

Digitális Technika

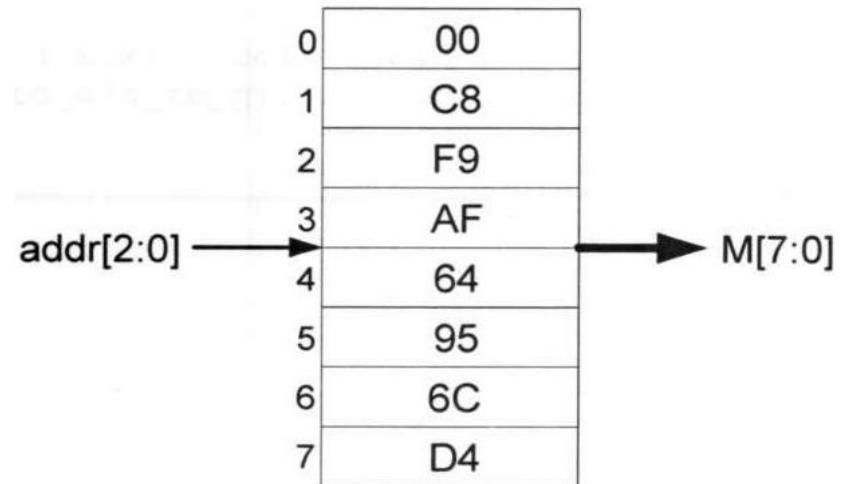
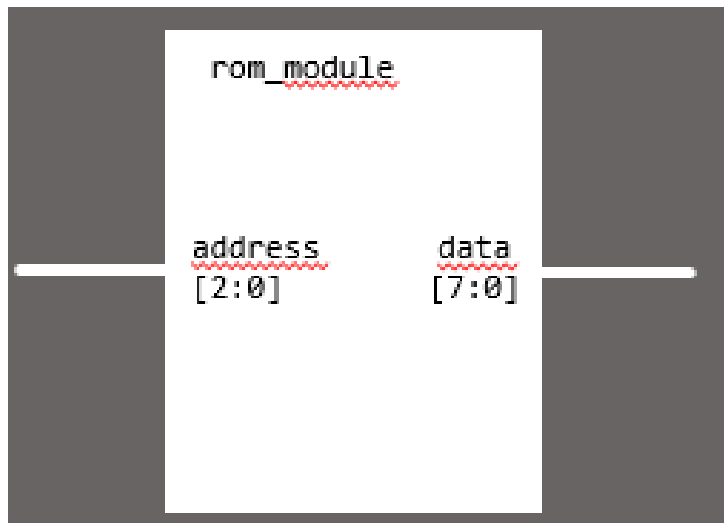
Dr. Oniga István
Debreceni Egyetem, Informatikai Kar

11. Laboratóriumi gyakorlat

- Memóriák
 - ROM modul - 8 bájtos címezhető ROM modul tervezése
 - Elosztott RAM/ROM (DISTRIBUTED RAM/ROM)
 - BLOCK RAM

Lab11_1 Egyszerű ROM modul

- Cél: egy ROM modul készítése, amiben tárolhatunk bájtokat és címezhetjük is őket
- A lényeg: 3 bites címtartomány: $2^3 = 8$ darab cím
- Minden címen egy 1 bájtos érték
- Ez összesen egy 8 bájtos ROM tároló modult jelent



Például, ha a bemenet 3'b110 akkor a kimenet 'h6C lesz.

