



BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
Mesterséges Intelligencia és Rendszertervezés Tanszék

Verilog HDL

Szántó Péter
Mesterséges Intelligencia és Rendszertervezés
Tanszék
FPGA laboratórium

HDL nyelvek

- **Verilog**

- 1984: Gateway Design Automation Inc.
- 1990: Cadence -> Open Verilog International
- 1995: IEEE szabványosítás
- 2001: Verilog 2001
- 2005: System Verilog

- **VHDL**

- 1983-85: IBM, Texas Instruments
- 1987: IEEE szabvány
- 1994: VHDL-1993
- 2008: VHDL-2008

HDL nyelvek célja

- **Hardver modellezés**
 - Mindkét nyelv jelentős része csak a hardver funkciók modellezésére ill. szimulációra használható
 - Szintetizálható részalmaz szintézer függő
- **Kapuszintű modulokból építkező, kapcsolási rajzon alapuló tervezési módszerek leváltása**
- **RTL (Register Transfer Level) szintű leírás**
 - Automatikus hardver szintézis a leírásból
 - Tervezői hatékonyság növelése

HDL nyelvek

- **Alapvetően moduláris felépítésű tervezést tesz lehetővé**
- **HDL modul**
 - Be-, kimenetek definiálása
 - Be-, kimenetek közötti logikai kapcsolatok és időzítések definiálása
- **NEM szoftver**
 - Alapvetően időben párhuzamos, konkurrens működést ír le

Verilog szintaktika

- **Megjegyzések (mint C-ben)**

- // egy soros
- /* */ több soros

- **Konstansok**

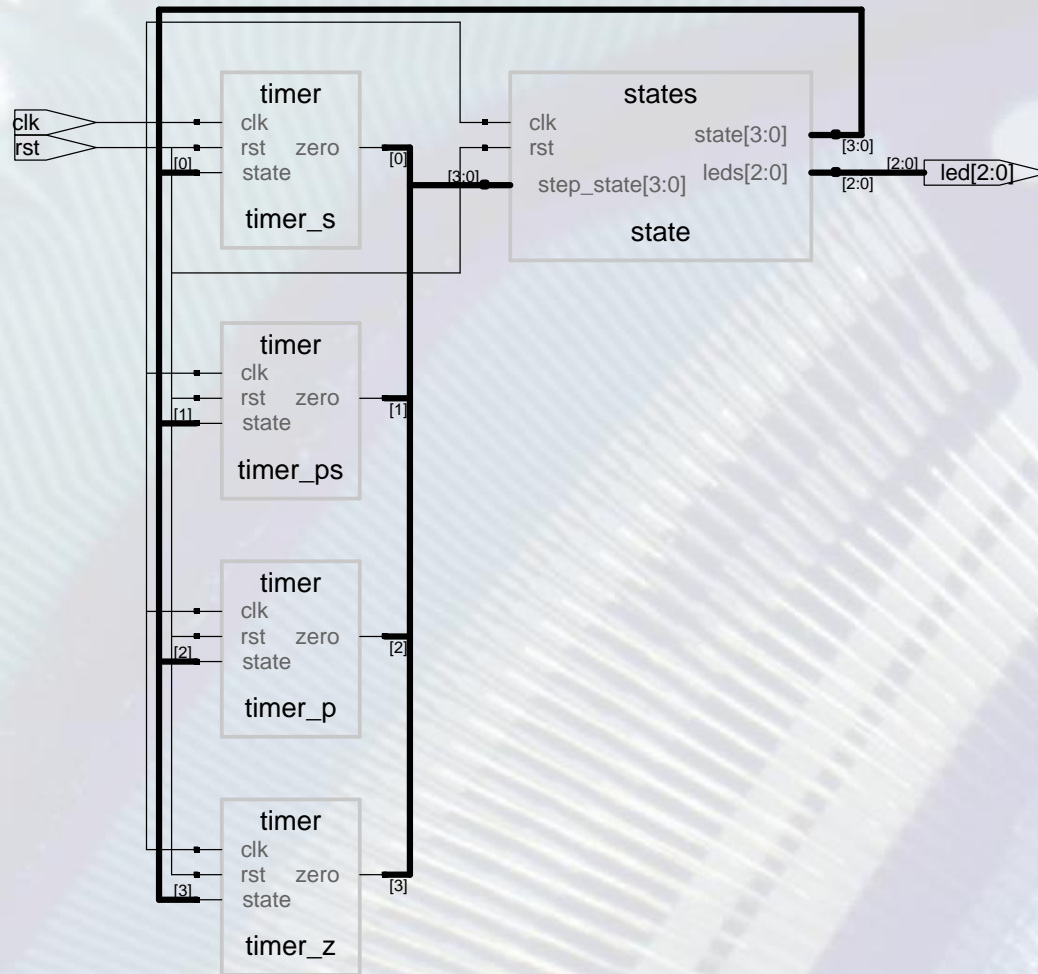
- <bitszám><'számrendszer><érték>

- 5'b00100: 00100 decimális érték: 4, 5 bites
- 8'h4e: 1001110 decimális érték: 78, 8 bites
- 4'bZ: ZZZZ nagy impedanciás állapot
- 1'bX: X don't care

