

NÉV:.....**Minta ZH megoldás**..... Neptun kód:.....

A feladatokat önállóan, meg nem engedett segédeszközök használata nélkül oldottam meg:

Olvasható aláírás:.....

Kedves Kolléga! **A kitöltést a név és aláírás rovatokkal kezdje!** Az alábbi kérdésekre a válaszokat - ahol lehet - mindig a feladatlapon adja meg! A feladatok megoldása során a részletes kidolgozást nagyfeladatonként, ha szükséges külön papíron végezze, (egyértelműen jelölje, hogy melyik lap melyik feladathoz tartozik, a papírra már a kezdetkor írja rá a nevét és Neptun kódját) és ezeket a papírokat is adja be a dolgozatával! A kérdésekre a táblázatok vagy a pontozott vonalak értelemszerű kitöltésével válaszoljon, ha csak külön másként nem kérjük. **Mindenütt a legegyszerűbb megoldás éri a legtöbb pontot.** Jó munkát!

E: 27
F1:13
F2:18
F3:17
Σ 75
IMSC2:5
IMSC3:6

Ellenőrző kérdések (27p)

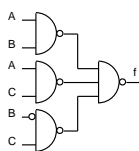
E1. (3p) Konvertálja az alábbi számokat a kért formátumra!

kiinduló formátum	konvertálandó szám	kért formátum	konvertált szám
2 digités decimális	59	8 biten BCD	0101 1001
6 bites 2-es komplement	100001	8 bites 2-es komplement	11100001
decimális	6.25	5 bites fix pontos előjel nélküli bináris, 2 db kettedes tört jeggyel	110 01

E2. (1p) Egyszerűsítse az alábbi Boole kifejezést!

$$/A + AB + BC = (/A+A)/(A+B)+BC = /A+B+BC = **/A+B**.....$$

E3. (4p) Töltse ki az alábbi kapcsolás rajzzal megadott f logikai függvény igazságtábláját!



A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0

A	B	C	f
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

E4. (3p) Igazságtáblájával adott az f logikai függvény.

a. Adja meg a függvény diszjunktív normál alakját (DNF tehát ne egyszerűsítsen)!

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1

A	B	C	f
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

f_{DNF} = ..**/A&B&C | A&~B&C | A&B&~C | A&B&C**;..

b. Milyen ismert funkcionális elem mely kimenetét állítja elő az f függvény?

.....**Teljes összeadó Co kimenete**.....

E5. (3p) A mellékelt blokkvázlatról hiányoznak a I0, I1, I2, I3 jelek. Adja meg ezeket úgy, hogy 4 bites törölhető, tölthető, engedélyezhető, balra shiftelő shiftregisztert valósítson meg!

MIMO:	00	01	10	11
funkció	tart	tölt	balra shiftel	töröl

I0:.....**Q**..... I1:.....**D**..... I2:.....**{Q[2:0], S1}**... I3: **4'h0**.....

E6. (2p) A MiniRISC CPU Éppen a 0x10 címen található **jsr** 0x08 utasítást hajtja végre. Az utasítás execute fázisában **IRQ** =1-et érzékel és elfogadja az interrupt kérést.

a. Milyen címről olvassa a következő utasítást? ...**0x01**.....

b. Milyen címre ugrik az **rti** utasítás hatására?.....**0x08**.....

E7. (3p) Egészítse ki a jobb oldali rajzot a következők szerint.

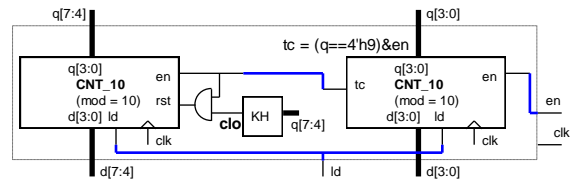
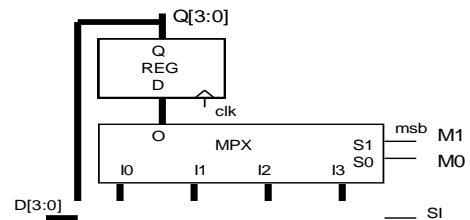
a. Csökkentse a bal oldali számláló modulusát 6-ra! Adja meg Verilog nyelven az ehhez szükséges clo kimenetet előállító logiát!

assign clo = q == 4'd5; vagy q[2]&~q[1]&q[0] vagy q[2]&q[0] (utolsó mert csak felfele számol)

b. Egészítse ki a rajzot úgy, hogy a két számláló együtt egy 60-as modulusú (0,1...59, 0...) engedélyezhető, tölthető számlálót valósítson meg!

