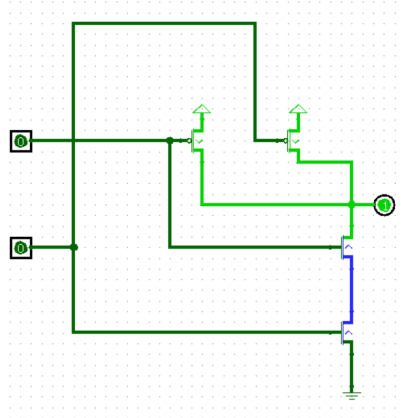
Proiect VLSI

Dumitrache Adrian-George, 342C1

R1. Explicați funcționarea porții NAND 2:

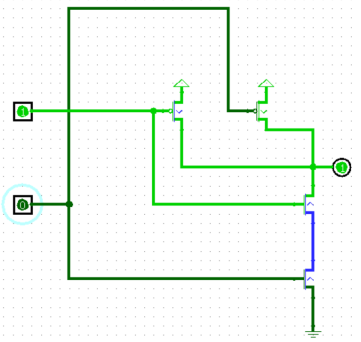


a) in1 = 0, in2 = 0

Cei 2 tranzistori PMOS vor fi deschisi, formand o cale de la alimentare (sursa celor 2 tranzistori) la iesire.

Cei 2 tranzistori NMOS vor fi inchisi, deci iesirea nu va avea cale la masa.

Concluzie: **out = 1**



b) in1 = 1, in2 = 0

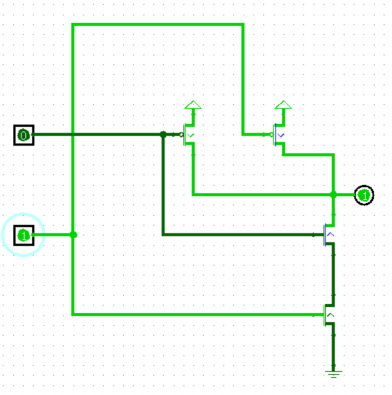
PMOS-ul conectat la in2 va fi deschis.
PMOS-ul conectat la in1 va fi inchis.

Cum cei 2 tranzistori PMOS sunt conectati in paralel, vom avea in continuare cale de la alimentare la iesire.

NMOS-ul conectat la in1 va fi deschis.
NMOS-ul conectat la in2 va fi inchis.

Dar cei 2 tranzistori NMOS sunt in serie, deci nu vom avea cale intre iesire si masa.

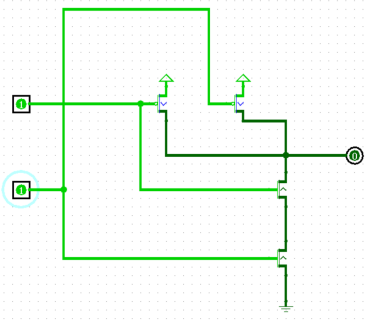
Concluzie: **out = 1**



c) in1 = 0, in2 = 1

Situatie similara cu cazul anterior, doar se interschimba care tranzistor e deschis in ambele perechi.

Concluzie: **out = 1**

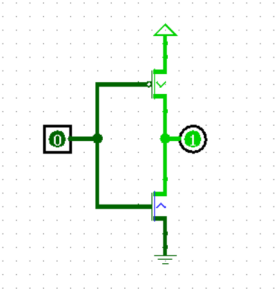


d) in1 = 1, in2 = 0

Tranzistorii PMOS vor fi inchisi, deci nu vom avea cale de la alimentare la iesire.
Tranzistorii NMOS vor fi deschisi, deci vom avea cale de la masa la iesire.

