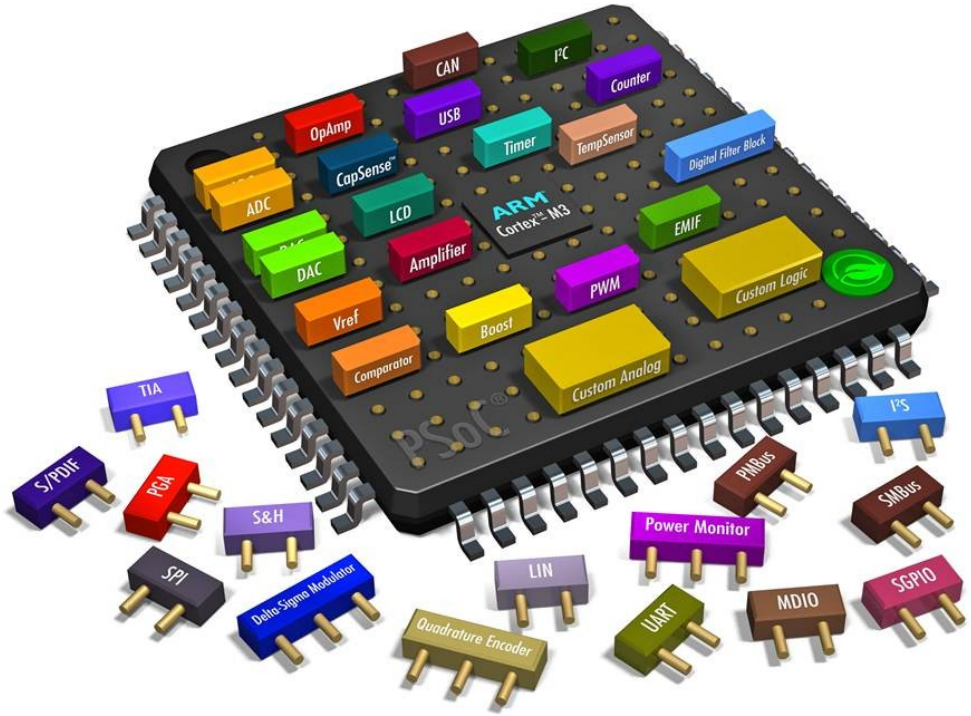
Újrakonfigurálható eszközök



5. A Verilog sűrűjében: véges állapotgépek

Felhasznált irodalom és segédanyagok

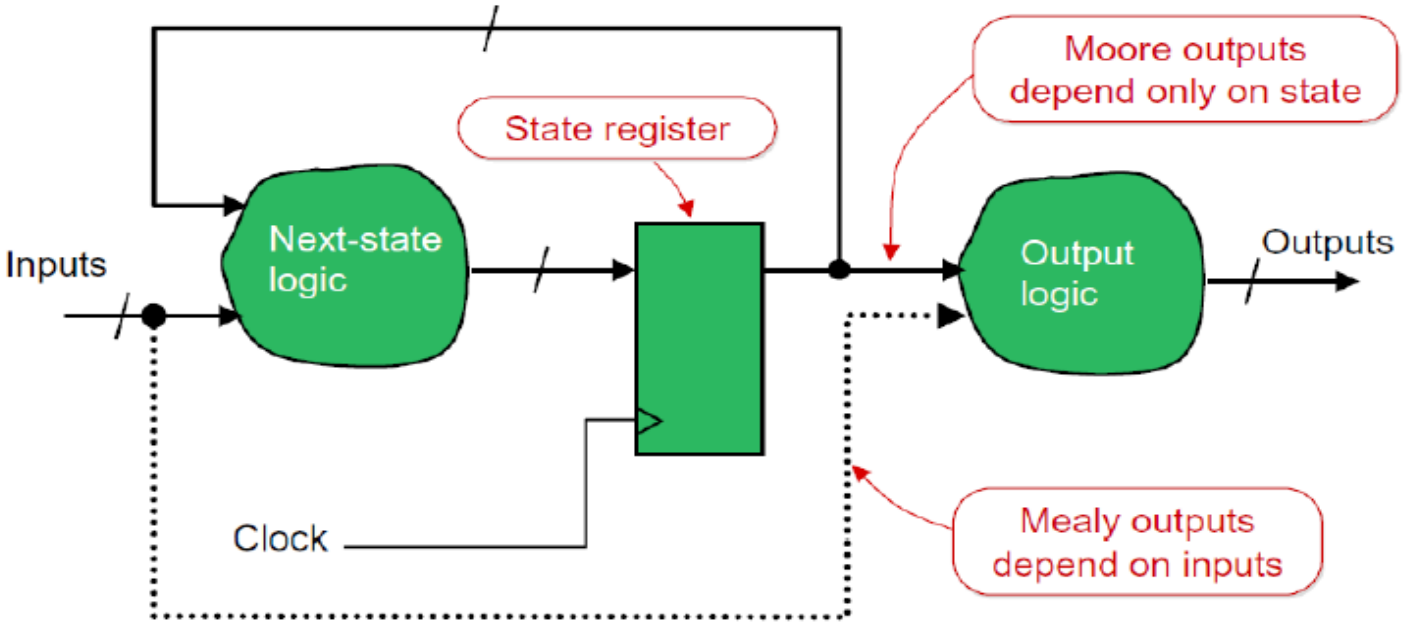
- Icarus Verilog Simulator: http://iverilog.icarus.com/
- GtkWave wave viever: http://gtkwave.sourceforge.net/
- Verilog online tutorial: vol.verilog.com/VOL/main.htm
- Verilog tutorial for beginners: www.referencedesigner.com/tutorials/verilog/verilog_01.php
- ASIC world – Verilog tutorial: asic-world.com/verilog/veritut.html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