Egyszerű ROM modul

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);
    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)

            rom[i] = init_data[63-i*8 -: 8];
    end

    assign data = rom[address];

endmodule
```

Modul fejléc, kimeneti és bemeneti paraméterekkel. Jelen esetben a bemenet a memóriacím (ez egy 3 bites érték, ezt Verilogban [2:0] jelzi. A kimenet pedig egy bájt, vagyis az adat, ami ezen a címen található. Ez egy 8 bites érték, jelölése [7:0] és a data néven hivatkozhatjuk

Egyszerű ROM modul

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);

    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)

            rom[i] = init_data[63-i*8 -: 8];
        end

        assign data = rom[address];

    endmodule
```

reg – regiszter
reg [7:0] – 8 bites regiszter
rom [0:7] – 8 bites regiszterek 8 bites rom nevű tömbje (fontos hogy [0:7] kell a tömb jelölésnél, ez jelzi, hogy a bájtok a memóriában hogyan indexelhetők)

Ilyenformán 1 kilobájtos ROM:
reg [7:0] rom [0:1024]

Egyszerű ROM modul

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);

    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)

            rom[i] = init_data[63-i*8 -: 8];
    end

    assign data = rom[address];

endmodule
```

8 bájtos inicializáló érték. A modul „létrehozásakor” kapja értékül a regisztertömb. Lényeges, hogy ez tetszőleges érték, lehetne pl. csupa 0 érték, de most a LED-eken szeretnénk megjeleníteni ezért nem árt látni, hogy jól működik-e, azok az értékek szerepelnek-e ott amik meg lettek adva.

A memóriának személyes adatokat kell tartalmaznia:
Neptun kód (6 byte) + MI (2 byte)
karakterének az ASCII kód hexadecimális értékeit
<https://cs.smu.ca/~porter/csc/ref/ascii.html>

Például: Neptun kód: A1B2C3MI=> 64'h4131423243334D49;

Egyszerű ROM modul

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);

    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)

            rom[i] = init_data[63-i*8 -: 8];
    end

    assign data = rom[address];

endmodule
```

Ciklusváltozó. A szintézer nem enged blokkon belüli változó deklarációt.

Egyszerű ROM modul

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);

    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)
            rom[i] = init_data[63-i*8 -: 8];
        end

    assign data = rom[address];

endmodule
```

initial blokk: egyszer fut le a modul élete során, a létrehozásakor.

Egyszerű ROM modul

```
`timescale 1ns / 1ps
module rom_module(input wire[2:0] address, output wire[7:0] data);
    reg [7:0] rom [0:7];
    parameter init_data = 64'h00C8F9AF64956CD4;
    integer i;
    initial
    begin
        for(i=0; i<8; i=i+1)
            rom[i] = init_data[63-i*8 -: 8];
    end
    assign data = rom[address];
endmodule
```

for ciklus: végig
megyünk a bájt tömbön
és beállítjuk az értékeket
a kezdőértéket
tartalmazó változóból.

init_data[63-i*8 -: 8]
ez iterációként kifejtve:
init_data[63:56]
init_data[55:48]
...
init_data[7:0]

Egyszerű ROM modul

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);

    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)
            rom[i] = init_data[63-i*8 -: 8];
        end

    assign data = rom[address];

endmodule
```

Az adat kimenethez (data) hozzárendeljük a kért címen lévő memória értéket

Top modul

```
module topmodule(input [2:0] SW, output [7:0] LED) ;  
    rom_module memory(.address(SW), .data(LED));  
  
endmodule
```

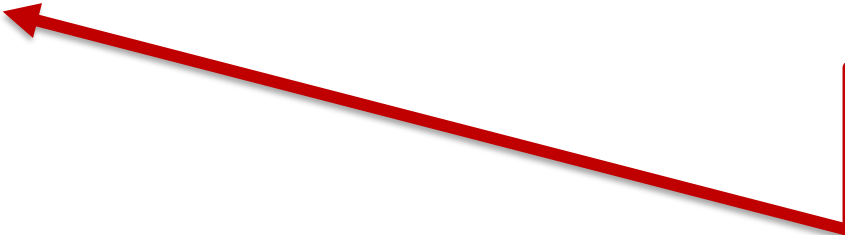
Bemenetet kapjuk a kapcsolókról (első 3 kapcsoló, egy-egy bit értéket jelentenek, címzés 000-től 111-ig (0-7))

Kimenet pedig 8 LED, amin a kapcsolók által beállított címen lévő bájt bitjei jelennek majd meg.