Concluzie: **out = 0**

R2. Desenați și explicați funcționarea unei porți NOT in cele 2 implementari (PMOS si NMOS in serie; NAND)

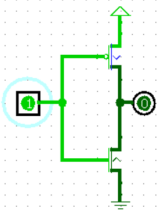


a) Implementare PMOS/NMOS in serie

Pentru intrare = 0:

- PMOS-ul este deschis => exista cale intre alimentare si iesire
- NMOS-ul este inchis => nu exista cale intre masa si iesire

Concluzie: **iesirea va fi 1**



Pentru intrare = 1:

- PMOS-ul este inchis => nu exista cale intre alimentare si iesire
- NMOS-ul este deschis=> exista cale intre masa si iesire

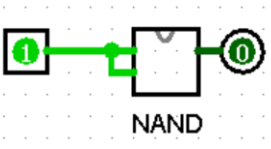
Concluzie: **iesirea va fi 0**

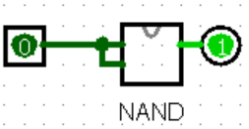
in	out
0	1
1	0

b) Implementare NAND

Observam din tabela de adevar a portii NAND (implementata anterior) ca pentru intrari egale (in1 = in2), poarta se comporta identic cu o poarta NOT. Astfel, putem lega o intrare la ambii pini ai portii pentru a obtine o poarta NOT.

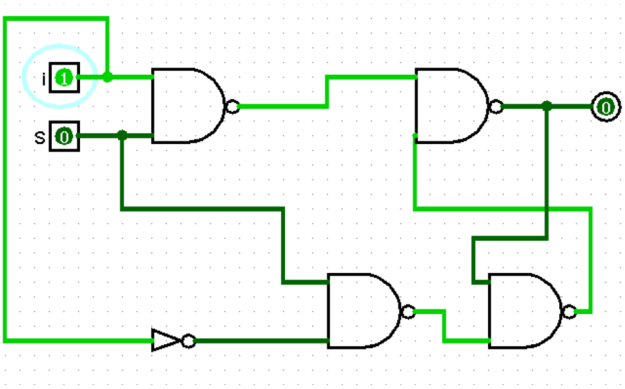
in1	in 2	out
0	0	1
0	1	1
1	0	1
1	1	0





R3) Demonstrați funcționarea unui registru de 1 bit

Schema initiala poate fi redusa la 5 NAND-uri, unul dintre ele fiind folosit ca un NOT, astfel, obtinem schema alaturata echivalenta pentru a simplifica circuitul.



Daca S = 1, primul NAND din etajul superior va intoarce !i, iar primul NAND din etajul inferior va intoarce i.

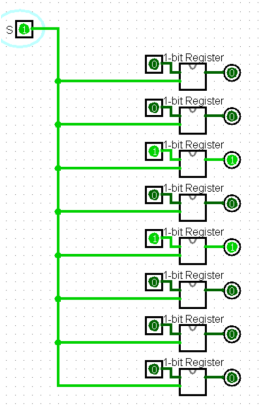
Data fiind o stare initiala S = 1, i = 1 si output = 1, schimbarea valorii lui i la 0 afecteaza circuitul in felul urmator:

- etajul inferior:
  - primul NAND isi schimba valoarea de la 1 la 0
  - al doilea NAND are in acest moment intrarile 0 de la primul NAND si 1 de la iesirea initiala al circuitului
  - deci al doilea NAND isi schimba valoarea de la 0 la 1
- etajul superior:
  - primul NAND isi schimba valoarea de la 0 la 1
  - al doilea NAND are in acest moment intrarile 1 de la primul NAND si 1 de la iesirea etajului inferior
  - deci iesirea circuitului este acum 0, valoare ce este propagata catre al doilea NAND din etajul inferior

Analog pentru cazul S = 1, i = 0 si output = 0.

Daca S ar fi fost 0, primul pas al fiecarui etaj nu s-ar fi intamplat, deci nu s-ar mai fi declansat scrierea, iar feedback loop-ul ar fi pastrat in continuare valoarea de iesire stabila

R4) Se consideră intrarea 00101000. Care este ieșirea unui registru pe 8 biți dacă S = 1 ? Dar dacă S = 0 ?



a) S = 1

Iesirea va urmari intrarea, adica vom avea 00101000 la iesire.