Véges állapotgépek

- Egy véges állapotgép (Finite State Machine, FSM) olyan digitális logikai áramkör, amelynek véges számú belső állapota van.
- Az állapotgép bemeneteinek értékét és az állapotgép pillanatnyi állapotát használja arra, hogy meghatározza kimeneteinek értékét és következő állapotát. Most csak a szinkron állapotgépekkel foglalkozunk, ahol az állapot egy órajel aktív élének hatására változik meg.
- A Verilog nyelven általában úgy írunk le egy véges állapotú automatát, hogy egy ***always*** utasításba ***case*** utasításokat teszünk. Az automata állapotát egy állapotregiszterben tároljuk, és valamennyi lehetséges állapotot paraméter értékekkel írunk le.
- A véges állapotgépeket explicit módon inicializálni kell egy ***reset*** jellel. Enélkül nincs garancia arra, hogy az automata ismert állapotba kerül, így az állapotok egyenlőségét sem vizsgálhatjuk.

Véges állapotgépek típusai

- A véges állapotgépek két fő típusát különböztetjük meg a kimenőjel előállítása szerint:
- A **Moore állapotgép** esetében a kimenőjel csak a pillanatnyi állapot függvénye.
- A **Mealy állapotgép** esetében a kimenőjel a pillanatnyi állapoton kívül a bemenetek pillanatnyi állapotától is függ.