Modulok

- **„Építőelem” komplex rendszerek létrehozására**
- **Hierarchikus leírás, feladat partícionálás**
- **Alkalmazható felülről lefelé, alulról felfelé tervezéskor**
- **Egy modul tetszőleges példányban beépíthető**
 - Nem szubrutin, mindegyik példány önálló valódi HW, saját erőforrásokkal
- **Adatkapcsolat az interfész leíráson keresztül**

Hierarchikus tervezés



Modul interfészlista

- Preferált a kompakt lista, kevesebb hiba

„module” kulcsszó

Module típusa

```
module test(  
    input clk,  
    input [7:0] data_in,  
    output [7:0] data_out,  
    output reg valid  
);  
.....  
.....  
.....  
endmodule
```

Modul bemeneti
portjai

Modul kimeneti
portjai

„endmodule”
kulcsszó

Kívánt
funkcionalitás

Bitműveletek

- **Logikai műveletek bitvektorokon (egy vagy több bites adatokon)**
 - \sim , $\&$, $|$, \wedge , (negálás, és, or, xor)
 - $\sim\&$, $\sim|$, $\sim\wedge$ (NAND, NOR, XNOR)
- **Vektorokon bitenkén, pl:**
 - $4'b1101 \& 4'b0110 = 4'b0100$
- **Ha a két operandus szélessége nem egyezik meg, a kisebbik az MSB biteken 0-val kiterjesztve**
 - $2'b11 \& 4'b1101 = 4'b0001$
- **A feltételes kifejezések logikai operátorai az igaz-hamis vizsgálatokhoz eltérőek:**
 - $!$, $\&\&$, $||$ (negálás, és, vagy)

Bit redukciós operátorok

- Egy operandusú művelet, a bitvektor összes bitjét önálló egybites változóként értelmezve, eredménye is egy bites
 - $\&$, $\sim\&$, $|$, $\sim|$, \wedge , $\sim\wedge$ (és, nem és, vagy, nem vagy)
- Példák:
 - $\&4'b1101 = 1'b0$
 - $|4'b1101 = 1'b1$
- Használat:
 - Számláló kimenet végérték? `assign tc = &cnt;`
 - ALU kimenet nulla? `assign z = ~|result;`

Komparátor operátorok

- **C-szintakszissal megegyező**
- **Egyenlőség**
 - `==`, `!=`
 - `===`: egyenlőség az „x, z” értékek figyelembevételével, azaz bizonyos bitek értéke tetszőleges
 - `!==`: nem egyenlő, „x, z” figyelembevételével
- **Nem egyenlőség**
 - `<`, `>`, `<=`, `>=`

Aritmetikai operátorok

- **C-szintakszissal megegyező**
- **Operátorok: +, -, *, /, %**
 - Nem mindegyik szintetizálható
 - Szintézer függő, de tipikusan / pl. csak akkor, ha az osztó kettő hatvány
 - Szorzásra választható implementációs stílus
 - Beépített blokk vagy LUT hálózat
 - Negatív számok kettes komplementes kódban

Egyéb operátorok

- **Konkatenálás (összefűzés): {} , pl:**
 - $\{4'b0101, 4'b1110\} = 8'b01011110$
 - $\{2\{3'b101\}, 2'b00\} = 8'b10110100$
- **Shift operátor**
 - \ll, \gg
 - \lll, \ggg Előjeles shift, MSB nem változik
- **Bit kiválasztás**
 - Kiválasztott rész konstans
 - `data[5:3]`

Adattípusok

- **A szintézis szempontjából kétfajta adat van**
- **A huzal típusú „wire”**
 - Nevének megfelelően viselkedik (vezeték)
 - Nincs saját állapota, azt mindig örökli
 - Pl. 8 bites vezeték: wire [7:0] data;
- **A változó típusú „reg”**
 - Két értékadás között állapotát tartja
 - Értékadás történhet eseményvezérlésre , vagy órajelre
 - Szintézis utáni eredmény nem mindig regiszter
 - Vezeték
 - Latch
 - Flip-flop
 - Pl. 8 bites regiszter: reg [7:0] data;

