BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Digitális technika (VIMIAA03)
3. gyakorlat és laboratórium

Raikovich Tamás
BME MIT

Bevezetés

- Feladat: 4 bites összeadó készítése
  1) Teljes összeadók felhasználásával
  2) A + aritmetikai operátor felhasználásával
- A feladatokhoz szükséges új Verilog ismeretek
  - Verilog modul példányosítása
  - Belső jelek deklarálása
  - Aritmerikai operátorok (összeadás, kivonás)
  - Szimuláció: for ciklus
- **A Xilinx ISE fejlesztői környezet használata (projekt létrehozás, szimuláció) megtalálható a 2. labor anyagában, ezt itt nem ismételjük meg újból**

Verilog HDL ismeretek

Modul példányosítása

- A felhasználni kívánt modul

```
module SomeFunction(input A, input B, output C);
    ⋮
endmodule
```

- Ezt más modulokban felhasználhatjuk az alábbi módon
  - Több példány is létrehozható eltérő néven

```
wire d, e, f;
SomeFunction Func1(.A(f), .B(e), .C(d));
```

Jelek hozzárendelése a portokhoz

A példányosítandó modul

A példány (egyedi) neve

Egy jel bekötése egy portra:

.port_név(jel_név)

Az *e* jel a *B* portra csatlakozik

Verilog HDL ismeretek

Belső jelek

- **A belső jelek deklarálásának szintaxisa**

```
wire [7:0] op1;
típus méret  név
```

- **Hasonló a portok deklarálásához, de nincs irány és a típus megadása nem hagyható el**
  - Típus: ***wire***, ***reg***
  - Méret: **[j : i]** → a jel mérete |j – i| + 1 bit
  - Ha nem adunk meg méretet, akkor a jel 1 bites lesz
- **A *wire* típusú jelhez a deklaráláskor érték is rendelhető, ez ekvivalens az *assign* utasítással**

```
wire [15:0] result = op1 & op2;
```

Verilog HDL ismeretek

Összeadás és kivonás operátor

- **Az összeadás (+) és kivonás (-) operátorok használata esetén a műveleti egység automatikusan generálódik**
- **Összeadó: használjuk a + (összeadás) operátort**
  - Az eredmény lehet 1 bittel szélesebb, ekkor az MSb a kimenő átvitel bit

```
wire [15:0] a, b;
wire        cin;
wire [15:0] sum1 = a+b;      //Nincs cin és cout
wire [15:0] sum2 = a+b+cin; //Van cin, nincs cout
wire [16:0] sum3 = a+b+cin; //Van cin és cout
```

- **Kivonó: használjuk a – (kivonás) operátort**
  - Nincs átvitel kimenet: használjunk helyette 1 bittel szélesebb kivonót (az MSb lesz a kimenő átvitel bit)

```
wire [15:0] a, b;
wire        cin;
wire [15:0] dif1 = a-b;      //Nincs cin és cout
wire [15:0] dif2 = a-b-cin; //Van cin, nincs cout
```

Verilog HDL ismeretek

Összeadás és kivonás operátor

- **Kivonó: használjuk a – (kivonás) operátort**
  - Nincs átvitel kimenet: használjunk helyette 1 bittel szélesebb kivonót (az MSb lesz a kimenő átvitel bit)

```
wire [16:0] a, b;
wire        cin;

assign a[15:0] = ……;          //”a” operandus
assign a[16]   = 1’b0;        //Előjel nélküli ”a”
//assign a[16] = a[15];       //Kettes kompl. ”a”

assign b[15:0] = ……;          //”b” operandus
assign b[16]   = 1’b0;        //Előjel nélküli ”b”
//assign b[16] = b[15];       //Kettes kompl. ”b”

wire [16:0] dif3 = a-b-cin; //Van cin és cout
```

  - Előjel nélküli operandusok esetén csak akkor kapunk helyes eredményt, ha a kisebbítendő nem kisebb a kivonandónál (azaz az eredmény nemnegatív)

Verilog HDL ismeretek

for ciklus

- A ***for*** ciklus megadásának szintaxisa

```
for([ini]; [felt]; [műv])
begin
   utasítás(ok);
end
```