Top modul

```
module topmodule(input [2:0] SW, output [7:0] LED) ;  
  
    rom_module memory(.address(SW), .data(LED));  
  
endmodule
```

Constraints fájl hozzáadása



Példányosítjuk a ROM modult „memory” néven (ez tetszőleges lehet, felfogható változónévként)

Lab11_2 DISTRIBUTED RAM/ROM (Elosztott RAM/ROM)

```
19 //  
20 //////////////////////////////////////  
21 module EX28_top(  
22     |input clk,  
23     input [3 : 0] addr,  
24     output [7 : 0] spo  
25 );  
26  
27  
28 endmodule  
29
```

1. Hozzuk létre egy új projektet, a nexys4 kártya paramétereivel
2. A projekthez adjunk hozzá egy új verilog modult pl „EX28_top” néven
3. Hozzuk létre a szükséges input és output változókat

Memory Initialisation file (MIF) létrehozása

A memória modul létrehozásához szükségünk lesz egy MIF fájlra. Ezzel tudjuk inicializálni a memóriánkat. Egy szövegszerkesztő programmal (Pl: Notepad++) hozzunk létre egy fájlt majd írjuk bele a következőt:

```
1 ; Example 28 Initialization file for a 16x8 distributed ROM
2 memory_initialization_radix = 16;
3 memory_initialization_vector =
4 0 C8 F9 AF
5 64 95 6C D4
6 39 E7 5A 96
7 84 37 28 4C;
8
```

A memóriának személyes adatokat kell tartalmaznia:

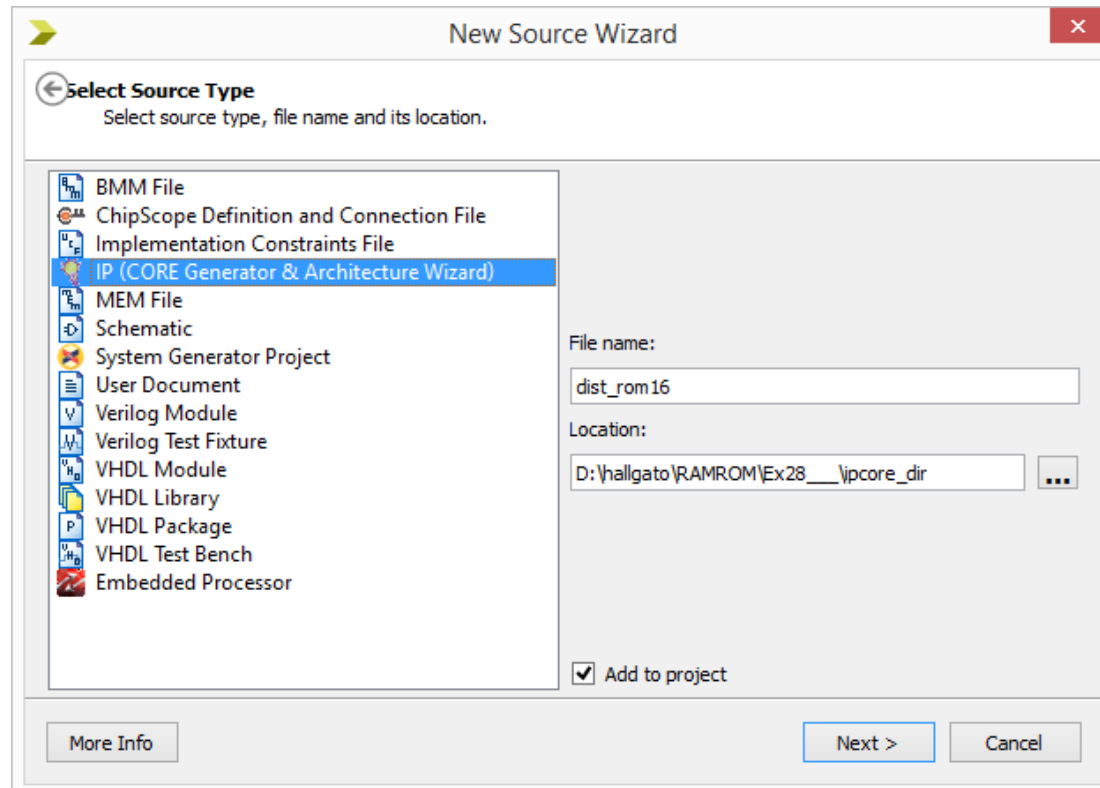
Neptun kód (6 byte) + MI (2 byte) + év + hónap + nap (4 + 2+ 2 byte)

