau până la întâlnirea în cod a unei comenzi de oprire a simulării) prin acționarea butonului *Run All*  sau cu comanda:

```
VSIM> run -all
```

Oprirea simulatorului comandat să ruleze la infinit se face prin apăsarea butonului *Break* .

Ferestre ale simulatorului *ModelSim*

Simulatorul *ModelSim* prezintă mai multe ferestre interne, dedicate diverselor funcții. Ferestrele mai frecvent utilizate sunt:

- Fereastra **Transcript**, în care sunt afișate mesaje referitoare la starea execuției comenzilor și mesajele de eroare sau atenționare. În această fereastră se pot lansa comenzi textuale individuale sau scripturi de comenzi (cu comanda *do <fisier.do>*).
- Fereastra **Objects**, în care se vizualizează semnalele asociate unui modul selectat. Există posibilitatea transferării acestora în fereastra *Wave* pentru a fi vizualizate ca forme de undă sau fereastra *List*.
- Fereastra **sim**, în care se prezintă o descriere textuală a ierarhiei proiectului. În această fereastră se poate selecta modulul curent la care se face referire pentru vizualizarea informațiilor în alte ferestre: lista de semnale, codul sursă, etc. În figura 4.7 se observă în fereastra *sim* prezentarea textuală a ierarhiei proiectului simulat. Modulul *vot3_test* instanțiază modulul *vot3* (cu nume de instanță *DUT*)

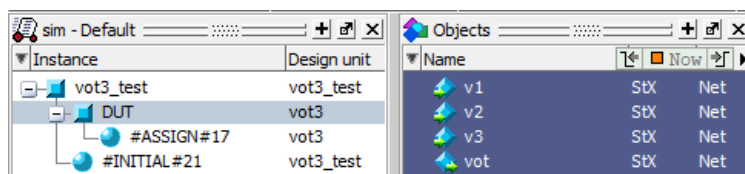


Figura 4.7 Ferestrele *sim* și *Objects*, în cazul selecției modulului *vot3*.

Fereastra formelor de undă ale semnalelor

Fereastra **wave**, figura 4.8, permite vizualizarea variației în timp a semnalelor selectate (forme de undă). Pe orizontală este axa timpului iar pe

verticală sunt afișate diversele semnale prezente în proiect.

Semnalele ce se doresc a fi vizualizate trebuie aduse explicit în fereastra *wave*. Există mai multe forme de a aduce un semnal în fereastra *wave*. Cea mai simplă procedură este:

- se selectează modulul dorit în fereastra *sim*,
- se selectează semnalele dorite în fereastra *Objects*,
- după selecția multiplă a semnalelor și clic dreapta pe unul din semnale apare un meniu de unde se selectează opțiunea *Add Wave*.

Semnalele selectate se vor regăsi în fereastra de forme de undă *wave*.

O alternativă constă în selectarea unui modul în fereastra *sim* și clic dreapta urmat de selecția opțiunii *Add Wave* din meniul apărut. Există și opțiunea utilizării tastei scurte *Ctrl+W* după selecția unui modul sau a unor semnale în ferestrele asociate (*sim*, *Objects*).

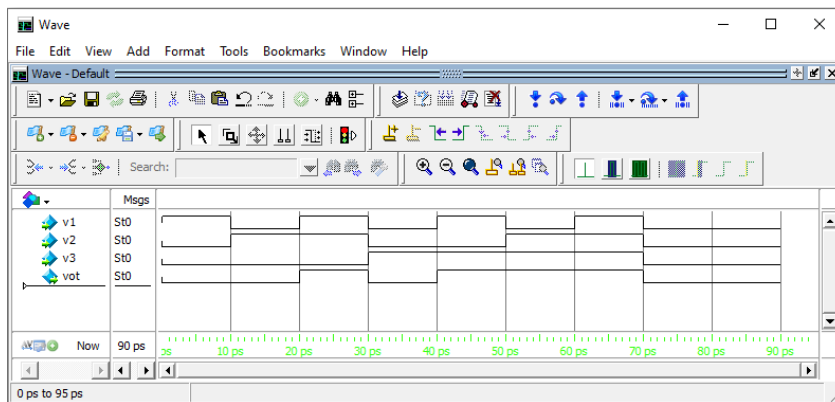


Figura 4.8 Fereastra *Wave* cu semnalele interne ale modulului de testat *vot3*.

Configurarea ferestrei *Wave* poate fi salvată într-un fișier extern, pentru a fi reutilizată într-o sesiune viitoare, cu meniul:

File > *Save format...* sau tasta scurtă *Ctrl+S*.

Fișierul salvat va avea extensia **.do**.

Se recomandă denumirea **wave.do**.

Scriptul va putea fi executat cu comanda lansată după simulare la promptul ferestrei *Transcript*:

VSIM> *do wave.do*

Cursoarele sunt utile pentru a marca timpul la care se investighează valoarea semnalelor și pentru a face măsurători de timp.

Adăugarea și eliminarea cursoarelor din fereastra de afișare a formelor de undă se face cu comenzi specifice. La prima deschidere a ferestrei *Wave* există un singur cursor, plasat la timpul 0. Cursorul se mută automat la timpul la care se face clic în fereastra *Wave*. Pot fi plasate pe ecran maximum 12 cursoare. Cursoarele sunt însoțite de casete care precizează intervalele de timp dintre cursoare.

Se pot adăuga și elimina cursoare cu butoanele *Add Cursor*  și *Delete Cursor* .

Realizarea unui script cu comenzi *ModelSim* pentru rularea automată a simulării

Comenzile generate prin meniuri au un corespondent textual. Acesta se prezintă în fereastra *Transcript*. Prin preluarea comenzilor textuale (cu *copy & paste*) se poate edita un fișier de comenzi, cu extensia *.do*. Fișierele de comenzi se pun în execuție, ca o singură comandă, la prompterul ferestrei *Transcript*, cu comanda:

```
VSIM> do <fisier.do>
```

În acest mod, se pot executa rapid multiple comenzi pentru configurarea mediului de simulare, compilare, simulare, rulare și particularizarea ferestrei cu formele de undă.

Fișierul *sim.do* conține comenzi scriptice *ModelSim* pentru:

- Crearea bibliotecii de lucru *work* (comanda **vlib**).
- Asocierea bibliotecii de lucru *work* cu numele generic *work* (comanda **vmap**).
- Compilarea fișierelor sursă Verilog, *DUT*, test-bench *TB* și test (comanda **vlog**).
- Lansarea în execuție a simulării (comanda **vsim**).
- Încărcarea formelor de undă ce se doresc a fi vizualizate (comanda **do** cu un fișier auxiliar de descriere a formelor de undă).
- Rularea simulării (comanda **run**).

```
vlib work
vmap work work
```

```
vlog ../hdl/vot3.v
vlog ../hdl/vot3_test.v

vsim -gui work.vot3_test

do wave.do

run -all
```

Automatizarea simulării nu este necesară pentru proiecte de dimensiuni mici. Pentru proiecte complexe, ce conțin zeci sau sute de module, este absolut necesară scrierea unor scripturi de comenzi prin care să se pregătească simularea.

4.2 Desfășurarea lucrării

1. Urmăriți metodologia de simulare prezentată în secțiunea 4.1.5 pentru simularea unui circuit combinațional care determină votul majoritar în cazul a 3 intrări. Preluati modelul RTL și mediul de depanare din folderele specifice lecției. Urmăriți metodologia de simulare: *compilare, simulare, rulare, observare forme de undă*.

În cadrul simulării, descoperiți funcționalitatea meniurilor și a ferestrelor *ModelSim*.

2. Modificați codul RTL pentru a modela un circuit de vot majoritar cu 5 intrări.
3. Modificați codul RTL pentru a modela un circuit de vot majoritar cu un număr parametrizabil de intrări.
4. Realizați un script de automatizare prin copierea comenzilor din fereastra *Transcript*. Denumiți scriptul *sim.do*. Includeți în script toate comenzile necesare pentru rularea simulării, începând cu crearea bibliotecii de lucru *work* și până la rularea simulării (*run -all*), inclusiv încărcarea unor forme de undă.

Verificați scriptul prin lansarea în execuție a acestuia din fereastra *Transcript* cu comanda: *do sim.do*.

5. Proiectați un model Verilog pentru un circuit care acceptă la intrare un număr pozitiv în domeniul 0 ... 7, reprezentat în binar pe 3 biți și activează ieșirea dacă numărul de la intrare este divizibil cu 3. Verificați modelul prin simulare.
6. Proiectați un model Verilog pentru un circuit care implementează funcția $F = A \cdot \overline{B} + C \oplus D$. Verificați modelul prin simulare.

4.3 Temă de casă

Proiectați un circuit convertor BCD (Binary Code Decimal) - 7 segmente. Circuitul primește la intrare un cod BCD (4 biți) iar la ieșire prezintă comenzi pentru afișajul cu 7 segmente pentru a afișa cifrele ca în figura 4.9.

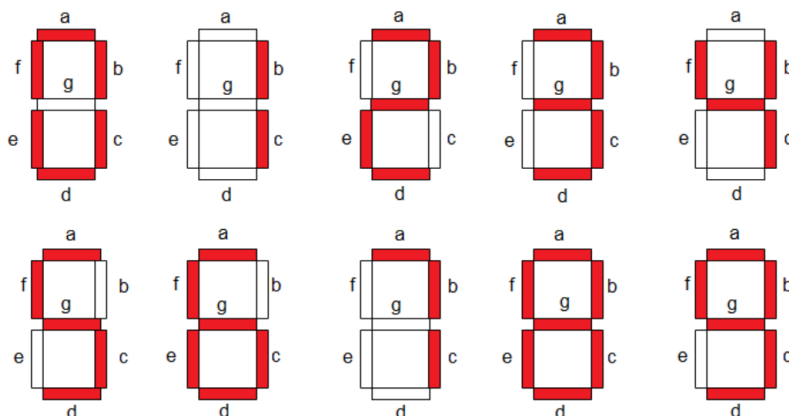


Figura 4.9 Cifrele afișate pe 7 segmente.

Etape:

- Model Verilog.
- Model de testare.
- Verificare prin simulare cu *ModelSim*.
- Script de automatizare.

Livrare: Fișier **.PDF** conținând:

- Modulul RTL: *bcd_7seg.v*.
- Modulul de test: *bcd_7seg_test.v*.
- Script de simulare *sim.do*.
- Forme de undă relevante, captate din fereastra *Wave* a simulatorului *ModelSim*.

Sugestie: descrierea Verilog a interfeței modulului:

```
module bcd_7seg(  
input      [3:0] bcd      , // range 0 ... 9  
output reg  a, b, c, d, e, f, g  
);  
endmodule // bcd_7seg
```

Lucrarea 5

Multiplexoare, demultiplexoare, codificatoare

Verilog este un limbaj de descriere hardware în care se pot modela atât circuite ce urmează a fi implementate în mod real sub forma unui circuit integrat digital (hardware) cât și medii de testare, utilizate exclusiv pentru simulare.

**Verilog nu este un limbaj de programare.
Verilog este un limbaj de modelare pentru hardware.**

Din acest motiv, este improprie noțiunea de „*program Verilog*”. Este recomandată utilizarea noțiunii de „*model Verilog*” sau „*cod Verilog*”.

Modelul Verilog se poate simula. Dacă modelul este compus exclusiv din specificații sintetizabile, modelul se poate și sintetiza, adică transforma într-o structură hardware prin programe software specializate.

Sinteza reprezintă transformarea modelului scris de proiectant într-o structură hardware de către un set de programe software specializate. Setul de programe software utilizate pentru proiectare în electronică este cunoscut sub abrevierea EDA ¹.

Modulul este blocul elementar de construcție Verilog. Un modul conține modelul unui circuit sau a unei părți de circuit. Limbajul Verilog are multe similitudini cu limbajul de programare C. De exemplu, similar cu instanțierea unei clase, un modul poate fi instanțiat în cadrul altui modul. În acest mod, se poate realiza o descriere a structurii unui modul complex.

Codul asociat unui modul Verilog este delimitat de cuvintele rezervate **module** și **endmodule**. Numele modulului este obligatoriu și apare

¹ EDA = *Electronic Design Automation* = suita de aplicații software utilizate pentru proiectare în electronică, engl.

imediat după cuvântul rezervat **module**. Se recomandă utilizarea unor nume sugestive pentru module.

```
module <nume> #(
  <declaratii parametrui>
) (
  <declaratii porturi>
);
<declaratii date>          ;
<specificatii concurente> ;
endmodule
```

Circuitele combinaționale simple (cu o funcție ușor de exprimat) se pot modela cu specificația **assign** și operatori Verilog. În acest caz, semnalul de ieșire al circuitului combinațional trebuie declarat ca fiind de tip **wire**.

Circuitele combinaționale complexe (cu funcții exprimate mai complex) se pot modela cu specificația **always** și specificații secvențiale (**if**, **case**). În acest caz, semnalul trebuie declarat ca fiind de tip **reg**. Este obligatoriu ca, în cazul modelării unui circuit combinațional, specificația **always** să cuprindă în lista de senzitivități toate semnalele ce determină valoarea ieșirii. O alternativă la enumerarea semnalelor într-o listă de semnale despărțite de virgule este de a se utiliza simbolul specific @ (*).

Se recomandă parametrizarea circuitelor combinaționale, în măsura în care aceasta ar permite re-utilizarea viitoare a codului, doar cu particularizarea valorilor parametrilor. De exemplu:

- lățimea bus-urilor de date pentru circuite aritmetice,
- dimensiunea multiplexoarelor ($2^N : 1$).

Pentru testarea circuitelor combinaționale, ar trebui generate toate combinațiile intrărilor. Acest lucru este greu de realizat pentru un număr mare de intrări. Totuși, se poate modela un generator de stimuli (cu numărul de biți parametrizabil) care acoperă toate combinațiile de intrări. Acesta poate fi ulterior reutilizat pentru testarea oricăror circuite combinaționale.

5.1 Desfășurarea lucrării

1. Realizați modele Verilog comportamentale pentru circuitele enumerate. Utilizați descrierile modulelor prezentate în fișierele furnizate ca model.

Verificați prin simulare cu *Model_Sim* comportamentul circuitelor.

A) Multiplexor (mux 2 : 1, mux 4 : 1, mux parametrizabil $2^N : 1$).

```

module mux_2Nx1 #(
parameter WIDTH_SEL          = 'd1      // numar de biti
                                     // pentru selectie
) (
input  [WIDTH_SEL-1:0] sel      , // selectie
input  [(1<<WIDTH_SEL)-1:0] data_in , // data de intrare
output                                data_out // data de iesire
);

assign data_out = data_in[sel];

endmodule // mux_2Nx1

```

B) Demultiplexor (dmux 1 : 2, dmux 1 : 4).

C) Codificator cu prioritate (8 biți).

D) Codificator cu prioritate (8 biți), cu codificarea ieșirii „one-hot”.

```

module encoder8b_onehot (
input  [8-1:0] data_in , // data de intrare
output [8-1:0] data_out , // data de iesire, codata one-hot
output                                valid // data de iesire valida
);

// codul aici

endmodule // encoder8b_onehot

```

E) Comparator de 1 bit (mai mic, egal, mai mare).

```

module comp_1b (
input  data_a , // operand A
input  data_b , // operand B
output a_less_b, // A < B
output a_eq_b , // A = B
output a_gt_b  // A > B
);

// codul aici

endmodule // comp_1b

```

F) Comparator de 8 biți, numere pozitive (mai mic, egal, mai mare).

G) Sumator (8 biți, parametrizabil N biți)

Realizați un generator de stimuli parametrizabil pentru a testa circuite combinaționale.

Realizați un mediu de testare pentru circuitele modelate.

2. Modelați un circuit de multiplexare 4 : 1 cu datele de lățime parametrizabilă și selecție codificată „one-hot” (selecția are 4 biți din care totdeauna unul singur este egal cu 1).

Realizați un mediu de testare. Simbolul multiplexorului este prezentat în figura 5.1.

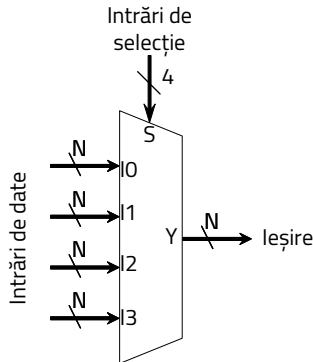


Figura 5.1 Circuit multiplexor 4:1 cu date de lățime parametrizabilă și selecție „one-hot”.

3. Modelați un circuit de multiplexare 5 : 1 cu datele de lățime parametrizabilă și selecție codificată „one-hot”.

5.2 Temă de casă

1. Modelați un circuit comparator de numere întregi reprezentate pe 8 biți, codificate în complement față de 2.

Circuitul are 3 ieșiri: mai mic, egal, mai mare. Simbolul este prezentat în figura 5.2.

Realizați un mediu de testare pentru a testa cel puțin următoarele comparații:

-3, -2
 -2, 0
 -2, 3
 0, 3
 2, 3
 -4, -4

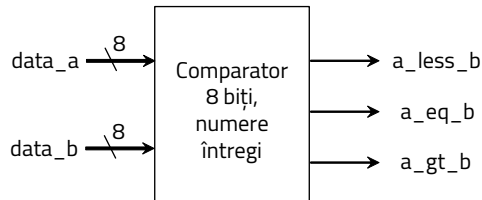


Figura 5.2 Circuit comparator de numere întregi reprezentate pe 8 biți, codificate în complement față de 2.

```

0, 0
4, 4
-2, -5
0, -3
2, -3
2, 0
5, 2
  
```

Extindeți mediul de testare pentru a testa numere aleatorii la infinit.
Cum faceți verificarea?

Interfața modului Verilog:

```

module comp_8b_int (
input  [8  -1:0]  data_a  , // operand A
input  [8  -1:0]  data_b  , // operand B
output          a_less_b, // A < B
output          a_eq_b  , // A = B
output          a_gt_b   // A > B
);

// codul aici

endmodule // comp_8b_int
  
```

Lucrarea 6

Modelarea circuitelor logice combinaționale în Verilog

Aceasta este lucrarea finală pentru capitolul de circuite logice combinaționale. Desigur, sistemele digitale actuale necesită și circuite logice secvențiale.

Însă, până lucrurile nu se complică prea mult, 😊 este momentul să ne încercăm forțele cu modelarea circuitelor logice combinaționale pe baza unor descrieri mai apropiate de ceea ce există în industrie.