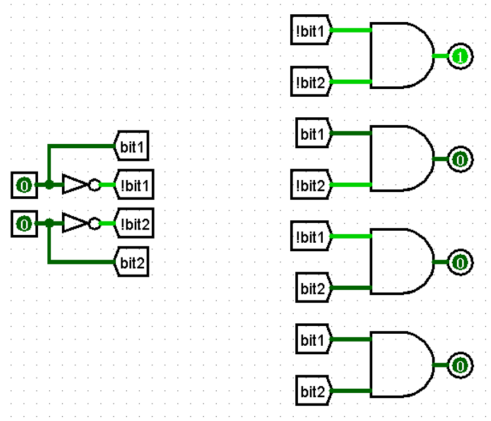
b) S = 0

Daca S = 0, output-ul este determinat in intregime de valorile ce se aflau in registru inainte ca valoarea lui S sa fie schimbata la 0. Deci putem avea in registru orice valoare (inclusiv intrarea de input, daca acesta era input-ul si cand S = 1).

R5) Din punct de vedere de funcționare se va demonstra că S = 1 doar dacă EN și CLK = 1, altfel S = 0

Cum valorile registrului se pot schimba doar daca EN = 1 (daca circuitul este pornit) si daca ne aflam pe un front pozitiv al ceasului (deci CLK = 1), S nu poate fi alta valoarea decat 1 (adica starea in care putem memora valori), iar daca circuitul e oprit sau nu suntem pe frontul pozitiv al ceasului, atunci S = 0 (deci valoarea de output ramane la fel indiferent de input).

R6) Explicați funcționarea unui DEC 2x4



Observam ca circuitul original este format din 10 NAND-uri aranjate in 3 coloane.

Prima coloana este formata din 2 NAND-uri folosite ca inversoare. De asemenea, NAND-urile din a 3-a coloana sunt folosite ca inversoare pentru iesirile NAND-urilor din a 2-a coloana. Astfel, cele 2 coloane formeaza AND-uri.

Aceste transformari ne duc la aceasta schema simplicata, in care se poate observa mai usor rolul componentelor, fiecare AND verifica una dintre posibilele intrari.

Exemple:
in1 = 0, in2 = 0 => primul AND va intoarce 1 deoarece !in1 & !in2 = 1, restul AND-urilor vor intoarce 0
in1 = 1, in2 = 0 => al doilea AND va intoarce 1 deoarece in1 & !in2 = 1, restul intorc 0
in1 = 0, in2 = 1 => al treilea NAND va intoarce 1 deoarece !in 1 & in2 = 1, restul intorc 0
in1 = 1, in2 = 1 => al patrulea NAND va intoarce 1 deoarece in 1 & in2 = 1, restul intorc 0

R7) Descrieți realizarea operației de scriere în memorie

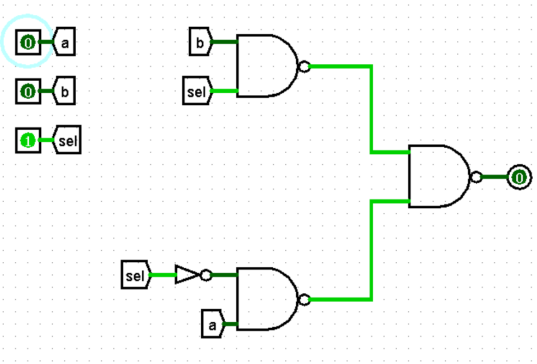
Procesorul pune adresa de scriere pe magistrala de adrese, deoarece legarea la o celula de memorie a acestei adrese utilizand un singur decodificator necesita un decodificator complex (16 x 64k !), utilizam doua decodificatoare 8 x 256. Unul determina linia celulei, celalalt coloana adresei.

Luand fiecare iesire a primului decodificator cu fiecare iesire a celui de al doilea, putem determina corect carei celule de memorie sa ii trimitem semnalul de EN, lucru ce va activa scrierea datelor aflate pe magistrala de date in celula de memorie corecta.

R8) Descrieți realizarea operației de citire din memorie

Modulul de memorie primeste o adresa pe magistrala din care trebuie sa citeasca, o parte din adresa (aici 2 biti) este trimisa catre un decodificator ce selecteaza linia adresei (adica selecteaza niste celule de memorie, aici 2). Ceilalti biti (aici 1) sunt folositi ca semnal de EN/SEL pentru a selecta o coloana din aceasta linie de memorie (adica celula de memorie ceruta)..

R9) Descrieți operarea circuitului MUX 2:1



La fel ca in intrebarile precedente, putem reduce circuitul la cateva porti logic de baza precum in figura alaturata.

