Digitális Technika

Dr. Oniga István
Debreceni Egyetem, Informatikai Kar

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta.
A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

3. Laboratóriumi gyakorlat

A gyakorlat célja:

- Négy változós AND, OR, XOR és NOR függvények realizálása
- Szimulátor használata ciklussal generált tesztvektorokkal

Lab3_1 feladat:

- **Az AND, OR, XOR és NOR függvények realizálása**
  - Bemeneti változók: DIP kapcsoló alsó 4 bitje
  - Kimenet: alsó 4 LED
- **Szimulátor használata a tesztvektorokat ciklussal generálva**

Előző heti munka alapján a „DTL_2.pdf” diasorozatban ismertetett módon

- ISE elindítása, új projekt létrehozása
- Lab3_1a.v forrásfájl mintakeret specifikálása
- Nexys4.UCF fájl másolatának hozzáadása és adaptálása a kívánt interfészekhez
- A Lab3_1a feladat specifikálása a funkcionális kódrészletekkel
- Funkcionális kód ellenőrzése szimulációval
- Konfiguráció generálás, letöltés, működés tesztelése a kártyán

Lab3_1 feladat:

- **Egybites változókként használva a bemeneti jeleket**

```
21  module  Lab3_1a (
22      input [3:0] sw,
23      output [3:0] ld
24      );
25
26  assign ld[0] = sw[3] & sw[2] & sw[1] & sw[0]  ;    // 4 változó ÉS függvénye
27  assign ld[1] = sw[3] | sw[2] | sw[1] | sw[0]  ;    // 4 változó VAGY függvénye
28  assign ld[2] = sw[3] ^ sw[2] ^ sw[1] ^ sw[0]  ;    // 4 változó XOR függvénye
29  assign ld[3] = ~sw[3] & ~sw[2] & ~sw[1] & ~sw[0]  ;   // 4 változó NOR függvénye
30
31  endmodule
```

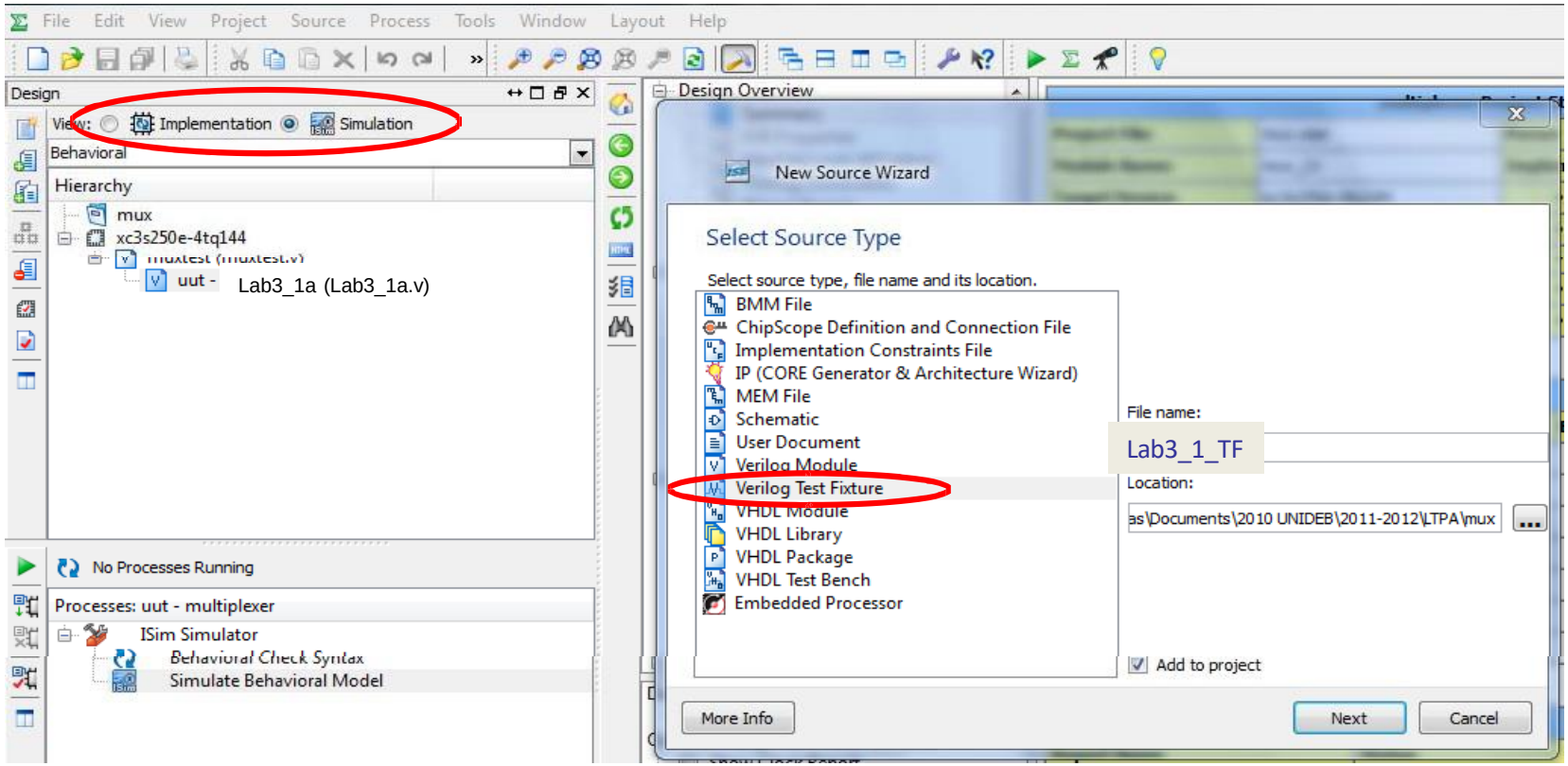
- **Ugyanez redukciós operátorokkal vektorváltozóra**

```
34  module  Lab3_1b (
35      input [3:0] sw,
36      output [3:0] ld
37      );
38
39  assign ld[0] = &sw[3:0];       // ÉS kapcsolat a 4 bites változó bitjeire // llll?
40  assign ld[1] = |sw[3:0];       // VAGY kapcsolat a 4 bites változó bitjeire
41  assign ld[2] = ^sw[3:0];       // XOR kapcsolat a 4 bites változó bitjeire
42  assign ld[3] = ~|sw[3:0];      // NOR kapcsolat a 4 bites változó bitjeire // 0000?
43
44  endmodule
```