E8. (3p) Válaszoljon a következő MiniRISC GPIO perifériával kapcsolatos kérdésekre! A periféria ADO regiszterébe 0x00-t írunk, a periféria lábain tápra húzó ellenállások vannak elhelyezve, kívülről semmi nem kapcsolódik rá. (Többlet információk a GPIO perifériáról az F3 feladat elején található.)

- A periféria biteit kimenetként állítva milyen logikai értéket lehet olvasni az ADI regiszteréből? **0x00**....
- A periféria biteit bemenetként állítva milyen logikai értéket lehet olvasni az ADI regiszteréből? **0xff**....
- Hány MiniRISC órajel alatt lehet a GPIO kimenetét megváltoztatni? **6 (2 utasítás 2x3 órajel)**....



E10. (5p) Mely állítások igazak és melyek hamisak? Jelölje **+ -al az igaz, - -al a hamis** állításokat!

1.	Csak AND kapukkal, OR kapukkal és inverterekkel minden logikai függvény előállítható.	+
2.	LOAD-STORE architektúrájú processzor esetén egy művelet eredménye csak regiszterben keletkezhet.	+
3.	Két UART összekapcsolásánál az azonos elnevezésű jeleket kell egymással összekötni.	-
4.	A 2 regiszter címes processzor architektúra több bites utasítás kód igényel, mint a 3 regiszter címes.	-
5.	Az egyszintű interrupt rendszerekben egy interrupt megszakíthatja egy másik végrehajtását.	-

Feladatok:

F1. (13p) Adott egy FSM az alábbi **kódozott állapotgráffival**. A kimenete az állapotkód és z. Végezze el az alábbi feladatokat! **Az állapot kódok helyett mindenütt állapot neveket használjon!**

a. (3p) Adja meg az állapotregiszter Verilog leírását!
//A konstans és változó deklarációk
 localparam [3:0] A = 3'b0001, B = 3'b0000,
 C = 3'b0010, D = 3'b0100, E = 3'b1000;
 reg [3:0] s, next_s;
 wire clk, rst;
 wire [1:0] x;
//Állapotregiszter
 always @ (posedge clk)
 if (rst) s <=A.....;
 else s <= ...next_s...;

b. (7p) Következő állapot logika
//Next_state logika
 always @ (.....*.....)
 case (s)
 A: if (~x[1] & ~x[0]) next_s <=A.....;
 else next_s <=C.....;
 B: case (.....x[1:0].....)
 2'b00: next_s <=A.....;
 2'b01: next_s <=C.....;
 2'b10: next_s <=D.....;
 2'b11: next_s <=E.....;
 endcase
 C: next_s <=D.....;
 D: next_s <=E.....;
 E: if (~x[1] & ~x[0]) next_s <=B.....;
 else next_s <=A.....;
 default: next_s <=A.....;
 endcase

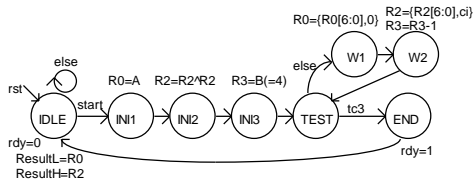
c. (1p) Milyen modell szerint működik? (Húzza alá a megfelelőt) Mealy Moore

d. (2p) Adja meg a z kimenet logikai függvényét Verilog-ban!
A legegyszerűbb SOP alakban: assign z = ~s[2] & ~s[1] & ~s[0];
Ha csak az s-t, az állapot neveket, az == operátort és | & műveletet használhatja:
 assign z =(s == B) | (s == E);.....