Mai mult, acum avem răgazul de a explora ce facilități ne oferă Internetul. Altfel spus, ce putem genera cu [ChatGPT](#) și ce chiar trebuie să învățăm, pentru că [ChatGPT](#) nu știe. Încă. 🗝️

Pentru acest laborator, sunteți încurajați să utilizați



pentru a genera o primă versiune a modelelor. Dar, vă previn că ChatGPT greșeste frecvent. Nu luați de bun tot ce generează. Trebuie să înțelegeți modelul generat pentru a-l putea justifica, să îl corectați, să îl explicați.

Este nevoie de **ECHILIBRU** în toate!

Aveți și timpul să descoperiți un altfel de GPT:



6.1 Desfășurarea lucrării

Enunț

Modelați în Verilog pentru un circuit logic combinațional care primește la intrare un număr pozitiv reprezentat pe 8 biți. La o ieșire prezintă numărul cel mai mare, mai mic sau egal cu numărul de la intrare și divizibil cu 16. La o altă ieșire prezintă numărul cel mai mic, mai mare sau egal cu numărul de la intrare și divizibil cu 16.

Ați înțeles enunțul? Nu din prima... Este scris intenționat pentru a nu fi evident.

Presupun că citiți acest rând înainte de a vă fi gândit măcar la rezolvare.



Poate ați avea nevoie de o formulare/formatare mai atentă a enunțului? Să îl **rescriem** astfel:

Modelați în Verilog pentru un circuit logic combinațional care primește la intrare un număr pozitiv reprezentat pe 8 biți.

La o ieșire prezintă numărul:

- cel mai mare,
- mai mic sau egal cu numărul de la intrare și
- divizibil cu 16.

La o altă ieșire prezintă numărul:

- cel mai mic,
- mai mare sau egal cu numărul de la intrare și
- divizibil cu 16.

Transmiteți ChatGPT enunțul inițial al problemei. În câteva secunde primiți un răspuns similar cu cel din figura 6.1:

Studiați dialogul cu ChatGPT. Nu-i așa că acest subiect a fost rezolvat de ChatGPT mai repede decât ați fi făcut-o voi?

Cereți [ChatGPT](#) și un mediu de testare pentru acest modul. Va răspunde similar cu răspunsul din figura 6.2.

Pare a fi extraordinar? O fi și corect? Vrei să aflii? Întrebați profesorul.



```
module div16_boundaries(  
    input  [7:0] num,  
    output [7:0] lower_bound,  
    output [7:0] upper_bound  
);  
  
    // Calculează limita inferioară (cel mai mare număr <= num și divizibil cu 16)  
    assign lower_bound = num & 8'hF0;  
  
    // Calculează limita superioară (cel mai mic număr >= num și divizibil cu 16)  
    assign upper_bound = (num & 8'hF0) + ((num[3:0] == 4'b0000) ? 8'h00 : 8'h10);  
  
endmodule
```



Iată un model Verilog pentru circuitul descris:

«» Verilog Div16

Explicație:

- `lower_bound` este obținut prin mascarea ultimilor 4 biți ai lui `num` cu `8'hF0` (ștergând acești biți, obținem cel mai mare număr divizibil cu 16 și mai mic sau egal cu `num`).
- `upper_bound` este calculat similar, dar dacă `num` nu este deja divizibil cu 16 (ultimii 4 biți nu sunt toți zero), adăugăm `8'h10` pentru a ajunge la următorul multiplu de 16.

Dacă ai nevoie de modificări sau clarificări, spune-mi! 😊

Figura 6.1 Răspunsul ChatGPT la enunțul problemei (neformatat).

Atenție!

Pentru chestiuni simple, ChatGPT vă poate depăși!
Trebuie să învățați să rezolvați chestiuni mai grele.



Rezolvați singuri problema!

```
// Testbench pentru div16_boundaries
module testbench;
    reg [7:0] num;
    wire [7:0] lower_bound;
    wire [7:0] upper_bound;

    div16_boundaries uut (
        .num(num),
        .lower_bound(lower_bound),
        .upper_bound(upper_bound)
    );

    initial begin
        $monitor("num = %d, lower_bound = %d, upper_bound = %d", num, lower_bound, upper_bound);

        num = 8'd5; #10;
        num = 8'd15; #10;
        num = 8'd16; #10;
        num = 8'd30; #10;
        num = 8'd48; #10;
        num = 8'd60; #10;
        num = 8'd75; #10;
        num = 8'd100; #10;
        num = 8'd200; #10;
        num = 8'd255; #10;

        $finish;
    end
endmodule
```

Figura 6.2 Răspunsul ChatGPT la cererea pentru mediul de testare asociat modulului.



6.2 Teme de casă

Rezolvați următoarele probleme, cât mai corect și cât mai repede, cu acces la resursele Internet. Refolosiți codul pe care deja îl aveți.

Proiectați modelele Verilog ale circuitelor logice combinaționale prezentate.

Pentru fiecare problemă testați toate cazurile în simulare.

Reutilizați codul Verilog, mai ales mediul de testare.

1. Circuit combinațional care înmulțește cu 5 numărul pozitiv de la intrare, fără a folosi operatorul de înmulțire (doar o adunare).
2. Circuit combinațional care înmulțește cu 9 numărul pozitiv de la intrare, fără a folosi operatorul de înmulțire (doar o adunare).
3. Circuit combinațional care înmulțește cu 9 un număr întreg exprimat în complement de 2, reprezentat pe 16 biți de la intrare și are o ieșire care semnalează depășirea. Ieșirea de date este tot pe 16 biți în complement față de 2.
4. Circuit combinațional care realizează extinderea de semn a numerelor întregi reprezentate în complement față de 2 de la 8 biți la 16 biți. Ulterior, parametrizați numerele de biți ale intrării și ieșirii circuitului. Inspectați codul și deduceți ce porți logice sunt necesare pentru realizarea acestui circuit.
5. Circuit combinațional care realizează înmulțirea cu -1 a numărului întreg prezentat la intrare, reprezentat pe 16 biți în complement față de 2.
6. Utilizând exclusiv sumatoare complete de un bit, proiectați un circuit care furnizează la ieșire numărul de biți egali cu 1 din cadrul cuvântului de 5 biți prezentat la intrare. Cod Verilog structural cu instanțieri de sumatoare complete de 1 bit.
7. Utilizând exclusiv sumatoare complete de un bit, proiectați un circuit care furnizează la ieșire numărul de biți egali cu 1 din cadrul cuvântului de 7 biți prezentat la intrare. Cod Verilog structural cu instanțieri de sumatoare complete de 1 bit.
8. Utilizând exclusiv sumatoare complete de un bit, proiectați un circuit care furnizează la ieșire numărul de biți egali cu 1 din cadrul cuvântului de 15 biți prezentat la intrare. Cod Verilog structural cu instanțieri de sumatoare complete de 1 bit.

Lucrarea 7

Modelarea circuitelor cu bistabile

Activitățile abordate la această lucrare sunt:

- Modelarea și simularea bistabilelor și a circuitelor cu bistabile.
- Verificarea comportamentului bistabilelor prin compararea formelor de undă simulate cu cele așteptate.
- Modelarea generării semnalelor de ceas și de reset.

Atentie! Această lucrare nu este despre „*cum facem rost mai repede de cod*”. Este despre înțelegerea deplină a modului de funcționare și a modelării bistabilelor.

Bistabilul D stochează un bit de informație în intervalul de timp dintre două fronturi active succesive ale semnalului de ceas.

În momentul determinat de apariția frontului activ de ceas, bistabilul eșantionează intrarea D și o „memorează” până la momentul următorului front activ de ceas.

În continuare, se va considera că toate bistabilele comută pe frontul activ crescător ($CK = \uparrow$).

În tabelul 7.1, se descrie funcționarea bistabilului D. S-a notat cu Q starea prezentă și cu Q^+ starea viitoare a bistabilului.

În momentul frontului crescător al semnalului de ceas, $CK = \uparrow$, bistabilul eșantionează starea intrării D , o memorează, și menține valoarea pe ieșire, pe durata unei perioade de ceas.

Modelul Verilog pentru DFF ¹ este prezentat în continuare. Condiția

¹ DFF = D Flip-flop = Bistabil D, engl.

D provine de la cuvântul „Delay” = Întârziere, engl.

CK	D	Q^+	Acțiune
↑	0	0	copiază intrarea, în momentul frontului crescător al ceasului
↑	1	1	copiază intrarea, în momentul frontului crescător al ceasului
0/1	–	Q	păstrează starea, între două fronturi active ale ceasului

Tabelul 7.1

Tabel de funcționare pentru bistabilul D.

„front crescător de ceas” este modelată cu specificația:

```
always @(posedge clk)

module dff (
input      clk , // semnal de ceas, activ pe frontul crescator
input      d   , // intrare D
output reg  q   // iesire Q
);

always @(posedge clk)
    q <= d ;

endmodule // dff
```

Un circuit format din mai multe instanțe de bistabile conectate în paralel se numește **registru**.

Registrul are un model Verilog similar cu al unui bistabil, însă intrarea și ieșirea sunt bus-uri de mai mulți biți.

```
module dff8 (
input      clk , // ceas, activ pe frontul crescator
input      [8-1:0] d , // intrare D
output reg  [8-1:0] q   // iesire Q
);
always @(posedge clk)
    q <= d ;

endmodule // dff8
```

În formele de undă prezentate de simulatoare, semnalele sunt idealizate:

- fronturile sunt ideale (verticale),
- timpul de propagare de la ceas la ieșire este neglijat (considerat 0).

Din acest motiv, semnalele de date D par că se modifică exact în momentul frontului activ de ceas, neîndeplinind condițiile de „*setup*” și „*hold*”. Aceasta este doar o simplificare de simulare.

De fapt, în realitate, datorită întârzierii inerentente la propagarea semnalelor prin bistabil, semnalele de ieșire din bistabil Q sunt întârziate față de frontul de ceas.

Într-un sistem sincron, toate semnalele comută ca efect al frontului pozitiv al semnalului de ceas, deci ulterior acestuia.

Ca o consecință a faptului că ieșirea bistabilului este întârziată față de frontul activ al semnalului de ceas, pe formele de undă provenite din simulare (care consideră timpul de propagare al bistabilelor ca fiind zero) **valoarea eșantionată a unui semnal este valoarea de dinaintea frontului de ceas.**

Valoarea intrării D **eșantionată înaintea frontului de ceas**, este menținută la ieșirea bistabilului Q , pe durata unei perioade de ceas.

În figura 7.1 se justifică motivul pentru care, privind forme de undă simulate, valoarea eșantionată în momentul frontului crescător al semnalului de ceas este întotdeauna valoarea de dinaintea frontului de ceas.

Desenul de sus (figura 7.1) prezintă momentele eșantionării:

- săgeată cu linie punctată, de la momentul frontului spre punctul de eșantionare a datelor, D ;
- cerc cu valoarea eșantionată a intrării D ;
- săgeată continuă de la valoarea eșantionată la palierul în care valoarea se menține pentru o perioadă pe ieșirea Q .

Cu hașur s-a marcat faptul că valoarea ieșirii este necunoscută.

Desenul de jos (figura 7.1) prezintă aceeași formă de undă, dar provenită din simulare cu *ModelSim*.

Cu linie roșie este marcată valoarea necunoscută a ieșirii $1'_{bx}$.

7.1 Desfășurarea lucrării

Pentru fiecare din circuitele menționate în continuare, realizați următoarele acțiuni:

- Desenați simbolul bloc al circuitului (intrări, ieșiri, ceas, reset).

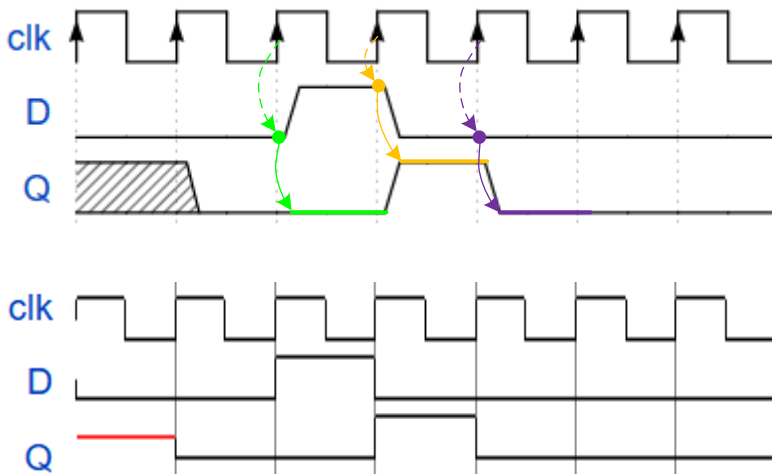


Figura 7.1 Înțelegerea formelor de undă față de frontul de ceas.

Sus: forme de undă desenate, justificare.

Jos: forme de undă simulate.

- Explicați comportamentul așteptat pentru fiecare circuit.
- Desenați formele de undă așteptate.
- Modelați circuitul în Verilog.
- Modelați un mediu de verificare pentru circuite.
- Concepeți un generator de secvențe care pune în evidență comportamentul circuitului.
- Vizualizați formele de undă în fereastra *Wave* a simulatorului *ModelSim* și comparați formele de undă obținute din simulare cu cele așteptate. Justificați comportamentul așteptat.

1. Modelați un registru de 8 biți cu validare/încărcare (DFFE)², a cărui funcționare este descrisă în tabelul 7.2.

2. Modelați un bistabil T (TFF)³.

²DFFE = D Flip-flop Enable = Bistabil D cu validare, engl.

³TFF = T Flip-flop = Bistabil T, engl.

T provine de la cuvântul „Toggle” = Comutare, engl.

CK	E	D	Q^+	Acțiune
$\uparrow$	1	0	0	copiază intrarea, în momentul frontului crescător al ceasului
$\uparrow$	1	1	1	copiază intrarea, în momentul frontului crescător al ceasului
$\uparrow$	0	–	Q	păstrează starea, indiferent de intrarea D (este invalidat)
0/1	–	–	Q	păstrează starea, între două fronturi active ale semnalului de ceas

Tabelul 7.2

Tabelul de funcționare al registrului cu validare/încărcare, DFFE.

Funcționarea bistabilului T este următoarea:

- dacă $T = 0$ bistabilul își păstrează starea, iar
- dacă $T = 1$ bistabilul își complementează/neagă starea.

Se remarcă faptul că bistabilul T are nevoie de a se inițializa asincron într-o stare cunoscută (0 sau 1), altfel va rămâne în starea nedefinită $1'bx$. Nu uitați să modelați un semnal de reset asincron, activ low, denumit în cod *rst_n*.

Tabelul 7.3 prezintă funcționarea bistabilului T (TFF).

CK	T	Q^+	Acțiune
$\uparrow$	0	Q	păstrează starea
$\uparrow$	1	$\bar{Q}$	complementează starea
0/1	–	Q	păstrează starea, între două fronturi active ale ceasului

Tabelul 7.3

Tabel de funcționare pentru bistabilul T.

7.1.1 Mediu de testare pentru circuite cu bistabile

Pentru simularea circuitelor cu bistabile este necesară modelarea unui semnal de ceas, prezent la intrarea tuturor bistabilelor.

Modelarea semnalului de ceas periodic se face pe baza specificației:

„are inițial valoarea 0 și își complementează valoarea la fiecare jumătate de perioadă”.

Acest lucru este modelat în Verilog astfel:

```
initial begin
    clk = 1'b0;           // valoare initiala 0
    forever #CK_SEMIPERIOD // valoare complementata
        clk = ~clk;       // la fiecare semi-perioada
end
```

Semnalul de reset asincron se recomandă a avea valoarea inițială inactivă și a se activa ulterior timpului zero de simulare. Deși modelează un semnal asincron (independent de semnalul de ceas), se recomandă ca semnalul de reset să comute în momente determinate de frontul activ de ceas, iar durata de activare să fie mai mare decât o perioadă de ceas.

Codul următor prezintă modelul complet pentru un generator de semnal de ceas și de reset asincron (sincronizat pe ceas).

```
module ck_rst_tb #(
    parameter CK_SEMIPERIOD = 'd5 // semi-perioada ceas
) (
    output reg clk           , // ceas
    output reg rst_n        // reset asincron, activ 0
);

initial begin
    clk = 1'b0;           // valoare initiala 0
    forever #CK_SEMIPERIOD // valoare complementata
        clk = ~clk;       // la fiecare semi-perioada
end

initial begin
    rst_n <= 1'b1;        // initial inactiv
    @(posedge clk);
    rst_n <= 1'b0;        // activare sincrona
    @(posedge clk);
    rst_n <= 1'b1;        // inactivare dupa doua perioade de ceas
    @(posedge clk);      // ramane inactiv pentru totdeauna
end

endmodule // ck_rst_tb
```

Formele de undă generate pentru semnalele de ceas și de reset sunt prezentate în figura 7.2.

Un generator de semnale pentru circuite cu bistabile trebuie să modeleze schimbarea semnalelor exclusiv pe frontul activ al semnalului de ceas. Pentru aceasta, se recomandă ca timpul să nu fie modelat cu întârzieri absolute (de exemplu #10), ci cu întârzieri egale cu multiplii ai perioadei de ceas: `@(posedge clk).`

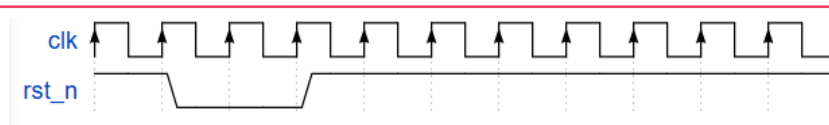


Figura 7.2 Forme de undă ale generatorului de semnal de ceas și de reset asincron.

De exemplu, porțiunea de cod următoare modelează forma de undă prezentată în figura 7.3.

```
initial begin
    t <= 1'bx;
    @(negedge rst_n);
    @(posedge rst_n);
    t <= 1'b0;
    @(posedge clk);
    @(posedge clk);
    t <= 1'b1;
    @(posedge clk);
    t <= 1'b0;
    repeat (3) @(posedge clk);
    t <= 1'b1;
    repeat (3) @(posedge clk);
    t <= 1'b0;
    @(posedge clk);
    t <= 1'b1;
    @(posedge clk);
    @(posedge clk);
    t <= 1'b0;
    @(posedge clk);
    @(posedge clk);
    t <= 1'b1;
    @(posedge clk);
    t <= 1'b0;
    @(posedge clk);

    $display ("%M %0t INFO: Final simulare.", $time);
    $stop;
end
```

Figura 7.4, obținută din fereastra *Wave* a simulatorului *ModelSim*, confirmă forma de undă așteptată (figura 7.3).