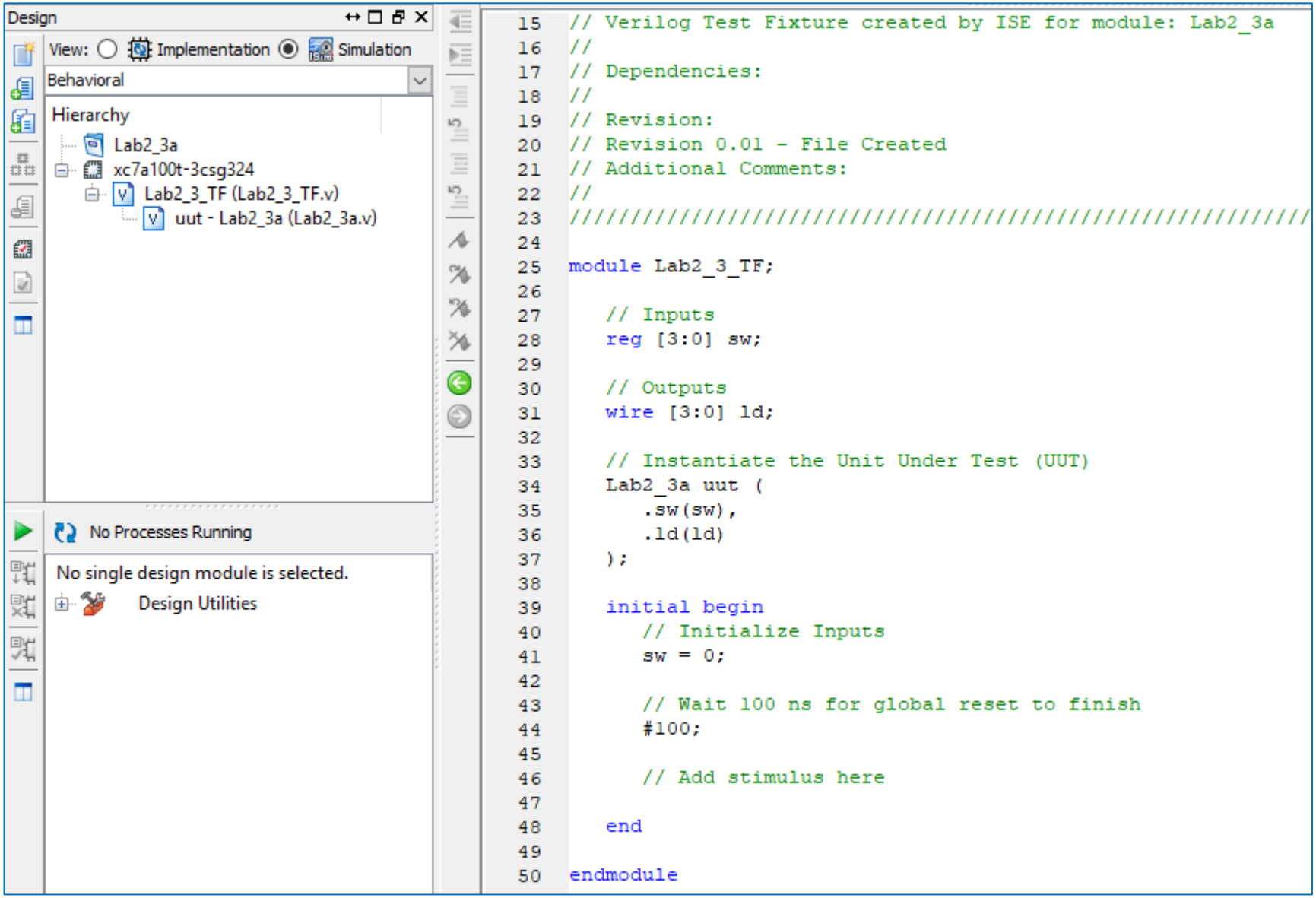
- **Nézzük meg a grafikus kapcsolásokat is (RTL + Tech.)**
  - A technológiai nézetben a LUT-ra: SCH+EQ+TT+KM

