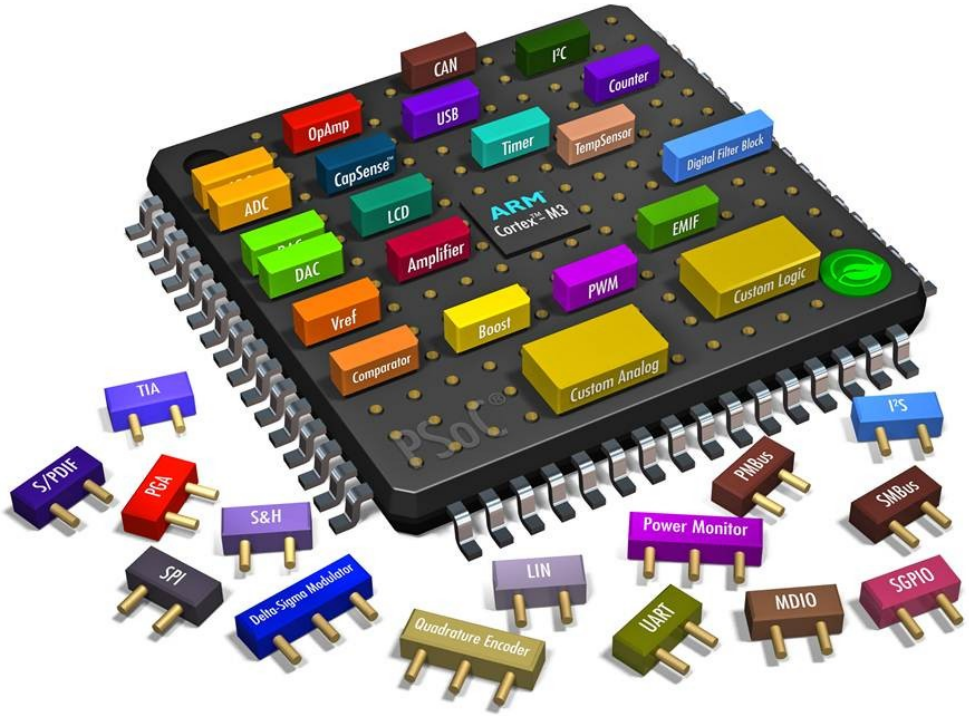
Újrakonfigurálható eszközök



2. Verilog HDL alapok

Végh János: Bevezetés a Verilog hardverleíró nyelvbe c. jegyzete nyomán

Tartalom

- Mi az a Verilog?
- Az Icarus Verilog használata szimulációhoz
- Verilog nyelvi elemek
  - Folyamatos értékadás
  - Vezetékek
- Verilog modulok hierarchiája
- Verilog próbapad (test bench)
- A GtkWave hullámalak-megjelenítő használata

Mi az a Verilog?

- A **Verilog HDL** egy szabványosított hardverleíró nyelv (Hardware Description Language), elektronikus áramkörök és rendszerek szöveges leírására szolgál.
- Széleskörűen használják digitális elektronikus áramkörök tervezésére és szimulációval történő verifikálásra, időzítés vizsgálatra, teszt jellegű analízisre, és logikai szintézisre.
- A **Verilog-AMS** nyelv analóg és kevertjelű áramkörök leírására alkalmas szerkezetekkel bővíti ki a Verilog szabványt.
- A **System Verilog** nyelv a Verilog HDL és a konkurrens VHDL előnyös tulajdonságait próbálja egyesíteni – megőrizve a visszafelé kompatibilitást a Verilog 2001-gyel.

A Verilog története röviden

- 1984 - Gateway Design Automation bevezeti Verilog-XL
- rendszert
- 1989 - Cadence Design Systems megvásárolja a Gateway-t
- 1990 - A Verilog HDL valamennyi jogát átruházzák az
- Open Verilog International-re (OVI, ma Accellera)
- 1995 - Az IEEE Standard 1364 közzététele
- 2001- "Verilog 2001"
- A jövő – Verilog-AMS, System Verilog
  - Verilog-AMS: Analóg és kevertjelű bővítmények a Verilog-hoz
  - Sytem Verilog: a VHDL-ben bevált eszközök beépítése a Verilog-ba

Icarus Verilog

- Az **Icarus Verilog** egy Verilog szimulációs és szintézis eszköz. A nyíltforrású programot Stephen Williams írta és GPL licenc alatt terjeszti. Honlap: http://iverilog.icarus.com/
- Fordítóként működik, IEEE-1364 szabvány szerinti Verilog HDL forráskódot fordít különféle formátumokra.
- **Szimulációhoz** egy közbenső (**vvp** assembly) formátumra fordít, amelyet a **vvp** nevű programmal dolgozhatunk fel.
- **Szintézishez** az **Icarus Verilog** fordító kötéslistákat generál, különféle formátumban.
- Az **Icarus Verilog** telepítője Windowshoz innen tölthető le: http://bleyer.org/icarus/ (e sorok írásának idején az aktuális változat: iverilog-10.1.1-x64_setup.exe)
- **GtkWave** – grafikus segédprogram a szimulációban kapott hullámforma jelek vizsgálatához. Honlap: gtkwave.sourceforge.net

Icarus Verilog: az első lépések

- Az **Icarus Verilog** használatához a telepítési könyvtár **bin** mappájának benne kel lennie az elérési útban. Például:

```
> PATH = %PATH%;C:\iverilog\bin
```

- Fordítsuk le a **hello.v** forráskódot közbenső kódra:

```
> iverilog -o hello hello.v
```

Itt az **-o** opció adja meg a kimenő fájl nevét.

- Végül futtassuk a szimulációt:

```
> vvp hello
Hello, World
```

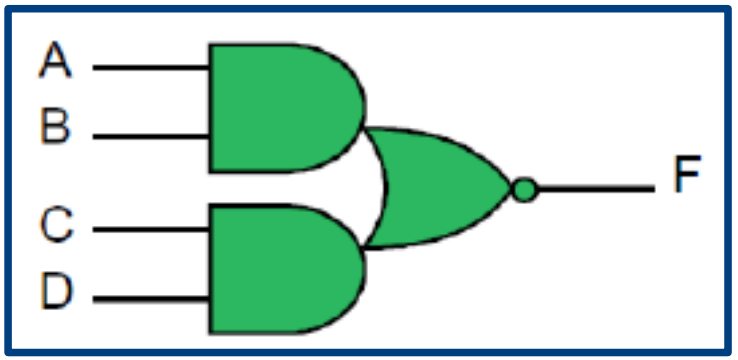
hello.v

```
module main;
  initial
    begin
      $display("Hello, World");
      $finish;
    end
endmodule
```