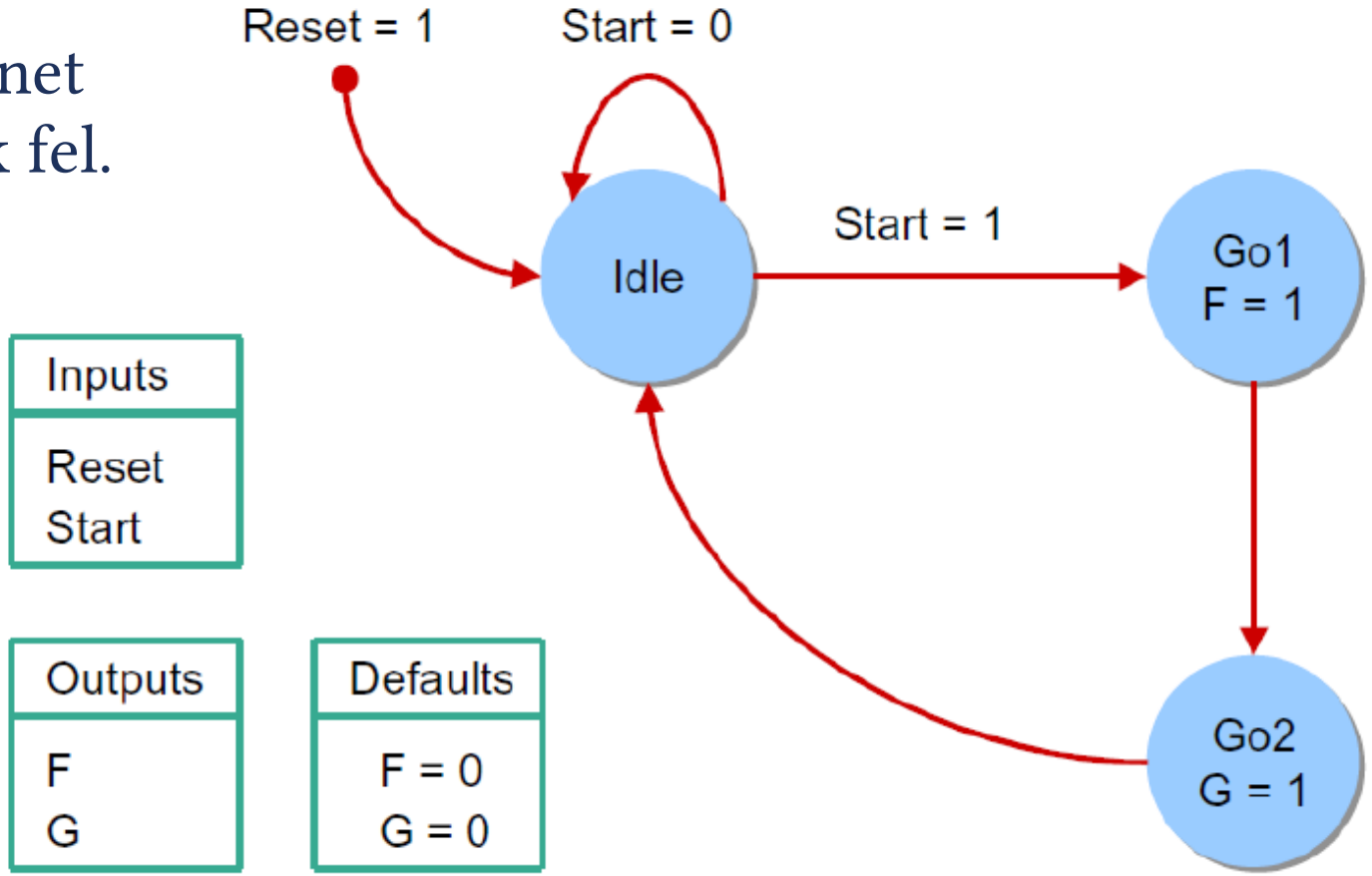


A véges állapotgép modellezése

- Az előző ábrán a következő állapot meghatározása mindig kombinációs logikai hálózattal történik, az állapot nyilvántartása, pedig szekvenciális logikát igényel.
- A véges állapotgép kódolásánál vegyük figyelembe, hogy a kombinációs és a szekvenciális áramköri részt két külön ***always*** blokkban kell definiálnunk.
- Az állapotok nyilvántartása sokféle kódolással történhet. A leggyakoribbak:
  - Bináris kódolás (az állapot kódja a sorszám)
  - „One Hot” kódolás (állapotonként csak egy 1-es)
  - „One Cold” kódolás (állapotonként csak egy 0)
  - Gray kódolás (szomszédos állapotoknál csak 1 bitben különbözik)

Ábrázolás állapotdiagrammal

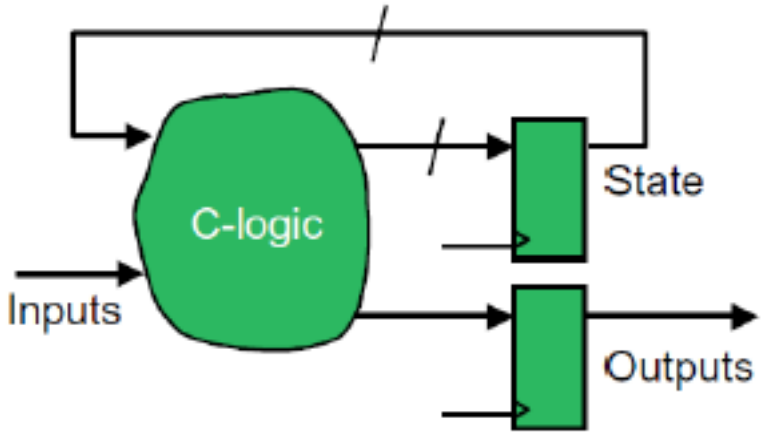
- A véges állapotgépek megadása szöveges leírás helyett történhet állapotdiagram használatával, mint az alábbi példában is.
- A pontból kiinduló Reset = 1 ún. *globális átmenet*, ami magasabb prioritású, mint a többi átmenet.
- Az éleken az átmenet feltételét tüntettük fel.
- A csomópontok a lehetséges állapotok.