Lab3_1 feladat: szimuláció

- Átkapcsolás szimulációs módba
- Az tesztelési környezet létrehozása az egységet működtető gerjesztő jelek megadásával történik
- Első lépésként adjunk hozzá a projekt-hez egy új forrást: Project / New Source ablakban - Verilog Test Fixture opció. A fájl neve legyen : Lab3_1_TF !
- Ki válasszuk azt a modult, amelyhez a testbench-et generáljuk – jelen esetben egyetlen modulunk van.



Lab3_1 feladat: szimuláció




```
15  // Verilog Test Fixture created by ISE for module: Lab2_3a
16  //
17  // Dependencies:
18  //
19  // Revision:
20  // Revision 0.01 - File Created
21  // Additional Comments:
22  //
23  ////////////////////////////////////////////////////////////////////////////////
24
25  module Lab2_3_TF;
26
27     // Inputs
28     reg [3:0] sw;
29
30     // Outputs
31     wire [3:0] ld;
32
33     // Instantiate the Unit Under Test (UUT)
34     Lab2_3a uut (
35        .sw(sw),
36        .ld(ld)
37     );
38
39     initial begin
40        // Initialize Inputs
41        sw = 0;
42
43        // Wait 100 ns for global reset to finish
44        #100;
45
46        // Add stimulus here
47
48     end
49
50  endmodule
```

Tesztvektorok generálása

- Az automatikusan generált Verilog Test Fixture fájlt módosítjuk
- Négyváltozós függvények
  - Max. 16 kombináció
  - Teljes lefedést ad