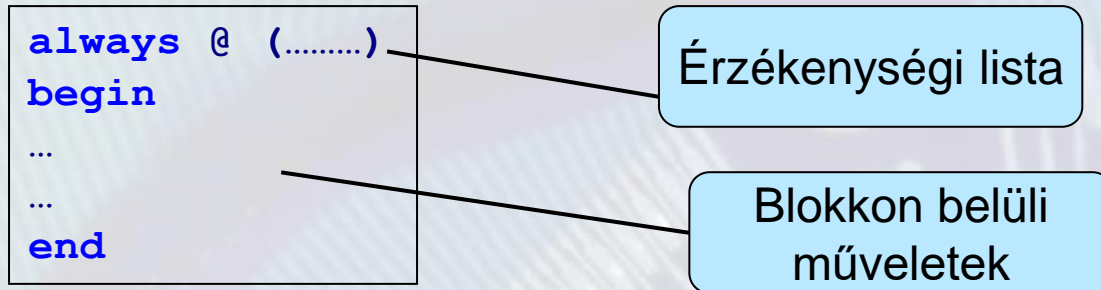
Assign

- **Tipikusan kombinációs logika leírására**
- „assign”-val csak „wire” típusú változónak lehet értéket adni
- **Konkurrens, folytonos értékadás**
 - A jobb oldali változó bármely változása a kifejezés kiértékelődését, új értékének meghatározását okozza
 - Pl. `assign c = a & b;`
- **Egy változó csak egy „assign” által kaphat értéket**



Always blokk

- Magas szintű viselkedési leírás
- Szintakszis:



- Reg típusú változó kaphat értéket
- Egy változó csak egy „always” blokkban kaphat értéket
- Always blokk nem lehet érzékeny a saját kimenetére
- Az „always” blokkok egymással párhuzamosan működnek

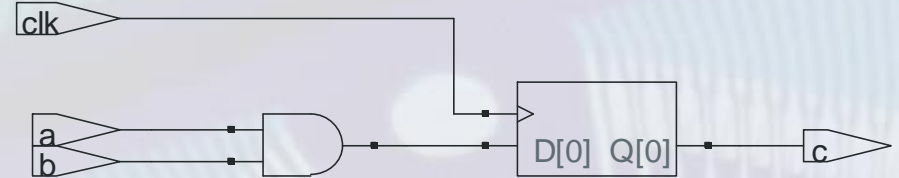
Always – értékadás

- **Eljáráson belül kétfajta értékadás**
- **Blokkoló értékadás: =**
 - Blokkolja az utána következő értékadásokat -> szekvenciális utasítás végrehajtás
- **Nem blokkoló értékadás: <=**
 - A nem blokkoló értékadások párhuzamosan hajtódnak végre, azaz a baloldali kifejezések kiértékelődnek az aktuális változó értékek szerint és az eredmény csak a fázis végén adódik át a bal oldali változónak
- **Blokkoló értékadás: =**
 - A Verilog egyik népszerű témája
 - Laboron TILOS használni

Always – Flip Flop

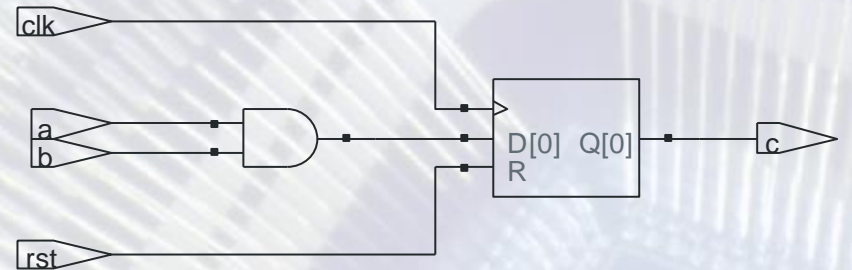
- Flip Flop: érzékeny D tároló

```
always @ (posedge clk)
    c <= a & b;
```



- Szinkron reset

```
always @ (posedge clk)
if (rst)
    c <= 1'b0;
else
    c <= a & b;
```



- Aszinkron reset

```
always @ (posedge clk, posedge
rst)
if (rst)
    c <= 1'b0;
else
    c <= a & b;
```

Always – Flip Flop

- **Clock enable (ce): a tároló frissítésének engedélyezése**
 - Ha nem minden órajelben szeretnénk frissíteni a tároló tartalmát, akkor ezt használjuk, nem pedig új órajelet hozunk létre!

```
always @ (posedge clk)
if (rst)
    c <= 1'b0;
else if (ce)
    c <= d;
```