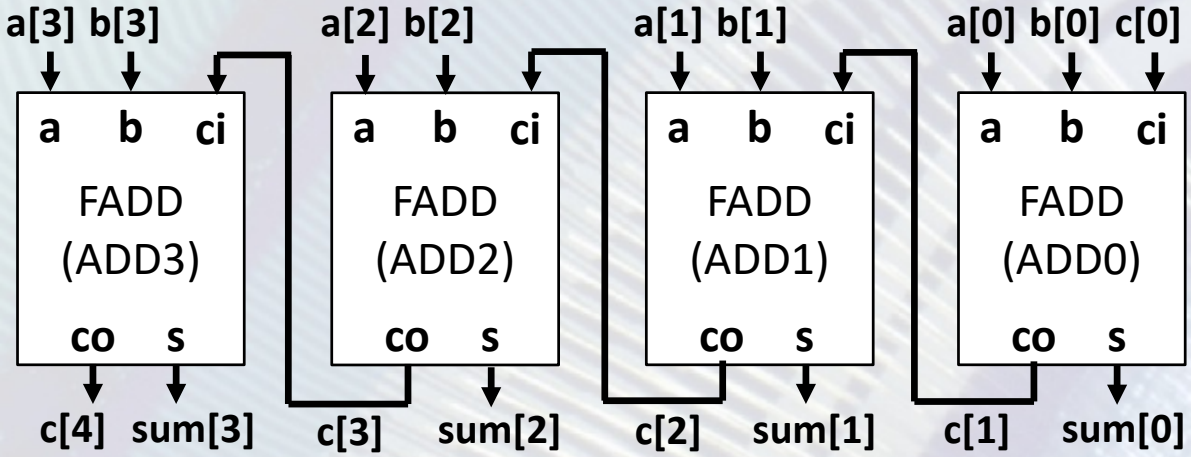
- Működése a következő
  1. Az **[ini]** inicializációs rész beállítja a ciklusváltozó kezdeti értékét, amit általában ***integer*** típusúnak deklarálunk az ***initial*** blokk előtt
  2. Kiértékelődik a **[felt]** feltétel, ha ez hamis, akkor kilépünk a ciklusból
  3. Végrehajtódik a megadott **utasítás (csoport)**.
  4. Végrehajtódik a **[műv]** művelet, majd ugrás a 2. pontra
- **Csak bizonyos blokkokban (pl. initial) használható!**
- **Hardver leírásához csak korlátozottan használható!**

1. feladat: hierarchikus tervezés

- **Teljes összeadókból készítünk egy 4 bites összeadót**
- **Elemezzük a bináris írásbeli összeadást**
- **A teljes összeadó a bináris írásbeli összeadás egy helyiértékéhez tartozó műveletet valósítja meg**
  - Összead 3 bitet: operandusok (a, b), bejövő átvitel (ci)
  - 2 bites eredmény: kimenő átvitel (co), összeg (s)
- **Töltsük ki a teljes összeadó igazságtábláját és ez alapján adjuk meg a kimenetek legegyszerűbb logikai függvényeit**
  - Mikor keletkezik átvitel?
  - Milyen tulajdonságú bemenetek esetén lesz az összeg 1?
- **Végezzük el az alábbi összeadásokat 4 bites eredménnyel. Mit tapasztalunk? Mi ennek az oka? Hogy tudjuk ezt „kijavítani”?**
  - Előjel nélküli: 12 + 7 ($1100_2 + 0111_2$)
  - Kettes komplemens: 5 + 6 ($0101_2 + 0110_2$)
  - Kettes komplemens: (-7) + (-3) ($1001_2 + 1101_2$)

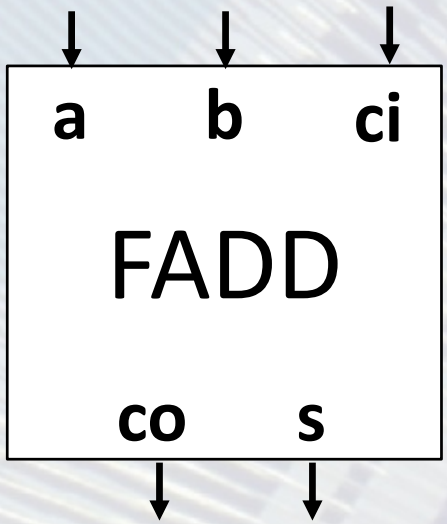
1. feladat: hierarchikus tervezés

- **Teljes összeadókból készítünk egy 4 bites összeadót**
- **A rendszer blokkvázlatát az alábbi ábra mutatja**
  - sw[3:0] → a[3:0] sw[7:4] → b[3:0]
  - sum[3:0] → ld[3:0] c[4] → ld[4]



1. feladat: hierarchikus tervezés

- **Nyissuk meg a kiindulási projektet**
- **Először a teljes összeadót (FADD modul) készítjük el**
  - Mik a kimenetek logikai függvényei?
  - Használjuk az ábrán látható port neveket