Verilog nyelvi elemek

- **Folyamatos értékadás**

```
// Egy AND-OR-INVERT áramkör
module AOI(A, B, C, D, F);
  input A, B, C, D;
  output F;
  assign F = ~((A&B)|(C&D));
endmodule
```



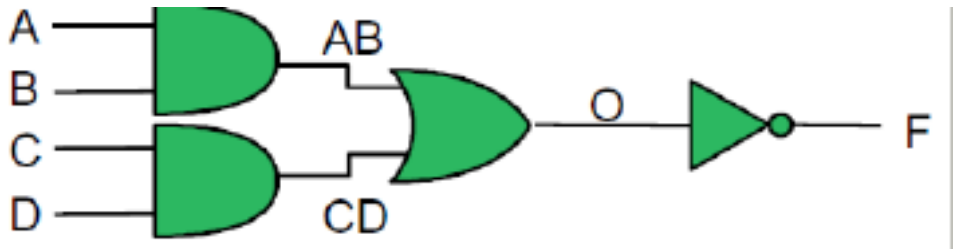
Ez a sor adja meg a funkcionalitást!

- A **`module`** kulcsszót a modul neve követi, majd zárójelben a **portok** (a modul ki- és bemeneti csatlakozási pontjai) következnek. Minden modul egy hardver blokkot ír le (be- és kimenetek, funkció).
- A fenti modul egy **folyamatos értékadó utasítást** tartalmaz, ami egyúttal a modul funkcióját is megadja. A bal oldal folyamatosan frissül, hogy tükrözze a jobb oldali kifejezés értékét.
- A szimuláció során a **folyamatos értékadó utasítás** akkor hajtódik végre, amikor a jobb oldalon szereplő négy bemenőjel (A, B, C és D) valamelyikének értéke megváltozik.

Vezetékek, értékadás

- A belső jeleket (hálózati csomópontokat) vezetékként kell definiálnunk.
- A portok alapértelmezetten vezetékek, tehát az F kimenetet nem szükséges vezetékként definiálni (opcionális).
- Az előző AND-OR-INVERT példa logikai kapukra lebontva így néz ki:

```
module AOI(A, B, C, D, F);
  input A, B, C, D;
  output F;
  wire F;              // Alapértelmezett (elhagyható)
  wire AB, CD, O;      // Ez viszont kell
  assign AB = A & B;
  assign CD = C & D;
  assign O = AB | CD;
  assign F = ~O;
endmodule
```



Többszörös értékadás vagy deklaráció

- Az előző oldalon definiált AOI modul ismétlődő elemei (értékadás, vezetékek deklarálása) összevonhatók:

Az AOI modul többszörös értékadással

```
module AOI(A, B, C, D, F);
  input A, B, C, D;
  output F;
  assign AB = A & B,
         CD = C & D,
         O = AB | CD,
         F = ~O;
endmodule
```

Az AOI modul többszörös deklarálással

```
module AOI(A, B, C, D, F);
  input A, B, C, D;
  output F;
  wire AB = A & B,
       CD = C & D,
       O = AB | CD,
       F = ~O;
endmodule
```

- Végeredményben mindhárom megadási mód ekvivalens.

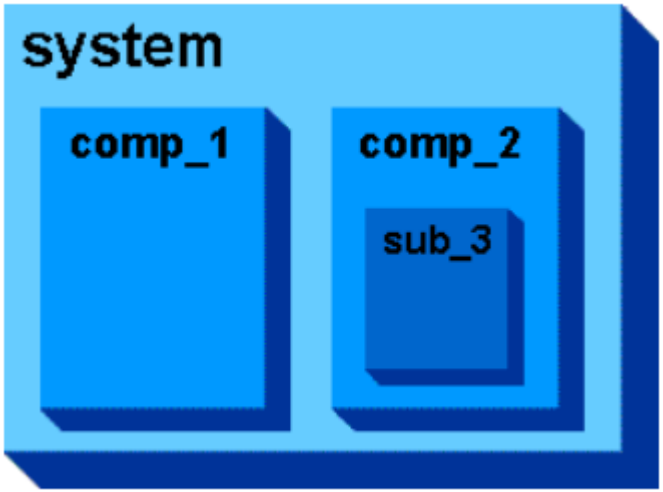
Az összekötések típusai

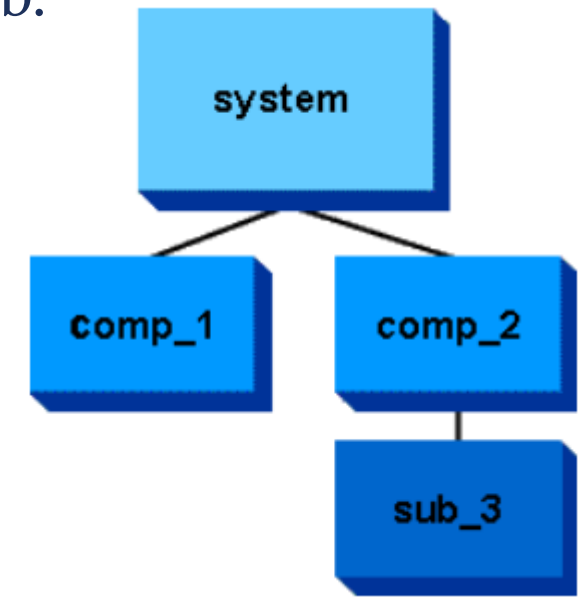
- Az összekötések (**net**) az áramkör pontjai közötti elektromos összekötéseket ábrázolják. A **net** (összekötés) típusai:

wire, tri	Közönséges vezeték, háromállapotú busz
wand, triand	Huzalozott ÉS
wor, trior	Huzalozott VAGY
tri0	Lehúzó ellenállás (pulldown)
tri1	Felhúzó ellenállás (pullup)
trireg	Kapacitív töltéstárolás
supply0	Tápegység közös pont (GND)
supply1	Tápfeszültség (POWER)

Modulok hierarchiája

- Minden Verilog modell (terv) modulokból épül fel.
- A modulok hivatkozhatnak más modulokra (példányosítás), így egy hierarchiát hoznak létre.
- Mindegyik példány független, egyidejűleg aktív.
- Az alábbi modell négy modulból áll *(system, comp_1, comp_2, sub_3)*. A modul hierarchia többféle képpen is ábrázolható. A jobboldali fadiagram az elterjedtebb.