Always – kombinációs logikához

- **Szemléletesen:**

- Az always blokk eseményvezérelt
- A bemenőjelek bármely változása ilyen esemény
- Ennek hatására az értékadások kiértékelődnek, a kimeneti változó értéke megváltozik

```
always @ (a, b)  
  c <= a & b;
```

```
always @ (*)  
  c <= a & b;
```

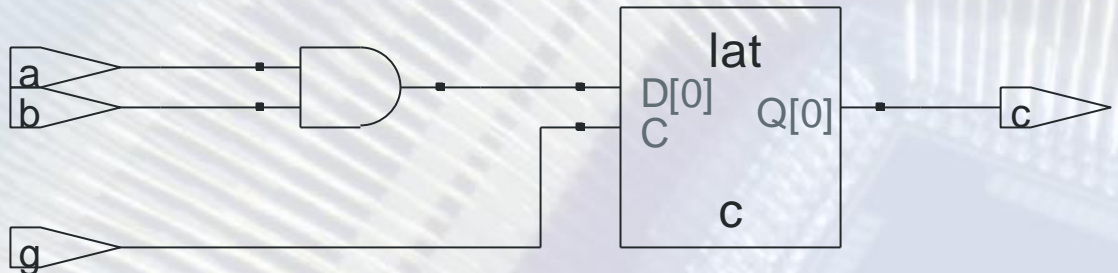


- Ha egy változó kimarad az érzékenységi listából, akkor véletlen latch keletkezhet

Always – latch

- **Latch tároló természetesen szándékosan is generálható:**
 - Az engedélyező „gate” bemenet magas értéke mellett a tároló transzparens, míg a „gate” bemenet alacsony értéke mellett zárt, tartja értékét.

```
always @ (*)  
if (g)  
    c <= a & b;
```