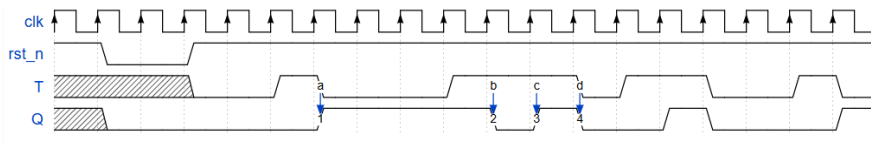


Figura 7.3 Stimul la intrarea T și ieșirea așteptată Q , pentru bistabilul TFF.

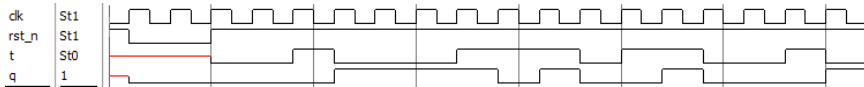


Figura 7.4 Stimul la intrarea t și ieșirea simulată q , pentru bistabilul TFF.

7.2 Teme de casă

Modelați în Verilog, simulați și studiați fiecare din circuitele prezentate în figurile 7.6, 7.7, 7.8, 7.9, 7.10, utilizând simulatorul *ModelSim*.

- Explicați comportamentul așteptat pentru fiecare circuit.
- Desenați formele de undă așteptate. Considerați la intrare forma de undă recomandată în figura 7.5. Pentru desenarea formelor de undă așteptate puteți utiliza *WaveDrom* cu descrierea următoare:

```
{signal: [
  {name: 'clk',      wave: 'P.....' },
  {name: 'rst_n',    wave: '10.1.....' },
  {name: 'INTRARE', wave: 'x..0.10..1.0..1..0..1....0' }
]}
```

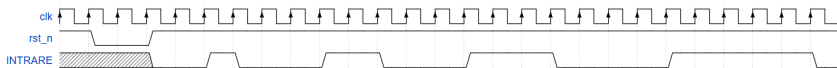


Figura 7.5 Forme de undă recomandate pentru studierea circuitelor propuse ca temă de casă.

- Modelați circuitul în Verilog.
- Modelați un mediu de verificare pentru circuite.
- Concepți un generator de secvențe care pune în evidență comportamentul circuitului.

- Vizualizați formele de undă în fereastra *Wave* a simulatorului *ModelSim* și comparați formele de undă obținute din simulare cu cele așteptate. Justificați comportamentul așteptat.

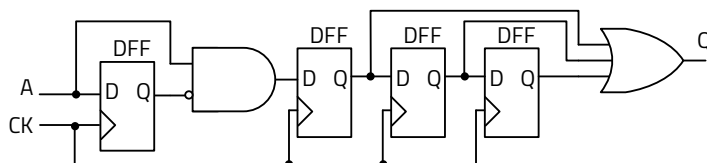


Figura 7.6 Circuit cu bistabile D și ieșire din circuit combinațional (poartă logică).

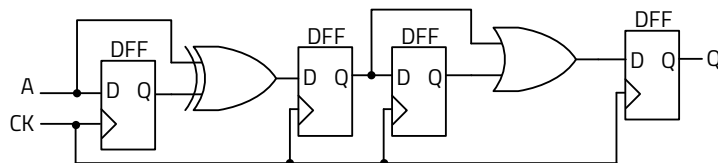


Figura 7.7 Circuit cu bistabile D și ieșire din bistabil.

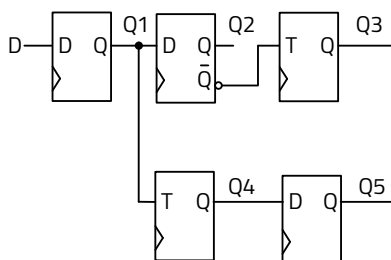


Figura 7.8 Circuit cu bistabile D și T.

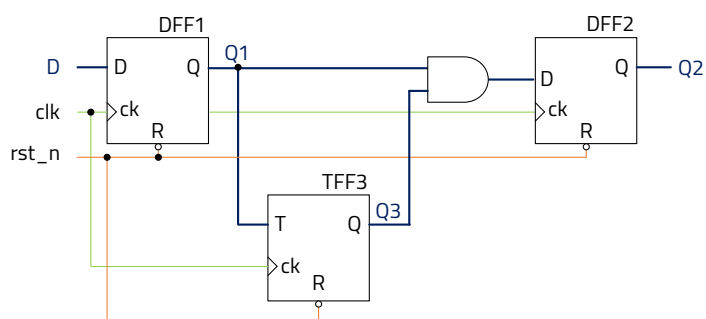
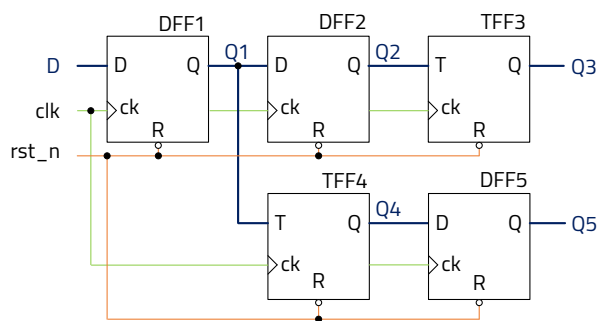


Figura 7.9 Circuit cu bistabile D și T și poartă logică.

**Figura 7.10** Circuit cu bistabile D și T.

Lucrarea 8

Modelarea circuitelor secvențiale cu bistabile

Subiectul acestei lucrări îl constituie proiectarea și verificarea prin simulare a unor circuite cu bistabile și porți logice, mici dar specializate și frecvent utilizate în practică.

8.1 Desfășurarea lucrării

1. **Bistabil T.** Modelați un bistabil T având comportamentul descris de graful din figura 8.1. Verificați modelul prin simulare Verilog. Deduceți tabelul de funcționare al bistabilului T.

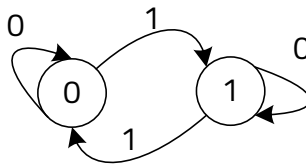


Figura 8.1 Graf de tranziții: bistabil T.

2. **Bistabil JK.** Modelați în Verilog un bistabil JK având comportamentul descris de graful din figura 8.2. Verificați modelul prin simulare Verilog. Deduceți tabelul de funcționare al bistabilului JK.

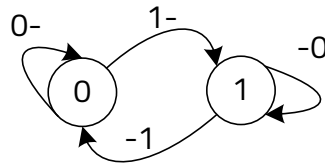


Figura 8.2 Graf de tranziții: **bistabil JK.**

3. **Detector de front pozitiv.** Modelați în Verilog un circuit cu bistabile și porți logice care prezintă la ieșire un puls cu lățime de o perioadă de ceas dacă intrarea a comutat din 0 în 1 în perioada anterioară.

Completați formele de undă cu semnale intermediare.

Faceți o simulare Verilog și comparați formele de undă simulate cu cele așteptate (figura 8.3).

```

{signal: [
  {name: 'clk',      wave: 'P.....' },
  {name: 'INTRARE', wave: '01...0..1.0..1.....0....10..'},
  {name: 'POS_EDGE', wave: '010.....10...10.....10..'}
]
}
  
```

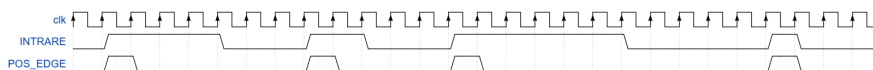


Figura 8.3 Forme de undă așteptate pentru „detector de front pozitiv”.

4. **Detector de front negativ.** Modelați în Verilog un circuit cu bistabile și porți logice care prezintă la ieșire un puls cu lățime de o perioadă de ceas dacă intrarea a comutat din 1 în 0 în perioada anterioară.

Completați formele de undă cu semnale intermediare.

Faceți o simulare Verilog și comparați formele de undă simulate cu cele așteptate (figura 8.4).

5. **Detector de front.** Modelați în Verilog un circuit cu bistabile și porți logice care prezintă la ieșire un puls cu lățime de o perioadă de ceas dacă intrarea a comutat în perioada anterioară.

Completați formele de undă cu semnale intermediare.

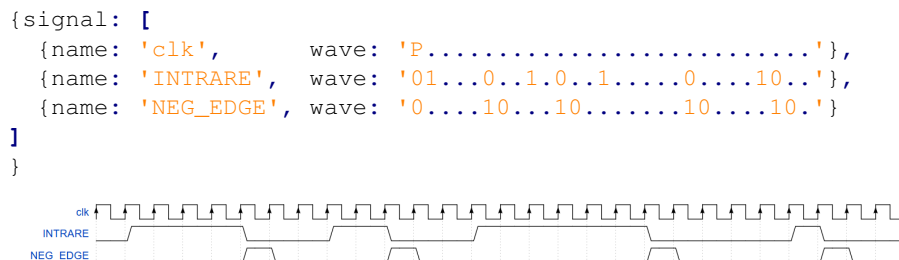


Figura 8.4 Forme de undă așteptate pentru „detector de front negativ”.

Faceți o simulare Verilog și comparați formele de undă simulate cu cele așteptate (figura 8.5).

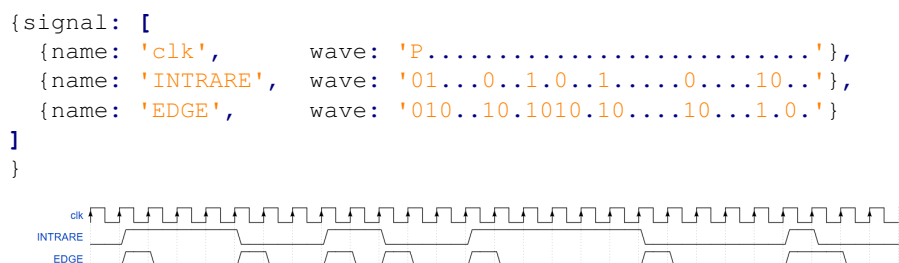


Figura 8.5 Forme de undă așteptate pentru „detector de front”.

6. **Detector de egalitate.** Modelați în Verilog un circuit cu bistabile și porți logice care prezintă la ieșire un puls cu lățime de o perioadă de ceas dacă cele două intrări au fost egale în perioada anterioară.

Adăugați circuitului o ieșire care prezintă valoarea 1 o perioadă de ceas dacă cele două intrări au fost egale în ultimele două perioade anterioare succesive.

Adăugați circuitului o ieșire care prezintă valoarea 1 o perioadă de ceas dacă cele două intrări au fost egale în ultimele trei perioade anterioare succesive.

Completați formele de undă cu semnale intermediare.

Faceți o simulare Verilog și comparați formele de undă simulate cu cele așteptate (figura 8.6).

```
{signal: [
  {name: 'clk', wave: 'P.....'},
  {name: 'IN1', wave: '010..10..1.0..1...010...10..'},
  {name: 'IN2', wave: '01.0..1...0..1.01.01...0.1...'},
  {name: 'EQ1', wave: 'x1.01.0...101.0101...0.1..0.'},
  {name: 'EQ2', wave: 'x.10.10.....10...1..0..1.0.'},
  {name: 'EQ3', wave: 'x..0.....1.0...10.'}
]}
```

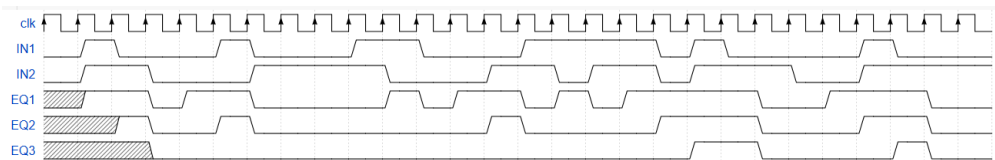


Figura 8.6 Forme de undă așteptate pentru „detector de egalitate”.

8.2 Temă de casă

Circuit de complementare față de 2. Modelați în Verilog un circuit cu bistabile și porți logice care completează față de 2 numere întregi primite secvențial, câte un bit pe tact, cu LSb primul bit.

Biții asociați unui număr sunt însoțiți de valoarea 1 pe o intrare adițională de validare.

Numărul de intrare înmulțit cu -1 este prezentat secvențial la ieșire, câte un bit pe tact, cu LSb primul bit. Biții asociați unui număr sunt însoțiți de valoarea 1 pe o ieșire adițională de validare.

Denumiți modulul `compl2.`

Completați formele de undă cu semnale intermediare.

Faceți o simulare Verilog și comparați formele de undă simulate cu cele așteptate (figura 8.7).

Utilizați  **ChatGPT** pentru a converti formele de undă descrise în format [WaveDrom](#) într-un modul Verilog care generează stimuli pentru modulul proiectat, ca în figura 8.7. Denumiți modulul care generează stimuli `compl2_deg_data.`

Ce numere au fost transmise prin semnalele prezentate în figura 8.7? Scrieți un modul „monitor” (denumit `compl2_moni`) care culege datele de pe o interfață și le scrie în fereastra *Transcript* a *Model_Sim*.

```
module compl2_moni
```

```

{signal: [
  {name: 'clk',      wave: 'P.....'},
  {name: 'DATA_in',  wave: 'x0.10..1.x..1.01x..10.1.0x.'},
  {name: 'VLD_in',   wave: '01.....0..1...0..1.....0.'},
  {name: 'DATA_out', wave: 'x.0.1...0.x..1010x..1..0.1x'},
  {name: 'VLD_o',    wave: '0.1.....0..1...0..1.....0'}
]
}

```

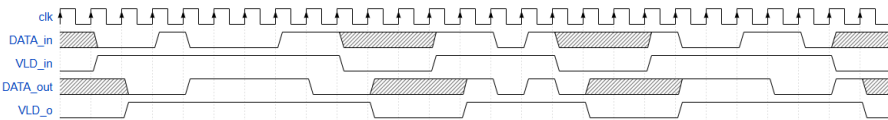


Figura 8.7 Forme de undă așteptate pentru „circuit de complementare față de 2”.

```

(
  input  clk      , // semnal de ceas, activ pe frontul crescator
  input  rst_n    , // reset asincron, activ low
  input  data     , // data, activa cand valid=1, LSB primul
  input  valid    , // 1 = date valide
);

// codul aici

endmodule // compl2_moni

```

Instantiați monitorul pe cele două interfețe ale modulului (de intrare și de ieșire) în mediul de testare `compl2_test`, ca în figura 8.8.

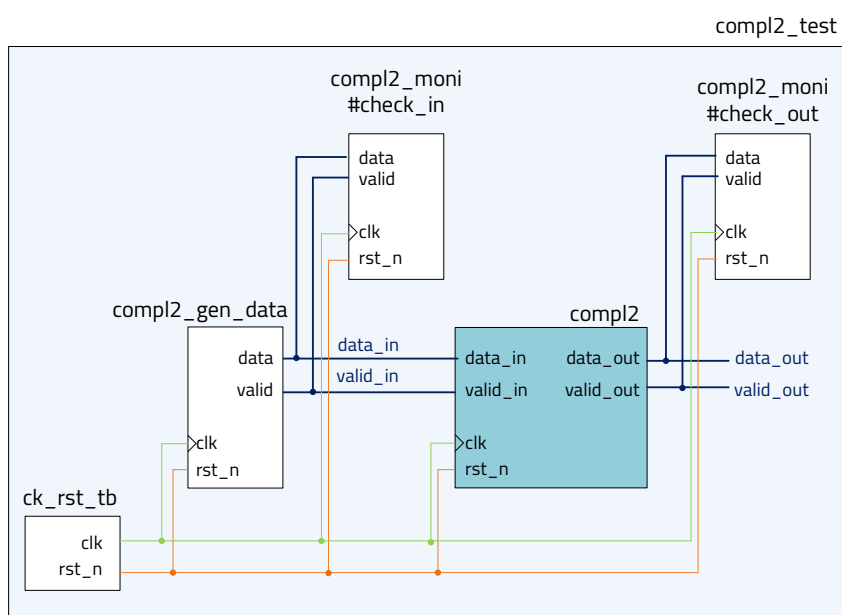


Figura 8.8 Mediu de test pentru „circuit de complementare față de 2” cu instanțiere de monitoare de date.

Lucrarea 9

Modelarea FSM

Această lecție prezintă modelarea în Verilog a automatelor secvențiale cu un număr finit de stări (FSM = *Finite State Machine*, engl.).

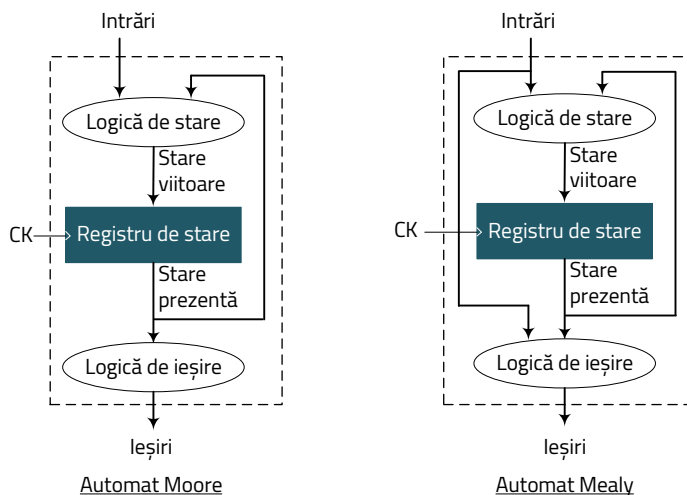


Figura 9.1 Structuri de automate imediate.

Automat Moore (ieșirile depind exclusiv de starea prezentă).

Automat Mealy (ieșirile depind atât de starea prezentă, cât și de intrări).

Automatele secvențiale cu număr finit de stări au în structura lor un registru pentru memorarea stării curente și un circuit combinațional pentru determinarea stării viitoare. Automatele au ca element esențial o buclă

realizată prin faptul că determinarea stării viitoare se face pe baza stării prezente.

Un automat care nu prezintă ieșiri se numește **semi-automat**. Registrul de stare are la ieșire codul stării prezente și la intrare codul stării viitoare. Starea viitoare este determinată prin circuitul combinațional, pe baza stării prezente și a intrărilor.

