Megoldás

Digitális technika II. (vimia111)
4. gyakorlat: Tesztelés alapjai, processzorok alapvető jellemzői

Elméleti anyag:

- Digitális áramkörök tesztelése: a tervezés verifikálása, a sorozatgyártás ellenőrzése (ezt az utóbit részleteztük)
  - A „jó" egység a tesztelőben: mesterpéldány (golden unit), modell (válaszok tárolása, algoritmus)
  - A tesztminták (tesztvektorok) előállítása: véletlen, kimerítés, hibamodell alapján tervezett
  - Sorrendi hálózat egyszerű vizsgálata: regiszter vagy RAM beírása-kiolvasása alternatív mintákkal (010101.., 101010…)
  - Kombinációs hálózat vizsgálata: egyszeres leragadási hibák (a hálózat valamelyik pontja logikai 0-ba vagy 1-be ragad
    Teszttervezés a Boole differenciák módszerével, elemi áramkörök tesztervezése
- Processzoros vezérlés általános tulajdonságai
  - Az induló készletben a vezérlőn kívül általános célú adatstruktúra elemek (aritmetikai-logikai egység, regiszterek stb.) is vannak
  - A külvilággal való kommunikáció BUSZ-on keresztűl történik
  - A BUSZ-on alapvetően egy rendszervezérlő, több de egyidejűleg csak egy aktív MASTER és több SLAVE van.
  - Tipikus SLAVE-ek: memória, periféria, ezek illesztése később
- Példák az anyag első részéből, ZH konzultáció

Irodalom:
Selényi Endre: A tesztelés alapjai, előadás jegyzet (a gyak könyvtárban)
Benesóczky Zoltán: Digitális tervezés funkcionális elemekkel és mikroprocesszorokkal, egyetemi tankönyv, MK55033, 103-112.

Gyakorló példák:

4.1. Készítsen mikroprogramozott vezérlő egységet, amely a START-tal való elindítás után nagyság szerint sorbarendezi egy RAM tartalmát, rendezés után a 0-ás címre kerül a legkisebb szám, a maximális címre pedig a legnagyobb szám. Célszerű az előadáson megismert „buborék" algoritmust használni.

a. Először olyan adatstruktúrát készítsen, amely képes két egymás utáni cím tartalmának nagyság szerint rendezésére. Készítse el ehhez a részhez a vezérlő programot és próbálja is ki.

b. Következő lépésben egészítse ki az adatstruktúrát a teljes rendezéshez szükséges további egységekkel és készítse el a teljes mikroprogramot!