Tesztvektorok generálása ***for*** ciklusban

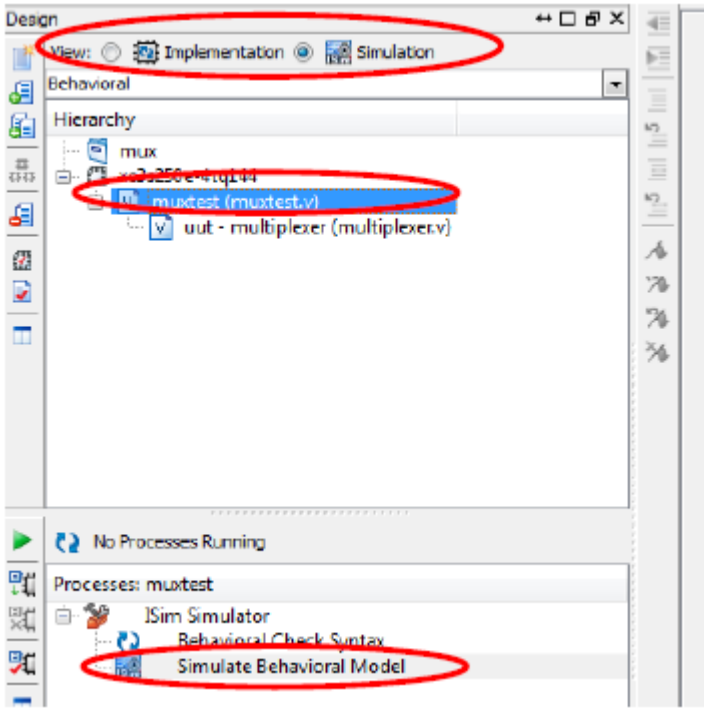
```
25  module Lab2_3_TF;
26     // Inputs
27     reg [3:0] sw;
28     // Outputs
29     wire [3:0] ld;
30     // Instantiate the Unit Under Test (UUT)
31     Lab2_3a uut (
32        .sw(sw),
33        .ld(ld)
34     );
35
36  integer i ;
37     initial begin
38        // Initialize Inputs
39        sw = 0;
40        // Wait 100 ns for global reset to finish
41        #100;
42        // Add stimulus here
43
44  // Teljes tesztvektorkészlet ciklussal generálva
45         for (i = 0 ; i<=15; i = i+1)
46         begin
47           #100 sw = i;
48         end
49
50     end
51  endmodule
```

Tesztvektorok generálása lineáris kóddal

```
25  module Lab2_3_TF;
26     // Inputs
27     reg [3:0] sw;
28     // Outputs
29     wire [3:0] ld;
30     // Instantiate the Unit Under Test (UUT)
31     Lab2_3a uut (
32        .sw(sw),
33        .ld(ld)
34     );
35
36  integer i ;
37     initial begin
38        // Initialize Inputs
39        sw = 0;
40        // Wait 100 ns for global reset to finish
41        #100;
42        // Add stimulus here
43        // Teljes tesztvektorkészlet lineáris felsorolással
44        #100    sw = 4'h0;
45        #100    sw = 4'h1;
46        #100    sw = 4'h2;
47        #100    sw = 4'h3;
48        #100    sw = 4'h4;
49        #100    sw = 4'h5;
50        #100    sw = 4'h6;
51        #100    sw = 4'h7;
52        #100    sw = 4'h8;
53        #100    sw = 4'h9;
54        #100    sw = 4'ha;
55        #100    sw = 4'hb;
56        #100    sw = 4'hc;
57        #100    sw = 4'hd;
58        #100    sw = 4'he;
59        #100    sw = 4'hf;
60
61     end
62  endmodule
```

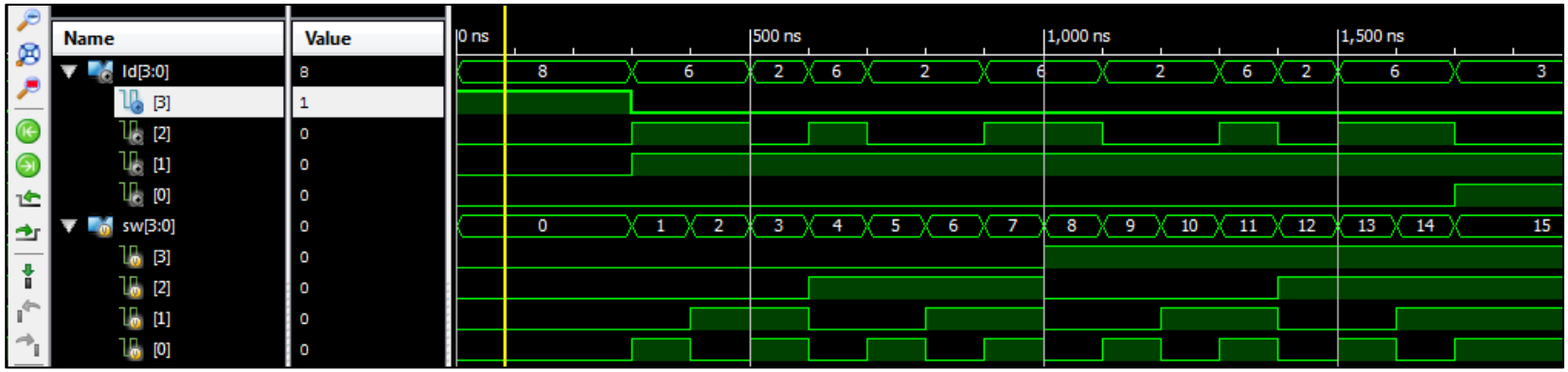
Szimuláció elindítása

- A **Project Navigator program View opciói közül válassza ki a Simulation-t, majd a Hierarchy** ablakban jelölje ki a testbench fájlt (*Lab3_1_TF).*
- *A **Processes ablakban indítsa el az ISim Simulator /**Simulate Behavioral Model programot.*
- **Szimulációhoz a Hierarchy ablakban a testbench file-t kell kiválasztani!!**