În funcție de modul de determinare a ieșirilor automatului, automatele se clasifică în (figura 9.1):

- **Automate Moore**, pentru care ieșirea este dependentă exclusiv de starea prezentă și
- **Automate Mealy**, pentru care ieșirea este dependentă atât de starea prezentă cât și de intrări.

9.1 Desfășurarea lucrării

1. Proiectați și modelați în Verilog un automat al cărui comportament este descris prin graful de tranziții prezentat în figura 9.2. Verificați modelul în simulare.

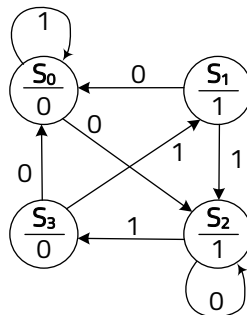


Figura 9.2 Graful de tranziții al automatului, problema 1.

Automatul are 4 stări (noduri), denumite S_0 , S_1 , S_2 , S_3 , posibil de codificat cu 2 biți.

Automatul are o intrare (1 bit menționat pe arce) și o ieșire dependentă exclusiv de stare (menționată în noduri).

Realizați un mediu de testare în care să verificați toate arcele. Pentru realizarea mediului de testare inițial puteți să vă inspirați din soluția furnizată de  ChatGPT .

2. Proiectați și modelați în Verilog un automat care prezintă comportamentul din graful de tranziții din figura 9.3.

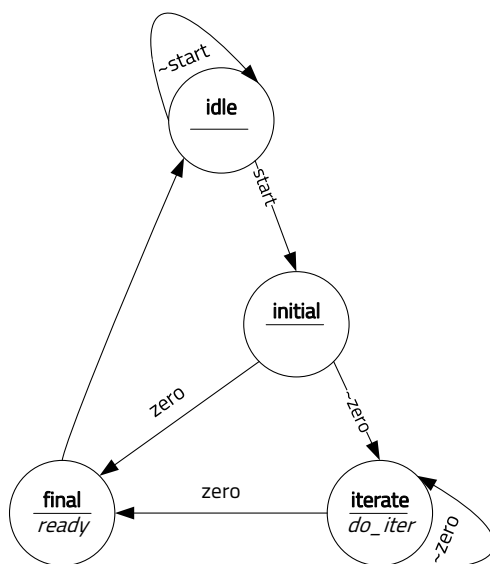


Figura 9.3 Graful de tranziții al automatului problema 2.

Automatul are 4 stări, denumite *idle*, *initial*, *iterate* și *final*, posibil de codificat pe un număr minim de 2 biți ($2^2 \leq 4$). Resetarea automatului va fi în starea *idle*.

Automatul are două intrări, denumite *start* și *zero*, însă nu ambele sunt relevante în toate stările pentru a decide stările viitoare.

Ieșirile automatului, denumite *ready* și *do_iter*, depind exclusiv de starea prezentă (numele lor apar în simbolurile de stare, noduri, sub numele stării în care este activă). Este un automat de tip Moore. Arcul dintre starea *final* și starea *idle* nu are nicio condiție și desemnează o tranziție în care intrările sunt indiferente.

Automatul are structura prezentată în figura 9.4. Poate fi modelat în una din cele două variante prezentate în figură.

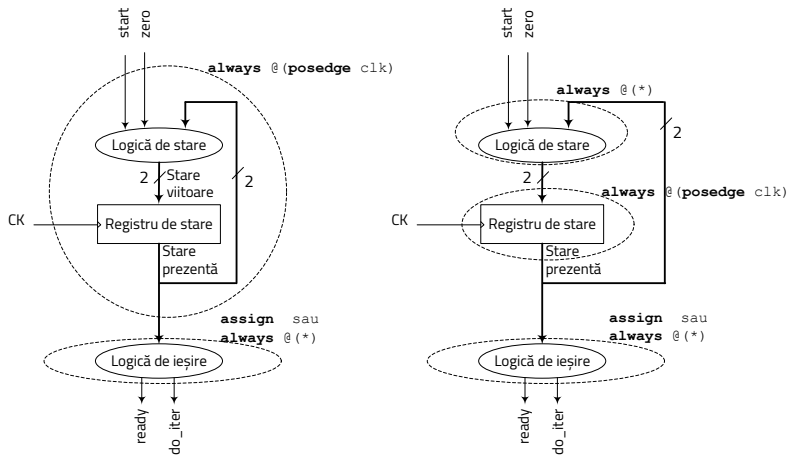


Figura 9.4 Structura automatului menționat la problema 2.

Realizați un mediu de testare în care să verificați toate arcele.

9.2 Temă de casă

Proiectați și modelați în Verilog un automat care controlează ciclul unui semafor.

- Semaforul pentru mașini are următorul ciclu:

roșu

 (30 sec),

verde_

 (minim 60 sec),

galben

 (5 sec).
- Semaforul pentru pietoni este

roșu

 când cel pentru mașini este verde sau galben și

verde_

 când cel pentru mașini este roșu.
- Buton pietonilor este activ 1. Butonul pietonilor este ignorat dacă semaforul pietonilor este verde. Apăsarea acestuia are efect exclusiv dacă semaforul pietonilor este roșu. Prin apăsarea butonului, pietonii cer ca semaforul de pietoni să devină verde.
- Cât timp semaforul pietonilor este roșu, orice apăsare pe buton este memorată. Apăsările succesive nu produc niciun efect.



Figura 9.5 Semafor cu buton pentru pietoni.

- Semaforul pentru mașini comută din verde în galben dacă a fost verde minimum 60 sec și a existat o apăsare pe buton în intervalul de timp în care acesta a fost verde.
- Culoarea verde pentru mașini este de minimum 60 sec.
- Frecvența ceasului de intrare este de 1Hz (perioada 1 secundă). Ulterior, adăugați un divizor de frecvență care să accepte o intrare de ceas de frecvență 1MHz și să producă un semnal de validare cu perioadă de o secundă (puls cu lățime de 1 microsecundă și perioadă de 1 secundă, sincron cu ceasul sistem). Acest puls va determina comutarea stării automatului.

Lucrarea 10

Modelarea registrelor și a numărătoarelor

Această lucrare are ca scop modelarea registrelor și a numărătoarelor în Verilog. Se vor modela diverse numărătoare: cu presetare, cu numărare în sens crescător sau descrescător, cu numărare modulo parametrizabil, cu indicatoare de stare (depășire, zero).

Se vor prezenta circuitele LFSR (*Linear Feedback Shift Register*, engl.), ca un caz particular de numărătoare.

10.1 Desfășurarea lucrării

Modelați următoarele circuite în Verilog.

Verificati corectitudinea modelelor în simulare.

Explicati comportamentul asteptat pentru fiecare circuit.

10.1.1 Registru cu validare

Modelați în Verilog un registru cu validare, cu lățime parametrizabilă. Registrul are o intrare de control, a cărei activare determină încărcarea datei de intrare. Data este menținută pe ieșirea registrului până la următoarea încărcare.

Interfata modulului este:

```
module reg_valid #(
parameter WIDTH = 8           // latime, in numar de biti
) (
input      clk                , // ceas, front crescator
input     rst_n               , // reset, activ in zero
```

```

input          load      , // semnal de incarcare
input [WIDTH-1:0] data_in , // date de intrare
output reg [WIDTH-1:0] data_out // date de iesire
);

// codul aici

endmodule // reg_valid

```

Formele de undă așteptate sunt prezentate în figura 10.1.

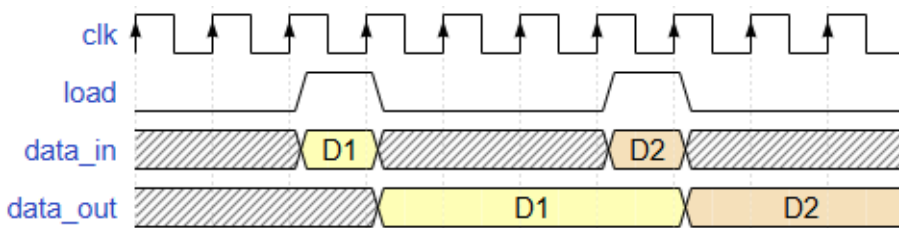


Figura 10.1 Forme de undă pentru registrul cu validare.

10.1.2 Numărător

Modelați în Verilog un numărător cu lățime parametrizabilă și cu interfața descrisă în continuare.

```

module counter_param #(
parameter WIDTH      = 8           // latime, in numar de biti
parameter MODULO     = 17          // lungime ciclu de numarare
) (
input          clk      , // ceas, front crescator
input          rst_n    , // reset activ in zero

input          ce       , // chip enable
input          ld       , // load
input          cu       , // count up
input          cd       , // count down
input [WIDTH-1:0] data_in , // date de intrare
output reg [WIDTH-1:0] data_out // date de iesire
);

// ce ld cu cd | data_out+ functie
// -----+-----
// 0 - - - | data_out   pastreaza starea
// 1 1 - - | data_in    presetare

```

```
// 1 0 1 - | data_out+1  numara in sens crescator MODULO
// 1 0 0 1 | data_out-1  numara in sens descrescator MODULO
// 1 0 0 0 | data_out    pastreaza starea

// codul aici

endmodule // counter_param
```

Formele de undă așteptate sunt prezentate în figura 10.2.

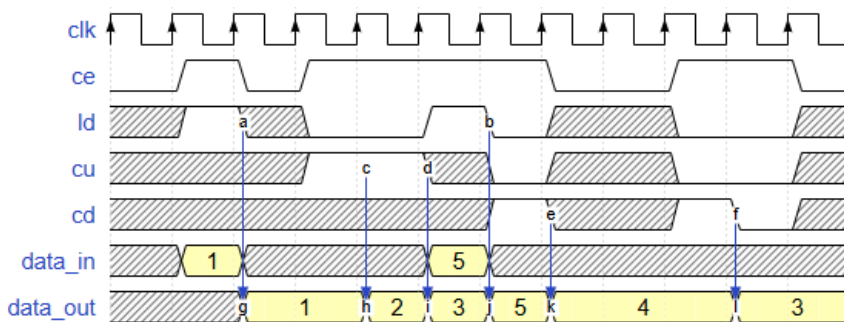


Figura 10.2 Forme de undă pentru numărător.

Instantiați numărătorul modelat, `counter_param`, eventual cu o logică exterioară acestuia, pentru a realiza următoarele:

- Numărător în sens crescător, modulo 27.
- Numărător în sens descrescător, modulo 27.
- Numărător în sens crescător, între 13 și 27.

Implementați următorii indicatori pentru numărător:

- `ovf`, activ 1, când numărătorul numără în sens crescător de la valoarea maximă la zero, sau când numărătorul numără în sens descrescător de la zero la valoarea maximă.
- `zero`, activ 1, când numărătorul are valoarea zero.

10.1.3 LFSR

Modelați în Verilog un circuit LFSR pe 4 biți. Circuitul se inițializează într-o stare diferită de 0000, de exemplu 0001.

Polinomul caracteristic al LFSR este $X^4 + X^3 + 1$.

Interfața modulului este:

```
module lfsr_4 (
input          clk      , // semnal de ceas,
input          rst_n    , // semnal de reset asincron activ 0
output reg [3:0] data_o  // iesirea lfsr
);

// polinom X^4 + X^3 + 1

// codul aici
endmodule // lfsr_4
```

Formele de undă așteptate sunt prezentate în figura 10.3.

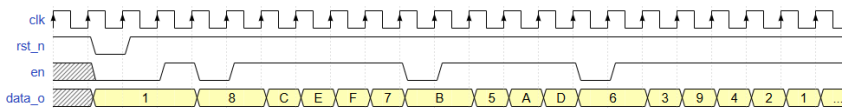


Figura 10.3 Forme de undă pentru LFSR.

10.2 Temă de casă

Modelați un circuit LFSR cu polinom parametrizabil cu interfața următoare:

```
module lfsr_param #(
parameter WIDTH      = 'd10      , // numarul de biti
parameter POLYNOM    = 'b00_1000_0001 // polinom caracteristic
                                     // implicit X10+X7+1
) (
input          clk      , // ceas
input          rst_n    , // reset asincron activ 0
output reg [WIDTH - 1:0] data_o  // iesirea LFSR
);

// codul aici

endmodule // lfsr_param
```

Numărul de biți ai parametrului POLYNOM trebuie să fie egal cu valoarea parametrului WIDTH. Valoarea polinomului prezintă bitul egal cu 1 dacă indexul respectiv este conținut de polinom.

Pentru polinomul $X^4 + X^3 + 1$, valorile parametrilor sunt:

```
parameter WIDTH      = 'd4  
parameter POLYNOM    = 4'b1001
```

Pentru polinomul $X^{10} + X^7 + 1$, valorile parametrilor sunt:

```
parameter WIDTH      = 'd10  
parameter POLYNOM    = 10'b00_1000_0001
```

Modelarea se bazează pe observațiile:

- Fiecare bit, cu excepția celui mai semnificativ, își preia valoarea viitoare din valoarea curentă a bitului de index superior.
- O operație AND pe biți între toți biții valorii curente și polinom determină un șir de biți în care diferiți de zero sunt doar biții din polinom cu valoarea 1. Altfel spus, parametrul `POLYNOM` realizează prin funcția logică AND mascarea biților care nu intră în calculul bitului cel mai semnificativ.
- Operația XOR pe baza biților mascați determină valoarea următoare a bitului cel mai semnificativ.

Verificați proiectarea prin simulare.

Lucrarea 11

Modelarea memoriilor

Lucrarea are ca scop modelarea memoriilor. Memoriile nu sunt implementate în hardware similar cu circuitele logice, prin sinteză. Din acest motiv, modelele de memorie nu trebuie să fie sintetizabile și pot conține specificații HDL fără corespondent în hardware. Ca de exemplu, accesul la fișiere de date și inițializarea variabilelor.

11.1 Noțiuni teoretice

O memorie se poate modela în Verilog ca o variabilă bidimensională:

```
reg [WIDTH-1:0] mem[DEPTH-1:0];
```

În etapa de sinteză RTL ¹, variabila bidimensională trebuie înlocuită cu resurse hardware reale. Dacă se dorește implementarea cu o memorie specifică unei tehnologii, atunci în biblioteca de tehnologie trebuie să existe modele potrivite, iar sintetizatorul să fie informat prin asocierea unui model de memorie cu o memorie fizică.

Alternativa o constituie sintetizarea variabilei bidimensionale sub forma unui set de registre, implementat cu bistabile D.

Diferența dintre **setul de registre** și **memorie** o constituie faptul că toate datele din setul de registre se pot accesa simultan și tot timpul, iar datele din memorie se pot accesa, la un moment dat, numai cele de la adresa curentă.

¹RTL = *Register Transfer Level* = nivel de detaliere care evidențiază transferurile de date între registre, engl.

11.1.1 Memorii

După modul în care sunt efectuate acțiunile de către o memorie (citire și scriere), memoriile pot fi clasificate în:

- **memorii sincrone**, pentru care accesarea datelor se face sincron cu frontul activ al unui semnal de ceas, sau
- **memorii asincrone**, pentru care accesarea datelor se face la activarea unor semnale de control specifice (citire sau scriere).

Numărul de porturi ale memoriei este reprezentat de numărul de adrese pe baza cărora se poate accesa memoria, la un moment dat. După numărul și tipul porturilor, memoriile pot fi clasificate astfel:

- **1RW** - memoria single-port, un singur port prin care se face atât citirea cât și scrierea (poate face o singură acțiune odată, fie citire, fie scriere),
- **1R1W** - memoria cu două porturi independente, unul de scriere și altul de citire (poate face simultan o citire și o scriere la adrese diferite),
- **2RW** - memoria cu două porturi independente prin care se pot face atât acțiuni de citire cât și de scriere (pot fi realizate orice două acțiuni simultan, inclusiv două scrieri la adrese diferite).

Se pot imagina diverse tipuri de memorii, cu mai multe porturi sincrone, pe același semnal de ceas sau pe semnale de ceas asincrone (fiecare port cu ceasul propriu).

Figurile 11.1, 11.2 și 11.3 prezintă simbolurile bloc al memoriilor 1RW, 1R1W și 2RW.

Memoria este modelată cu 3 parametri:

```
parameter ADDR_WIDTH = 8 ; // numar de biti adresa
parameter MEM_DEPTH = 256 ; // numar de locatii
parameter WORD_BYTES = 8 ; // numar de bytes date
```

- **ADDR_WIDTH** - numărul de biți ai adresei. Spațiul de adresare este de maximum 2^{ADDR_WIDTH} .
- **MEM_DEPTH** - adâncimea memoriei (exprimată în număr de locații), maximum 2^{ADDR_WIDTH} , dar poate fi mai mică.
- **WORD_BYTES** - lățimea datelor memoriei (exprimată în bytes). Memoria are date de lățime **WORD_BYTES** bytes, adică $8 * WORD_BYTES$ biți. Dacă memoria are o intrare pentru mascarea datelor la scriere (byte-enable), aceasta are dimensiune **WORD_BYTES** biți (câte un bit pentru fiecare byte de date).

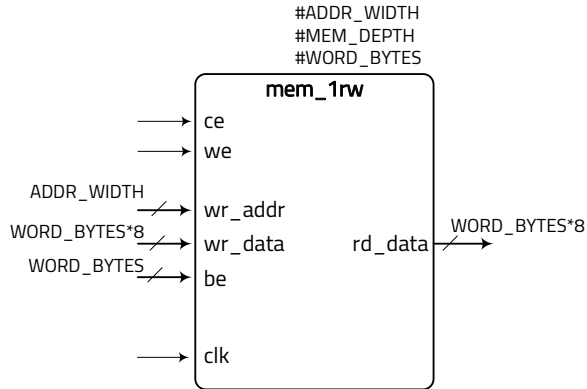


Figura 11.1 Simbol bloc pentru memoria 1RW, un port de scriere și citire („single-port”, engl.).