F2. (18p) Funkcionális blokkvázlatával (adatstruktúra + vezérlő) adott egy a bemenő adatokkal többféle művelet elvégzésére alkalmas számító modul. Az adatstruktúra **3 db 8 bites adat bemenettel** rendelkezik: **A, B, C** (előjel nélküli számok). Az aritmetikai logikai egység (ALU) 8 műveletet tud elvégezni az **opa** és **opb** operandusokkal az alább megadott táblázat szerint. Az **elvégzendő művelet** az ALU **op[2:0]** bemenetén állítható be. **Shifelés esetén a belépő bit 0, ha eci = 0, ci, ha eci = 1. Összeadás/kivonás esetén a ci-t csak akkor veszi figyelembe, ha eci = 1. A művelet eredményét az ALo kimenet adja. Az ALU z kimenet 1 értéke jelzi, ha a művelet eredménye 0. A c kimenet összeadás esetén a túlszorzást, kivonás esetén a negatív eredményt, shifelés esetén a kilépő bit értékét jelzi. A REG_ARR egy 4 regisztert tartalmazó regiszter tömb. Ennek opa kimenetén az a[1:0]-val kiválasztott sorszámu regiszter tartalma jelenik meg, ha a[1:0]=i, akkor Ri. Az adat beírását wr=1 engedélyezi és az in bemenetére kiválasztott adat (mpxo) az a[1:0]-val kiválasztott regiszterbe íródik az órajel felfutó élére. A wr jel ezen kívül az ALU c kimenetét beírja az ALU ci bemenetére kapcsolt 1 bites regiszterbe.**

Az **opb** kimenetén a **b[1:0]-val kiválasztott regiszter tartalma** jelenik meg, ha **b[1:0]=i, akkor Ri**. Az ALU **eci** bemenetére **b[1]** van kötve, ezért a **ci** bemenetét csak akkor veszi figyelembe az ALU, ha **opb-n R2** vagy **R3** van kiválasztva. A **REG_ARR** bemenetére kerülő adat (mpxo) az **MPX** multiplexer **s[1:0]** bemenetével választható ki. A regisztertömb **R3** regisztere speciális, mert **fel/le számlálóként is funkcionál**. Normál regiszterként írható (**a[1:0]=2'b11** és **wr = 1** esetén beíródik R3-ba az MPX-el kiválasztott érték) és normál regiszterként olvasható. Azonban, ha éppen nem írják, akkor **up3,dw3 = 0,1** esetén **lefele számol (R3=R3-1)**, **up3,dw3 = 1,0** esetén pedig **felfele számol (R3=R3+1)**, ha megjön a clk aktív éle. Az írás erősebb, mint a számlálás engedélyezés. Az R3 regiszter fel vagy le számláltatásával egyidőben bármely más regiszterbe lehetséges írni. A **tc3** kimenet akkor jelez, ha **R3==0**. A vezérlőt egy kívülről jövő, 1 órajel hosszú **start** parancs indítja. A vezérlő az adatstruktúrát a **V[11:0]** vezérlő jelekkel működteti. A működése közben figyelni képes az adatstruktúrából jövő **z, c** és **tc** feltétel jeleket. Egy számítási feladat elkészültét 1 órajel hosszú **rdy** státus jellel kell jeleznie. Az eredménynek meg kell maradnia a következő start jelig. **Az aktuális művelet szempontjából érdektelen vezérlő jelek értékét 0-nak kell választani**. Egy maximum 8 bites végeredményt a regiszter tömb opa kimenetén (**result_L**) kell megjeleníteni. Ha a végeredmény megjelenítéséhez 8-nál több bit kell, akkor a többi bitet az opb kimeneten (**result_H**) kell megjeleníteni.

a. (8p) Az alábbi Moore jellegű HLSM állapot diagram által leírt feladatot a fenti adatstruktúrával kell megvalósítani. (R0-at szorozza 16-al, eredmény:R2R0-ban)

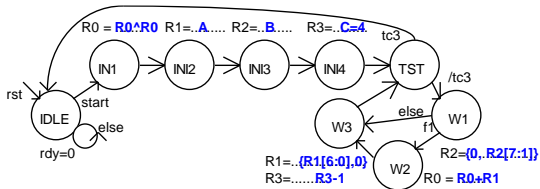


Adja meg a vezérlő állapotgráfiájának megadott állapotaiban kiadandó vezérlőjeleket a V[11:0] bitek értékének (0, 1) alábbi táblázatba való beírásával! (Segítségképp néhány értéket megadtunk. A rdy-t most nem kérjük.)

b. (1p) Milyen értékű lesz R2 az INI3 állapotban ?
.....0.....

c. (7p) A számító modullal két 4 bites szám szorzatát kell kiszámítani, **ResultL = A*B**.