Astfel, daca sel = 1, atunci valoarea NAND-ului de jos va fi intotdeauna 1, iar valoarea NAND-ului de sus va fi mereu !b. Cum 1 NAND !b = b, sel = 1 garanteaza ca iesirea va fi mereu egala cu b.

Analog, daca sel = 0, NAND-ul de sus va avea mereu valoarea 1, iar NAND-ul de jos va avea mereu valoarea !a. Cum 1 NAND !a = a, sel = 0 garanteaza ca iesire va fi mereu egala cu a.

R10) Descrieți modalitatea de realizare a sumei dintre valorile 3 și 4

Stocam valoarea 3 in registrul A:

- punem in MAR adresa unde se afla valoarea 3 si activam semnalul LD
- selectam valoarea data_out utilizand MUX-ul si o punem pe magistrala
- trimitem semnal de enable la registrul A, stocand valoarea aflata pe magistrala

Analog pentru valoarea 4 si registrul B.

Daca ALU-ul primeste semnalul CIN = 0, atunci va intoarce suma celor doua numere, valoarea ce poate fi selectata de catre MUX pentru a fi stocata in alt registru sau scrisa in memorie.

R11) Implementați restul instrucțiunilor dorite a fi implementate

a) LDA value

Cicluri de executie:
1. A <- IR
2. CLC <- 1

A <- IR

0 010 0010 - 0x22
1 010 0010 - 0xA2

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	111	0	0	0	1	0	0	0	0	0	0	000	0	0

Deci in celulele 22 si A2 vom stoca 0x38800.

CLC <- 1

0 011 0010 - 0x32
1 011 0010 - 0xB2

Vom stoca 0x00002 in aceste celule.

b) **MOV B, A**

Cicluri de executie:
B <- A
CLC <- 1

B <- A

0 010 0011 - 0x23
1 010 0011 - 0xA3

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	010	0	0	0	0	1	0	0	0	0	0	000	0	0

Deci in celulele 23 si A3 vom avea valoarea 0x10400.

CLC <- 1

0 011 0011 - 0x33
1 011 0011 - 0xB3

Vom stoca 0x00002 in aceste celule.

c) **MOV C, A**

Cicluri de executie:
C <- A
CLC <- 1

C <- A

0 010 0100 - 0x24
1 010 0100 - 0xA4

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	010	0	0	0	0	0	1	0	0	0	0	000	0	0

Deci in celulele 24 si A4 vom avea valoarea 0x10200.

CLC <- 1

0 011 0100 - 0x34
1 011 0100 - 0xB4

Vom stoca 0x00002 in aceste celule.

d) **MOV A, B**

Cicluri de executie:
A <- B
CLC <- 1

A <- B

0 010 0101 - 0x25
1 010 0101 - 0xA5

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	011	0	0	0	1	0	0	0	0	0	0	000	0	0

Deci in celulele 25 si A5 vom avea valoarea 0x18800.

CLC <- 1

0 011 0101 - 0x35
1 011 0101 - 0xB5

Vom stoca 0x00002 in aceste celule.

e) **MOV A, C**

Cicluri de executie:
A <- C
CLC <- 1

A <- C

0 010 0110 - 0x26
1 010 0110 - 0xA6

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	100	0	0	0	1	0	0	0	0	0	0	000	0	0

Deci in celulele 26 si A6 vom avea valoarea 0x20800.

CLC <- 1

0 011 0110 - 0x36
1 011 0110 - 0xB6

Vom stoca 0x00002 in aceste celule.

f) **MOV OUT, A**

Cicluri de executie:
OUT <- A
CLC <- 1

OUT <- A

0 010 0111 - 0x27
1 010 0111 - 0xA7

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	010	0	0	0	0	0	0	0	1	0	0	000	0	0

Deci in celulele 27 si A7 vom avea valoarea 0x10080.

CLC <- 1

0 011 0111- 0x37
1 011 0111 - 0xB7

Vom stoca 0x00002 in aceste celule.

g) **STR A**

Cicluri de executie:
MAR <- A
[MAR] <- C
CLC <- 1

MAR <- A

0 010 1000 - 0x28
1 010 1000 - 0xA8

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	010	1	0	0	0	0	0	0	0	0	0	000	0	0

Deci in celulele 28 si A8 vom avea valoarea 0x1C000.