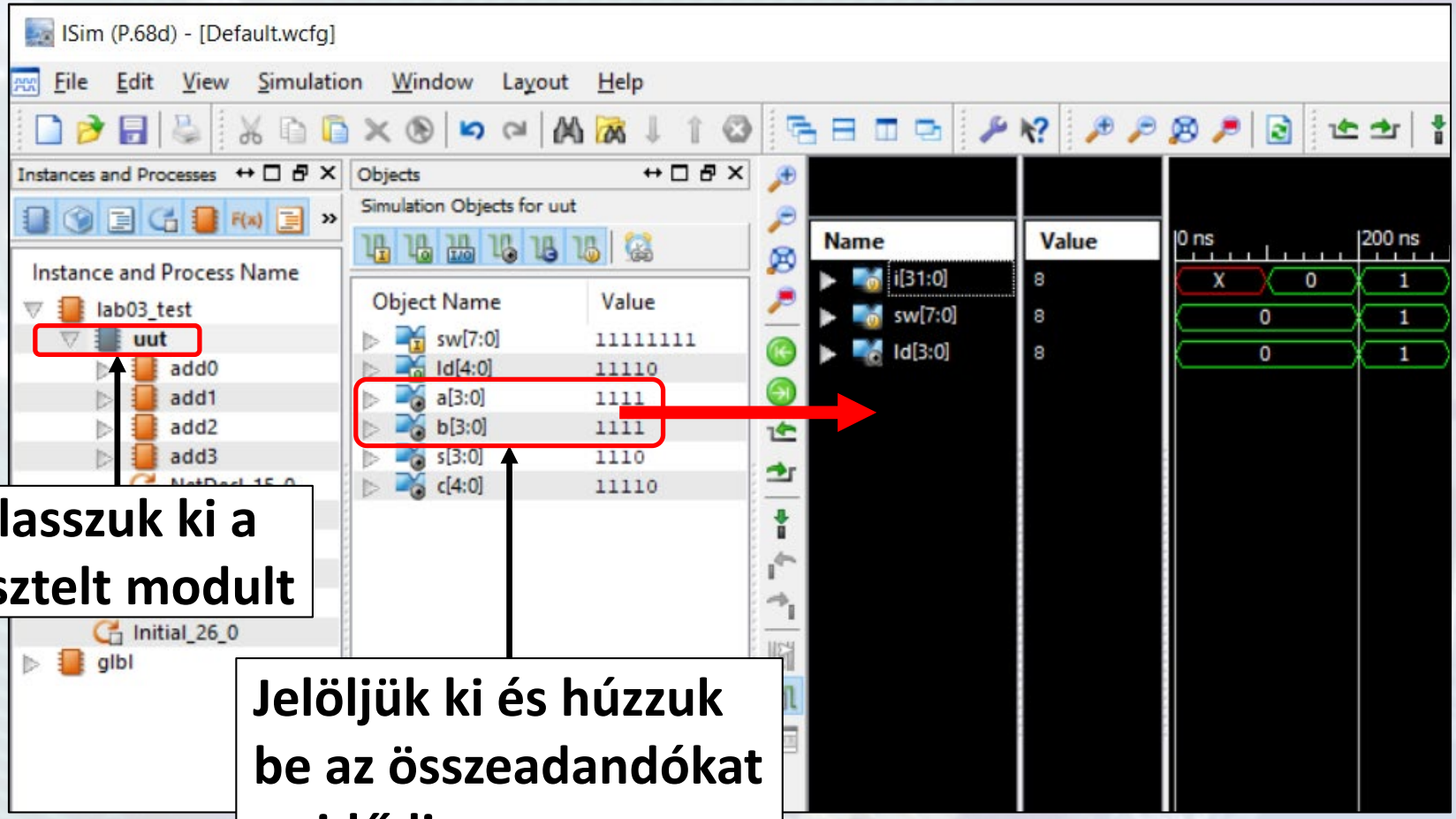
1. feladat: hierarchikus tervezés

- **A top-level modulban (lab03_1) hozzunk létre 4 fadd modul példányt és kössük be a portjaikat**
  - A kaszkádosítást az átvitel bitekkel végezzük el
    ***c[0] (=?)* → ADD0 → c[1] → ADD1 → c[2] → ADD2 → c[3] → ADD3 → c[4]**
  - Operandusok: sw[3:0] és sw[7:4]
  - Eredmény 5 biten: ld[4:0] (ld[4] = c[4])
- **Modul példányosítása**
  modul_név példány_név(port-jel társítás lista);
- **Port-jel társítási lista**
  .port_név1(jel_név1), .port_név2(jel_név2), …

1. feladat: hierarchikus tervezés

- **Szimulációval ellenőrizzük a működést**
  - A bemenetet *for* ciklussal állítsuk elő a tesztkörnyezetben
  - Tegyük ki a 4 bites összeadandókat az időDiagramra
    - Lásd a következő diát
  - A 4 bites összeadandók és az 5 bites kimenet számrendszerére (radix) állítsunk be előjel nélküli decimálist majd pedig előjeles decimálist (kettes komplemens)
    - Ellenőrizzük, hogy helyesek-e az eredmények
- **Próbáljuk ki az összeadót az FPGA kártyán előjel nélküli és kettes komplemens kódolású adatokkal**
- **Megjegyzés**
  - ***Az aritmetikai műveleteket Verilog nyelven sohasem így, hanem a megfelelő aritmetikai operátorokkal valósítjuk meg***
  - A feladat célja csak a hierarchikus tervezés bemutatása volt

1. feladat: hierarchikus tervezés



2. feladat: a + operátor használata

- **Az összeadó megvalósításához most az összeadás (+) operátort fogjuk használni**
- **Állítsuk be top-level modulként a lab03_2 modult és töltsük ki benne a hiányzó részeket**
- **Szimulációval ellenőrizzük a működést**
  - Az előző tesztkörnyezet jó itt is, a tesztelendő modul nevét kell csak benne módosítani
- **Próbáljuk ki az összeadót az FPGA kártyán előjel nélküli és kettes komplemens kódolású adatokkal**