A megvalósítást: **Kezdéskor a szorzandót R1, a szorzót, R2 regiszterbe tesszük, az eredmény regisztert 0-ázzuk. A szorzást ciklusba szervezzük.**



elvégzéséhez szükséges műveleteket a Moore jellegű HLSM állapotaikhoz. Az alább felsorolt műveletekből válasszon (szükségtelenek is vannak közöttük). Írja az elvégzendő művelet(ek) sorszámát a megfelelő állapot neve mellé! Nem biztos, hogy van elvégzendő művelet, ilyenkor írjon x-et! (A HLSM gráf segítő információkat tartalmaz. A gráf pontozott részreit nem kell kitölteni, de az is segítő információ.) A többlet és hiányzó művelet sorszámokért pontlevonás jár!

- 1: R0 = {R0[6:0],0} 2: R0 = R0^R0 3: R0 = {0, R0[7:1]}
 - 4: R0 = R0+R1 5: R1 = A 6: R1 = B 7: R1 = C
 - 8: R1 = R0+R1 9: R1 = {R1[6:0],0} 10: R2 = B
 - 11: R2 = C 12: R2 = {0, R2[7:1]} 13: R3 = R3+1
 - 14: R3 = R3-1, 15: R3 = R3+R1 16: R3=C 17: rdy = 1
- IDLE: ...x... INI1: ...2..... INI2: ...5... INI3: ...10....
INI4: ...16....TST: ...x..... W1:.....12..... W2:
...4..... W3:....9, 14.....

f. IMSC_F2 (5p) Készítsen HLSM diagram részletet a feladatban szereplő adatstruktúrához, a következő feladatra: Dekrementálja a R2R0-ban levő 16 bites számot. Röviden magyarázza el szóvegesen is a megoldását!



ALU funkció vezérlés: op[2:0]	ALU out (Alo)	z =1, ha	c = 1, ha
000	opa + opb + ci&eci	Alo==0	átvitel van
001	opa - opb - ci&eci	Alo==0	a-b < 0
010	{opa[6:0], ci&eci}	Alo==0	opa[7]
011	{ci&eci, opa[7:1]}	Alo==0	opa[0]
100	opa & opb	Alo==0	Soha (c=0)
101	opa opb	Alo==0	Soha (c=0)
110	opa ^ opb	Alo==0	Soha (c=0)
111	opb	Alo==0	Soha (c=0)

	wr	dw3	up3	s[1:0]	b[1:0]	a[1:0]	op[2:0]
V[11:0]	11	10	9	8 7	6 5	4 3	2 1 0
IDLE	0	0	0	0 0	1 0	0 0	0 0 0
INI1	1	0	0	0 1	0 0	0 0	0 0 0
INI2	1	0	0	0 0	1 0	1 0	1 1 0
INI3	1	0	0	1 0	0 0	1 1	0 0 0
TEST	0	0	0	0 0	0 0	0 0	0 0 0
W1	1	0	0	0 0	0 0	0 0	0 1 0
W2	1	1	0	0 0	1 0	1 0	0 1 0
END	0	0	0	0 0	0 0	0 0	0 0 0

Ciklusszámlálóként a speciálisan számlálóként is használható R3 regisztert használjuk. A szorzást LSB-vel kezdjük. A szorzó LSB-jét megjelenítjük c-ben. Ha c=1, akkor az eredményként használt regiszterhez hozzáadjuk a szorzandó aktuális 2 hatványát, ha c=0, nem adjuk hozzá. Mindkét eset után előállítjuk a szorzó következő 2 hatványát és ezzel egyidőben csökkentjük a ciklusszámlálót. Ezután visszamegyünk a ciklus elejére, ahol ellenőrizzük a ciklus számlálót. Rendelje a számítás

d. (1p) Adja meg a kért feltételeket a blokkvázlaton szereplő feltétel jel(ek) felhasználásával. (A parancs és feltétel bitek negáltját is fel lehet használni.)

fl:c.....

e. (1p) Adja meg az adatstruktúra C bemenetének értékét! (Az algoritmusban fel kell használni.)