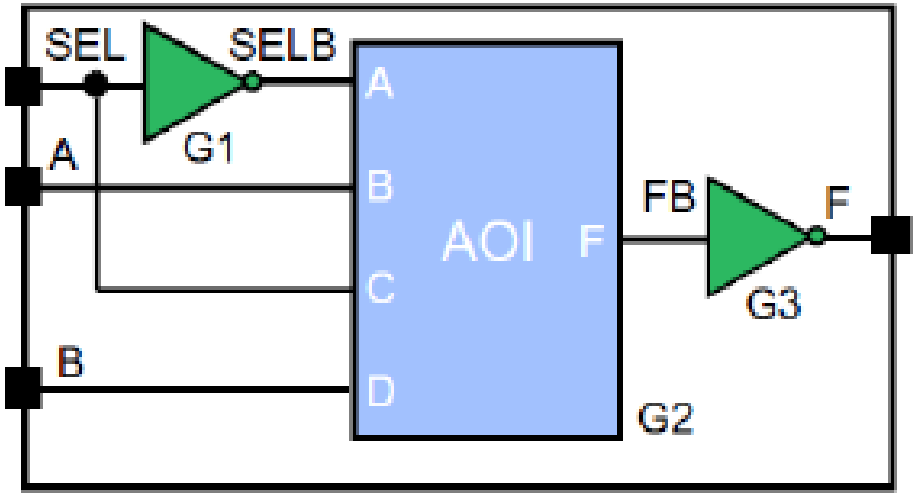


Forrás: Verilog online tutorial http://vol.verilog.com/VOL/main.htm

Modul hierarchia, sorrendi leképezéssel

- Terv: 2-ből 1-re multiplexer, a korábbi AOI modul felhasználásával

```
module AOI(A,B,C,D,F);
. . .
endmodule

module INV(A,F);
. . .
endmodule

module MUX2(SEL,A,B,F) ;
input SEL,A,B;
output F;
  INV G1(SEL,SELB) ;          //Modul példányosítás
  AOI G2(SELB,A,SEL,B,FB); //SELB és FB implicit wire típusú
  INV G3(FB,F) ;              //Másik modul példányosítás
endmodule
```

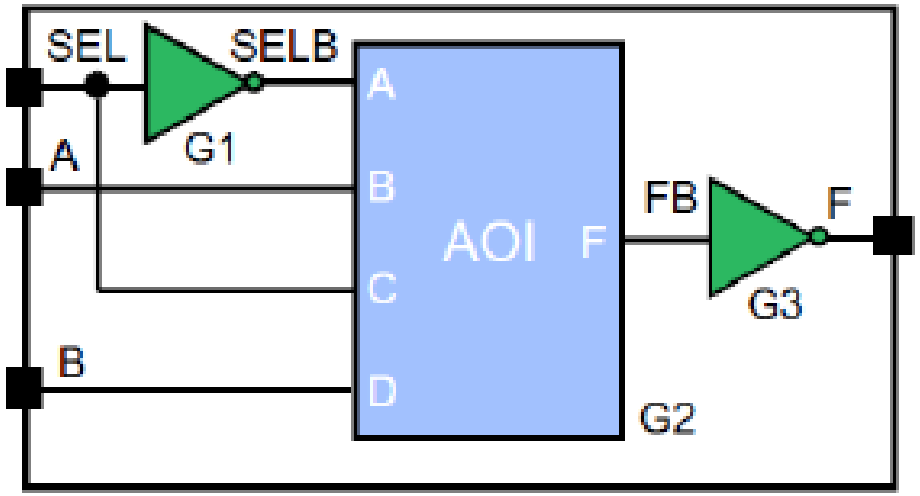


- A példányosításnál a portok megfeleltetése itt a **sorrend** alapján történt.
- A **név szerinti** megfeleltetést a következő oldalon mutatjuk be.

Modul hierarchia, név szerinti leképezéssel

- Terv: 2-ből 1-re multiplexer, a korábbi AOI modul felhasználásával

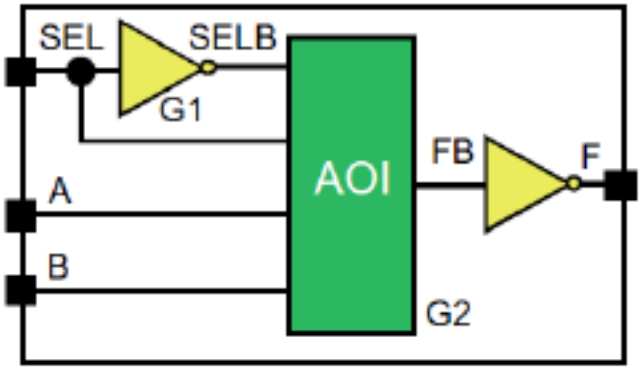
```
module AOI(A,B,C,D,F);
. . .
endmodule
module INV(A,F);
. . .
endmodule

module MUX2(SEL,A,B,F) ;
input SEL,A,B;
output F;
  INV G1(.A(SEL),.F(SELB)); // Név szerinti leképezés
  AOI G2(.A(SELB),.B(A),.C(SEL),.D(B),.F(FB));
  INV G3(.F(F),.A(FB));     // A sorrend nem számit
endmodule
```



- A név szerinti megfeleltetésnél mindegy, hogy milyen sorrendben adjuk meg a paramétereket.

Primitívek használata

- A **Verilog primitívek** (amelyeket kapuknak is nevezhetünk) előredefiniált modulok. Példányosításuknál csupán egy eltérés van: a példányoknak nem kötelező nevet adni.




```
module MUX2(SEL,A,B,F);
input SEL,A,B;
output F;
  not G1(SEL,SELB);              //primitív példányosítása
  AOI G2(SELB,A,SEL,B,FB);
  not (FB,F);                    //a név opcionális
endmodule
```

- A primitívek típusai:

1-kimenet	1-bemenet	tristate	pull
and, nand or, nor xor, xnor	buf, not	bufif0,notif0 bufif1,noif1	pullup pulldown

Be nem kötött portok

- Példányosításkor a modul példány bizonyos portjait bekötetlenül hagyhatjuk.