Lab3_1 feladat: ellenőrzés

- **A szimuláció eredménye**
  - LD[0] → ÉS, LD[1] → VAGY, LD[2] → XOR, LD[3] → NOR



- Generáljuk a specifikációhoz tartozó konfigurációs adatfájlt
- Letöltés után ellenőrizzük a működést
- A *Laboratórium 3. eredmények* kérdőíven jegyezzük fel a tapasztalatokat

Lab3_2a feladat

Érvénytelen BCD kód detektálása

- Ha SW[3:0] nem BCD, akkor mind a 4 LED világít, egyébként az érvényes bemeneti kód jelenjen meg
  - A 4 bites bemeneti kód 6 esetben nem felel meg
  - Tehát 6 minterm detektálása a feladat
- Felírható lenne másképpen is, a szintézer majd egyszerűsíti

```
50  ////////////////////////////////////////////////////////////////////////////////
51  // 2_1_2   Érvénytelen BCD kód detektáló függvény
52  //         Az SW[3:0] biteken beállított értéket átmásolja az LD[3:0] LED-ekre,
53  //         ha az kisebb, mint decimális 10, azaz érvényes BCD kódú digit
54  //         Ha nem, akkor az összes LED-et kigyújtja.
55  ////////////////////////////////////////////////////////////////////////////////
56  wire error;                     // Ez a jel jelzi a hibás BCD kódot
57
58  ////////////////////////////////////////////////////////////////////////////////
59  // A hiba 6 esetben fordulhat elő. Felírhatjuk ezt a hat esetet, azaz a 4 változós
60  // logikai függvény 6 mintermjét. A szintézer program a redundáns kifejezéseket
61  // ki fogja optimalizálni.
62  ////////////////////////////////////////////////////////////////////////////////
63  assign error =  (sw[3] & ~sw[2] &  sw[1] & ~sw[0]) |   // 10
64                  (sw[3] & ~sw[2] &  sw[1] &  sw[0]) |   // 11
65                  (sw[3] &  sw[2] & ~sw[1] & ~sw[0]) |   // 12
66                  (sw[3] &  sw[2] & ~sw[1] &  sw[0]) |   // 13
67                  (sw[3] &  sw[2] &  sw[1] & ~sw[0]) |   // 14
68                  (sw[3] &  sw[2] &  sw[1] &  sw[0]);    // 15
```

```
50  ////////////////////////////////
51  // 2_1_2   Érvénytelen BCD kód
52  //         Az SW[3:0] biteken b
53  //         ha az kisebb, mint d
54  //         Ha nem, akkor az öss
55  ////////////////////////////////
56  wire error;                    /
57
58  ////////////////////////////////
59  // A hiba 6 esetben fordulhat e
60  // logikai függvény 6 mintermjé
61  // ki fogja optimalizálni.
62  ////////////////////////////////
63  assign error = (sw == 4'd10) |
64                 (sw == 4'd11) |
65                 (sw == 4'd12) |
66                 (sw == 4'd13) |
67                 (sw == 4'd14) |
68                 (sw == 4'd15);
```

Lab 3_2a feladat

- **Felírhatjuk egyszerűbb formában is**

```
70  ////////////////////////////////////////////////////////////////////////////////
71  // Előre is gondolkodhatubnk. Mikor lesz hiba? Ha a 8+2 vagy a 8+4 esetek,
72  // azaz a 3. és 2. vagy a 3. és 1. biteken lesz egyidőben 1-es bit.
73  ////////////////////////////////////////////////////////////////////////////////
74  assign error = sw[3] & (sw[2] | sw[1]);     // Ez jelzi, ha sw[3:0] nem BCD kód
```