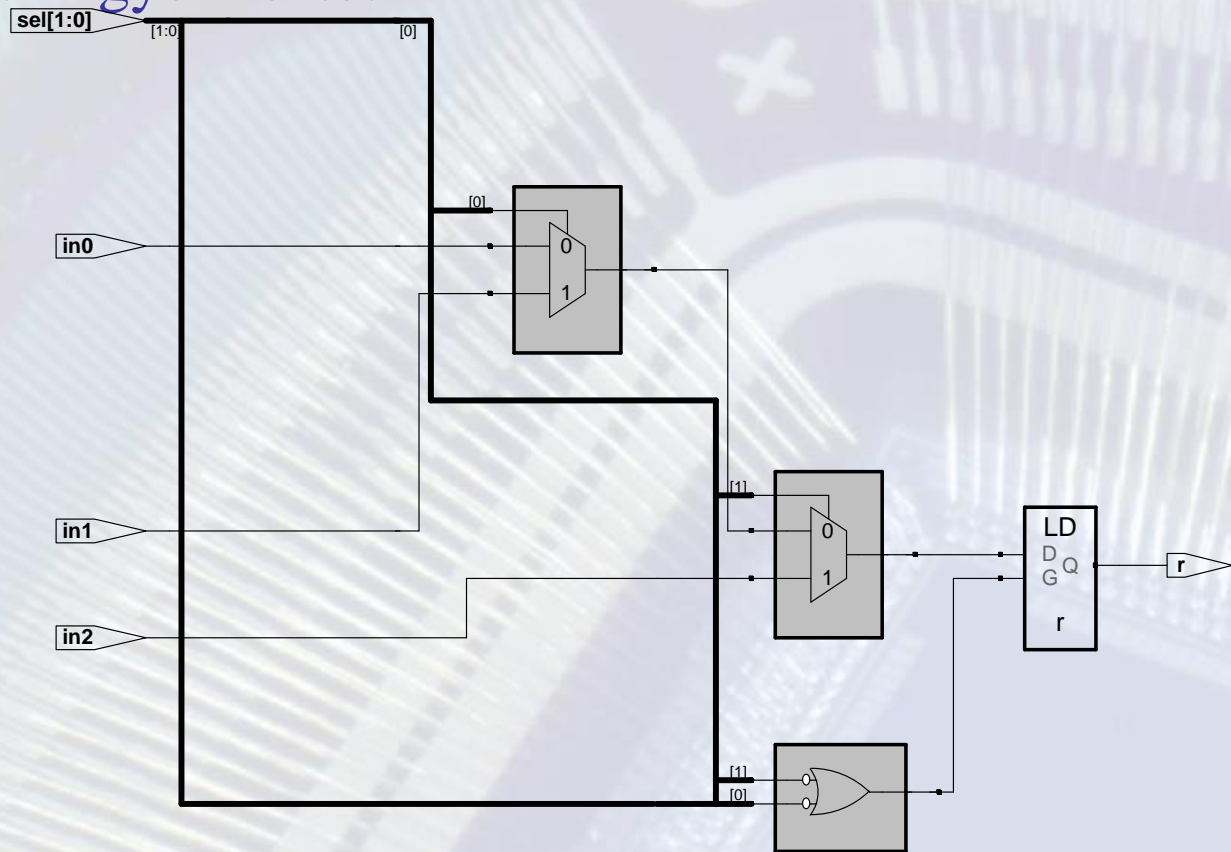


Always – latch hiba

- A tipikus véletlen „Latch”
 - Nem teljes “if” vagy „case” szerkezet
 - Szintézizer általában figyelmeztet

```
always @ (*)
case (sel)
  2'b00: r <= in0;
  2'b01: r <= in1;
  2'b10: r <= in2;
endcase
```

```
always @ (*)
if (sel==0)
  r <= in0;
else if (sel==1)
  r <= in1;
else if (sel==2)
  r <= in2;
```

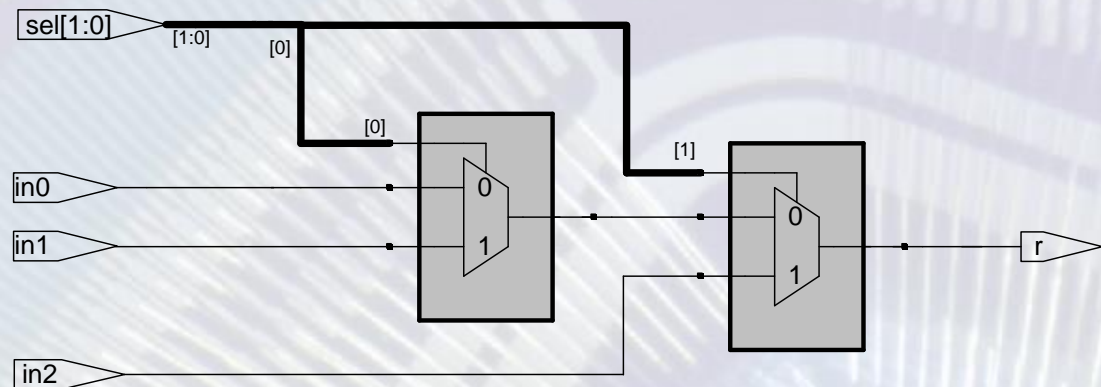


Always – helyes

- Helyes kód

```
always @ (*)
case (sel)
  2'b00: r <= in0;
  2'b01: r <= in1;
  2'b10: r <= in2;
  default: r <= 'bx;
endcase
```

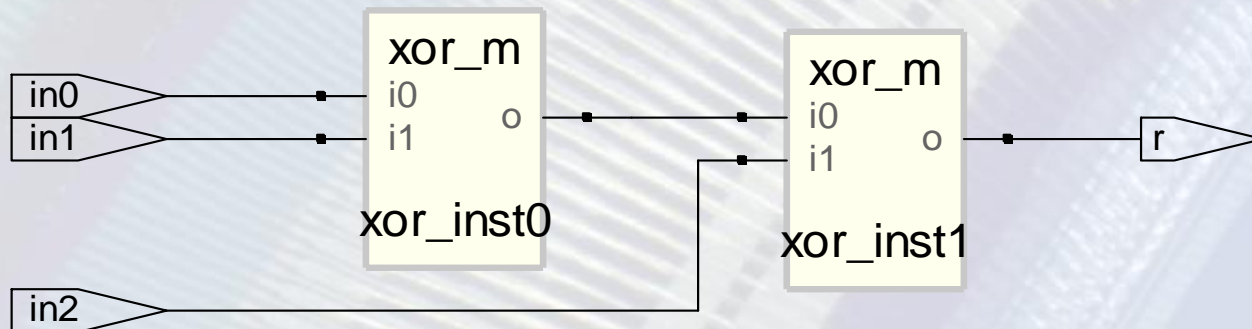
```
always @ (*)
if (sel==0)
  r <= in0;
else if (sel==1)
  r <= in1;
else
  r <= in2;
```



Strukturális leírás

- **Hierarchia felépítése: modulok összekapcsolása**
 - Almodul kimenete csak wire típusú változóra csatlakozhat

```
module top_level (input in0, in1, in2, output r);  
  wire xor0;  
  xor_m xor_inst0 (.i0(in0), .i1(in1), .o(xor0));  
  xor_m xor_inst1 (.i0(xor0), .i1(in2), .o(r));  
endmodule
```



Példa – MUX (1.)