C = ...4.....

f. folytatása:

Az R0-ból 1-et kivonunk, ez az átvitelt a ci regiszterben tárolja. Rögtön utána az R2-ből R3=0-át kivonunk. Az op2 R3 miatt b[1]=eci=1, ezért az összeadás a ci-it is figyelembe veszi. A felhasznált konstansokat az előbbieket előtt tároljuk a regiszterekben.

További segítség az F2 a.-hoz

Minták különféle műveletek esetén a táblázat kitöltésére:

Ha a művelet eredménye beíródik op1-be, akkor a wr bit 1, egyébként 0. (A továbbiakban a beíródó eset szerint töltjük ki a wr bitet a táblázatban.)

Ha a művelet az R3 regiszter speciális fel vagy le számláló képességét használja: Ha $R3 = R3+1$, akkor $up3 = 1$. Ha $R3 = R3-1$, akkor $dw3 = 1$. Ezek a műveletek a többi művelettel párhuzamosan végezhetőek, ha a másik művelet az R3-ba nem ír.

(A továbbiakban a többi művelet példáiban az up3 és dw3 bitet 0-ba állítjuk.)

Ha a művelet az ALU eredmény kimenetét írja egy regiszterbe (nem valamelyik konstans töltjük egy regiszterbe), akkor $s[1:0] = 00$. (Az MPX az ALU kimenetét választja ki a regisztertömb bemenetére.)

Ri = Ri művelet Rj, ha a 2 operandusos műveletben a c flag nem szerepel (pl. AND, OR, EXOR).

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	0 0	j[1] j[0]	i[1] i[0]	k ó d		
1	0	0	0 0	1 0	0 1	1 0 0		

(k ó d a művelet 3 bites kódja)

Pl. R1 = R1 & R2

Ha az 1 operandusos művelet az adatmozgatás (Ri = Rj).

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	0 0	j[1] j[0]	i[1] i[0]	1 1 1		
1	0	0	0 0	1 0	0 1	1 1 1		

Pl. R1 = R2

Ha az 1 operandusos műveletben a c flag szerepelhet ({opa[6:0], ci&eci} és {ci&eci, opa[7:1]}) de nem akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 0-ba kell állítani és b[0] értéke közömbös (x, de a feladatokban x helyett 0 beírását kérjük).

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	0 0	0	x	i[1] i[0]	k ó d	
1	0	0	0 0	0	0	0 1	0 1 0	

Pl. R1 = ({R1[6:0], 0})

Ha az 1 operandusos műveletben a c flag szerepelhet ({opa[6:0], ci&eci} és {ci&eci, opa[7:1]}) és azt akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 1-be kell állítani és b[0] értéke közömbös (x, de a feladatokban x helyett 0 beírását kérjük).

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	0 0	1	x	i[1] i[0]	k ó d	
1	0	0	0 0	1	0	0 1	0 1 0	

Pl. R1 = ({R1[6:0], c})

Ha a 2 operandusos műveletben a c flag szerepelhet ({opa+opb+ci&eci} és {opa-opb-ci&eci}) de nem akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 0-ba kell állítani, ezért ilyenkor op2-ként csak R0 vagy R1 használható!

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	0 0	0	x	i[1] i[0]	k ó d	
1	0	0	0 0	0	0	0 1	0 0 0	

Pl. R1 = R1+R0

Ha a 2 operandusos műveletben a c flag szerepelhet ({opa+opb+ci&eci} és {opa-opb-ci&eci}) de azt akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 1-be kell állítani, ezért ilyenkor op2-ként csak R2 vagy R3 használható!

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	0 0	1	x	i[1] i[0]	k ó d	
1	0	0	0 0	1	0	0 1	0 0 0	

Pl. R1 = R1+R2+c

Ha a művelet valamelyik konstans betöltése egy regiszterbe (Ri = K), akkor s[1:0]-al a konstansot kell kiválasztani. (Az A esetén s[1:0]=01, B esetén s[1:0]=10, C esetén s[1:0]=11), a wr bitet 1-be kell állítani, az a[1:0]-val kell kiválasztani, hogy melyik regiszterbe íródjon. (Az op2 és az ALU művelet közömbös.)

wr	dw3	up3	s[1:0]	b[1:0]	(Rj)	a[1:0]	(Ri)	op[2:0]
11	10	9	8 7	6 5	4 3	2 1 0		
1	0	0	K[1] K[0]	x x	i[1] i[0]	x x x		
1	0	0	1 1	0 0	0 1	0 0 0		

Pl. R1 = C

F3. (17p) A MiniRISC **GPIO A** egysége segítségével kell megoldania egy feladatot.