Az állapotok explicit megadása

```
parameter Idle = 2'b00,   Go1 = 2'b01,   Go2 = 2'b10;
reg [1:0] State;
...
always @(posedge Clock or posedge Reset)
  if (Reset)
  begin
    State <= Idle;  F <= 0; G <= 0;
  end
  else
    case (State)
      Idle : if (Start)
        begin
          State <= Go1;   F <= 1;
        end
      Go1 : begin
        State <= Go2;  F <= 0; G <= 1;
        end
      Go2 : begin
        State <= Idle;  G <= '0';
        end
    endcase
```

Az állapotgép-ben kell reset lehetőségnek lenni

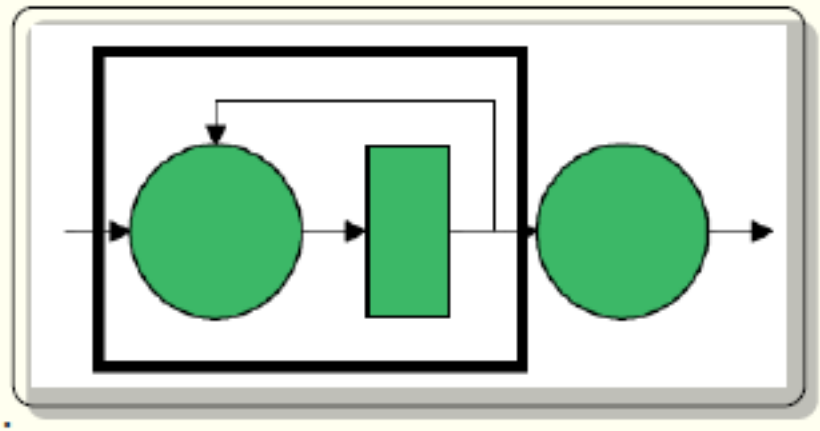


- A kimeneti regiszter használata nem mindig kívánatos!
- Kiküszöböléséhez egy másik ***always*** blokk kell.

A kimeneti regiszter kiküszöbölése

```
parameter Idle = 2'b00,
Go1 = 2'b01,
Go2 = 2'b10;
reg [1:0] State;
...
always @(posedge Clock or posedge Reset)
  if (Reset)
  State <= Idle;
else
  case (State)
    Idle : if (Start) State <= Go1;
    Go1 : State <= Go2;
    Go2 : State <= Idle;
  endcase
...
always @(State)
begin
  F = 0; G = 0;
  if (State == Go1) F = 1;
  else if (State == Go2)  G = 1;
end
```

- Az első ***always*** blokk az állapotvektort és az állapotátmeneteket írja le, a másik pedig a kimenetet dekódoló logikát.

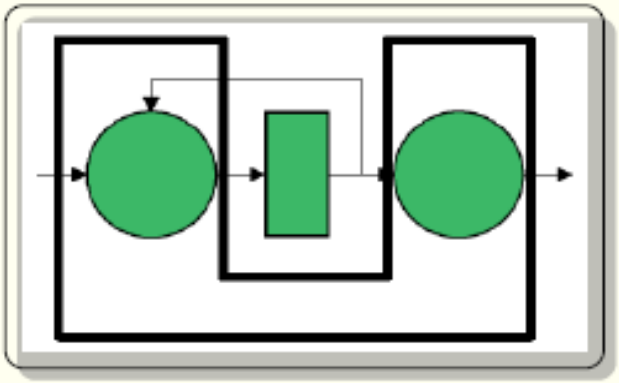


Szeparált kimenetdekódolás

- Az állapotautomaták leírásának egy másik szokásos stílusa, hogy a kombinációs logikából eltávolítjuk a regisztereket.
- A következő kódolási példában egy ***always*** blokk írja le az állapot-regisztereket, egy másik pedig a következő állapotot és a kimenetet dekódoló logikát.
- Ebben az esetben egy további regiszter, **NextState,** szükséges ahhoz, hogy a kombinációs ***always*** utasítás tudjon kommunikálni az órajelvezérelt ***always*** utasítással.
- Mivel a **NextState** állapot a **State** és **Start** kombinációs függvénye, annak teljesen meghatározottá kell válnia az ***always*** blokkban (különben latch áramkörökre van szükség). A ***NextState = State*** értékadás biztosítja, hogy ilyen latch áramkörökre ne legyen szükség.