[MAR] <- C

0 011 1000 - 0x38
1 011 1000 - 0xB8

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
1	0	100	0	0	0	0	0	0	0	0	0	0	000	0	0

Deci in celulele 38 si B8 vom avea valoarea 0xA0000.

CLC <- 1

0 100 1000 - 0x48
1 100 1000 - 0xC8

Vom stoca 0x00002 in aceste celule.

h) **STR value**

Cicluri de executie:
MAR <- A
[MAR] <- IR
CLC <- 1

MAR <- A

0 010 1001 - 0x29
1 010 1001 - 0xA9

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	010	1	0	0	0	0	0	0	0	0	0	000	0	0

Deci in celulele 29 si A9 vom avea valoarea 0x14000.

[MAR] <- IR

0 011 1001 - 0x39
1 011 1001 - 0xB9

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
1	0	111	0	0	0	0	0	0	0	0	0	0	000	0	0

Deci in celulele 39 si B9 vom avea valoarea 0xB8000.

CLC <- 1

0 100 1001 - 0x49
1 100 1001 - 0xC9

Vom stoca 0x00002 in aceste celule.

j) **ADD (C <- A + B)**

Cicluri de executie:
C <- ALU
CLC <- 1

C <- ALU

0 010 1010 - 0x2A
1 010 1010 - 0xAA

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	110	0	0	0	0	0	1	0	0	0	1	000	0	0

Deci in celulele 2A si AA vom avea valoarea 0x30220.

CLC <- 1

0 011 1010 - 0x3A
1 011 1010 - 0xBA

Vom stoca 0x00002 in aceste celule.

k) **SUB (C <- A - B)**

Cicluri de executie:
C <- ALU
CLC <- 1

C <- ALU

0 010 1011 - 0x2B
1 010 1011 - 0xAB

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	110	0	0	0	0	0	1	0	0	0	1	010	0	0

Deci in celulele 2B si AB vom avea valoarea 0x30228.

CLC <- 1

0 011 1011 - 0x3B
1 011 1011 - 0xBB

Vom stoca 0x00002 in aceste celule.

l) **JMP addr**

Cicluri de executie:
PC <- IR
CLC <- 1

PC <- IR

0 010 1100 - 0x2C
1 010 1100 - 0xAC

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	111	0	0	1	0	0	0	0	0	0	0	000	0	0

Deci in celulele 2C si AC vom avea valoarea 0x39000.

CLC <- 1

0 011 1100 - 0x3C
1 011 1100 - 0xBC

Vom stoca 0x00002 in aceste celule.

m) **JC addr**

Cicluri de executie:
PC <- IR (optional, doar in cazul in care avem carry)
CLC <- 1

PC <- IR

1 010 1101 - 0xAD

STR	LD	BUS	MAR	COUNT	LDCOUNT	A	B	C	IN	OUT	IR	FLAGS	OPC	CLC	X
0	0	111	0	0	1	0	0	0	0	0	0	0	000	0	0

Deci in celula AD vom avea valoarea 0x39000.

CLC <- 1

0 010 1101 - 0x2D
0 011 1101 - 0x3D
1 011 1101 - 0xBD

Vom stoca 0x00002 in aceste celule.

R12) Implementați un program ales

Pseudocod (program care aduna doua numere si stocheaza in memorie 0 pentru adunare cu carry si 1 in caz contrar):

a = [addr1]
b = [addr2]
c = a + b
[addr3] = 0
if c < 255 [addr3] = 1
while 1 ; loop infinit

Instructiuni:

```
; stocam in registrele a si b numerele ce trebuie adunate
LDA [0x0f]
MOV B, A
LDA [0x0e]

; adunare a si b
ADD C <- A + B

; stocam in a adresa unde vrem sa punem rezultatul si stocam raspunsul in caz ca avem
; carry
LDA 0x0d
STR 0

; daca avem carry sarim direct la finalul programului, loop-ul infinit
JC 08

; altfel, stocam rezultatul 1, neavand carry
STR 1

; infinite loop
NOP
jmp 8
```

Memorie:

0 1f	1 30	2 1e	3 a0
4 2d	5 90	6 d8	7 91
8 00	9 c8	a	b
c	d	e valoare	f valoare