Variabila bidimensională care modelează memoria este declarată cu parametrii astfel:

```
reg [8*WORD_BYTES -1 : 0] mem[MEM_DEPTH -1 : 0];
```

Modelul de memorie **mem_1rw** descrie parametrii memoriei, interfața și acțiunile de citire și de scriere efectuate asupra matricii de memorie.

```
module mem_1rw #(
    parameter ADDR_WIDTH = 8 ,
    parameter MEM_DEPTH = 256 , // MEM_DEPTH <= 2^ADDR_WIDTH
    parameter WORD_BYTES = 8 // latimea datelor (biti) =
                                // 8 * WORD_BYTES
) (
    input clk , // ceas
    input ce , // chip enable
    input we , // write enable
    input [ADDR_WIDTH -1 : 0] addr , // adresa
    input [8*WORD_BYTES -1 : 0] wr_data , // date scrie
    input [WORD_BYTES -1 : 0] be , // byte enable
    output reg [8*WORD_BYTES -1 : 0] rd_data // date citite
);
reg [8*WORD_BYTES -1 : 0] mem[MEM_DEPTH -1 : 0];

wire [8*WORD_BYTES-1:0] write_mask;

// extinde bitul de masca la cate 8 biti
genvar i;
```

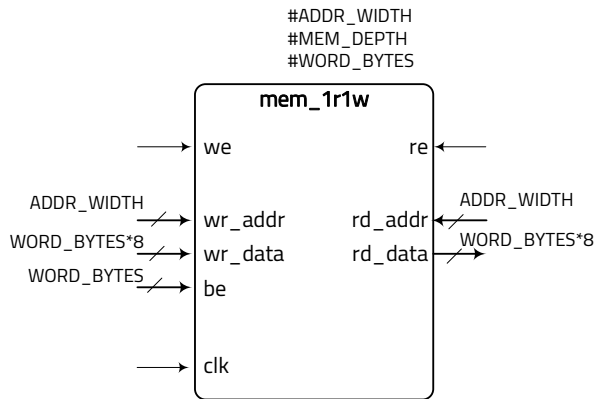


Figura 11.2 Simbol bloc pentru memoria 1R1W, un port de citire și unul de scriere.

```

generate
    for (i = 0; i < WORD_BYTES; i = i + 1) begin : gen_mask
        assign write_mask[i*8 +: 8] = {8{be[i]}};
    end
endgenerate

// scriere memorie
always @(posedge clk)
if (ce & we)
    mem[addr] <= (mem[addr] & ~write_mask) | // pastreaza datele
                  (wr_data & write_mask);   // modifica datele

// citire memorie
always @(posedge clk)
if (ce & (~we)) rd_data <= mem[addr]; else // date citite
                rd_data <= 'bx;           // date necunoscute

endmodule // mem_1rw

```

11.1.2 Inițializarea memoriei cu o constantă

În mod normal, o memorie se scrie de către circuit înainte de a fi citită. Din acest motiv, nu este nevoie ca memoria să fie inițializată cu date cunoscute. Dacă nu este inițializată, ca orice variabilă Verilog, se poate considera că valorile memorate sunt toate 'bx.

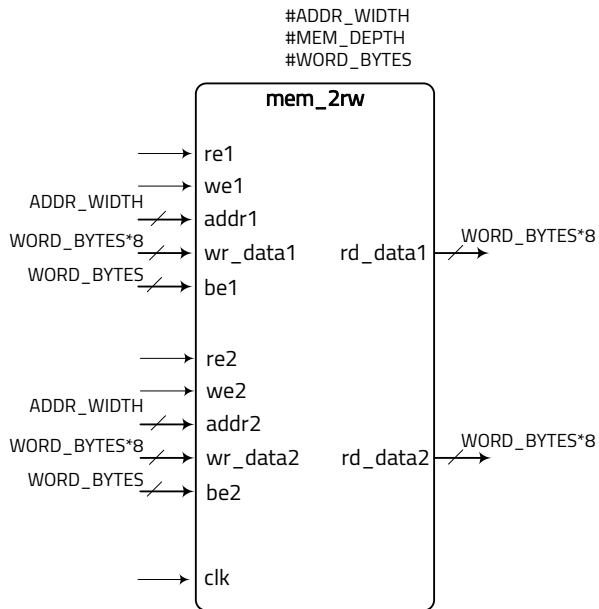


Figura 11.3 Simbol bloc pentru memoria 2RW, două porturi de scriere și citire („true dual-port”, engl.).

Există însă cazuri când este necesară inițializarea memoriilor, în afara circuitului de accesare normală.

Exemple de memorii ce trebuie inițializate sunt memoriile ROM sau modulele LUT („Look-Up Table”, engl.).

Alteori, module de memorie RAM (care se scriu înainte de a fi citite) sunt utilizate pentru a testa exclusiv circuitul care preia date din memorie. Se poate presupune că datele au fost scrise anterior (de către un alt modul) și nu se dorește simularea perioadei de scriere a memoriei. De exemplu, dacă se testează un circuit care preia datele din memorie pentru a fi afișate („driver video”).

Inițializarea variabilelor bidimensionale care modelează memoriile este un proces exclusiv de simulare (nu are corespondent în hardware real).

Inițializarea tuturor locațiilor memoriei cu aceeași valoare constantă se poate face într-o specificație `initial` în care toate locațiile sunt inițializate secvențial, una câte una, la timpul 0 de simulare, cu valoarea constantă:

```
integer    idx ;                      // index
initial begin

    // initializare totala cu o valoare constanta
    for (idx=0; idx < MEM_DEPTH; idx=idx+1)
        mem[idx] = 'hAB;             // valoare constanta

end
```

11.1.3 Inițializarea memoriei

Funcțiile Verilog `$readmemb` și `$readmemh` inițializează o matrice de memorie (variabilă bidimensională) cu valori dintr-un fișier extern. Fișierul conține caractere ASCII ce reprezintă valori binare („0” sau „1”) sau în baza 16 („0-9”, „A-F”). Valorile trebuie să aibă aceeași lățime cu cea a memoriei (exprimată în biți) și să fie separate cu spații. Valorile adreselor de început și de sfârșit sunt valori în baza 16 (atât pentru `$readmemb` cât pentru și `$readmemh`) precedate de caracterul „@”.

Sintaxa specificațiilor de inițializare a memoriilor este:

```
$readmemb("nume_file", mem);
$readmemh("nume_file", mem);
```

Pentru memoria `mem_1rw`, inițializarea cu specificație `$readmemh` se realizează prin includerea acesteia într-o specificație `initial`:

```
// initializare din fisier extern cu date in hexa
$readmemh("mem.hex", mem);    // initializare memorie
```

11.1.4 Funcții pentru accesul la fișiere de intrare/ieșire

Verilog permite accesul la fișierele de pe calculatorul local pentru a ușura implementarea unor medii de verificare complexe și pentru a înlesni depanarea. Specificațiile de acces la fișiere **nu sunt sintetizabile** și trebuie utilizate exclusiv în module ce modelează mediul de verificare.

Funcțiile de acces la fișierele de pe mediul de stocare al calculatorului sunt apelate prin funcții specifice Verilog. Numele acestor funcții este precedat de simbolul `$`. Un fișier local, identificat prin cale și nume în sistemul de fișiere al calculatorului, este asociat cu un descriptor de fișier în Verilog. Calea fișierului se recomandă a fi relativă la folderul de proiect, în care se face simularea, pentru a asigura portabilitatea modelului.

Sintaxa asocierii dintre un fișier și un descriptor este următoarea:

```
integer file_id;           // descriptor de fisier
file_id = $fopen("file_name"[, type]);
```

Tipul fișierului semnifică scopul pentru care este deschis fișierul. Cele mai utilizate tipuri sunt menționate în tabelul următor:

Tip fișier	Descriere
„r” sau „rb”	Deschidere pentru citire
„w” sau „wb”	Creare pentru scriere
„a” sau „ab”	Scriere în continuarea unui fișier existent (<i>append</i> , engl.)

Funcția `$fopen` deschide un fișier și îi asociază un descriptor, declarat ca un întreg pe 32 de biți. Descriptorul va fi utilizat ulterior de către specificațiile de acces la fișier. În caz de eroare, descriptorul are valoarea zero. Se recomandă ca după o specificație de deschidere de fișier să se verifice dacă fișierul a putut fi deschis, prin testarea descriptorului de fișier:

```
if (data_file == 0) begin
    $display("Nu se poate deschide fisierul <cale/nume fisier>.");
    $stop;
end
```

Caracterul „b” prezent în tipul fișierului desemnează un „fișier binar” (conține valori binare). În lipsa acestuia, fișierul este de „tip text” (conține caractere ASCII).

Comanda de închidere a unui fișier este `$fclose`:

```
$fclose(file_id);
```

Pentru citirea din fișier există comenzi specifice. Scrierea în fișier se face cu comanda `$fdisplay` (similară comenzii `$display`). Scrierea se face temporar într-un buffer de ieșire. Bufferul este depus în fișier fie cu comenzi explicite `$flush(file_id)`; fie la întreruperea sau finalul simulării. Există și comenzi de control pentru lucru cu fișiere.

11.1.5 Inițializarea memoriei din fișier

Funcțiile de inițializare a memoriilor `$readmemb` și `$readmemh` au ca argument un fișier extern ce conține caractere ASCII. Dacă se dorește inițializarea unei memorii dintr-un fișier binar (de exemplu o imagine în format **.raw**), această opțiune nu este potrivită.

În schimb, se poate deschide fișierul în citire ca fiind de tip binar (opțiunea „rb - read, binar”) și atribui câte o valoare fiecărei locații de memorie, cu funcția `$fread`.

```

integer    file_id                ; // handler fisier

integer    status                 ; // stare operatie cu fisiere

initial begin
    // initializeaza memoria sursa
    file_id = $fopen("input_data/img_256x128.raw", "rb");
    if (file_id == 0) begin
        $display("Nu se poate deschide fisierul img_256x128.raw");
        $stop;
    end
    for (idx=0; idx<(256*128); idx=idx+1)
        status = $fread(i_mem_src.mem[idx], file_id);
    $fclose(file_id);

end

```

11.1.6 Salvarea conținutului memoriei în fișier

De multe ori este necesară inspectarea conținutului unui memorii la un anumit timp de simulare. Sau, este nevoie de interpretarea datelor existente în memorie (de exemplu ca fiind o imagine în format **.raw**). Soluția constă din transferul conținutului memoriei într-un fișier binar. Fișierul se va deschide pentru scriere de tip binar (opțiunea „*wb* - write, binar”). Printr-o specificație `for` se poate copia în fișier fiecare locație de memorie, formatată cu opțiunea „%c” (caracter), cu funcția `$fwrite`.

```

integer    file_id                ; // handler fisier

initial begin

    // scrie memoria destinatie in fisier binar
    file_id = $fopen("output_data/img_32x32.raw", "wb");
    if (file_id == 0) begin
        $display("Nu se poate deschide fisier img_32x32.raw");
        $stop;
    end
    for (idx=0; idx<(32*32); idx=idx+1)
        $fwrite(file_id, "%c", i_mem_dst.mem[idx]);
    $fclose(file_id);

end

```

11.2 Desfășurarea lucrării

11.2.1 Testarea memoriei single-port

Realizați un mediu de testare pentru memoria single-port **mem_1rw**, similar celui prezentat în figura 11.4. Mediu conține două instanțe ale aceleiași memorii, cu parametrii diferiți. Valorile actuale ale parametrilor asociați celor două instanțe sunt prezentați în figura 11.4.

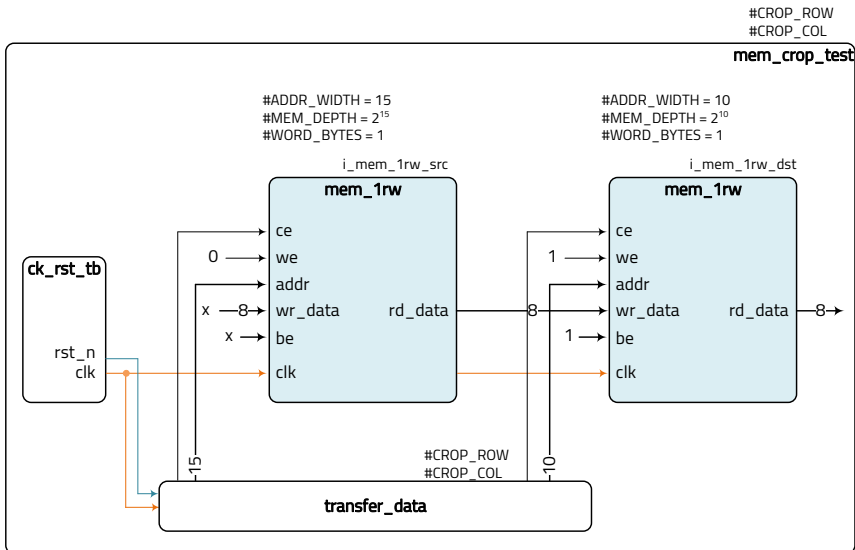


Figura 11.4 Mediu de testare pentru memorii single-port.

Prima instanță de memorie are capacitatea de memorare $2^{15} \times 8$ și poate stoca o imagine de dimensiune 256×128 , cu pixelii reprezentați pe 8 biți.

A doua instanță de memorie are capacitatea de memorare $2^{10} \times 8$ și poate stoca o imagine de dimensiune 32×32 , cu pixelii reprezentați pe 8 biți.

- Simulați mediul de testare furnizat în lecție (**mem_crop_test**) și observați structurile de date asociate memoriilor prin inspectarea ferestrelor *ModelSim*.
- Inspectați codul sursă al modului de test **mem_crop_test** și al generatorului de stimuli **transfer_data** și observați ce regiune din memoria sursă este transferată în memoria destinație.

11.2.2 Operațiuni cu fișiere

Mediul de testare **mem_crop_test** implementează câteva operațiuni cu fișiere:

- deschidere fișier cu `$fopen`,
- închidere fișier cu `$fclose`,
- inițializare variabilă de memorie cu `$fread`,
- scriere date din variabila de memorie în fișier cu `$fwrite`.

Identificați specificațiile de accesare a fișierelor în codul sursă. Identificați fișierele de intrare și de ieșire. Fișierele generate au extensii sugestive pentru tipul de date stocate: **.raw**, **.hex**, **.bin**.

- Fișierele cu extensia **.raw** conțin date care reprezintă imagini. Aceste fișiere pot fi deschise cu un editor de imagini, de exemplu **IrfanView** . Este necesară cunoașterea dimensiunii imaginii stocate.
- Fișiere cu extensia **.raw** pot fi deschise și cu un editor binar, de exemplu **HxD** .
- Fișierele cu extensia **.hex** sau **.bin** sunt fișiere cu caractere ASCII asociate valorilor numerice. Aceste fișiere pot fi deschise cu editoare de texte, de exemplu **Notepad++** .

Modulul **transfer_data** simulează mediul exterior memoriilor care comandă porturile acestora pentru a face transferul de date prin citirea memoriei sursă și transmiterea datelor spre a fi scrise în memoria destinație.

Memoria sursă, are dimensiune $2^{15} \times 8$ și conține o imagine de dimensiune 256×128 pixeli, 8bpp („bits per pixel”, engl.).

Memoria destinație, are dimensiune $2^{10} \times 8$ și poate stoca o imagine de dimensiune 32×32 pixeli, 8bpp.

Modulul **transfer_data** comandă decuparea unei porțiuni de imagine din memoria sursă și copierea acesteia în memoria destinație. Coordonatele colțului stânga-sus al zonei decupate sunt transmise modulului **transfer_data** ca parametrii:

```
parameter CROP_ROW    = 'd26      , // coordonate colt
parameter CROP_COL    = 'd122     //   in imagine sursa
```

Rulați simularea prin execuția fișierului **sim.do** și verificați formele de undă în fereastra *Wave*. **Sunt corecte?**

Verificați conținutul fișierelor din folderele de intrare și ieșire:

```
debug\sim\input_data și  
debug\sim\output_data.
```

Identificați poziția imaginii decupate în imaginea originală.

Este corectă?

11.2.3 Modificări asupra mediului de testare

- Observați cu atenție imaginea decupată. Există **imperfecțiuni** în partea din stânga. Cu se explică? Vizualizați cu atenție formele de undă și transferul primilor pixeli din memoria sursă în cea destinație. Reparați mediul de testare pentru a genera o imagine decupată **perfect**.
- Modificați mediul de testare pentru a decupa din imaginea sursă o porțiune de 32×32 pixeli aflați cu colțul din stânga sus la coordonatele (ROW=64, COL=188) și a depune imaginea rezultată în memoria destinație, cu colțul din stânga sus la coordonatele (0, 0).
- Modificați mediul de testare pentru a decupa din imaginea sursă o porțiune de 32×32 pixeli aflați cu colțul din stânga sus la coordonatele (ROW=26, COL=122) și a depune imaginea rezultată în memoria destinație, cu colțul din stânga sus la coordonatele (0, 0), dar rotită cu 90 de grade în sens trigonometric.
- Modificați mediul de testare pentru a decupa din imaginea sursă o porțiune de 64×64 pixeli aflați cu colțul din stânga sus la coordonatele (ROW=26, COL=122) și a depune imaginea rezultată în memoria destinație, micșorată cu factor cu $1/4$, adică 32×32 pixeli. Se vor citi din memoria sursă câte o zonă pătrată 2×2 pixeli, pentru fiecare pixel scris în memoria de destinație. Pixelul destinație va avea valoarea mediei aritmetice a celor 2×2 pixeli asociați.

Este nevoie modificarea mediului de testare **mem_crop_test** și a modului generator de stimuli **transfer_data** pentru ca datele citite din memoria sursă să fie trimise spre generatorul de stimuli pentru a fi procesate și ulterior rezultatul scris în memoria destinație.

Mediul de testare va fi similar celui prezentat în figura 11.5.

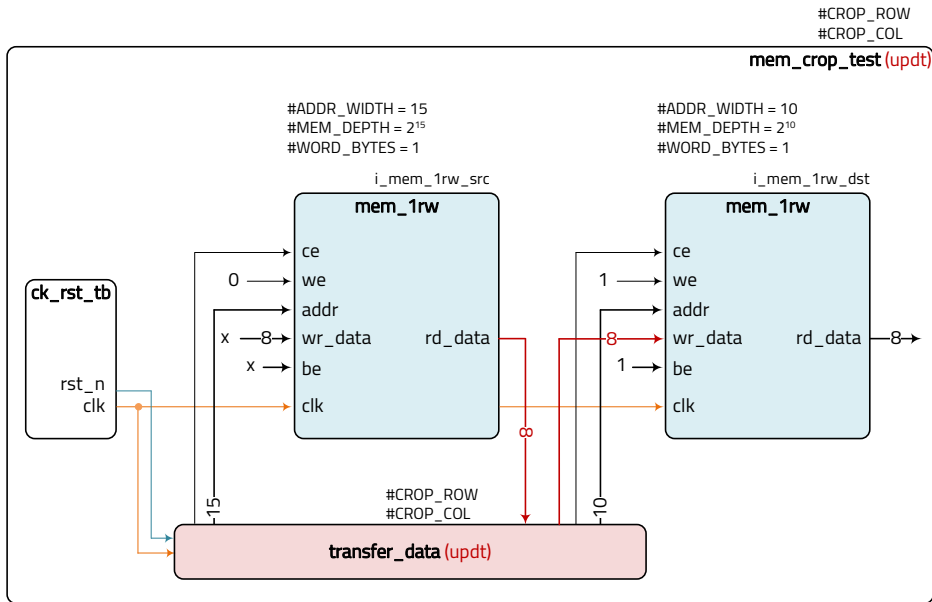


Figura 11.5 Mediu de testare modificat pentru prelucrarea datelor transferate între memorii.