A GPIO A adatregiszterébe (**ADO**) kell írni a kimenetek értékét. Az irányregiszterével (**ADR**) állítható be bitenként a port iránya. Amely bitekre 1-et írunk, azok kimenetek lesznek, vagyis az I/O lábakon megjelenik az adatregiszterbe írt bitek értéke. (ADR az adatregiszter kimeneti bitjeire kapcsolódó 3 állapotú meghajtókat engedélyezi vagy tiltja.) Az ADO és ADR regiszterek tartalma visszaolvasható. Az I/O lábak aktuális értékét az **ADI** regiszterből lehet beolvasni. Első feladatként a **GPIO A** periféria hardver egység egy részét kell megtervezni (újratervezni) Verilog nyelven. A **MiniRISC busz felhasználható jelei: (A[7:0], Din[7:0], Dou[7:0], RD, WR, IRQ, clk, rst)**. A GPIO A periféria báziscíme **0xA0**.

A GPIO A periféria programozói felülete:

Funkció	Cím	D[7:0]	olvasható/írható
Kimeneti adat regiszter ADO	Báziscím + 0	ADO[7:0]	W/R (<i>ADO_w</i> , <i>ADO_r</i>)
Adat az I/O lábakon ADI	Báziscím + 1	ADI[7:0]	R (<i>ADI_r</i>)
Írány regiszter ADR (0: be, 1: ki)	Báziscím + 2	ADR[7:0]	W/R (<i>ADR_w</i> , <i>ADR_r</i>)

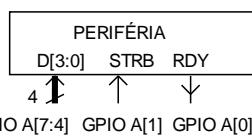
a. (5p) Tervezze meg Verilog nyelven a szükséges engedélyező jeleket: ADO regiszter írás engedélyező *ADO_w*, olvasás engedélyező *ADO_r*, ADI regiszter olvasás engedélyező *ADI_r*, ADR regiszter írás engedélyező *ADR_w*, olvasás engedélyező *ADR_r*!

b. parameter base_addr =8'hA0.....; // A periféria regiszter címtartomány kezdőcíme hexadecimálisan

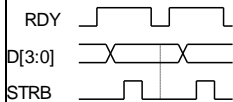
```
assign psel = (base_addr>>2)==(A>>2)...vagy (base_addr[7:2]==A[7:2]) ...;
assign ADO_w = psel&(A[1:0]==2'b00)&WR...vagy psel&~A[1]&~A[0]&WR.....;
assign ADO_r = psel&(A[1:0]==2'b00)&RD...vagy psel&~A[1]&~A[0]&RD.....;
assign ADI_r = psel&(A[1:0]==2'b01)&RD...vagy psel&~A[1]&A[0]&RD.....;
assign ADR_w = psel&(A[1:0]==2'b10)&WR...vagy psel&A[1]&~A[0]&WR.....;
assign ADR_r = psel&(A[1:0]==2'b10)&RD...vagy psel&A[1]&~A[0]&RD.....;
```

b. (3p) Definiálja Verilog nyelven az GPIO A visszaolvasható regisztereinek olvasásához szükséges logikát az alábbiak szerint! Elkezdttük, folytassa!
reg [7:0] ADO, ADI, ADR;
reg [7:0] Din;

```
always @ (.....*.....)
  case ({ADR_r, ADI_r, ADO_r})
    3'b100: .....Din.....<= .....ADR.....;
    3'b010: .....Din.....<= .....ADI.....;
    3'b001: .....Din.....<= .....ADO.....;
    default: .....Din.....<= .....8'h00.....;
  endcase
```



IDŐDIAGRAM:



A c pont után következő feladatok (d, e)

Egy periféria 4 bites adatokat vár. Az RDY jellel jelzi, ha új adat írható bele. Ekkor a D[3:0] bementére kell adni a beírandó adatot és

egy STRB impulzust kell adni, miközben a D[3:0] nem változik. A periféria a GPIO_A perifériával van összekötve, a fenti rajz szerint. A kommunikáció idődiagramját alatta mutatjuk. Programozott bit billegtetéssel (bit-banging) kell megoldania a feladatot. Feltételezzük, hogy az időzártási követelmények automatikusan teljesülnek a MiniRISC-kel történő kezelés esetén.

