Példa feladatok a kisZH6-hoz

Digitális technika (VIMIAA02)

A példa kisZH feladatok célja, hogy a kisZH-ra történő felkészülés során segítséget nyújtson pár tipikus feladat ismertetésével, valamint hogy a hallgatók számára egyenlő esélyeket biztosítson azáltal, hogy a példa kisZH feladatokat a legkorábbi labor kurzus időpontja előtt pár nappal közzétesszük. (A kisZH-n nem pontosan ilyen feladatok lesznek, de némileg hasonlóak.)

Az alábbiakban a kisZH6 által számonkért témakörhöz (*periféria illesztés a MiniRISC buszra*) található egy-két példa feladat. A feladatok megoldása során a MiniRISC busz felhasználható jelei: A[7:0], Din[7:0], Dou[7:0], RD, WR, IRQ, clk, rst.

1 Cím és parancs dekóder (Verilog implementáció)

a) A MiniRISC buszra illesztendő periféria bázis címe: **0xB8**
Ennek megfelelően egészítse ki az alábbi Verilog kódrészletet (hexadecimális konstans megadásával):

```
parameter base_addr = ____;
```

Tipp: Verilog nyelven más szintaktikával jelöljük a hexadecimális számrendszert, mint a MiniRISC assemblyben (vagy a C nyelvben).

b) A periféria **7 db adat regiszterrel (DR0 … DR6)** rendelkezik (mindegyikük írható/olvasható). A címeik pedig sorrendben vannak a **(bázis cím + 0) … (bázis cím + 6)** tartományban.

Tételezzük föl, hogy a MiniRISC CPU épp a periféria valamelyik regiszteréhez fér hozzá. Az általa kiadott cím bitjei közül melyek azok, amik a periféria kiválasztásában játszanak szerepet (jelölje **p**-vel) és melyek azok, amik a periférián belüli regiszter kijelöléséért felelősek (jelölje **r**-rel)?

A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]

c) Az alábbi Verilog kódrészlet kiegészítésével készítsen egy olyan segéd jelet, amely akkor igaz, ha a CPU **az adott perifériához fér hozzá**! (Kicsit részletesebben fogalmazva: a CPU az adott periféria címtartományában lévő valamelyik regiszterhez fér hozzá.)

```
assign p_sel = (A[ : ] == ________[ : ]);
```

d) Az alábbi Verilog kódrészlet kiegészítésével készítsen egy olyan segéd jelet, amely akkor igaz, ha a CPU által kiadott címnek az adott periférián belüli regiszter kijelöléséért felelős bitjei a **DR3** a**datregisztert jelölik ki**!

```
assign d3_sel = (A[ : ] == 'b   );
```

e) A fenti két jel, valamint a MiniRISC busz valamelyik vezérlő jelének felhasználásával készítsen egy olyan jelet, ami alkalmas lehet a **DR3-as regiszter írás engedélyező jelének** funkcióját ellátni!

```
assign d3_wr = ____________;
```

f) A perifériánk címtartományában még 1 db cím szabad. Szeretnénk azonban még két regisztert: egy **parancs regisztert (CR)** /csak írható/ és egy **státusz regisztert (SR)** /csak olvasható/. Megoldható-e a feladat (Igen / Nem + Indoklás)?

2 Írható regiszter (Verilog implementáció)

Adott egy timer periféria, aminek az alábbi (csak írható) parancs regisztere (**CR**) van:

CR[7]	CR[6]	CR[5]	CR[4]	CR[3]	CR[2]	CR[1]	CR[0]
EN	PER	–	–	–	PRES[2:0]		

EN a működést engedélyező bit (0: tiltás, 1: engedélyezés). **PER** a periodikus üzemmódot kiválasztó bit (0: egyszeri, 1: periodikus). **PRES** az timer órajel előosztását meghatározó mező (az órajelet 2^{PRES} értékkel ossza le). A **–** karakterrel jelölt bitek nem rendelkeznek funkcióval.

Reset hatására azt szeretnénk, hogy a timer legyen tiltva, az üzemmódja legyen egyszeri és az órajelének frekvenciája egyezzen meg a rendszer órajelével.

Normál működés során pedig, ha a MiniRISC CPU a timer parancs regiszterét írja (**cr_wr** aktív), akkor a processzor kimeneti adatbuszának tartalmát kell eltárolni (a fenti táblázat szerinti bit kiosztással).

A fent leírtak implementálására egészítse ki az alábbi Verilog kódrészletet:

```
reg en;         // Engedélyezés
reg per;        // Periodikus működési mód kiválasztása
reg [2:0] pres; // Órajel előosztó

always @(              )
   if (rst)
      {en,per,pres} <= 'b        ;
   else if (cr_wr)
      {en,per,pres} <= {      ,      ,      };
```

3 Olvasható regiszter (Verilog implementáció)

Adott egy timer periféria, aminek az alábbi (csak olvasható) státusz regisztere (**SR**) van:

SR[7]	SR[6]	SR[5]	SR[4]	SR[3]	SR[2]	SR[1]	SR[0]
EN	PER	TOUT	–	–	PRES[2:0]		

Az egyes bitek funkciója most nem fontos: számunkra mint Verilog jelek **en**, **per**, **tout** és **pres** nevek alatt elérhetőek. Ha a CPU ezen regisztert olvassa (**sr_rd** jel aktív), akkor a fenti táblázatnak megfelelően kell visszaadnunk a biteket (a nem használt pozíciókon adjunk vissza 0-ákat). Ha nem aktív az **sr_rd** jel, akkor minden bit pozíción 0-át adjunk vissza.