Szeparált kimenetdekódolás

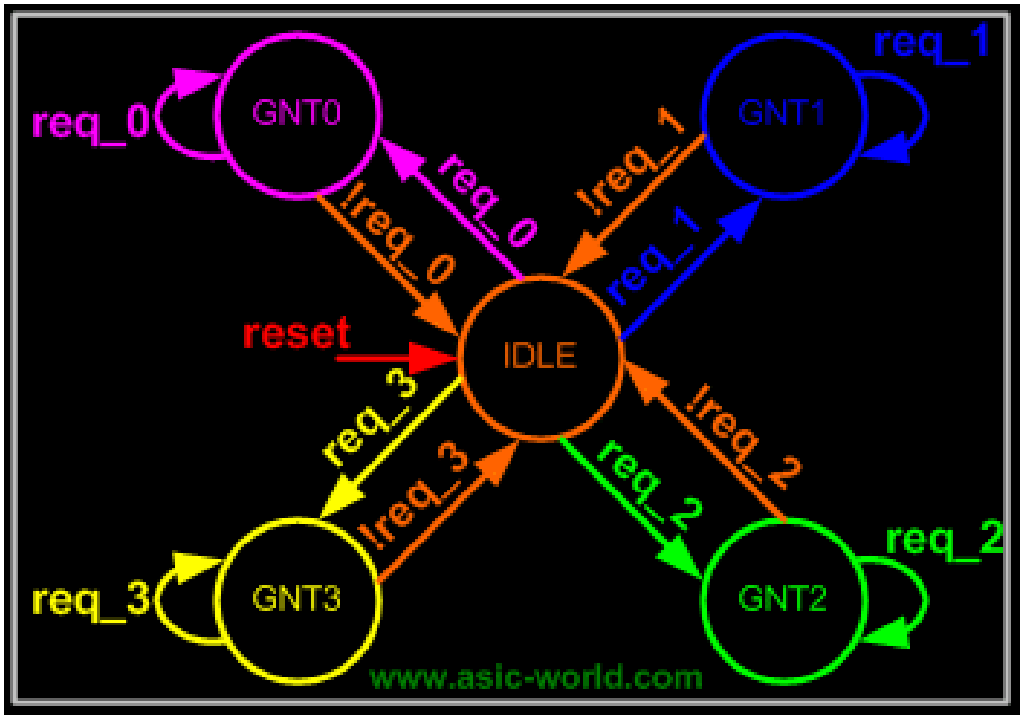
```
parameter Idle = 2'b00, Go1 = 2'b01, Go2 = 2'b10;
reg [1:0] State, NextState;
...
always @(posedge Clock or posedge Reset)
  if (Reset)
    State <= Idle;
  else
    State <= NextState;
...
always @(State or Start)
begin
  F = 0; G = 0; NextState = State;
  case (State)
    Idle :
      if (Start)  NextState = Go1;
    Go1 : begin
      F = 1;  NextState = Go2;
    end
    Go2 : begin
      G = 1;  NextState = Idle;
    end
  endcase
end
```



- Ennél a megoldásnál előnyös lehet, hogy a következő állapotot és a kimenetet dekódoló logika közös részeit nem kell duplikálni.

Mintapélda: prioritásos döntnök (arbiter)

- Az alábbi mintapéldában szereplő arbiter (döntnök) négy független kérelmet fogad, s mindig a magasabb prioritású (alacsonyabb sorszámú) kérelem érvényesül elsőként.
- Az állapotok: IDLE (tétlen), GNT0, GNT1, GNT2, GNT3 (a megfelelő sorszámú kérelem érvényesült).
- A ***reset*** bemenet alaphelyzetbe (IDLE) állítja a rendszert.



Az állapotok kódolása

- A véges állapotgép Verilog kódja 3 részből áll:

1) Az állapotok kódolása

2) A kombinációs logika (a következő állapot meghatározása)

3) A szekvenciális logika (a pillanatnyi állapot módosítása)

Binary Encoding

```
1 parameter [2:0] IDLE = 3'b000;
2 parameter [2:0] GNT0 = 3'b001;
3 parameter [2:0] GNT1 = 3'b010;
4 parameter [2:0] GNT2 = 3'b011;
5 parameter [2:0] GNT3 = 3'b100;
```

Ezt fogjuk használni!

One Hot Encoding

```
1 parameter [4:0] IDLE = 5'b0_0001;
2 parameter [4:0] GNT0 = 5'b0_0010;
3 parameter [4:0] GNT1 = 5'b0_0100;
4 parameter [4:0] GNT2 = 5'b0_1000;
5 parameter [4:0] GNT3 = 5'b1_0000;
```

Gray Encoding

```
1 parameter [2:0] IDLE = 3'b000;
2 parameter [2:0] GNT0 = 3'b001;
3 parameter [2:0] GNT1 = 3'b011;
4 parameter [2:0] GNT2 = 3'b010;
5 parameter [2:0] GNT3 = 3'b110;
```