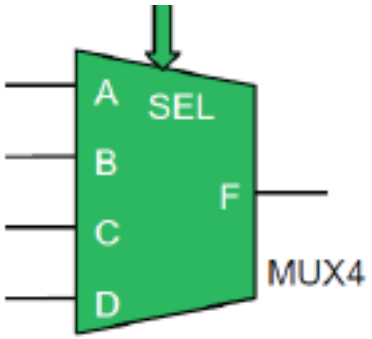
```
module AOI(A,B,C,D,F);
. . .
endmodule

module ...
//Nincs bekötve a C port
  AOI AOI1 (W1,W2, ,W3,W4);
  ...
endmodule
```

- Név szerinti leképezés esetén kétféle módon is bekötetlenül hagyhatunk egy portot:
  - Teljesen elhagyjuk a portot: `AOI AOI1(.F(W4),.D(W3),.B(W2),.A(W1));`
  - üresen hagyjuk a zárójelet: `AOI AOI1(.F(W4),.D(W3),.B(W2),.A(W1),.C());`

Vektor portok

- A portok és vezetékek nemcsak egyes biteket ábrázolhatnak, hanem buszokat is.

```
module MUX4 (SEL,A,B,C,D,F);
  input [1:0]SEL;  // különálló
  input A,B,C,D;   // input deklaráció
  output F;
  // . . .
endmodule
```



- A fenti 4-bemenetű multiplexernél a kétbites kiválasztó bemenetet egyetlen egységbe, egy kételemű vektorportba fogtuk össze.
- A 2016/2017 évad **digi10** előadásában bemutatott **10_key_digital_tube_display** nevű projekben is láttunk már vektor portokat:

```
input [3:0] key;      // value of the input key code
output [3:0] digit;   // digit control bit selection
output [7:0] segment; // digital tube segment code ABCDEFGH
```

Verilog logikai értékek és értékvektorok

- A logikai érték lehet:

  1‘b0 - logikai 0, false, föld (GND)

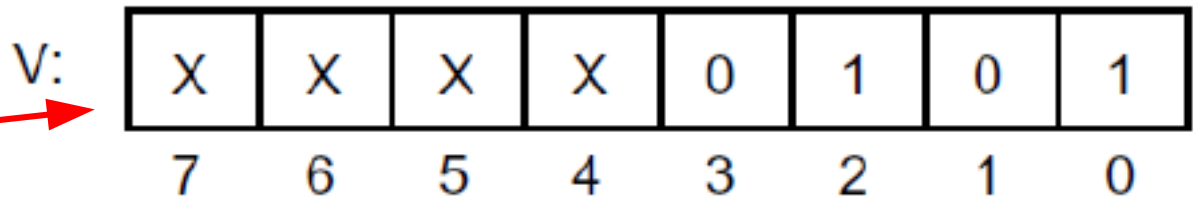
  1‘b1 - logikai 1, true, tápfeszültség

  1‘bX - ismeretlen vagy inicializálatlan

  1‘bZ - nagy impedanciájú, lebegő

- A Verilog értékvektor logikai értékek sorozata (0, 1, X és Z). Az értéket megelőzi a vektor bitjeinek száma és a számrendszer alapjának jele.

```
reg [7:0] v;
v = 8'bxxxx0101;
```



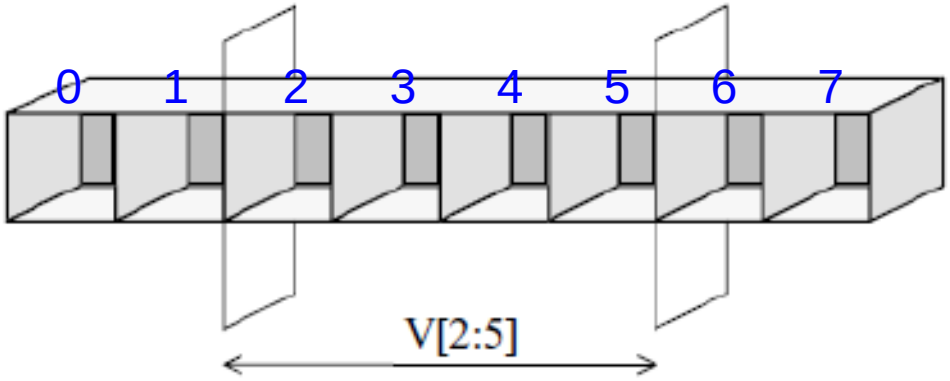
- Egy vektor egyes bitjeit külön is kezelhetjük, ha szögletes zárójelben megadjuk a bit sorszámát (bit select).

```
v[7] = 1'b0;
v[6] = B & v[0];
```

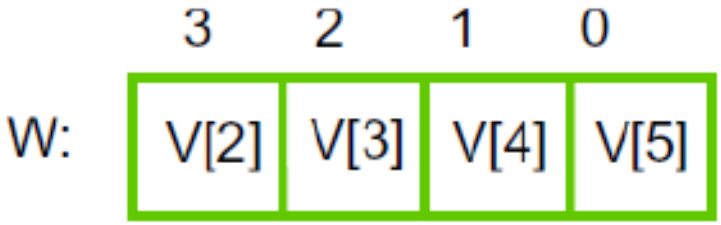
Vektor részének kiválasztása

- Egy *rész* kiválasztás egy vektor folytonos index tartományú részének kiválasztását jelenti. A kiválasztott résztartomány csak ugyanolyan irányú lehet, mint amilyen az eredeti deklarációban szerepelt.

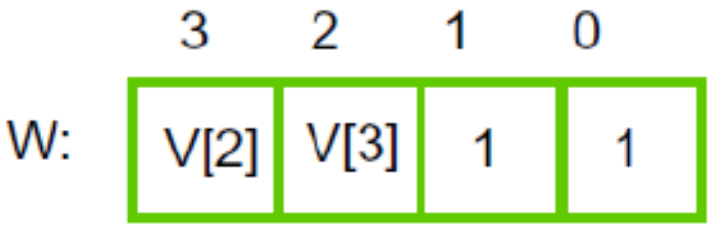
```
reg [0:7] v;
reg [3:0] w;
```




```
w = v[2:5];
```

