Digitális rendszertervezés

RTL tervezés és szimuláció - Szinkron áramkörök

Ismétlés

Verilog egy hardver leíró nyelv, amivel digitális áramköröket tudunk modellezni és szimulálni. Olvassátok el még egyszer a Verilog c. fejezetet az előző laboratórium segédletében. Ezen a laboron szinkron hálózatok modellezését fogjátok megismerni a Verilog hardver leíró nyelv segítségével.

Szinkron hálózatok modellezése

A folytonos értékadással (*assign*) lehetőség volt valamilyen kifejezést hozzárendelni egy vezetékhez (*wire*) vagy kimeneti porthoz. Ezzel a nyelvi elemmel olyan hálózatok modellezhetőek, amelynek a kimenete kizárólag a bemeneti értékektől függ (azaz kombinációs hálózatok). Sorrendi hálózatok kimenete a bemenő kombináció mellett előző eredményektől, az áramkör állapotától is függ, azaz szükség van valamilyen tároló elem modellezésére.

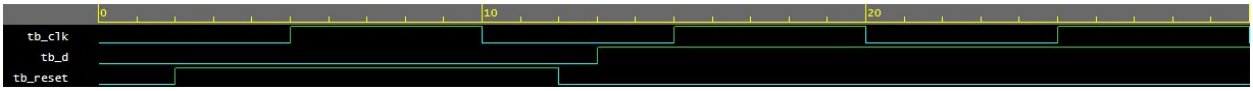
Ehhez az ún. *reg* típusú jeleket fogjuk használni, amelyet eddig csak tesztbenchben láthattunk. Ezeknek a *reg* típusú jeleknek csak kódblokkban (*initial* vagy *always* blokk) adhatunk értéket. Áramkör modellezésére az *always@* blokkot használjuk[1]. A következő kódrészlet egy egyszerű sorrendi hálózat modelljét mutatja be.

```
module dff (clk, reset, d, q, qn);
  input      clk;
  input      reset;
  input      d;
  output     q;
  output     qn;

  reg        q;

  assign qn = ~q;

  // always@ blokk egy „esemeny hatasara lefuto fuggveny”
  // az esemenyt az erzekenysegi listan tudjuk megadni
  // jelen esetben orajel vagy a reset felfuto ele eseten „fut le”
  always @(posedge clk or posedge reset)
  begin
    if (reset) begin
      // aszinkron, magas aktiv reset
      q <= 1'b0;
    end else begin
      // az orajel felfuto elere a q kimenet megkapja d bemenet erteket
      q <= d;
    end
  end
endmodule
```

[1] Az egyszerű *initial* és *always* blokkokat tesztelés céljából szoktuk alkalmazni

A különbség az egyszerű *always* és az *always@* blokk között az érzékenységi lista jelenléte. Ha az értékenységi listán jelölt esemény bekövetkezik, akkor az *always@* blokk kiértékelődik[2]. Az always blokk belsejében a kódot sorról sorra olvashatjuk, ahogy hagyományos programnyelvek esetén is tennénk. Ha a reset értéke 1, akkor a q jelnek 0 értéket adunk. Ellenkező esetben a q értéke felveszi a d bemenet értékét. Ez a jellegű értékadást *nem blokkoló* értékadásnak hívjuk, mert a bal oldal az értéket **nem azonnal kapja meg**, hanem csak az *always@* **blokk lefutása után**. Tehát az értékadás utáni sorban kiolvasnánk a q jel értékét, akkor az *always@* blokk meghívásakor tárolt értéket adná vissza. Az *always@* blokk következő hívásánál már az új értéket kapjuk vissza. Az *always* blokkok helyzete és a kiértékelésük sorrendje hasonlóan az *assign* értékadáshoz független a kódban lévő pozíciójuktól. Két órajelre érzékeny *always@* blokk a mi szempontunkból **párhuzamosan** értékelődik ki, ugyanúgy ahogy a folytonos értékadások is.

Gyakorlásképpen próbáljuk megfejteni, hogy mi fog történni a q és qn kimenetekkel az alábbi bemenő hullámforma esetén:



Feladatok

A feladatokat ModelSim segítségével végezheted el.

1. D-flip-flopból 4 bites tároló

a. A már bemutatott D-flip-flop modul felhasználásával készítsünk egy 4 bites tárolót. Írjunk hozzá tesztbenchet is, és szimulációval ellenőrizzük a működését! Az alap D-flip-flop példa megtalálható az EDU rendszerben.
b. Egészítsd ki a tároló modult egy load bemenettel. Kizárólag akkor mintavételezze a d bemenetét a tároló, ha ez a load jel aktív (logikai ’1’). Egészítsd ki a tesztbenchet, hogy az új funkciót is kipróbálja.

2. Számláló bővítése

a. Vizsgáld meg a számláló modult! Mire valók az egyes portjai? Szöveges formában foglald össze a portok funkcióit és a számláló modul működését.
b. Bizonyos bemenő port nincs felhasználva a forráskódban. Mi lenne a célja? Bővítsd a számlálót, hogy ezt a portot is használod! Az új funkciót tesztbenchben is próbáld ki.

3. Shift regiszter

a. Vizsgáld meg a shift regiszter modult. Mire valók az egyes portjai? Szöveges formában foglald össze a portok funkcióit és a shift regiszter működését.
b. Bizonyos bemenő port nincs felhasználva a forráskódban. Mi lenne a célja? Bővítsd a shift regisztert, hogy ezt a portot is használod! Az új funkciót tesztbenchben is próbáld ki.

[2] Az egyszerű *always* blokk gyakorlatilag egy végtelen ciklusnak tekinthető.