A kombinációs rész

```
1  always @ (state or req_0 or req_1 or req_2 or req_3)
2  begin
3    next_state = 0;
4    case (state)
5      IDLE : if (req_0 == 1'b1) begin
6           next_state = GNT0;
7             end else if (req_1 == 1'b1) begin
8           next_state= GNT1;
9             end else if (req_2 == 1'b1) begin
10          next_state= GNT2;
11            end else if (req_3 == 1'b1) begin
12          next_state= GNT3;
13       end else begin
14          next_state = IDLE;
15            end
16     GNT0 : if (req_0 == 1'b0) begin
17          next_state = IDLE;
18            end else begin
19          next_state = GNT0;
20       end
21     GNT1 : if (req_1 == 1'b0) begin
22          next_state = IDLE;
23            end else begin
24          next_state = GNT1;
25       end
26     GNT2 : if (req_2 == 1'b0) begin
27          next_state = IDLE;
28            end else begin
29          next_state = GNT2;
30       end
31     GNT3 : if (req_3 == 1'b0) begin
32          next_state = IDLE;
33            end else begin
34          next_state = GNT3;
35       end
36     default : next_state = IDLE;
37   endcase
38 end
```

- A kombinációs rész függvényekkel, ***assign*** értékadással vagy ***always*** blokkal és ***case*** utasítással valósítható meg.
- Az ***always*** blokk érzékenységi listájában a kombinációs hálózat bemenő jeleit kell felsorolni, vagy használhatjuk helyette az **`always @ (*)`** konstrukciót.
- A ***next_state*** regiszter ahhoz kell, hogy a kombinációs és a szekvenciális **always** blokkok kommunikálni tudjanak egymással.
- Itt blokkoló értékadásokat kell használnunk!

A szekvenciális rész

```
1  always @ (posedge clock)
2  begin : OUTPUT_LOGIC
3    if (reset == 1'b1) begin
4      gnt_0 <= #1  1'b0;
5      gnt_1 <= #1  1'b0;
6      gnt_2 <= #1  1'b0;
7      gnt_3 <= #1  1'b0;
8      state <= #1  IDLE;
9    end else begin
10     state <= #1  next_state;
11     case (state)
12        IDLE : begin
13                  gnt_0 <= #1  1'b0;
14                  gnt_1 <= #1  1'b0;
15                  gnt_2 <= #1  1'b0;
16                  gnt_3 <= #1  1'b0;
17             end
18     GNT0 : begin
19               gnt_0 <= #1  1'b1;
20             end
21         GNT1 : begin
22                   gnt_1 <= #1  1'b1;
23                 end
24         GNT2 : begin
25                   gnt_2 <= #1  1'b1;
26                 end
27         GNT3 : begin
28                   gnt_3 <= #1  1'b1;
29                 end
30       default : begin
31                   state <= #1  IDLE;
32                 end
33     endcase
34   end
35 end
```

- Ez a rész csak élre érzékeny logikát tartalmazhat, mint például egy ***always*** blokk, ***posedge clock*** vagy ***negegdge clock*** érzékenységi listával.
- Ebben a részben nem blokkoló értékadást használunk.
- A késleltetés opcionális.

A teljes kód (fsm_full.v) 1. rész

```
module fsm_full(
 clock , // Clock
 reset , // Active high reset
 req_0 , // Active high request from agent 0
 req_1 , // Active high request from agent 1
 req_2 , // Active high request from agent 2
 req_3 , // Active high request from agent 3
 gnt_0 , // Active high grant to agent 0
 gnt_1 , // Active high grant to agent 1
 gnt_2 , // Active high grant to agent 2
 gnt_3   // Active high grant to agent 3
);

input clock ; // Clock
input reset ; // Active high reset
input req_0 ; // Active high request from agent 0
input req_1 ; // Active high request from agent 1
input req_2 ; // Active high request from agent 2
input req_3 ; // Active high request from agent 3

output gnt_0 ; // Active high grant to agent 0
output gnt_1 ; // Active high grant to agent 1
output gnt_2 ; // Active high grant to agent 2
output gnt_3 ; // Active high grant to agent

reg    gnt_0 ; // Active high grant to agent 0
reg    gnt_1 ; // Active high grant to agent 1
reg    gnt_2 ; // Active high grant to agent 2
reg    gnt_3 ; // Active high grant to agent 3
```

```
parameter  [2:0]  IDLE  = 3'b000;
parameter  [2:0]  GNT0  = 3'b001;
parameter  [2:0]  GNT1  = 3'b010;
parameter  [2:0]  GNT2  = 3'b011;
parameter  [2:0]  GNT3  = 3'b100;

reg [2:0] state, next_state;
```

A **parameter** direktívával név szerint hívható konstansokat deklarálhatunk.