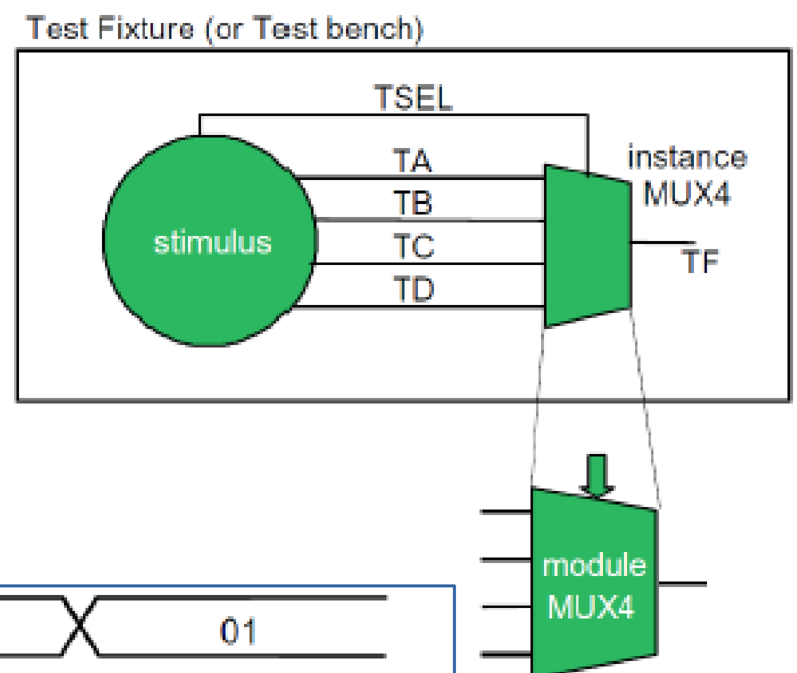



```
w[1:0] = 2'b11;
```



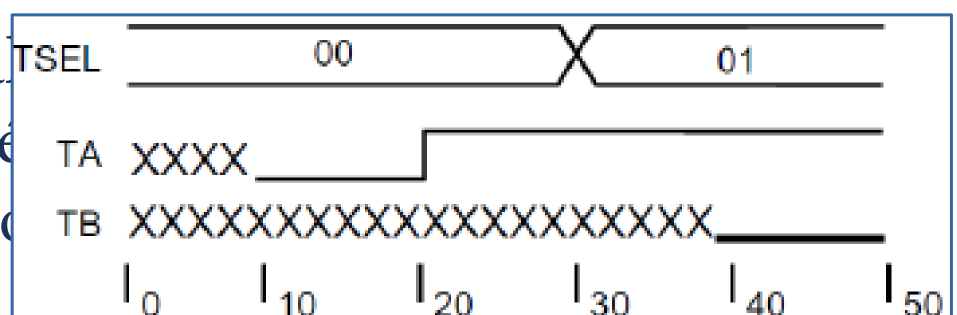
Próbapad (test bench) készítése

- A Verilog modellek szimulátorban történő kipróbálásához gondoskodnuk kell a bemenetek meghajtásátról (stimulus) és a kimenetek megfigyeléséről (pl. naplózás). Ehhez egy speciális modult kell definiálnunk, amit próbapadnak (test bench) vagy teszt fixtúrának hívnak.




```
module MUX4TEST;
  ...
  initial
  ...        //Bemenetek beállítása
  MUX4 M (.SEL(TSEL),.A(TA),
  .B(TB),.C(TC),.D(TD),.F(TF));
  initial
  ...        //Eredmények kiíratása
endmodule
```

- A própabad két ’initia
(stimulus és az eredm
valamint a vizsgáland

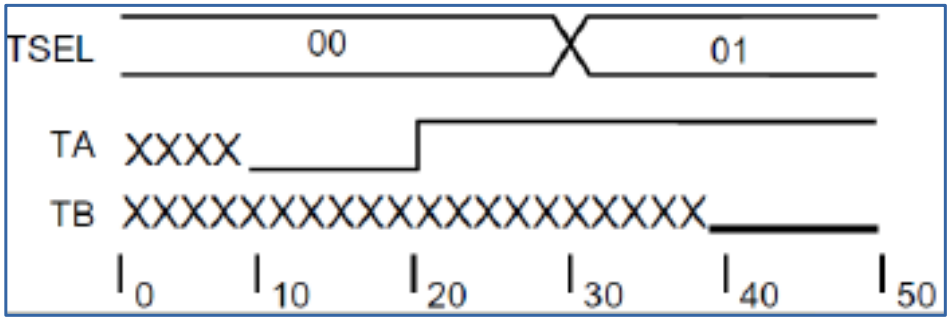


- A própadpadok nem tartalmaznak portokat.

Példa a vizsgálójelek előállítására

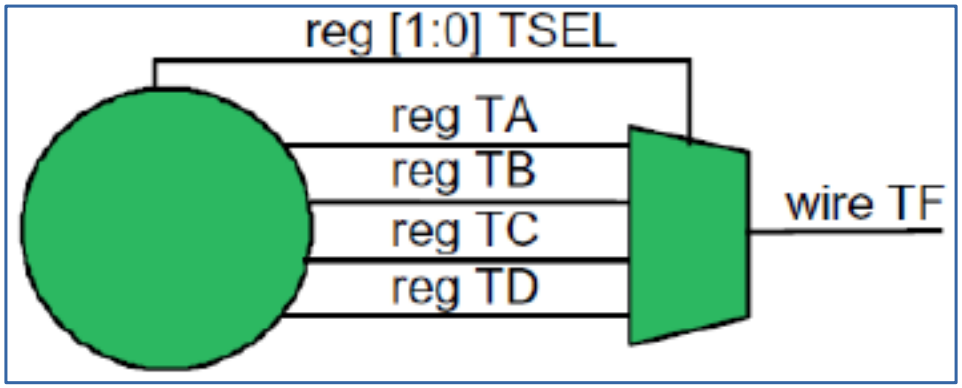
- Az **initial** egy procedurális blokk (a másik procedurális blokk az **always** blokk). Az **`initial`** blokk végrehajtása a szimuláció kezdetekor indul (nulla idopontban) és a **`begin`** és **`end`** közötti utasítások felülről lefelé, sorban egymás után hajtódnak végre.

```
// Egyszer hajtódik végre, felülről lefelé
initial                    // Stimulus
  begin
    TSEL = 2'b00;
    #10 TA = 0;            // 1'b0 röviden
    #10 TA = 1;
    #10 TSEL = 2'b01;
    #10 TB = 0;            // Procedurális értékadás
    //^ Procedurális késleltetés
  end
```

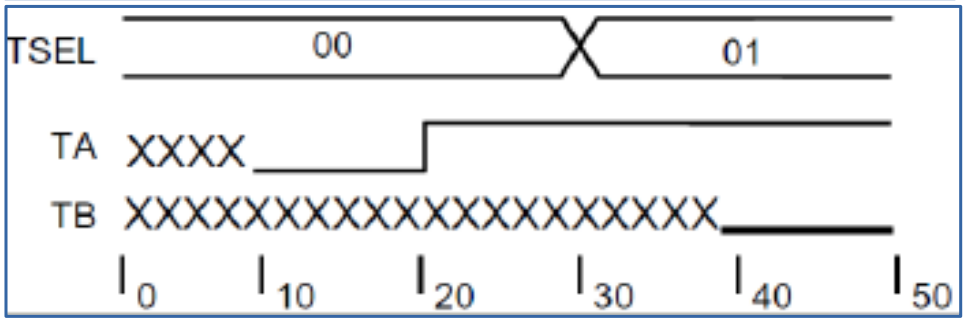


Regiszterek használata

- A procedurális blokkokban **regisztereket** használunk, s csak regiszteknek adhatunk értéket. A próbapadot tehát így kell kiegészítenünk (**mux4test.v**):




```
module MUX4TEST;
  reg TA, TB, TC, TD;
  reg [1:0] TSEL;
  initial
    begin
      TSEL = 2'b00;
      #10 TA = 0;
      #10 TA = 1;
      #10 TSEL = 2'b01;
      #10 TB = 0;
    end
  MUX4 M (.SEL(TSEL),.A(TA),
  .B(TB),.C(TC),.D(TD),.F(TF));
  initial
    $monitor($time, ,TSEL, ,TA,TB,TC,TD, ,TF);
endmodule
```




```
 0  0  XXXX  X
10  0  0XXX  0
20  0  1XXX  1
30  1  1XXX  X
40  1  10XX  0
```