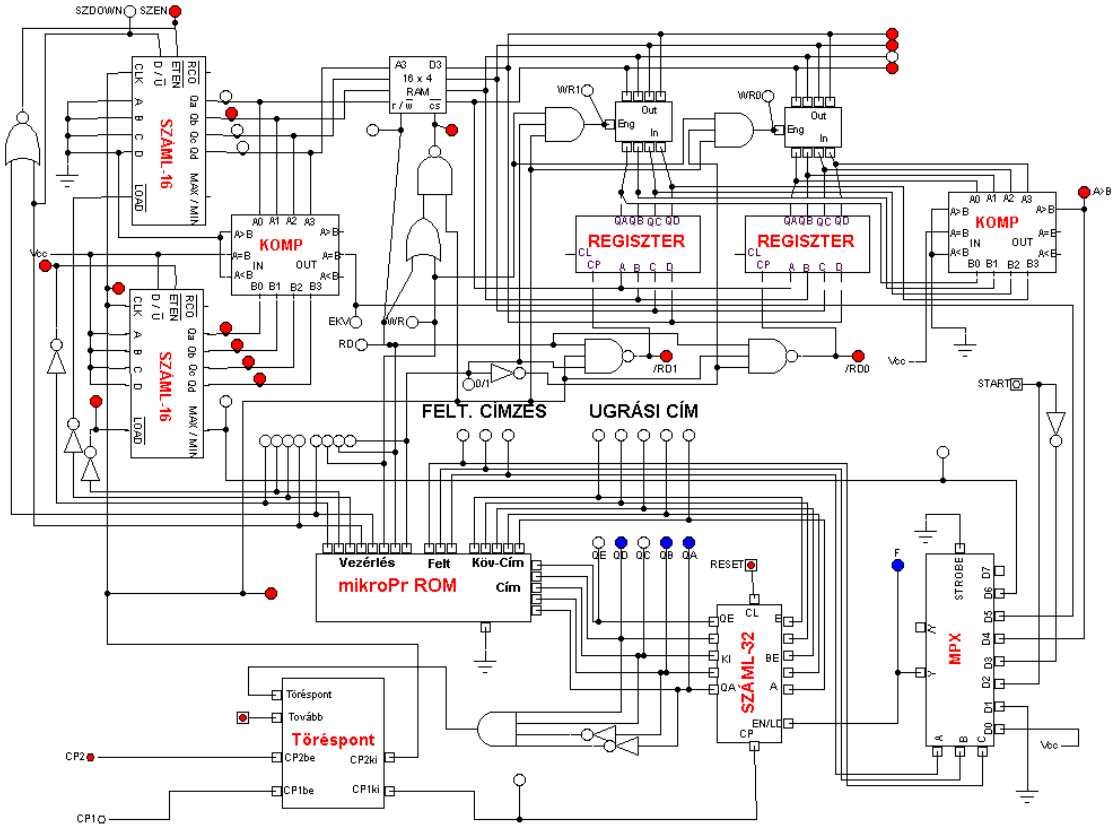
A feladat megoldásának dokumentációja megtalálható a RENDEZ_RAM_mintadoku_2011.pdf anyagban, maga a kapcsolás pedig **RAM_rendezes.dwm**

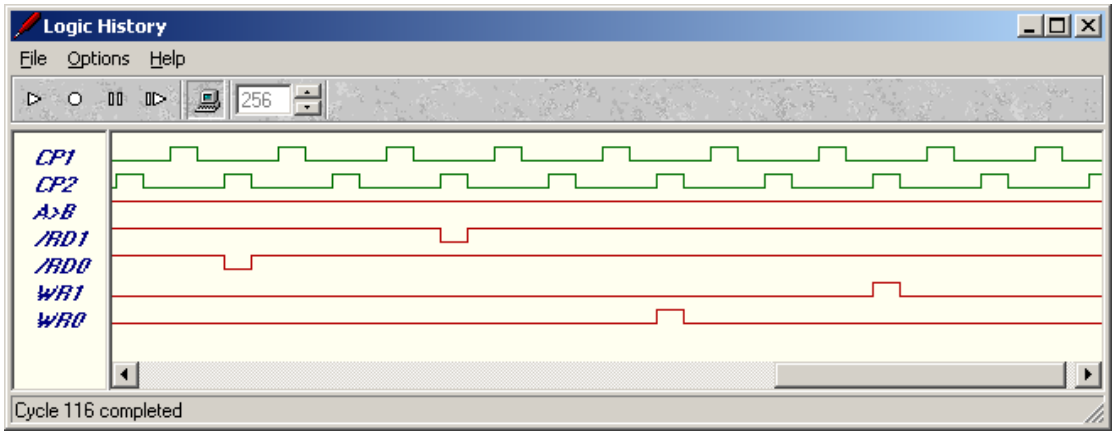
Ebben először a két cella-kiolvasás és ha kell, a fordított visszaírás részletet megnézni.

A következő ábra a könyvtári kapcsolás módosítását mutatja a fenti vizsgálathoz.
Beiktattuk a Töréspontot és leállási feltételt fogalmaztunk meg a x1100 címre
Ezzel egy RAM cella-pár eseményeit nyomon lehet követni: van tartalomcsere vagy nincs.