A teljes kód (fsm_full.v) 2. rész

```
always @ (state, req_0, req_1, req_2, req_3)
begin   next_state = 0;
  case(state)
    IDLE : if (req_0 == 1'b1) begin
           next_state = GNT0;
            end else if (req_1 == 1'b1) begin
           next_state= GNT1;
            end else if (req_2 == 1'b1) begin
           next_state= GNT2;
            end else if (req_3 == 1'b1) begin
           next_state= GNT3;
         end else begin
           next_state = IDLE;
            end
    GNT0 : if (req_0 == 1'b0) begin
           next_state = IDLE;
            end else begin
           next_state = GNT0;
       end
    GNT1 : if (req_1 == 1'b0) begin
           next_state = IDLE;
            end else begin
           next_state = GNT1;  end
    GNT2 : if (req_2 == 1'b0) begin
           next_state = IDLE;
            end else begin
           next_state = GNT2;  end
    GNT3 : if (req_3 == 1'b0) begin
           next_state = IDLE;
            end else begin
           next_state = GNT3; end
   default : next_state = IDLE;
  endcase
end
```

```
//--- Sequential Section ---------

always @ (posedge clock)
begin : OUTPUT_LOGIC
  if (reset) begin
    gnt_0 <= #1 1'b0;
    gnt_1 <= #1 1'b0;
    gnt_2 <= #1 1'b0;
    gnt_3 <= #1 1'b0;
    state <= #1 IDLE;
  end else begin
    state <= #1 next_state;
    case(state)
     IDLE : begin
                gnt_0 <= #1 1'b0;
                gnt_1 <= #1 1'b0;
                gnt_2 <= #1 1'b0;
                gnt_3 <= #1 1'b0;
            end
     GNT0 :     gnt_0 <= #1 1'b1;

     GNT1 :     gnt_1 <= #1 1'b1;

     GNT2 :     gnt_2 <= #1 1'b1;

     GNT3 :     gnt_3 <= #1 1'b1;

     default :  state <= #1 IDLE;
    endcase
  end
end

endmodule
```

A próbapad modul (fsm_full_tb.v)

```
`include "fsm_full.v"
module fsm_full_tb();
reg clock , reset ;
reg req_0 , req_1 ,  req_2 , req_3;
wire gnt_0 , gnt_1 , gnt_2 , gnt_3 ;
initial begin
  $display("Time\t R0 R1 R2 R3 G0 G1 G2 G3");
  $monitor("%g\t   %b %b %b %b %b %b %b %b",
    $time, req_0, req_1, req_2, req_3,
    gnt_0, gnt_1, gnt_2, gnt_3);
  clock = 0;
  reset = 0;
  req_0 = 0;
  req_1 = 0;
  req_2 = 0;
  req_3 = 0;
  #10 reset = 1;
  #10 reset = 0;
  #10 req_0 = 1;
  #20 req_0 = 0;
  #10 req_1 = 1;
  #20 req_1 = 0;
  #10 req_2 = 1;
  #20 req_2 = 0;
  #10 req_3 = 1;
  #20 req_3 = 0;
  #10 $finish;
end
```

```
always
 #2 clock = ~clock;


fsm_full U_fsm_full(
clock , // Clock
reset , // Active high reset
req_0 , // Active high request from agent 0
req_1 , // Active high request from agent 1
req_2 , // Active high request from agent 2
req_3 , // Active high request from agent 3
gnt_0 , // Active high grant to agent 0
gnt_1 , // Active high grant to agent 1
gnt_2 , // Active high grant to agent 2
gnt_3   // Active high grant to agent 3
);


endmodule
```

A szimuláció eredménye

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

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

> iverilog -o test fsm_full_tb.v

> vvp test

Time         R0 R1 R2 R3 G0 G1 G2 G3
0            0  0  0  0  x  x  x  x
7            0  0  0  0  0  0  0  0
30           1  0  0  0  0  0  0  0
35           1  0  0  0  1  0  0  0
50           0  0  0  0  1  0  0  0
55           0  0  0  0  0  0  0  0
60           0  1  0  0  0  0  0  0
67           0  1  0  0  0  1  0  0
80           0  0  0  0  0  1  0  0
87           0  0  0  0  0  0  0  0
90           0  0  1  0  0  0  0  0
95           0  0  1  0  0  0  1  0
110          0  0  0  0  0  0  1  0
115          0  0  0  0  0  0  0  0
120          0  0  0  1  0  0  0  0
127          0  0  0  1  0  0  0  1
140          0  0  0  0  0  0  0  1
147          0  0  0  0  0  0  0  0
```

A projekt adaptálása a C-M240 kártyára

- Az előző mintapélda adaptálásakor a **C-M240** (ALTERA EPM240) kártyán a 4 db **req_*n*** bemenetet a **K1...K4** nyomógombokkal vezéreljük, a **gnt_*n*** kimeneteket pedig a **D37...D34** LED-ekre kötjük. A ***reset*** jelet a **PIN_47** lábra kötöttük.
- Vegyük figyelembe, hogy a nyomógombok és a LED-ek negatív logika szerint működnek!
- Az állapotgép működtetéséhez az 50 MHz-es rendszer órajel leosztásával állítunk elő 1 Hz-es órajelet.
- A kivezetések hozzárendelése:

req_0	K1	PIN_29
req_1	K2	PIN_28
req_2	K3	PIN_27
req_3	K4	PIN_26

gnt_0	D37	PIN_58
gnt_1	D36	PIN_57
gnt_2	D35	PIN_56
gnt_3	D34	PIN_55

project09.v 1. rész