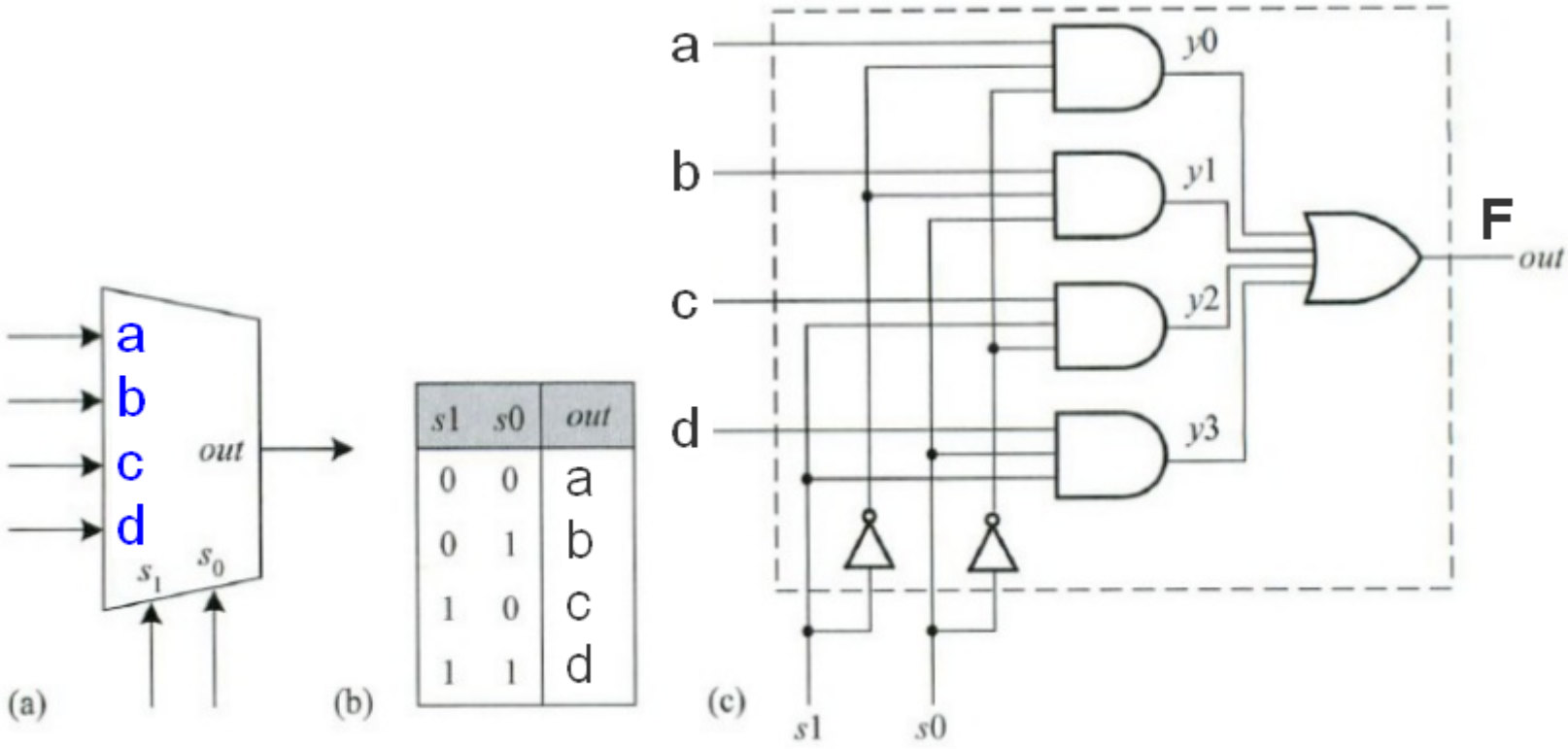
- A **$monitor** rendszertaszk
- A **$time** rendszerfüggvény

4-to-1 multiplexer

- Állítsuk össze a 4 vonalról 1 vonalra multiplexelő modellünket!

mux4.v

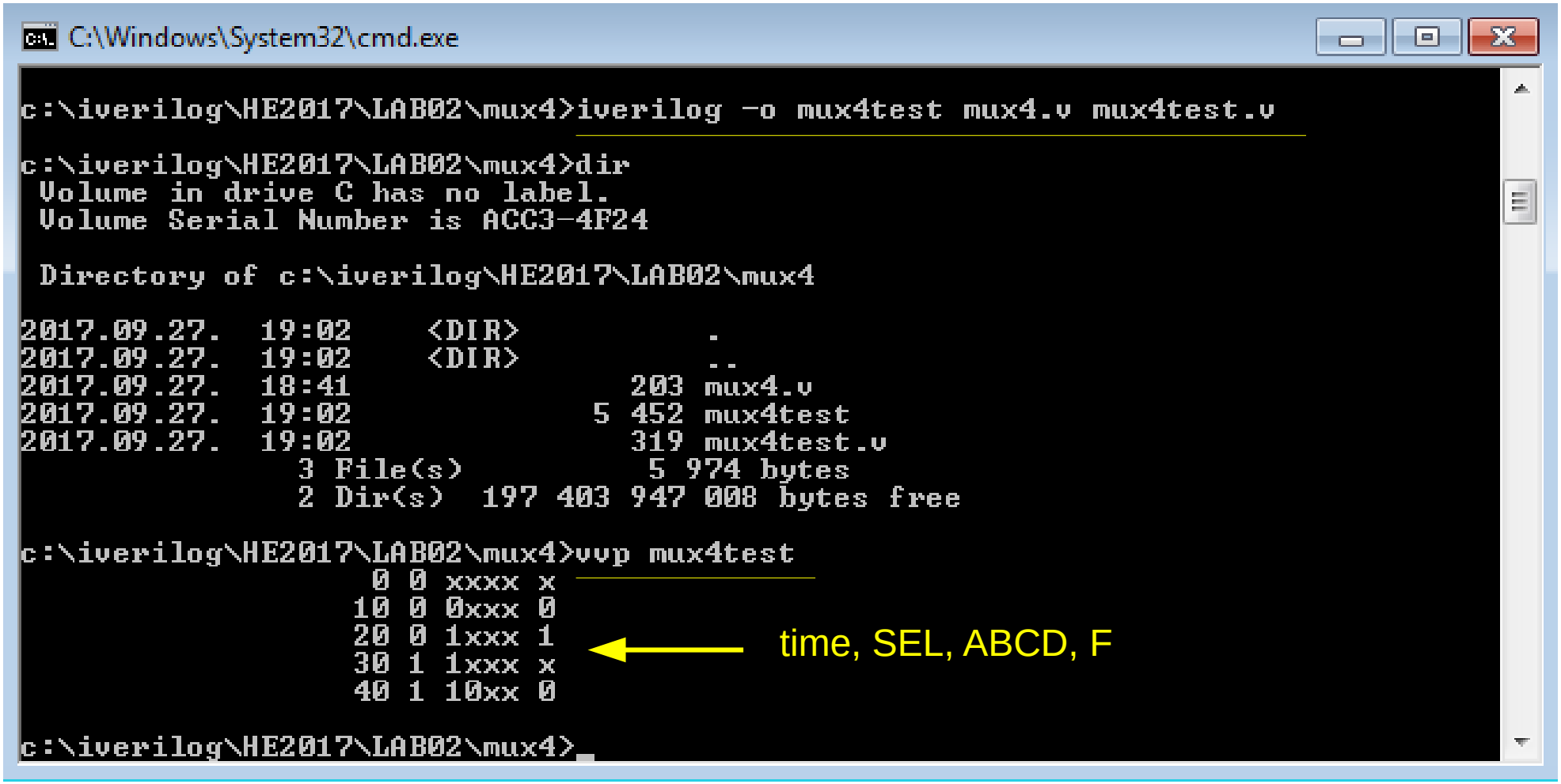
```
module MUX4 (SEL,A,B,C,D,F);
  input [1:0]SEL;
  input A,B,C,D;
  output F;
  assign F = (A&~SEL[1]&~SEL[0]) | (B&~SEL[1]&SEL[0]) |
             (C&SEL[1]&~SEL[0]) | (D&SEL[1]&SEL[0]);
endmodule
```