- Különböző leírási stílusok a 2:1 multiplexerre

```
module mux_21 (input in0, in1, sel, output r);  
  assign r = (sel==1'b1) ? in1 : in0;  
endmodule
```

```
module mux_21 (input in0, in1, sel, output reg r);  
  always @ (*)  
  if (sel==1'b1) r <= in1;  
  else          r <= in0;  
endmodule
```

```
module mux_21 (input in0, in1, sel, output reg r);  
  always @ (*)  
  case(sel)  
    1'b0: r <= in0;  
    1'b1: r <= in1;  
  endcase  
endmodule
```

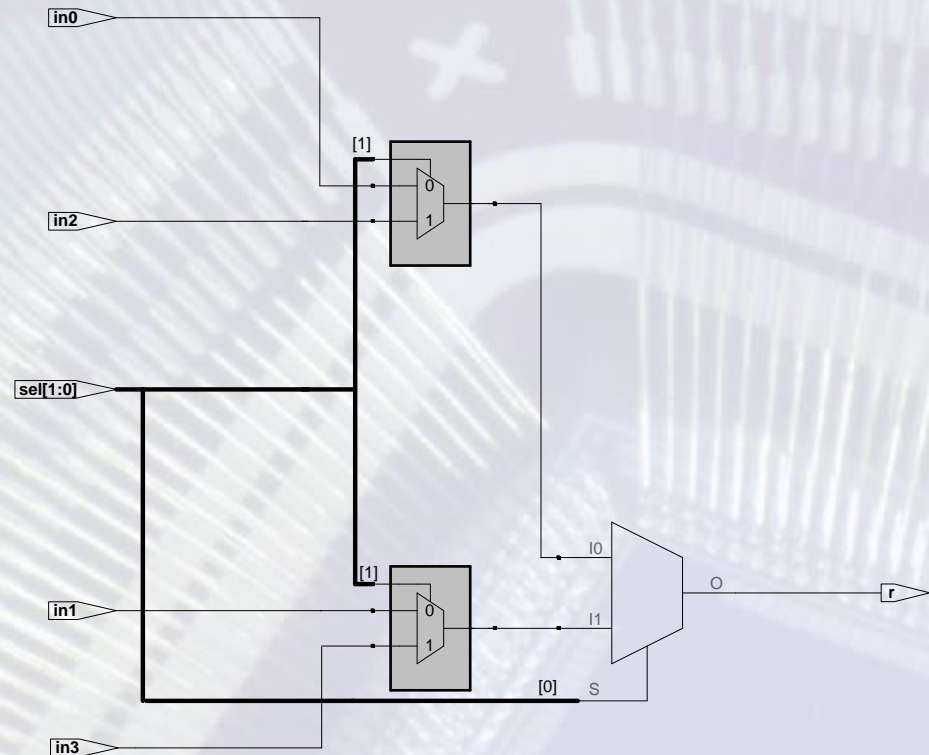
Példa – MUX (2.)

- 4:1 multiplexer

```
module mux_41 (input in0, in1, in2, in3,  
               input [1:0] sel, output reg r);  
  
always @ (*)  
case(sel)  
    2'b00: r <= in0;  
    2'b01: r <= in1;  
    2'b10: r <= in2;  
    2'b11: r <= in3;  
endcase  
endmodule
```

- 1 bites esetben

```
module mux_41 (  
    input [3:0] in,  
    input [1:0] sel,  
    output r);  
  
assign r = in[sel];  
  
endmodule
```



Példa: Shift regiszter

- 16 bites shift regiszter,
 - Engedélyezhető
 - Soros bemenet, soros kimenet

```
module shr_module (input clk, ce, din, output dout);  
  
    reg [15:0] shr;  
    always @ (posedge clk)  
        if (ce)  
            shr <= {shr[14:0], din};  
  
    assign dout = shr[15];  
  
endmodule
```