- **Vagy aritmetikai feltételként**

```
76  ////////////////////////////////////////////////////////////////////////////////
77  // Ez ugyanaz, mintha felírnánk a teljes 4 bites értékre vonatkozó aritmetikai
78  // feltételt
79  ////////////////////////////////////////////////////////////////////////////////
80  assign error = sw[3:0] > 4'd9;              //Ez is ugyanaz a feltétel!
```

- **Bármelyiket is választjuk, a teljes megoldás ilyen lesz**

```
82  ////////////////////////////////////////////////////////////////////////////////
83  // Ezután az error jellel kapuzzuk a megfeleleő LD[i] = SW[i] vezérlést.
84  // Ha ERROR = 1, akkor minden kimenet 1 lesz.
85  ////////////////////////////////////////////////////////////////////////////////
86  assign ld[0] = sw[0] | error;
87  assign ld[1] = sw[1] | error;
88  assign ld[2] = sw[2] | error;
89  assign ld[3] = sw[3] | error;
```

Lab 3_2a feladat

A legtömörebb felírási mód

```
91  ////////////////////////////////////////////////////////////////////////////////
92  // Egyetlen sorban is specifikálhatjuk a feltételes kifejezést.
93  ////////////////////////////////////////////////////////////////////////////////
94  assign ld[3:0] = (sw[3:0] > 4'd9) ? 4'b1111 : sw[3:0];
```

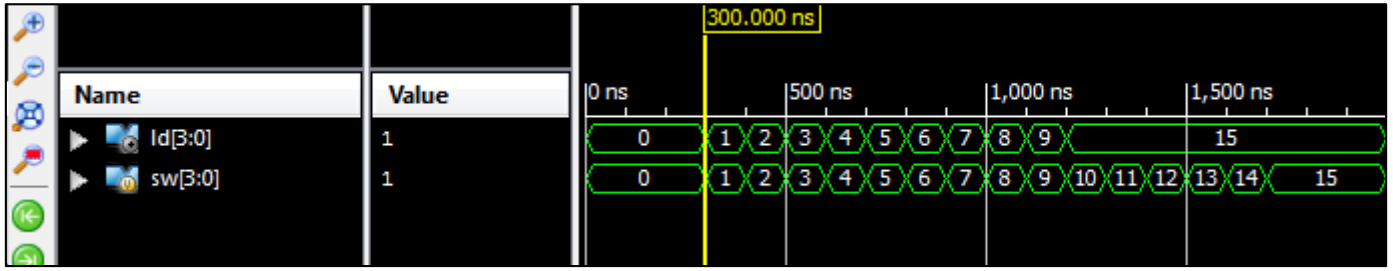
- Itt az error jel meg sem jelenik explicit módon
- A szintézer persze minden bitre a lehető legegyszerűbb logikát generálja

```
ld[3] = sw[3]                                          //1 vált.
        ;
ld[2] = sw[2] | sw[3] & sw[1];                         //3 vált.
ld[1] = sw[1] | sw[3] & sw[2];                         //3 vált.
ld[0] = sw[0] | sw[3] & sw[2] | sw[3] & sw[1]; //4 vált.
```

- Nézzük meg → View Technology Schematic

Lab 3_2a feladat: szimuláció, ellenőrzés

- Váltsunk át szimulációs módra
- A szimulációhoz a korábban már használt Lab3_1_TF.v Verilog Test Fixture már készen van, a 4 bites bemenethez 16 kombinációt generál



- Generáljuk a specifikációhoz tartozó konfigurációt
- Letöltés után ellenőrizzük a működést
- A *Laboratórium 3. eredmények* kérdőíven jegyezzük fel a tapasztalatokat