s1	s0	out
0	0	a
0	1	b
1	0	c
1	1	d

Fordítás és szimuláció

- Fordítsuk le a **mux4.v** és **mux4test.v** állományt az **Icarus Verilog** fordítójával és futtassuk a **mux4test** szimulációs állományt!

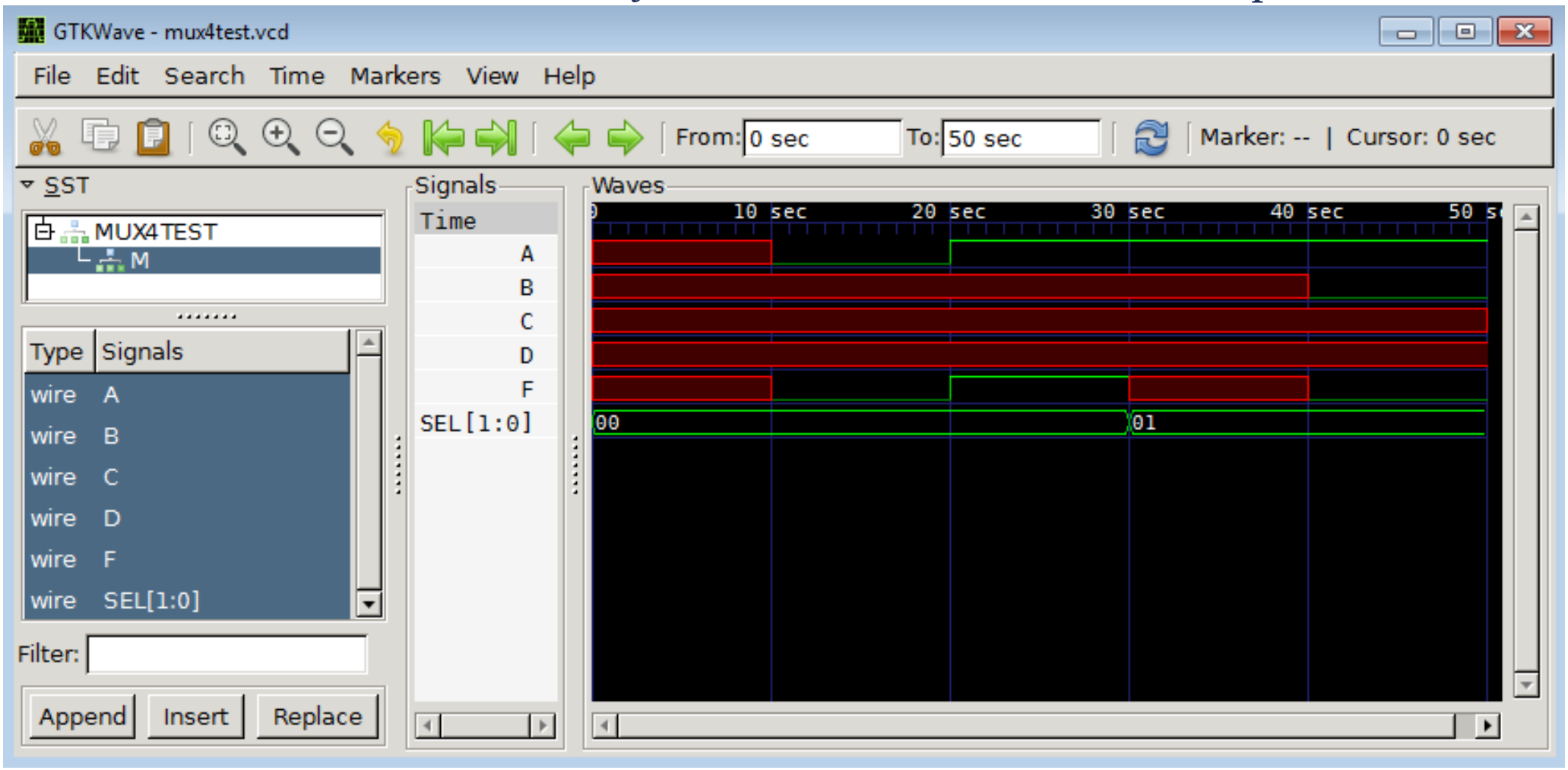


GtkWave megjelenítő

- Szúrjuk be az alábbi két sort a `mux4test.v` állományba:

```
$dumpfile("mux4test.vcd");
$dumpvars(0,MUX4TEST);
```

- Fordítás és szimuláció után adjuk ki a `gtkwave mux4test.vcd` parancsot!