d. (3p) Adja meg a periféria kezeléséhez szükséges assembly nyelvű definíciókat, majd programozza fel a GPIO_A[7:4]-et és GPIO A[1]-et kimeneti portnak, és írja ki az idődiagram szerinti kezdőértéket (D[3:0] kezdetben legyen 0). Az ADR, ADO regiszterek címei már definiált konstansok, azokat nem kell definiálnia..

```
DEF RDY 0b00000001 ;RDY mask
DEF STRB 0b00000010 ;STRB mask
DEF NSTRB 0b11111101 ;STRB mask negáltja
CODE
Itt még ne használja fenti DEF konstansokat, a kódban adja meg a megfelelő hexadecimálisan!
RST: mov r0, #0x00 ;GPIO_A kezdőérték
mov ADO, r0 ;kezdőérték az adat reg.-be
mov r0, #0xf2 ;GPIO_A [7:4] és [1] irány
mov ADR, r0 ;kimenetnek állítása
```

c. (2p) Tervezze meg a 8 bites adat regisztert (ADO)! A regisztert a rst törl. Ha a processzor a címére ír, beíródik az adat.

```
reg [7:0] ADO;
always @ (.....posedge clk.....)
  if (...rst...) ADO <= .....8'h00.....
  else
    if (...ADO_w...) ADO <= .....Dou.....
```

További feladatok:

e. (4p) Írja meg a PER_wr szubrutint, mely az r10 regiszterben megadott adat alsó és felső 4 bitjét ebben a sorrendben kiiírja a perifériába az idődiagrammal megadott módon! A feladatban nem használt GPIO bitekre az adat regiszterben mindig 0-át írjon!

```
PER_wr: mov r11, ADI..... ; PIN (státus) beolvasása
tst r11, #RDY..... ; RDY bit ellenőrzése
jz PER_wr..... ; vissza, amíg nem 1
mov r11, r10..... ; adat másolása
and...r11, #0x0f..... ; adat alsó 4 bit kimaszkolása
swp r11..... ; adat alsó 4 bit a felső 4 bitre
mov ADO, r11..... ; adat kiírása a GPIO-ba (STRB:0)
or r11, #STRB..... ; STRB bit 1-be állítása
mov ADO, r11..... ; adat kiírása a GPIO-ba (STRB:1)
and r11, #NSTRB..... ; STRB bit 0-ba állítása
mov ADO, r11..... ; adat kiírása a GPIO-ba (STRB:0)
wait: mov r11, ADI..... ; PIN. beolvasása
tst r11, #RDY..... ; RDY bit ellenőrzése
jz wait..... ; vissza, amíg nem 1
and r10, #0xf0..... ; adat felső 4 bit kimaszkolása
mov ADO, r10..... ; adat kiírása a GPIO-ba (STRB:0)
or r10, #STRB..... ; STRB bit 1-be állítása
mov ADO, r10..... ; adat kiírása a GPIO-ba (STRB:1)
and r10, #NSTRB..... ; STRB bit 0-ba állítása
mov ADO, r10..... ; adat kiírása a GPIO-ba (STRB:0)
rts..... ; visszatérés a szubrutinból
```

IMSC_F3 (6p) Írja meg külön lapon az e-pontbeli szubrutin olyan változatát, amely csak az r10-ben megkapott adat felső 4 bitjét írja ki a perifériára, de a GPIO_A adatregiszterének nem használt bitjeit nem változtatja meg.

```
PER_wr: mov r11, ADI ; PIN. beolvasása
tst r11, #RDY ; RDY bit ellenőrzése
jz PER_wr ; vissza, amíg nem
mov r11, ADO ;ADO mentése
and r11, # 0b00001100 ;használt bitek 0-ázása
and r10, #0xf0 ;adat felső 4 bit maszkolása
or r11, r10 ;adat a felső 4 bitre
mov ADO, r11 ;kiírás
or r11, #STRB ;STRB bit = 1
mov ADO, r11 ;kiírás
and r11, #NSTRB ;STRB bit = 0
mov ADO, r11 ;kiírás
rts ;egyéb jó megoldás is lehetséges!
```

Rendelkezésre álló idő: 100 perc Összes pont: 75