RAM_rendezes_torespont.dwm

Az alábbi idődiagramon egy RD0-RD1, WR0-WR1 sorozat látszik



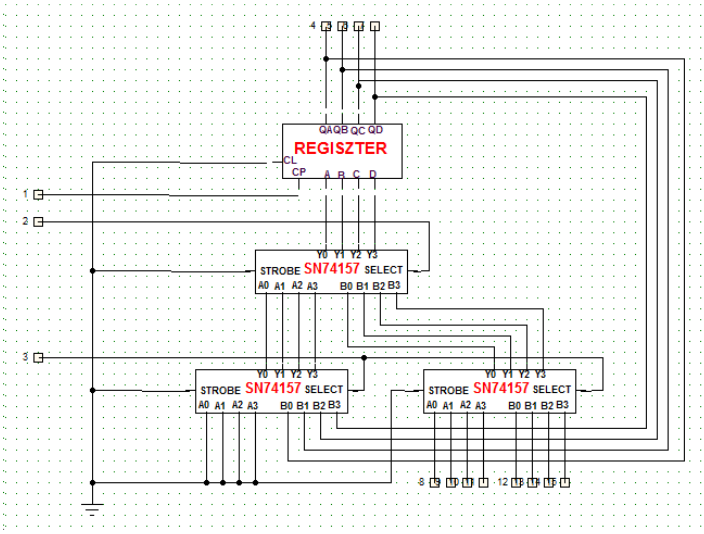


A teljes példa részletes kidolgozása megtalálható a RAM_RENDEZES projektben!
Ott a RAM kezdotartalom.map fájlban található a kezdeti RAM tartalom, ha valaki menetközben „megrendezi” a RAMot, akkor ezt célszerű egy új rendezés előtt betölteni.
Maga a teljes rendezés sokáig (10-20 percig) tart, ezen lehet spórolni, ha az adatstruktúrát úgy módosítjuk, hogy ne F-et töltsön a 2. számlálóba, hanem pl, 4-5-öt.

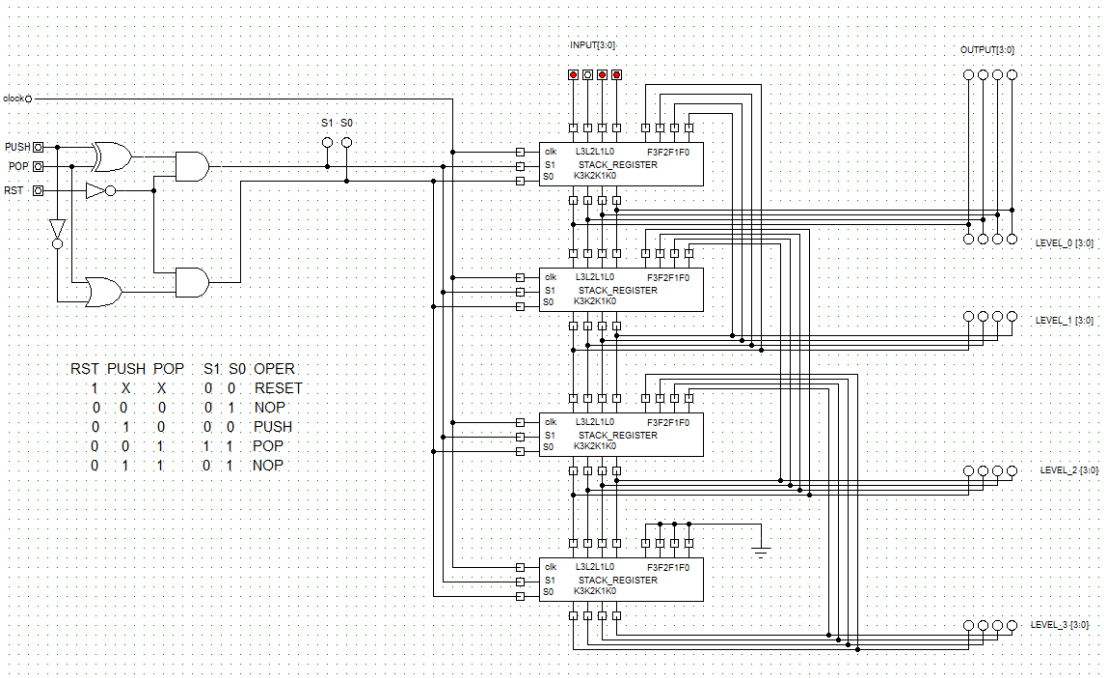
4.2. Regiszter alapú HW STACK egység tervezése