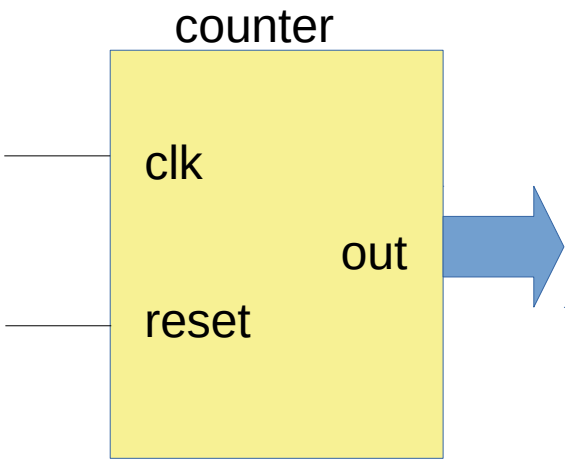


Számláló projekt

- 8-bites számlálót tervezünk és szimulátorban vizsgáljuk a működését

counter.v

```
module counter(out,clk,reset);
  parameter WIDTH = 8;
  output [WIDTH-1:0]  out;
  input          clk,reset;
  reg [WIDTH-1:0]     out;
  wire           clk, reset;

  always @(posedge clk)
    out <= out + 1;

  always @reset
    if (reset)
      assign out = 0;
    else
      deassign out;
endmodule // counter
```

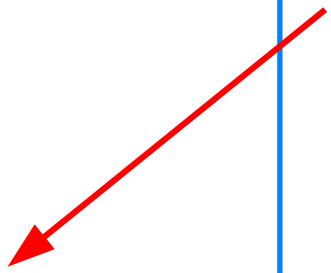


Számláló projekt

- A próbapad gondoskodik a bemenetek vezérléséről és a kimenet naplózásáról

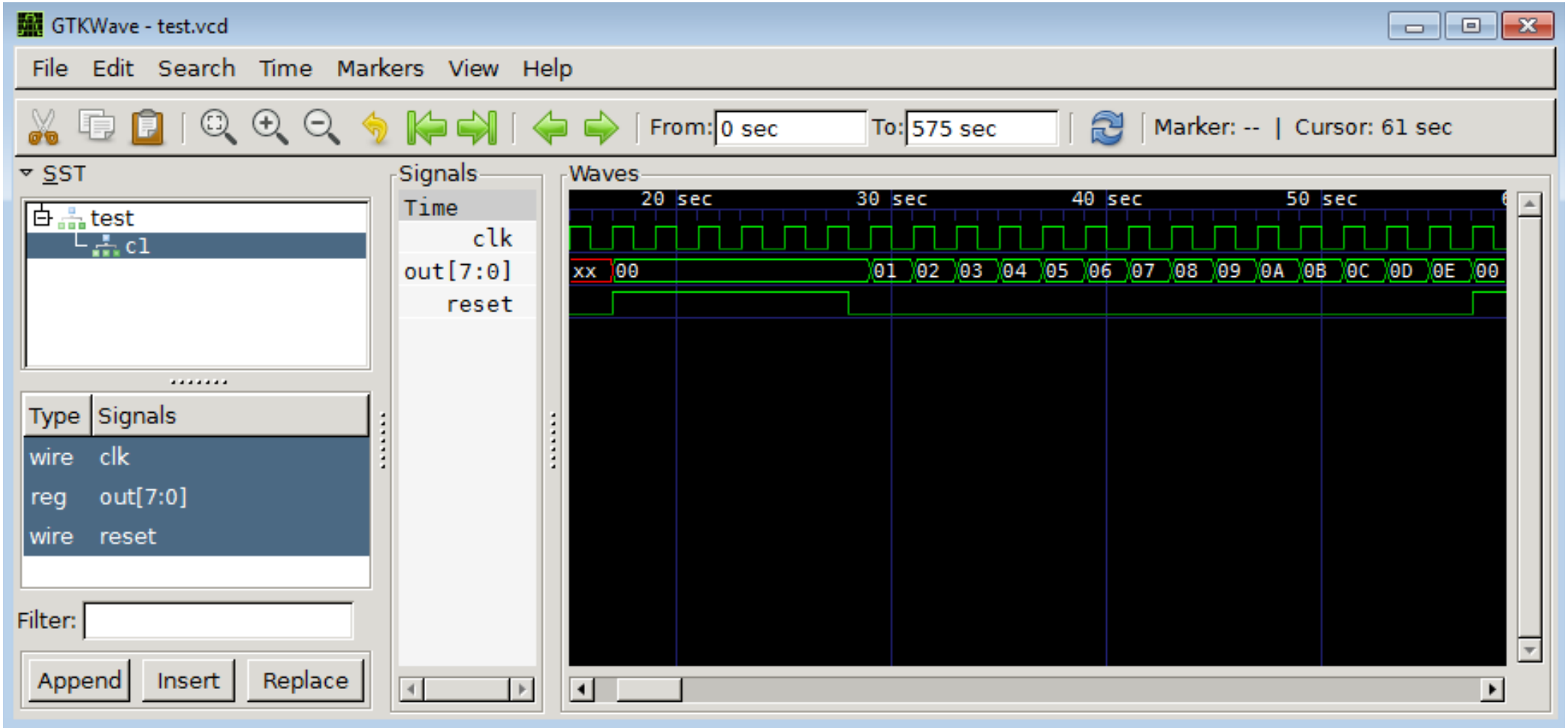
counter_tb.v

```
module test;
  reg reset = 0;
  initial begin
     $dumpfile("test.vcd");
     $dumpvars(0,test);
     # 17 reset = 1;
     # 11 reset = 0;
     # 29 reset = 1;
     # 5  reset =0;
     # 513 $finish;
  end
  reg clk = 0;
  always #1 clk = !clk;
  wire [7:0] value;
  counter c1 (value, clk, reset);
  initial
     $monitor("At time %t, value = %h (%0d)",
              $time, value, value);
endmodule // test
```



Számláló projekt

```
> PATH=%PATH%;C:\iverilog\bin;C:\iverilog\gtkwave\bin
> iverilog -o counter counter.v counter_tb.v
> vvp counter
> gtkwave test.vcd
```



Felhasznált irodalom és segédanyagok

- Icarus Verilog Simulator: http://iverilog.icarus.com/
- GtkWave wave viever: http://gtkwave.sourceforge.net/
- Verilog online tutorial: vol.verilog.com/VOL/main.htm
- Verilog tutorial for beginners: www.referencedesigner.com/tutorials/verilog/verilog_01.php
- Végh János: Bevezetés a **Verilog** hardverleíró nyelvbe
- Végh János: Segédeszközök az **Altera DE2** tanulói készlethez
- Végh János: Bevezetés a **Quartus II V13** fejlesztő rendszerbe