11.3 Temă de casă

Numerele lui Fibonacci. Implementați un circuit care generează numărul lui Fibonacci asociat indexului primit la intrare. Numerele lui Fibonacci sunt definite astfel:

- Dacă $i = 0$, $F_0 = 0$,
- Dacă $i = 1$, $F_1 = 1$,
- Altfel, pentru $i \geq 2$, $F_i = F_{i-1} + F_{i-2}$.

Simbolul generatorului de numere Fibonacci este prezentat în figura 11.6.

Interfața modului `fib.v` este următoarea:

```
module fib (
input      clk      , // ceas (front pozitiv)
input      rst_n    , // reset asincron activ 0
```

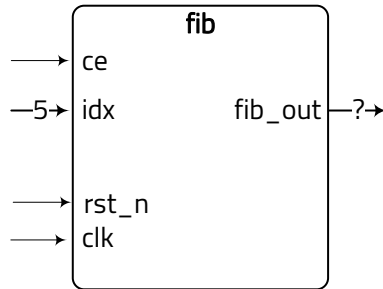


Figura 11.6 Simbolul generatorului pentru numerele lui Fibonacci.

```

input          ce          , // chip enable, activ 1
input [4 : 0] idx         , // index numar, in domeniul 0..20
output reg [? : 0] fib_out // numarul lui Fibonacci
                                // asociat indexului idx
);

// codul aici

endmodule // fib

```

Simultan cu `ce=1`, circuitul primește un număr pe intrarea `idx`. În următorul tact, modulul **fib** generează pe ieșirea `fib_out` numărul lui Fibonacci cu indexul `idx`. Indexul primit reprezintă un număr pozitiv între 0 și 20.

- Utilizați o memorie single-port **mem_1rw** inițializată dintr-un fișier binar extern. Memoria va simula o memorie ROM, accesibilă în sistem doar în citire (similar cu citirea dintr-un tabel, *LUT - Look Up Table*). Latența memoriei va fi de un singur tact.
- Generați conținutul fișierului extern prin export dintr-un tabel Excel.
- Determinați lățimea necesară (număr de biți) pentru ca datele de ieșire să poată reprezenta primele 20 de numere ale lui Fibonacci.
- Modificați modelul pentru a fi capabil să genereze primele 100 de numere ale lui Fibonacci. Ce trebuie schimbat?
- Explicați limitările programului Excel în a calcula corect numerele lui Fibonacci.

11.4 Fișiere cu modelul mediului de test

11.4.1 Modulul generatorului de stimuli

Acest paragraf prezintă codul Verilog care modelează modulul generator de stimuli, `transfer_data`. Se observă că acest modul are porturi de ieșire asociate intrărilor memoriilor testate.

```

module transfer_data #(
parameter CROP_ROW  = 'd26      , // coordonate colt
parameter CROP_COL  = 'd122     //   in imagine sursa
) (
input          clk          ,
input          rst_n        ,

output reg      ce_src      ,
output reg [15 -1: 0] addr_src ,
output reg      ce_dst      ,
output reg [10 -1: 0] addr_dst
);

reg [6 -1:0] row_src, row_dst; // index for
reg [6 -1:0] col_src, col_dst;

// citire memorie sursa
initial begin
    // initializeaza semnale si asteapta reset
    ce_src      <= 'bx;
    addr_src     <= 'bx;
    @(negedge rst_n);
    ce_src      <= 1'b0;
    @(posedge rst_n);
    @(posedge clk);

    ce_src <= 1'b1;
    for (row_src=0; row_src<32; row_src=row_src+1)
        for (col_src=0; col_src<32; col_src=col_src+1) begin
            addr_src <= (row_src+CROP_ROW)*256+(col_src+CROP_COL);
            @(posedge clk);
        end
    ce_src <= 1'b0;
    @(posedge clk);

end

// scriere memorie destinatie
initial begin
    ce_dst      <= 'bx;

```

```

    addr_dst      <= 'bx;
    @(negedge rst_n);
    ce_src        <= 1'b0;
    ce_dst        <= 1'b0;
    @(posedge rst_n);

    @(posedge clk);
    ce_dst <= 1'b1;
    for (row_dst=0; row_dst<32; row_dst=row_dst+1)
        for (col_dst=0; col_dst<32; col_dst=col_dst+1) begin
            addr_dst <= row_dst*32+col_dst; // pozitie normala
            @(posedge clk);
        end
    ce_dst <= 1'b0;
    @(posedge clk);

end

endmodule // transfer_data

```

11.4.2 Modulul mediului de testare

Acest paragraf prezintă codul Verilog care modelează modulul cu mediul de testare, `mem_crop_test`. Se observă structura acestui modul, conform mediului prezentat în figura 11.5.

```

module mem_crop_test;
    parameter CROP_ROW  = 'd26    ; // coordonate colt
    parameter CROP_COL  = 'd122   ; // in imagine sursa

    wire      clk          ;
    wire      rst_n        ;

    integer   file_id      ; // handler fisier

    integer   idx           ;
    integer   row           ;
    integer   col           ;

    integer   status        ; // stare operatie cu fisiere

    wire      ce_src        ;
    wire [14 : 0] addr_src  ;
    wire      ce_dst        ;
    wire [9  : 0] addr_dst  ;
    wire [7  : 0] data_intern ;

```

```

ck_rst_tb #(
    .CK_SEMIPERIOD ('d10)
) i_ck_rst_tb (
    .clk      (clk      ),
    .rst_n    (rst_n    )
);

initial begin
    // initializeaza memoria sursa
    file_id   = $fopen("input_data/img_256x128.raw", "rb");
    if (file_id == 0) begin
        $display("Nu se poate deschide fisierul img_256x128.raw");
        $stop;
    end
    for (idx=0; idx<(256*128); idx=idx+1)
        status = $fread(i_mem_src.mem[idx], file_id);
    $fclose(file_id);

    repeat (32*32+10) @(posedge clk); // asteapta transferul

    // scrie memoria destinatie in fisier binar
    file_id   = $fopen("output_data/img_32x32.raw", "wb");
    if (file_id == 0) begin
        $display("Nu se poate deschide fisier img_32x32.raw");
        $stop;
    end
    for (idx=0; idx<(32*32); idx=idx+1)
        $fwrite(file_id, "%c", i_mem_dst.mem[idx]);
    $fclose(file_id);

    // scrie memoria destinatie in fisier text, hexa
    file_id   = $fopen("output_data/img_32x32.hex", "w");
    if (file_id == 0) begin
        $display("Nu se poate deschide fisier img_32x32.hex");
        $stop;
    end
    for (idx=0; idx<(32*32); idx=idx+1)
        $fwrite(file_id, "%h \n", i_mem_dst.mem[idx]);
    $fclose(file_id);

    // scrie memoria destinatie in fisier text, binar
    file_id   = $fopen("output_data/img_32x32.bin", "w");
    if (file_id == 0) begin
        $display("Nu se poate deschide fisier img_32x32.bin");
        $stop;
    end
    for (idx=0; idx<(32*32); idx=idx+1)
        $fwrite(file_id, "%b \n", i_mem_dst.mem[idx]);
    $fclose(file_id);

```

```

    $display ("%M %0t INFO: Final simulare.", $time);
    $stop;
end

// memorie sursa, accesibila doar in citire
mem_lrw #(
    .ADDR_WIDTH (15          ),
    .MEM_DEPTH  (1 << 15     ),
    .WORD_BYTES (1           ) // latimea datelor 8 biti
) i_mem_src(                // 2^15 x 8 (256x128 bytes)
    .clk      (clk           ),
    .ce       (ce_src        ),
    .we       (1'b0          ),
    .addr     (addr_src       ),
    .wr_data   (8'bx          ),
    .be       (1'bx          ),
    .rd_data   (data_intern   )
);

// memorie destinatie, accesibila doar in scriere
mem_lrw #(
    .ADDR_WIDTH (10          ),
    .MEM_DEPTH  (1 << 10     ),
    .WORD_BYTES (1           ) // latimea datelor 8 biti
) i_mem_dst(                // 2^10 x 8 (32x32 bytes)
    .clk      (clk           ),
    .ce       (ce_dst        ),
    .we       (1'b1          ),
    .addr     (addr_dst       ),
    .wr_data   (data_intern   ),
    .be       (1'b1          ),
    .rd_data   (              )
);

transfer_data #(
    .CROP_ROW (CROP_ROW),
    .CROP_COL (CROP_COL)
) i_transfer_data (
    .clk      (clk           ),
    .rst_n    (rst_n         ),
    .ce_src   (ce_src        ),
    .addr_src (addr_src),
    .ce_dst   (ce_dst        ),
    .addr_dst (addr_dst)
);

endmodule // mem_crop_test

```

Lucrarea 12

Modelarea sistemelor digitale complexe

Această lucrare propune studenților realizarea a trei exerciții de proiectare și modelare a unor sisteme digitale mai complexe, apropiate de complexitatea sistemelor reale.

- **Multiplicator secvențial**, modelat pe baza arhitecturii RTL ¹ proiectate de un alt inginer, cu interfață *request-acknowledge*.
- **Circuit watchdog**, a cărui funcționare este prezentată prin forme de undă ale semnalelor generate.
- **Circuit pipeline pentru determinarea maximului dintre 3 date**, modelat pe baza unor specificații sumare, cu interfețe *valid-ready*.

Modelarea celor trei circuite nu se poate finaliza în două ore. Însă, studenții interesați au opțiunea alocării unui timp suplimentar pentru rezolvarea acestor subiecte.

Pentru toate cele trei proiecte propuse se recomandă și utilizarea



pentru a genera codul Verilog care modelează circuitele cerute sau mediul de testare pentru acestea.

Atentie! Informațiile generate de ChatGPT trebuie să fie verificate deoarece nu sunt garantate a fi corecte.

Desigur, pentru proiectare este extrem de încurajată utilizarea inteligenței naturale a viitorului inginer.

¹RTL = Register Transfer Level = nivel de detaliere care evidențiază transferurile de date între registre, engl.

12.1 Multiplicator secvențial

Circuitul de multiplicare secvențială, cu interfață *request-acknowledge* are simbolul bloc prezentat în figura 12.1.

Algoritmul propus pentru implementare este cel de adunare repetată. Algoritmul se bazează pe observația că produsul a două numere se poate obține prin adunarea primului operand cu el însuși de un număr de ori egal cu valoarea celui de-al doilea operand.

Multiplicatorul are intrări pentru semnalul de ceas (activ pe frontul creșcător) și pentru semnalul de reset asincron, activ în 0.

Interfața multiplicatorului respectă protocolul *request-acknowledge*. Simultan cu cererea, din exterior sunt prezentați cei doi operanzi, de dimensiune parametrizabilă DW (*Data Width*). Simultan cu confirmarea, multiplicatorul livrează spre exterior rezultatul, de dimensiune parametrizabilă $2 \times DW$.

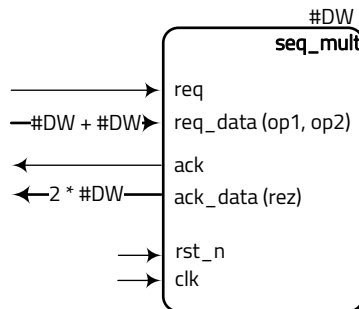


Figura 12.1 Simbol bloc pentru multiplicatorul secvențial cu interfață *request-acknowledge*.

Implementarea hardware a algoritmului este realizată de circuitul prezentat în figura 12.2.

În schemă sunt prezente porți logice și registre. Elementele secvențiale sunt simbolizate cu dreptunghiuri marcate cu simbol de ceas.

Registru care determină ieșirea *ack_data* are funcționalitatea descrisă de următorul tabel:

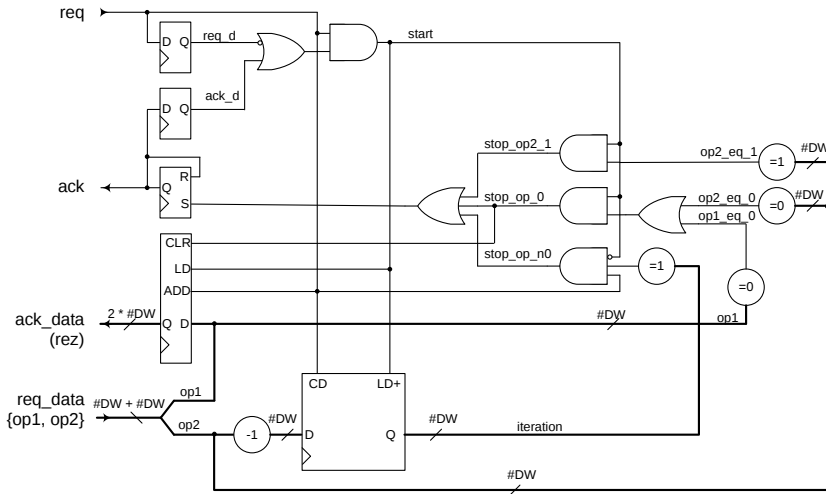


Figura 12.2 Arhitectura multiplicatorului secvențial (porți și bistabile, RTL).

CLR	LD	ADD	Q^{++}	Funcție
1	-	-	0	Inițializare cu 0
0	1	-	D	Încărcare cu D
0	0	1	$Q + D$	Acumulare D
0	0	0	Q	Păstrează starea

Numărătorul de iterații are funcționalitatea descrisă de următorul tabel:

LD	CD	Q^{++}	Funcție
1	-	D	Încărcare cu D
0	1	$Q - 1$	Numărare în sens descrescător
0	0	Q	Păstrează starea

op1 și op2 provin din intrarea req_data, op1 fiind partea mai semnificativă.

Figura 12.3 prezintă formele de undă așteptate pentru multiplicatorul secvențial cu protocol *request-acknowledge*. Sunt prezentate cazuri particulare de operanzi egali cu 0, sau cu 1 și cazul unei cereri „back-to-back”. Porțiunile hașurate desemnează semnale de valori indifferente/nerelevante.

Modelul multiplicatorului secvențial are următoarea interfață:

```
module seq_mul #(
```

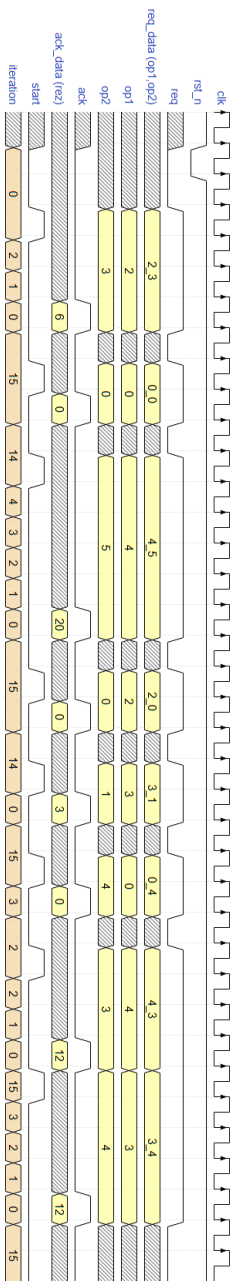


Figura 12.3 Forme de undă așteptate pentru multiplicatorul secvențial cu interfață request-acknowledge.

```

parameter DW = 'd4 // Data width, numar de biti ai operandilor
)(
input          clk          , // semnal de ceas,
input          rst_n        , // reset asincron activ 0
input          req          , // cerere
output reg     ack          , // confirmare, puls activ 1
input          [DW+DW-1:0] req_data , // {op1, op2}
output reg     [2*DW-1:0] ack_data  // rezultat
);

// codul aici

endmodule // seq_mul

```

Modelați circuitul multiplicator secvențial, pe baza arhitecturii prezentate în figura 12.2, având interfața prezentată. Modelul va conține câte o specificație `always` pentru fiecare registru (sau bistabil). Circuitele combinaționale care generează semnalele prezente în schemă se vor modela cu specificație `assign`.

Pentru depanarea modelului de multiplicator secvențial este necesar un mediu de testare. Mediul va instanția următoarele componente:

- instanța multiplicatorului testat **seq_mul**, cu parametrul *DW* preluat din valoarea parametrului cu același nume al mediului de verificare;
- instanța modulului **ck_rst_tb** care generează semnalele de ceas și de reset;
- instanța modulului **checker_rq_ack** care verifică protocolul *request-acknowledge*.

În modelul de top, se va defini un `task` utilizat pentru trimiterea de comenzi spre multiplicator și verificarea datelor. Task-ul acceptă ca intrări două valori asociate operandilor înmulțirii și un număr reprezentând intervalul de timp (exprimat în perioade de ceas) până la următoarea comandă. Dacă această ultimă valoare este zero, comanda ce va urma se va lansa back-to-back.