Lab3_2b feladat

- 4 bites érték 3-mal vagy öttel oszthatóságának jelzése
- Ez sajnos semmilyen módon nem egyszerűsíthető, nincsenek összevonható mintermek
- Esetleg felírhatjuk aritmetikai formában, de így sem tűnik egyszerűbbnek

```
96  ////////////////////////////////////////////////////////////////////////////////
97  // 2_1_3  4 bites érték 3-mal vagy 5-tel oszthatóságának jelzése
98  //        A bemeneti értékek tartománya 0...15. A 'jó' értékek 0,3,5,6,9,10,12,15.
99  //        Ez közvetlenül a mintermek felsorolása is, azaz közvetlenül a
100 //        diszjunktív normál alak. Y = SZUMMA(mi), ahol i a fentieknek felel meg.
101 ////////////////////////////////////////////////////////////////////////////////
102 assign ld[0] = (~sw[3] & ~sw[2] & ~sw[1] & ~sw[0]) |     //  0
103                (~sw[3] & ~sw[2] &  sw[1] &  sw[0]) |     //  3
104                (~sw[3] &  sw[2] & ~sw[1] &  sw[0]) |     //  5
105                (~sw[3] &  sw[2] &  sw[1] & ~sw[0]) |     //  6
106                ( sw[3] & ~sw[2] & ~sw[1] &  sw[0]) |     //  9
107                ( sw[3] & ~sw[2] &  sw[1] & ~sw[0]) |     // 10
108                ( sw[3] &  sw[2] & ~sw[1] & ~sw[0]) |     // 12
109                ( sw[3] &  sw[2] &  sw[1] &  sw[0]);      // 15
110
111 assign ld[3:1] = 3'b000;           // A többi LED kikapcsolva
```

```
96  ////////////////////////////////////
97  // 2_1_3  4 bites érték 3-mal vagy 5-
98  //        A bemeneti értékek tartomá
99  //        Ez közvetlenül a mintermek
100 //        diszjunktív normál alak. Y
101 ////////////////////////////////////
102 assign ld[0] = ( sw[3:0] == 4'h0) |
103                ( sw[3:0] == 4'h3) |
104                ( sw[3:0] == 4'h5) |
105                ( sw[3:0] == 4'h6) |
106                ( sw[3:0] == 4'h9) |
107                ( sw[3:0] == 4'ha) |
108                ( sw[3:0] == 4'hc) |
109                ( sw[3:0] == 4'hf);
110
111 assign ld[3:1] = 3'b000;
```

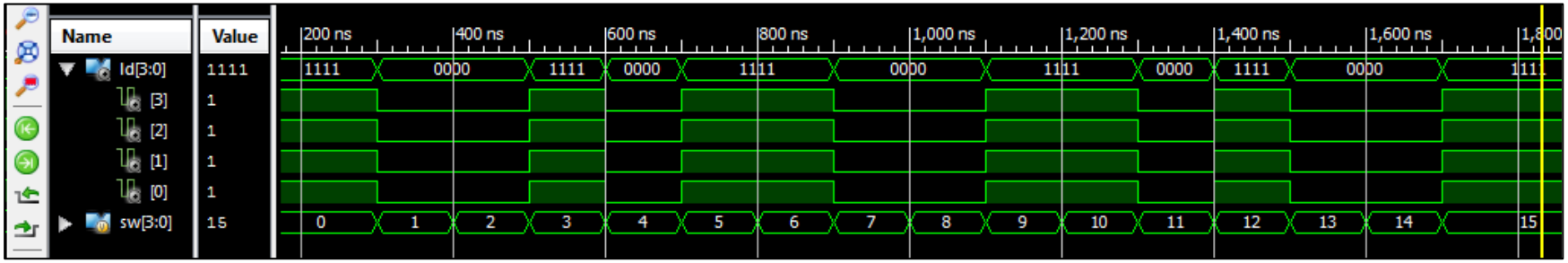
Lab3_2b feladat

- **A komplexitás ijesztőnek látszik**
  - View RTL Schematic → 8 db AND4, 1 db OR8
  - View Technology Schematic → 1 db LUT ?????????
- **Alaposabban megvizsgálva észrevehetünk valamit**

```
113  //////////////////////////////////////////////////////////////////////////////////
114  // A bemeneti kódszavakat felírva azonban észrevehetjük, hogy mindig 2 bit aktív,
115  // tehát a kódok páros paritásúak. Erre viszont van közvetlen logikai függvényünk
116  // az XNOR kapcsolat
117  //////////////////////////////////////////////////////////////////////////////////
118
119  assign ld[2] = ~(sw[3] ^ sw[2] ^ sw[1] ^ sw[0]); //4 változó XNOR függvénye
120
121  assign ld[3] = ~^sw[3:0];                        //Ugyanez redukciós operátorral
122
```

Lab3_2b feladat: szimuláció, ellenőrzés

- **Váltsunk át szimulációs módra**
- **A szimulációhoz a korábban már használt Lab3_1_TF.v Verilog Test Fixture már készen van**
  - A négyfajta leírás a 4 LED-en azonosan működik



- Generáljuk a specifikációhoz tartozó konfigurációt
- Letöltés után ellenőrizzük a működést
- A *Laboratórium 3. eredmények* kérdőíven jegyezzük fel a tapasztalatokat