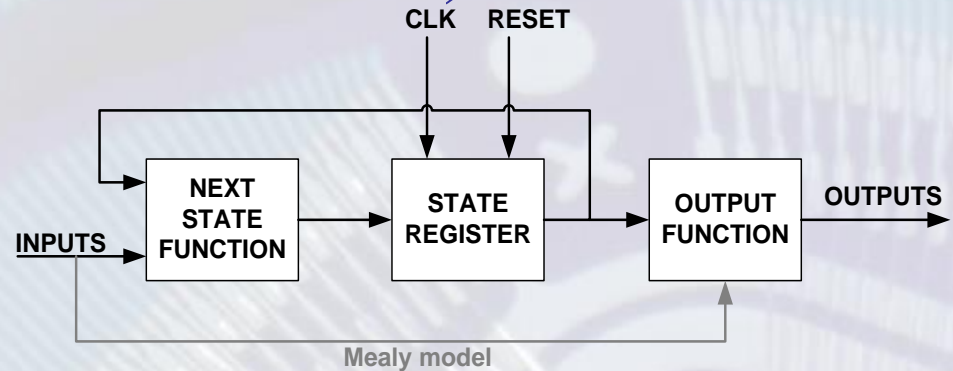
Példa: Számláló

- Számláló minta leírás
 - Szinkron, 8 bites
 - Szinkron RESET
 - Tölthető
 - Engedélyezhető
 - fel/le számláló

```
module m_cntr (  
    input clk, rst, ce, load, dir,  
    input  [7:0] din,  
    output [7:0] dout);  
  
    reg [7:0] cntr_reg;  
    always @ (posedge clk)  
    if (rst)  
        cntr_reg <= 0;  
    else if (ce)  
        if (load)  
            cntr_reg <= din;  
        else if (dir)  
            cntr_reg <= cntr_reg - 1;  
        else  
            cntr_reg <= cntr_reg + 1;  
  
    assign dout = cntr_reg;  
  
endmodule
```

FSM – Finite State Machine

- Állapotgép – vezérlési szerkezetek kialakítása
- Általános struktúra (Moore modell)

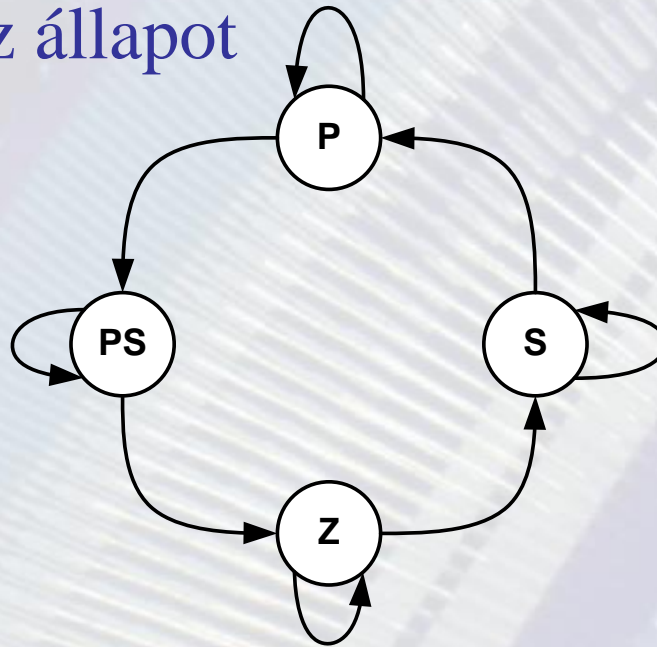


- State register: állapotváltozó
- Next state function: következő állapotot dekódoló logika
- Output function: kimeneti jeleket előállító logika
 - Moore: állapotváltozó alapján
 - Mealy: állapotváltozó + bemenetek alapján

FSM példa

- **Közlekedési lámpa**

- Állapotok: piros, sárga, zöld, piros-sárga (a villogó sárga nem implementált)
- Bemeneti változók: időzítő az egyes állapotokhoz
- Kimenet: az állapot



FSM példa – Verilog (1)

```
module lampa(  
    input      clk, rst,  
    output reg [2:0] led);
```

```
parameter PIROS   = 2'b00;  
parameter PS      = 2'b01;  
parameter ZOLD    = 2'b10;  
parameter SARGA   = 2'b11;
```

```
reg [15:0] timer;  
reg [1:0] state_reg;  
reg [1:0] next_state;
```

```
always @ (posedge clk)  
if (rst)  
    state_reg <= PIROS;  
else  
    state_reg <= next_state;
```

```
always @ (*)  
case(state_reg)  
    PIROS: begin  
        if (timer == 0)  
            next_state <= PS;  
        else  
            next_state <= PIROS;  
        end  
    PS: begin  
        if (timer == 0)  
            next_state <= ZOLD;  
        else  
            next_state <= PS;  
        end  
    SARGA: begin  
        if (timer == 0)  
            next_state <= PIROS;  
        else  
            next_state <= SARGA;  
        end  
    ZOLD: begin  
        if (timer == 0)  
            next_state <= SARGA;  
        else  
            next_state <= ZOLD;  
        end  
    default:  
        next_state <= 3'bxxx;  
endcase
```