```
module fsm_full(
 clk_50M , // 50 MHz system clock
 reset , // Active low reset
 req_0 , // Active low request from agent 0
 req_1 , // Active low request from agent 1
 req_2 , // Active low request from agent 2
 req_3 , // Active low request from agent 3
 gnt_0 , // Active low grant to agent 0
 gnt_1 , // Active low grant to agent 1
 gnt_2 , // Active low grant to agent 2
 gnt_3   // Active low grant to agent 3
);

input clk_50M, // 50 MHz system clock
input reset ;  // Active low reset
input req_0 ;  // Active low request from agent 0
input req_1 ;  // Active low request from agent 1
input req_2 ;  // Active low request from agent 2
input req_3 ;  // Active low request from agent 3

output gnt_0 ; // Active low grant to agent 0
output gnt_1 ; // Active low grant to agent 1
output gnt_2 ; // Active low grant to agent 2
output gnt_3 ; // Active low grant to agent

reg    gnt_0 ; // Active low grant to agent 0
reg    gnt_1 ; // Active low grant to agent 1
reg    gnt_2 ; // Active low grant to agent 2
reg    gnt_3 ; // Active low grant to agent 3
```

```
parameter  [2:0]  IDLE  = 3'b000;
parameter  [2:0]  GNT0  = 3'b001;
parameter  [2:0]  GNT1  = 3'b010;
parameter  [2:0]  GNT2  = 3'b011;
parameter  [2:0]  GNT3  = 3'b100;

reg [2:0] state, next_state;
reg [24: 0] count;
reg clock;

//--- Clock signal generation ---
  always @ (posedge clk_50M)
    begin
      if (count == 25000000)
        begin
          clock <= ~clock;
          count <= 0;
        end
      else
        count <= count + 1;
    end
```

project09.v 2. rész

```
always @ (state, req_0, req_1, req_2, req_3)
begin   next_state = 0;
  case(state)
    IDLE : if (req_0 == 1'b0) begin
           next_state = GNT0;
            end else if (req_1 == 1'b0) begin
           next_state= GNT1;
            end else if (req_2 == 1'b0) begin
           next_state= GNT2;
            end else if (req_3 == 1'b0) begin
           next_state= GNT3;
         end else begin
           next_state = IDLE;
            end
    GNT0 : if (req_0 == 1'b1) begin
           next_state = IDLE;
            end else begin
           next_state = GNT0;
       end
    GNT1 : if (req_1 == 1'b1) begin
           next_state = IDLE;
            end else begin
           next_state = GNT1;  end
    GNT2 : if (req_2 == 1'b1) begin
           next_state = IDLE;
            end else begin
           next_state = GNT2;  end
    GNT3 : if (req_3 == 1'b1) begin
           next_state = IDLE;
            end else begin
           next_state = GNT3; end
   default : next_state = IDLE;
  endcase
end
```

```
//--- Sequential Section ---------

always @ (posedge clock)
begin : OUTPUT_LOGIC
  If (!reset) begin
    gnt_0 <= #1 1'b1;
    gnt_1 <= #1 1'b1;
    gnt_2 <= #1 1'b1;
    gnt_3 <= #1 1'b1;
    state <= #1 IDLE;
  end else begin
    state <= #1 next_state;
    case(state)
     IDLE : begin
                 gnt_0 <= #1 1'b1;
                 gnt_1 <= #1 1'b1;
                 gnt_2 <= #1 1'b1;
                 gnt_3 <= #1 1'b1;
            end
     GNT0 :     gnt_0 <= #1 1'b0;

     GNT1 :     gnt_1 <= #1 1'b0;

     GNT2 :     gnt_2 <= #1 1'b0;

     GNT3 :     gnt_3 <= #1 1'b0;

     default :  state <= #1 IDLE;
    endcase
  end
end

endmodule
```