A fent leírtak implementálására egészítse ki az alábbi Verilog kódrészletet:

```
always @(              )
   if (      )
      ______ <= {      ,      ,      ,      ,      };
   else
      ______ <= 'b        ;
```

4 Assembly rutin regiszter értékek tesztelésére

a) Adott egy utazó bőrönd informatikusok számára. A **bőrönd kinyitásához** egy 8 db kapcsolóból álló panelen kell az elvárt kombinációt (**0xAC**) előállítani. Illetéktelen kezek véletlen próbálkozásait elkerülendő van egy másik kombináció is (**0xE8**). Ha ebben az állásban vannak a kapcsolók, akkor **beindul az önmegsemmisítő**! A kapcsolók a **0x81-es címről** olvashatóak be.

Írjunk egy assembly programot az alábbi kódrészlet kiegészítésével, ami egy ciklusban folyamatosan vizsgálja a kapcsolók állapotát. Ha a kinyitáshoz szükséges kombinációt tapasztalja, akkor átugrik az **OPEN** címke mögötti kódra, ha az önmegsemmisítéshez szükséges kombináció tapasztalható, akkor pedig a **SELF_DESTRUCT** címke mögötti kódra.

A bőrönd vezérlésére két relé áll rendelkezésre. A reléket aktivizálni a **0x80-as címen** található **relé vezérlő regiszterrel** lehet (az adott reléhez tartozó bit 1-be állításával). A zárat a 0. bithez tartozó relé vezérli, míg az önmegsemmisítő biztonsági reteszét a 7. bithez tartozó.

Az **OPEN** címkét követő kódrészlet a megfelelő relé meghúzásával kinyitja a bőrönd zárját. Ezt követően arra vár, hogy lecsukják a bőröndöt. Ha le van csukva a bőrönd fedele, az megnyom egy gombot. A gomb állapota a **0x84-es címen** elérhető **gomb regiszterből** olvasható ki, a 0. biten található (lenyomott:1, elengedett:0). Miután lecsukták a bőröndöt, a szubrutin elengedi a zárat nyitva tartó relét majd visszatér.

A **SELF_DESTRUCT** címkét követő kódrészlet egy másik relé meghúzásával kihúzza a bőröndbe épített önmegsemmisítő biztonsági reteszét. Ezt követően… tulajdonképpen mindegy is…

```
DEF RL 0x80              ; Relék (MSB: önmegsemmisítő, LSB: zárnyitó)
DEF SW 0x81              ; Kapcsolók
DEF BT 0x84              ; Gomb (LSB: fedél zárás érzékelő)

SW_LOOP:
    mov r0, [          ] ; Kapcsolók beolvasása
    [  ] r0, [         ] ; Nyitáshoz szükség kombináció ellenőrzése
    [  ] OPEN            ; Ugrás a nyitó kódra megfelelõ kombináció esetén
    [  ] r0, [         ] ; Önmegsemmisítéshez szükség kombináció ellenőrzése
    [  ] SELF_DESTRUCT   ; Ugrás az önmegsemmisítõ kódra, ha kell
    jmp SW_LOOP          ; Ugrás vissza a kapcsolók beolvasásához

OPEN:                    ; Nyitó kódrészlet
    mov r0, [          ] ; Zár nyitó relé aktivizálása
    mov [ ], r0          ;
LID_LOOP:                ; Várakozás, amíg nyitva van a bőrönd
    mov r0, [ ]          ; Gomb állapot beolvasása
    [  ] r0, [         ] ; Lecsukás érzékelő gombra tesztelés
    [  ] LID_LOOP        ; Ugrás vissza, ha még nyitva
    mov r0, [          ] ; Zár nyitó relé deaktiválása
    mov [ ], r0          ;
    jmp SW_LOOP          ; Ugrás vissza a kapcsolók beolvasásához

SELF_DESTRUCT:           ; Önmegsemmisítő kódrészlet
    mov r0, [          ] ; Biztonsági reteszt kihúzó relé aktiválása
    mov [ ], r0          ;
    ...
```

b) A fenti kódban nem használtunk szubrutin hívást, de szervezhettük volna úgy is a programot.

- Mi a különbség egy szubrutin hívás (**jsr**) és egy feltétel nélküli ugrás (**jmp**) között?
- Milyen utasítással kell visszatérni egy szubrutinból?
- Honnan tudja ez az utasítás, hogy hova kell visszatérni? (Avagy hol tároljuk el a visszatérési címet?)
- A visszatérési címek tárolására FIFO vagy verem jellegű struktúra a célszerű? Miért?