A feladat regiszter alapú STACK egység tervezése, 4 szintre, 4 bites regiszterekkel. A stack szinkron, órajelvezérelt regisztereket tartalmaz, vezérlőjelei CLK, PUSH, POP, RST () a RST nem lenne szükséges, hiszen a stack használata nem igényli, de esetleg teszteléskor hasznos lehet), az adat interfészei INPUT[3:0], OUTPUT[3:0].

A terv alapja egy 4 üzemmóddal rendelkező regiszter. Az üzemmódok a bemeneti 4x1-es multiplexerrel választhatók ki. Ennek megfelelően 4 adat tölthető a regiszterbe: Nulla (S1S0=00), saját tartalma (S1S0=01), az előző szint (S1S0=10), vagy a következő szint (S1S0=11).



Figyeljük meg az adatok mozgását. Minden regiszter az őt megelőző ill. követő szinttel van kapcsolatban. A LEVEL_0 az állandó bemenet és az állandó kimenet is egyben. A szintek száma tetszőlegesen bővíthető az adott minta szerint.



A STACK 3 vezérlő bemenettel rendelkezik, a bemenőjelek értelmezését a vezérlési táblázat specifikálja. A bal oldali kis kombinációs logika a 4 üzemmódú STACK regiszter egység vezérlő jeleit állítja elő, az egyedi PUSH, POP, RST jelek alapján. A RST jel prioritása a legnagyobb, a másik két jel (PUSH és POP) azonos prioritású. Az együttes PUSH és POP nem okoz változást a STACK tartalmában.
A tervezői fájlok: STACK4x4.dwm, STACK_REG.dwm.

4.3. Egészítsük ki a STACK-et egy EMPTY és FULL jelzéseket biztosító logikával.

A STACK használata során a legutolsó adatot kivéve nincs explicit információ a STACK-ben lévő adatokról, továbbá ezek mennyiségéről. Alapvetően ez csak lesz akkor érdekes, amikor esetleg a STACK túlcsordulna, viszont jellegéből adódóan ezt csak akkor vesszük észre, amikor az adatok kivételével elérjük az „alulcsordulás” állapotot.

Az állapotjelző kiegészítés lehetséges verziói:

a, Egy 3 bites fel/le számláló, a max. 4 és min. 0 értékeknél megállítva
b, Egy 5 bites shiftregiszter, szintén szélső állapotban megállítva.

Indulásnál, illetve később a számláló 000 (az SHR 10000) állapotában aktiválja az EMPTY jelet, jelezve az üres állapotot. A számláló 100 (az SHR 00001) állapotában kiadja a FULL jelet. Ezek megjelenése még nem hiba, tehát rendszer szinten felhasználható megelőző ellenőrzésre. Ha azonban érvényes üres STACK esetén POP műveletet hajtunk végre, az azonnali hibát eredményez. Ha tele STACK esetén PUSH műveletet adunk ki, akkor az szintén hibát eredményez, de nem azonnal, csak amikor az ezutáni működés során (esetleg sorozatos POP/PUSH műveletek után) egyszer újra visszaáll az üres állapot, és a rendszer még egy POP műveletet adna ki. Tehát az igazán precíz megoldás az lenne, hogy a FULL esetén kiadott PUSH parancs beállít egy belső ERROR flag-et, amit vagy azonnal aktivál (STACK ERROR), vagy csak akkor, amikor a végrehajtott műveletekben 4-gyel több POP művelet fordult elő, mint PUSH (ERROR „HIBÁS ADAT”).