FSM példa – Verilog (2)

```
always @ (posedge clk)
case(state_reg)
  PIROS: begin
    if (timer == 0)
      timer <= 500;      //next_state <= PS;
    else
      timer <= timer - 1;
    end
  PS: begin
    if (timer == 0)
      timer <= 4000;     //next_state <= ZOLD;
    else
      timer <= timer - 1;
    end
  SARGA: begin
    if (timer == 0)
      timer <= 4500;     //next_state <= PIROS;
    else
      timer <= timer - 1;
    end
  ZOLD: begin
    if (timer == 0)
      timer <= 500;      //next_state <= SARGA;
    else
      timer <= timer - 1;
    end
endcase
```

Időzítő

- Állapotváltáskor egy állapotfüggő kezdőértéket tölt be
- Lefelé számol
- == 0 : állapotváltás

```
always @ (*)
case (state_reg)
  PS: led <= 3'b110;
  PIROS: led <= 3'b100;
  SARGA: led <= 3'b010;
  ZOLD: led <= 3'b001;
endcase

endmodule
```

Szimuláció

- **Verilog Test Fixture**
 - Verilog kódban megírt stimulus
 - Nincs ki/bemenete
 - A tesztelt modul számára generál bementi jeleket
- **Szimulátor**
 - Xilinx Simulator (Vivado)
 - Mentor ModelSim/Quarta
 - Aldec Active-HDL
 - Synopsys VCS

Verilog Test Fixture

- **Test Fixture**

- A Test Fixture egy Verilog modul, ez a legfelső szintű modul
- A tesztelendő modul almodulként van beillesztve
- Minden, a szintézisnél használt nyelvi elem felhasználható
- Nem szintetizálható nyelvi elemek is

Időalap

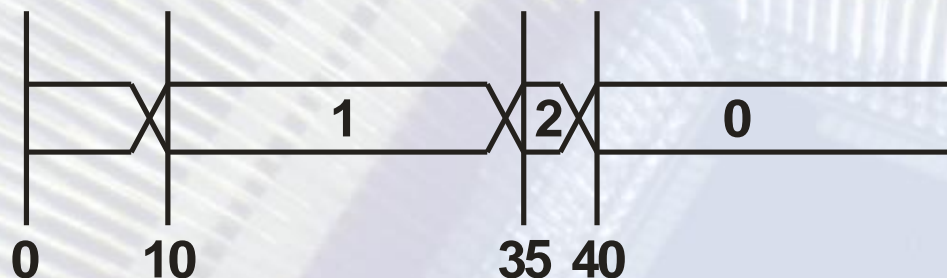
- ‘timescale 1ns/1ps
 - Megadott idők ns-ban értendők
 - Szimulációs lépésköz: 1 ps

Test Fixture - initial

- „initial” blokk

- 0. időpillanatban kezdődik a végrehajtása
- Egyszer fut le, belső időzítések akkumulálódnak
- Az „initial” blokkok egymással, és az „always” blokkokkal párhuzamosan működnek

```
initial
begin
    a <= 0;
    #10 a <= 1;
    #25 a <= 2;
    #5   a <= 0;
end
```



Test Fixture - always

- Tipikus feladatok
- Órajel generálás

```
initial
```

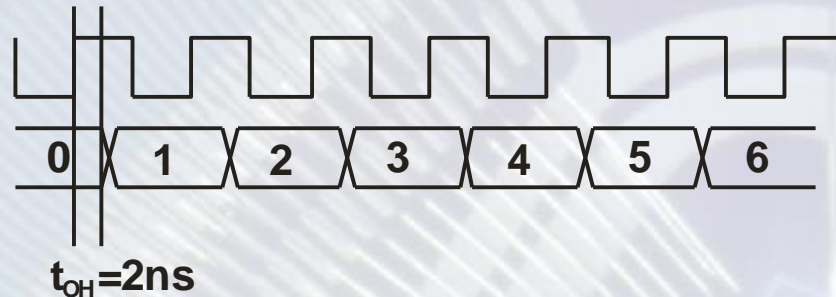
```
    clk <= 1;
```

```
always #5
```

```
    clk <= ~clk;
```

- Órajelre működő bemenetek (pl. számláló)

```
initial cntnr <= 0;  
always @ (posedge clk)  
    #2 cntnr <= cntnr + 1;
```



- Élre várakozás

```
initial begin  
    #101  
    @ (negedge clk)  
        cntnr <= cntnr + 1;  
    .....  
end
```