Verificarea rezultatelor se va face prin compararea valorii așteptate (calculate în task ca produs al operandilor lansați pentru multiplicator) cu valoarea produsă de multiplicator. Compararea se va face în momentul primirii confirmării (`ack=1`).

```

task send_req;
input [DW-1:0] op1      ; // date transmise
input [DW-1:0] op2      ; // date transmise

```

```

input [32-1:0] delay_p ; // pauza intre cereri succesive
begin
    // activeaza cererea
    req      <= 1'b1;
    req_data <= {op1, op2};
    @(posedge clk);
    // asteapta confirmarea
    while (~ack) @(posedge clk);
    // verifica datele
    if (ack_data !== (op1*op2)) begin
        $display ("%M %0t EROARE %0d * %0d = %0d, asteptat %0d.",
            $time, op1, op2, ack_data, (op1*op2));
        $stop;
    end
    else begin
        $display ("%M %0t CORECT %0d * %0d = %0d.", $time,
            op1, op2, ack_data);
    end
    // asteapta pauza
    if (delay_p != 'd0) begin
        req      <= 1'b0;
        req_data <= 'bx;
        repeat (delay_p) @(posedge clk);
    end
end
endtask

```

Apelul task-ului se face prin numele task-ului și valori pentru intrări, între paranteze rotunde și despărțite prin virgulă. De exemplu:

```

// send_req(op1, op2, pauza);

$display ("%M %0t INFO: Cereri pentru depanare.", $time);
send_req(2, 3, 1);
send_req(0, 0, 1);
send_req(4, 5, 1);
send_req(2, 0, 1);
send_req(3, 1, 1);
send_req(0, 4, 1);
send_req(4, 3, 0);
send_req(3, 4, 10);

```

Apelați task-ul cu parametrii determinați, astfel încât să verificați cazurile externe („corner cases”, engl.):

- operatori egali cu 0 (unul sau ambii);
- operatori egali cu 1 (unul sau ambii);
- operatori egali cu valoarea maximă (unul sau ambii);

- operatori de parități diferite.

Verificați cazurile menționate în figura 12.4.

Apelați task-ul de multiple ori, cu parametrii aleatorii pentru verificarea intensivă a modului.

Apelați task-ul pentru a verifica în mod determinist toate cazurile pentru valorile operatorilor modului de multiplicare.

Verificați modelul multiplicatorului secvențial cu diverse valori ale parametrului DW (de exemplu 8 și 16). Modificați valoarea doar în mediul de testare. Valoarea din preambulul mediului de testare se propagă la instanța modelului de verificat.

Verificați modelul pentru $DW = 8$ și $DW = 16$.

Evaluați timpul de rulare a verificărilor pentru toate cazurile posibile, $2^{2 \times DW}$.

Utilizați  ChatGPT pentru a genera codul Verilog care modelează circuitul multiplicator secvențial. Oferiți o descriere textuală și interfața modului.

O fi bun modelul generat? Vă ajută? Este scris în mod profesionist?

Verificați codul generat de ChatGPT în mediul de verificare furnizat la laborator.

Este corect?

Este respectat protocolul *request-acknowledge*?

12.2 Circuit watchdog

Circuitul watchdog este utilizat în sistemele integrate cu microprocesor („*embedded systems*”, engl.) pentru a reinițializa sistemul, în ipoteza blocării acestuia datorită unor situații neprevăzute sau periferice care nu răspund la cereri conform așteptărilor microprocesorului.

Circuitul watchdog generează un semnal de întrerupere către procesor în cazul în care nu a primit un mesaj de la acesta, într-un interval prestabilit de timp.

Circuitul se programează cu o valoare ce reprezintă numărul de perioade de ceas pe care le va număra (intrarea `delay_value` eșantionată când `set_delay=1`). Valoarea prestabilită poate fi modificată de circuitul exterior numai când circuitul watchdog este inactiv (`enable=0`).

Simplu spus, după ce este activat (`enable=1`), circuitul numără în sens

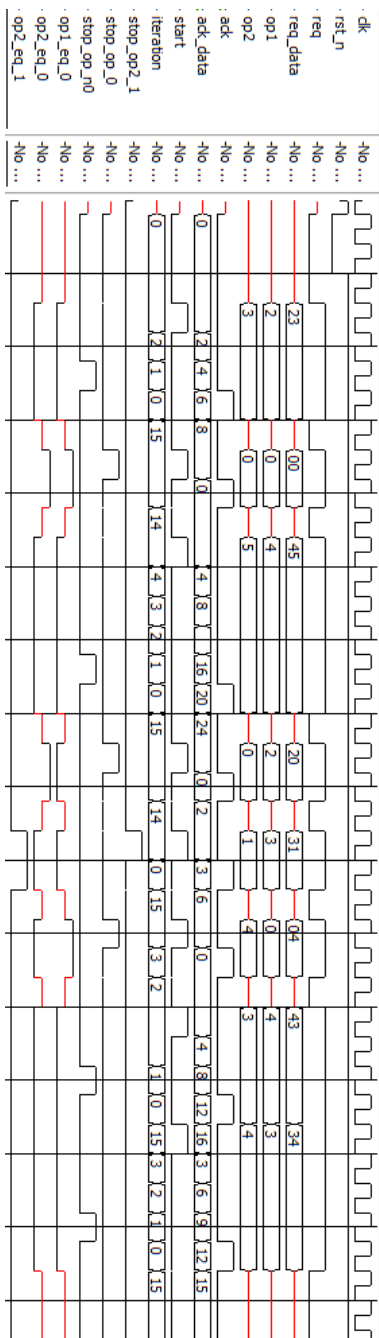


Figura 12.4 Forme de undă rezultate din simulare pentru multiplicatorul secvențial cu interfață request-acknowledge.

descrescător de la valoarea prestabilită. Dacă în cursul numărării apare un puls de declanșare (`trigger=1`), numărarea este reluată de la valoarea maximă prestabilită. Dacă numărătorul ajunge la 0, se setează întreruperea (`intr=1`).

Odată setată întreruperea, numărătorul așteaptă inactivarea circuitului (`enable=0`). La inactivarea funcționării, circuitul watchdog își resetează ieșirea de întrerupere (`intr=0`).

Circuitul își realizează funcția doar când este activat (`enable=1`). Dacă este dezactivat în timpul numărării, circuitul abandonează numărarea și așteaptă o nouă activare pentru a relua numărarea în sens descrescător de la valoarea prestabilită.

Simbolul circuitului watchdog este prezentat în figura 12.5.

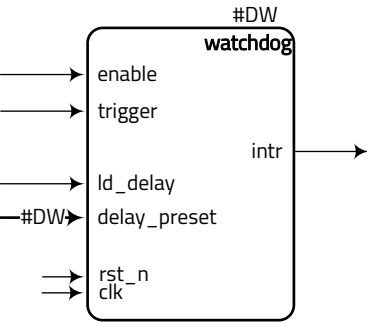


Figura 12.5 Simbolul circuitului watchdog.

Interfața circuitului watchdog este prezentată în tabelele următoare.

Nume param.	Val.	Dome- niu	Descriere
<i>DW</i>	8	≥ 1	Dimensiunea numărătorului intern (în biți)

Nume port	Dir.	Dim.	Descriere
clk	IN	1	Semnal de ceas
rst_n	IN	1	Semnal asincron de reset, activ 0
enable	IN	1	Validare circuit, activ 1
trigger	IN	1	Semnal puls de triggerare, activ 1
set_delay	IN	1	Încărcare date, activ 1, exclusiv când enable=0
delay_value	IN	DW	Valoare maximă de numărare, relevantă exclusiv când enable=0 și set_delay=1
intr	OUT	1	Ieșire de întrerupere, setată la finalizare numărare, resetată la dezactivare

Figura 12.6 prezintă forme de undă caracteristice pentru circuitul watchdog. Se observă încărcarea valorii prestabilite `delay_value=7`, cu comanda `set_delay=1`, lansată în timp ce circuitul nu este activat `enable=0`. Cu excepția acestui moment, valoarea `delay_value` este indiferentă. Apoi circuitul este activat `enable=1`. La activare, începe numărarea internă în sens descrescător, de la valoarea prestabilită (7). În cursul numărării, apar două pulsuri de triggerare (`trigger=1`) care determină reluarea numărării de la valoarea maximă prestabilită (7). După o numărare completă (valoarea numărătorului intern ajunge la zero), numărarea se oprește și se activează întreruperea `intr=1`. Întreruperea este dezactivată în momentul dezactivării circuitului `enable=0`.

O nouă activare a circuitului, `enable=1`, determină reluarea numărării în sens descrescător de la valoarea maximă prestabilită (7).

Modelați circuitul watchdog timer, pe baza descrierii prezentate în tabele și a formelor de undă din figura 12.6.

Modelați în Verilog un mediu de testare pentru model, care să genereze semnalele de intrare (`enable`, `trigger`, `set_delay` și `delay_value`).



ChatGPT

Utilizați pentru a genera codul Verilog care modelează circuitul watchdog. Oferiți doar tabelul cu porturile modulului și funcția acestuia: „*watchdog timer*”.

Verificați modelul RTL generat de ChatGPT în mediul de testare descris de voi.

Ulterior, cereți ChatGPT să genereze un mediu de testare pentru watchdog timer.

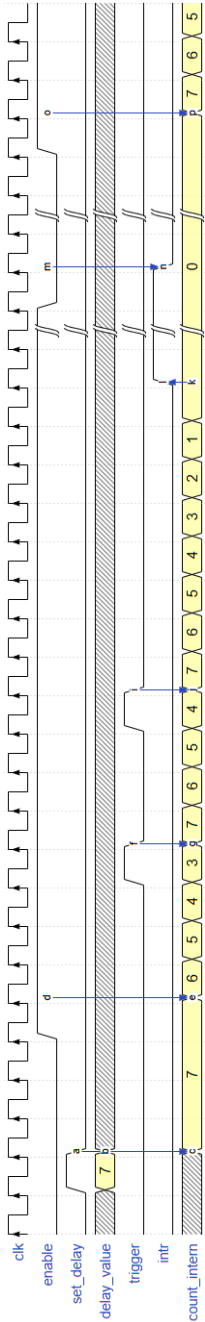


Figura 12.6 Forme de undă care caracterizează circuitul watchdog.

12.3 Circuit pipeline pentru determinarea maximului dintre 3 date

Circuitul primește pe o interfață cu protocol *valid-ready* un set de 3 date ce reprezintă numere pozitive. Circuitul calculează maximul celor 3 numere și îl transmite pe o altă interfață, cu același protocol *valid-ready*.

Numărul de biți de date este parametrizabil DW , astfel încât numerele pozitive sunt în intervalul

$$[0 \dots 2^{DW} - 1]$$

Determinarea numărului maxim se va face în pipe-line, cu un algoritm cât mai eficient, ales de proiectant. În condițiile în care modulele exterioare sunt disponibile tot timpul ($\text{max_rdy}=1$), iar datele de intrare se primesc continuu ($\text{client_val}=1$) atunci datele de ieșire se livrează în mod continuu, cu o latență egală cu numărul de nivele de pipeline din acest modul.

Simbolul circuitului care determină maximul este prezentat în figura 12.7.

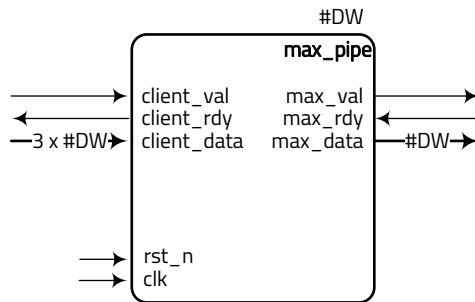


Figura 12.7 Simbolul circuitului pentru determinarea maximului a 3 date, implementare pipeline.

Interfața circuitului care determină maximul este prezentată în tabelele următoare.

Nume param.	Val.	Domeniu	Descriere
DW	8	≥ 1	Dimensiunea datelor (în biți)

Nume port	Dir.	Dim.	Descriere
clk	IN	1	Semnal de ceas
rst_n	IN	1	Semnal asincron de reset, activ 0
client_val	IN	1	Valid din partea clientului
client_rdy	OUT	1	Ready pentru client
client_data	IN	$3 \times DW$	Date de la client, interpretate ca 3 date de câte DW biți.
max_val	OUT	1	Date valide livrate
max_rdy	IN	1	Modulul receptor este pregătit să accepte date
max_data	OUT	DW	Date reprezentând maximul, DW biți.

Atentie!

Protocolul *valid-ready* impune `ready=1` când pot fi primite date, indiferent de valoarea `valid`. În ipoteza unui algoritm de determinare a maximului prin 2 comparații (câte una pe ciclu), circuitul are latență de 2 tacte. Inițial `client_rdy=1`. Dacă se primesc succesiv 2 date (`client_val=1`), trebuie resetat `client_rdy=0` pentru a nu se accepta datele, în ipoteza `max_rdy=0`.

Problema se poate rezolva combinational. Însă, logica de determinare a maximului se va adăuga la logica existentă în alte module și frecvența maximă va fi scăzută.

```

module max3_0 #(
parameter DW = 'd4 // Data width, numar de biti ai operanzilor
) (
input          clk          , // semnal de ceas,
input          rst_n        , // reset asincron activ 0
// interfata client
input          client_val   ,
output         client_rdy   ,
input [3*DW-1:0] client_data , // 3 date de cate DW biti
// interfata maxim
output         max_val      , //
input          max_rdy      , //
output [DW-1:0] max_data    // maximul, DW biti
);

wire [DW-1:0] data0 ;
wire [DW-1:0] data1 ;
wire [DW-1:0] data2 ;
wire [DW-1:0] max_0_1 ;

assign data0 = client_data[ 0+:DW];
assign data1 = client_data[ DW+:DW];

```

```

assign data2 = client_data[2*DW+:DW];

assign max_val    = client_val;
assign client_rdy = max_rdy    ;
assign max_0_1    = data0 > data1 ? data0 : data1;
assign max_data   = max_0_1 > data2 ? max_0_1 : data2;

endmodule // max3_0

```

Timingul este cel prezentat în figura 12.8. Se observă conectarea directă a semnalelor de control de pe cele două interfețe:

```

max_val = client_val și
client_rdy = max_rdy.

```

Logica de determinare a maximului este combinațională.

O alternativă o constituie implementarea cu latență egală cu un tact. Logica de determinare a maximului se realizează într-un tact, însă va exista un tact latență într-acele două interfețe (logica se trece printr-un registru). Beneficiul așteptat este creșterea frecvenței maxime de operare.

Timingul este cel prezentat în figura 12.9. Se observă că datele se propagă cu o latență de un tact.

O alternativă și mai bună o constituie împărțirea logicii de determinare a maximului pe două perioade de tact. De exemplu, implementarea unei comparații într-un tact și a comparației dintre maximul primelor două date și cea de-a treia dată în al doilea tact.

Un scenariu de forme de undă care caracterizează circuitul de determinare a maximului a 3 date, cu latență de 2 tacte, este prezentat în figura 12.10.

- La momentul 1, clientul depune prima cerere, acceptată la momentul 2.
- Pentru că modulul este disponibil (`client_rdy=1`) sunt acceptate 2 cereri back-to-back la momentele 2 și 3. Ulterior circuitul nu mai poate accepta mai multe date (deja are 2 seturi) și resetează `client_rdy=0`.
- Primele date sunt disponibile la ieșire la momentul 3 și acceptate la momentul 5. Există o relație combinațională între `client_rdy` și `max_rdy`. Ca efect, `client_rdy=1` la momentul 4, simultan cu `max_rdy=1`, datele fiind preluate de interfața `max` la momentul 5.
- Datele acceptate la momentele 3 și 6 sunt transmise la ieșire cu 2 tacte întârziere, pentru că precedentele nu s-au acceptat, deoarece modulul conectat la interfața `max` nu a fost disponibil un tact.
- Se observă că dacă `max_rdy=1` și `client_val=1`, după momentul 7, datele sunt acceptate și transmise la fiecare ciclu, obținându-se o cadență

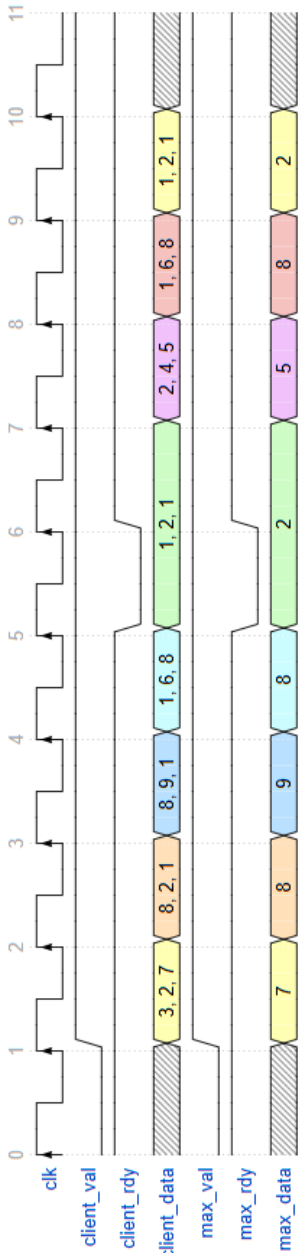


Figura 12.8 Timing pentru circuitul de determinare a maximului, implementarea combinatională, cu latență = 0 .

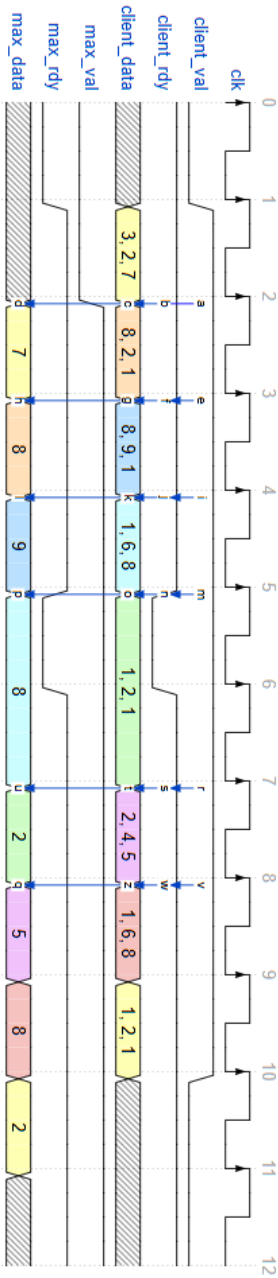


Figura 12.9 Timing pentru circuitul de determinare a maximumului, implementarea cu latență = 1.

de procesare continuă de o dată pe tact. Latența datelor prin modul este de 2 tacte.

Modelați circuitul pentru determinarea maximului în cele 3 variante: cu latențe 0, 1 și 2.

Testați modelele în același mediu de testare furnizat. Modelul de test instanțiază câte un modul verificator al protocolului *valid-ready* pe fiecare din cele două interfețe.

```
module vld_rdy_checker #(
parameter      DW  = 8      // Data width;
)
(
input          clk          ,
input          valid        ,
input          ready        ,
input [DW - 1: 0] data
);
```

Mediul de test, prezentat în continuare, are și facilitatea de a scrie în fișiere datele transmise pe cele două interfețe. Prin inspectarea și compararea fișierelor, se poate face depanarea modelului RTL.

```
module max3_test #(
parameter CK_SEMIPERIOD = 10 ,
parameter DW            = 8   // Data width;
);

wire          clk          ;
wire          rst_n        ;
reg           client_val   ;
wire          client_rdy   ;
reg [3*DW - 1:0] client_data ;

wire          max_val      ;
reg           max_rdy      ;
wire [DW - 1:0] max_data    ;

ck_rst_tb #(
  .CK_SEMIPERIOD (CK_SEMIPERIOD)
) i_ck_rst_tb (
  .clk          (clk          ),
  .rst_n        (rst_n        )
);

//max3_2 #(
//max3_1 #(
max3_0 #()
```

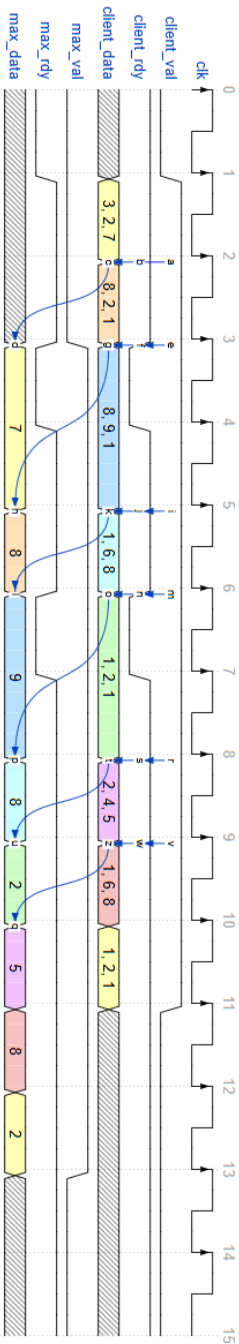


Figura 12.10 Timing pentru circuitul de determinare a maximumului, implementarea cu latență = 2.

```

.DW (DW)
) i_max3(
.clk      (clk      ),
.rst_n    (rst_n    ),
.client_val (client_val ),
.client_rdy (client_rdy ),
.client_data (client_data),
.max_val   (max_val   ),
.max_rdy   (max_rdy   ),
.max_data  (max_data  )
);

vld_rdy_checker #(
.DW (3*DW )
) client_vld_rdy_checker (
.clk      (clk      ),
.valid     (client_val ),
.ready     (client_rdy ),
.data      (client_data )
);

vld_rdy_checker #(
.DW (DW )
) max_vld_rdy_checker (
.clk      (clk      ),
.valid     (max_val   ),
.ready     (max_rdy   ),
.data      (max_data  )
);

/// client initiator
initial begin
    // asteptare reset asincron
    client_val  <= 1'bx;
    client_data <= 'bx;
    @(negedge rst_n);
    client_val  <= 1'b0;
    @(posedge rst_n);
    @(posedge clk);

    forever begin
        if ($urandom_range(0,4)<4) begin
            client_val  <= 1'b1;
            client_data <= $urandom_range(0, (1<<3*DW));
            @(posedge clk);
            while (~client_rdy) @(posedge clk);
        end
        else begin
            client_val  <= 1'b0;

```

```

        client_data <= 'bx;
        @(posedge clk);
    end
end
end

// client max
initial begin
    max_rdy <= 1'bx;
    @(negedge rst_n);
    max_rdy <= 1'b0;
    @(posedge rst_n);
    @(posedge clk);

    forever begin
        max_rdy <= ($urandom_range(0,10)<10);
        @(posedge clk);
    end
end

integer    file_src_id  ; // handler fisier
integer    file_dst_id  ; // handler fisier

initial begin
    // src
    file_src_id  = $fopen("output_data/src.txt", "w");
    if (file_src_id == 0) begin
        $display("Nu se poate deschide fisier src.txt");
        $stop;
    end
    forever begin
        if (client_val & client_rdy)
            $fwrite(file_src_id,"%h \n", client_data);
        @(posedge clk);
    end
end

initial begin
    // dst
    file_dst_id  = $fopen("output_data/dst.txt", "w");
    if (file_dst_id == 0) begin
        $display("Nu se poate deschide fisier dst.txt");
        $stop;
    end
    forever begin
        if (max_val & max_rdy)
            $fwrite(file_dst_id,"%h \n", max_data);
        @(posedge clk);
    end
end

```

end

endmodule // max3_test

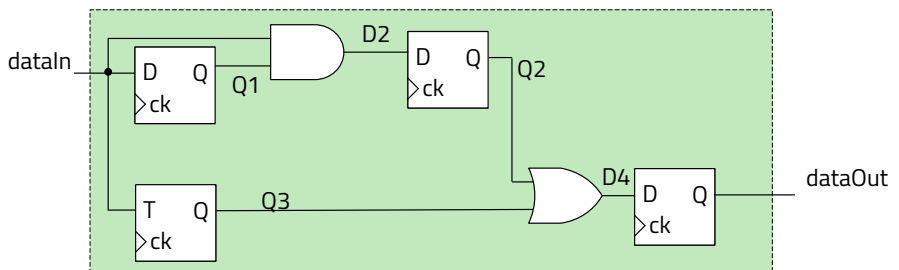
Anexa 13

Subiecte pentru colocviul de laborator

Anexa prezintă un exemplu de subiect pentru colocviul de laborator cu rezolvarea asociată.

Exemplu

Se consideră circuitul cu porți logice și bistabile.



Cerințe

- Explicații referitoare la funcția și utilizarea circuitului. Justificări.
- Forme de undă ale intrărilor, semnalelor intermediare și ieșirilor așteptate (desen și cod [WaveDrom](#)).
- Model Verilog RTL, cu comentarii.
- Mediu de testare Verilog, cu comentarii.
- Explicații și justificări referitoare la funcția și comportamentul circuitului.

Soluție

- Pentru intrarea *dataIn* s-au considerat pulsuri de diferite lățimi.
- Forma de undă a semnalului *Q1* este identică cu cea a *dataIn*, întârziată cu un ciclu de ceas.
- $D2 = Q1 \cdot dataIn$. Se poate interpreta că *D2* este egală cu 1 dacă *dataIn* a fost egal cu 1 pentru două perioade succesive.
- Forma de undă a semnalului *Q2* este identică cu *D2*, dar întârziată cu un ciclu de ceas (bistabil D).
- Forma de undă a semnalului *Q3* este obținută ca ieșire a unui bistabil T pe a cărui intrare este conectat semnalul *dataIn*: dacă *dataIn* = 1, *Q3* își completează valoarea, dacă *dataIn* = 0, *Q3* își păstrează valoarea.
- $D4 = Q2 + Q3$ (SAU logic).
- *dataOut* este întârziat cu ciclu față de *D4*.

Formele de undă ale circuitului sunt prezentate în figura 13.1.

Atenție! Formele de undă conțin două greșeli **intenționate**, cu efecte propagate.

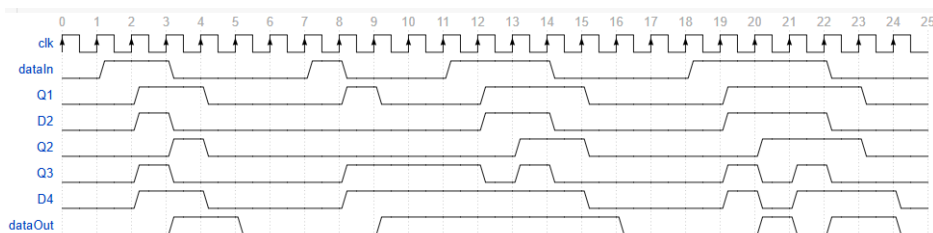


Figura 13.1 Forme de undă așteptate. **Atenție!** Formele de undă conțin două greșeli **intenționate**, cu efecte propagate.

Modelul RTL Verilog este prezentat în continuare.

```

module colocviu_0 (
input      clk      , // semnal de ceas,
input      rst_n    , // reset asincron activ 0
input      dataIn   , // intrare
output reg dataOut   // iesire
);

reg  Q1 ;
reg  Q2 ;
reg  Q3 ;
wire D2 ;
wire D4 ;

always @(posedge clk or negedge rst_n)
if (~rst_n) Q1 <= 1'b0; else
    Q1 <= dataIn;

assign D2 = dataIn & Q1;

always @(posedge clk or negedge rst_n)
if (~rst_n) Q2 <= 1'b0; else
    Q2 <= D2;

always @(posedge clk or negedge rst_n)
if (~rst_n)  Q3 <= 1'b0; else
if (dataIn)  Q3 <= ~Q3;

assign D4 = Q2 | Q3;

always @(posedge clk or negedge rst_n)
if (~rst_n) dataOut <= 1'b0; else
    dataOut <= D4;

endmodule // colocviu_0

```

Mediu de testare Verilog prezintă instanțierea circuitului proiectat și generatoare de stimuli pentru intrarea *dataIn*.

```

module colocviu_0_test;

wire  clk      ;
wire  rst_n    ;

reg   dataIn   ; // intrare
wire  dataOut  ; // iesire

ck_rst_tb #(
    .CK_SEMIPERIOD (10)
) i_ck_rst_tb (

```

```

.clk      (clk      ),
.rst_n    (rst_n    )
);

colocviu_0 i_colocviu_0(
.clk      (clk      ),
.rst_n    (rst_n    ),
.dataIn   (dataIn   ),
.dataOut  (dataOut  )
);

initial begin
    dataIn  <= 'bx;
    @(negedge rst_n);
    dataIn  <= 'b0;
    @(posedge rst_n);
    @(posedge clk);

    dataIn  <= 'b1;
    @(posedge clk);
    @(posedge clk);
    dataIn  <= 'b0;
    repeat (4)@(posedge clk);
    dataIn  <= 'b1;
    @(posedge clk);
    dataIn  <= 'b0;
    repeat (3)@(posedge clk);
    dataIn  <= 'b1;
    repeat (3)@(posedge clk);
    dataIn  <= 'b0;
    repeat (4)@(posedge clk);
    dataIn  <= 'b1;
    repeat (4)@(posedge clk);
    dataIn  <= 'b0;
    repeat (6)@(posedge clk);

    $display ("%M INFO: Terminare simulare.");
    $stop;
end

endmodule // colocviu_0_test

```

Formele de undă rezultate din simulare (figura 13.2), confirmă așteptările și evidențiază cele două greșeli din figura 13.1.

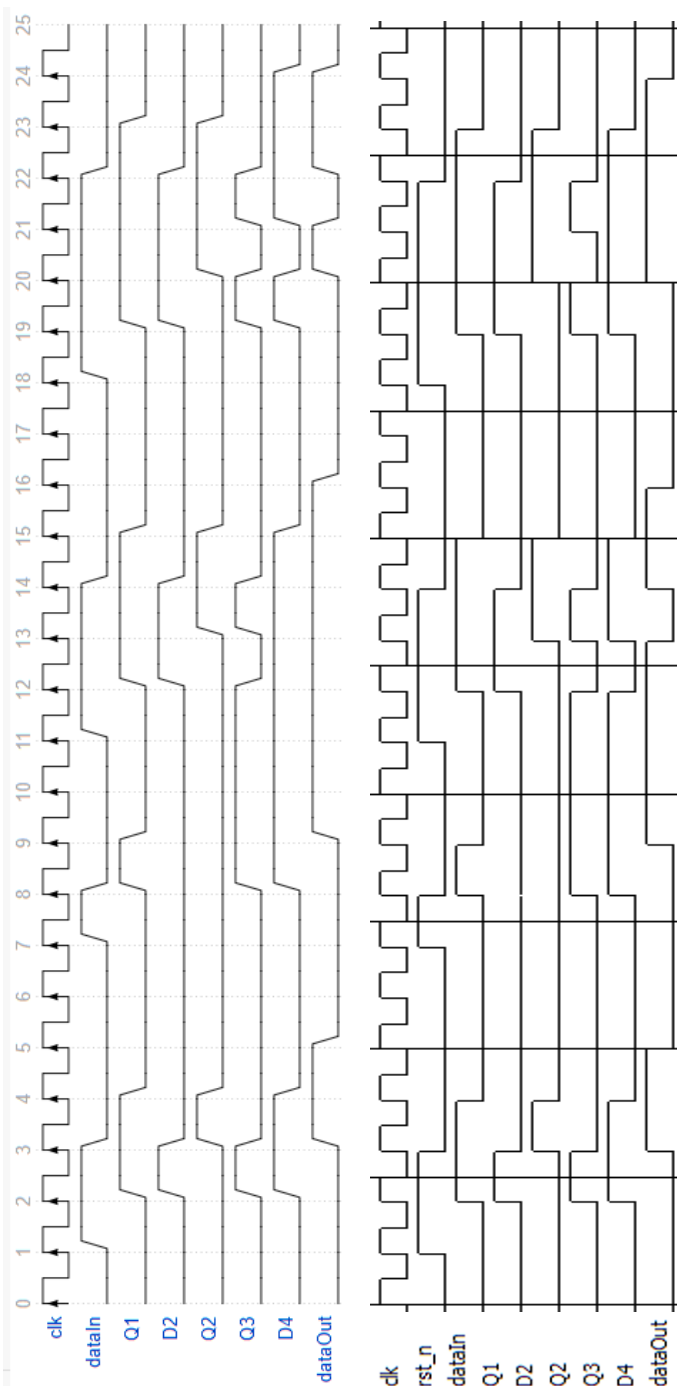
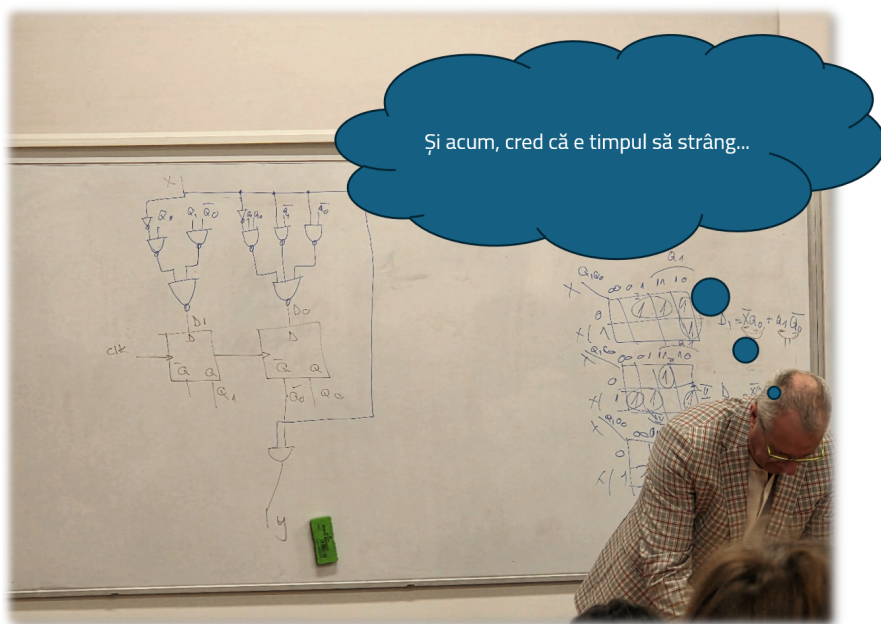


Figura 13.2 Forme de undă așteptate (sus, greșite) și simulate (jos, corecte) pentru circuitul propus.

Grăirea de încheiere

Într-o perioadă în care performanța profesorilor este măsurată
exclusiv prin numărul de articole publicate,
dăruiesc generațiilor viitoare o

„Carte de învățătură”.



ELECTRONICĂ DIGITALĂ

CARTE DE ÎNVĂȚĂTURĂ

TOMUL PENTRU MEȘTERIT ȘI MUNCIT DAN NICULA

📖 Această carte face parte din vestita serie de autor „Carte de învățatură de Electronică Digitală”, zămislită cu trudă 🧐, gând adânc 😊 și un pic de curent electric ⚡.

Întrupată în anul de grație 2025 🗓️, ediția de față vine cu rânduială nouă 📅, cu gânduri bine înșiruite 📖 și cu o chibzuială mai strânsă între toate cele ce țin de tainica lume a electronicii digitale 🔍. Se vrea sprijin pentru orice suflet studios 🧑🏫, fie că-i la curs 📚, la seminar 🗣️, la laborator 🧪 sau în colțul camerei 🏠, cu laptopul în brațe 💻 și cafeaua în a treia ceașcă ☕☕☕.

👤 Cartea se adresează celor din neamurile:

- Electronicii Aplicate 🔧 – cu mintea în scheme 📐 și sufletul în osciloscop 📖,
- Telecomunicațiilor 📡 – care dau cu semnalul peste mări și zări 🌐,
- Calculatoarelor 🖨️ – ce dezbat biți și bytes până la răsărit 🌅,
- Automaticii 🤖 – ce leagă lumea în bucle cu feedback 🔄,
- Tehnologiei Informației 🌐 – unde datele se poartă cu fast digital 🚀.

📖 Întreg cuprinsul este împărțit în patru tomuri, fiecare cu rostul său:

1. 📖 **Tomul pentru zăbovit și gândit** – suportul de curs, plin cu noțiuni teoretice și cu modelări în limba Verilog 📄💡, pentru cei ce vor să știe nu doar ce, ci și de ce.
2. 🧑🏫 **Tomul pentru chibzuit și buchisit** – pentru seminarii și proiecte de făcut cu creionul 🖋️, guma 🟢 și ceva răbdare de arici în iarnă 🐿️❄️.
3. 🔧 **Tomul pentru meșterit și muncit** – unde se pun mâinile la treabă 🙌, firele la locul lor 🧑🔧 și logica în practică 🧠⚙️.
4. ⌚ **Tomul pentru grabă mare** – pentru zilele cu examen 📄 sau uitare 😊🗨️, cu chestionare sprintene 🏃🏻💨, ce-ți arată ce știi și ce-ai mai bine să repeți degrabă 📖.

💡 Așadar, oricine voiește să învețe cu haz 😊, cu rost 🗣️ și cu plăcere 💖 despre firele care gândesc 🧠, porturile care vorbesc 🗣️ și circuitele care își amintesc 📖 – să deschidă această carte fără teamă 📖.

👉 Căci nu mușcă 🦷 — doar luminează. 💡🌟

Dan NICULA este profesor la Universitatea Transilvania din Brașov.
www.dannicula.ro



Editura
Universității
Transilvania
din Brașov

ISBN 978-606-19-1794-5



9 786061 917945