karaktereinek az ASCII kód hexadecimális értékeit <https://cs.smu.ca/~porter/csc/ref/ascii.html>

Például: Neptun kód: A1B2C3MI20210526 => 41 31 42 32 43 33 4D 49 32 30 32 31 30 35 32 36;

Mentsük el a fájlt a projekt könyvtárba, pl EX28_init.coe néven. FONTOS, hogy a fájl kiterjesztése .coe legyen!

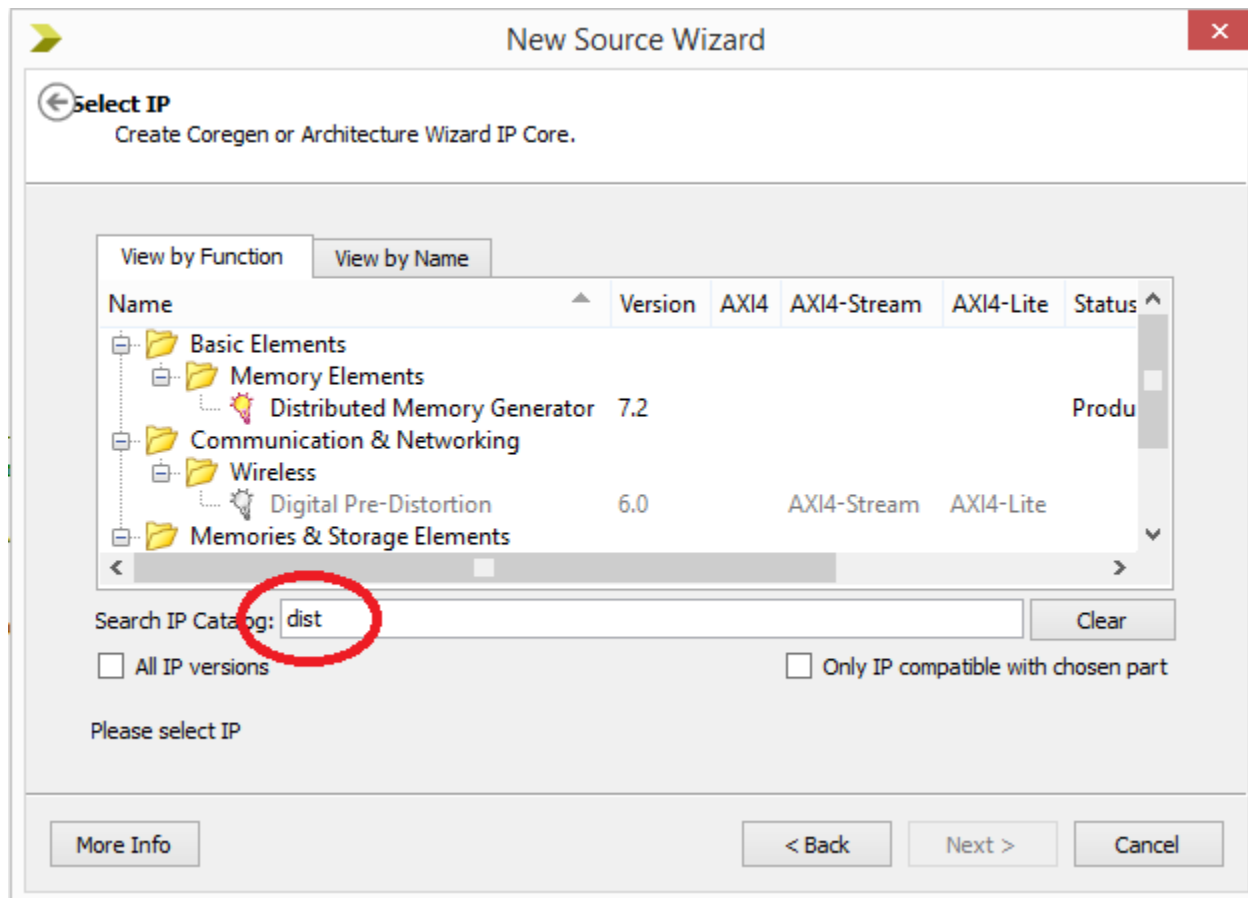
Core Generator használata



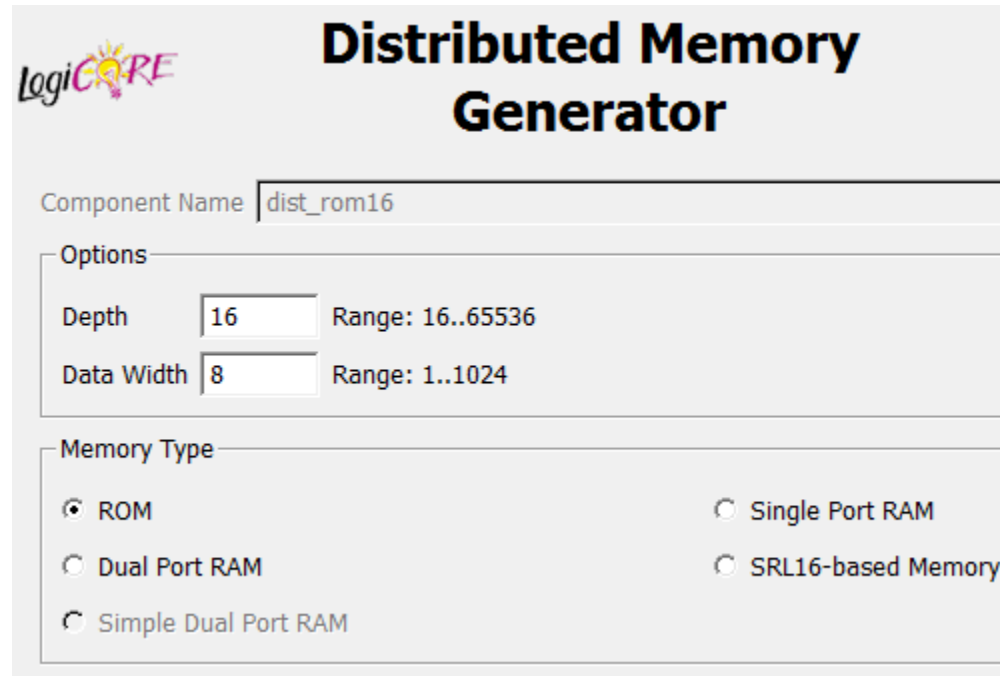
1. Adjunk hozzá egy új forrást a projekthez, majd a „New Source Wizard” ablakban válasszuk az IP Core forrás típust.
2. Adjunk nevet az új forrásnak (pl „dist_rom16”) majd kattintsunk a Next > gombra.

Distributed RAM/ROM létrehozása

A megjelenő ablakban keressük meg a *Distributed Memory Generator-t*. (Használjuk a keresőmezőt)



A Memória paramétereinek beállítása



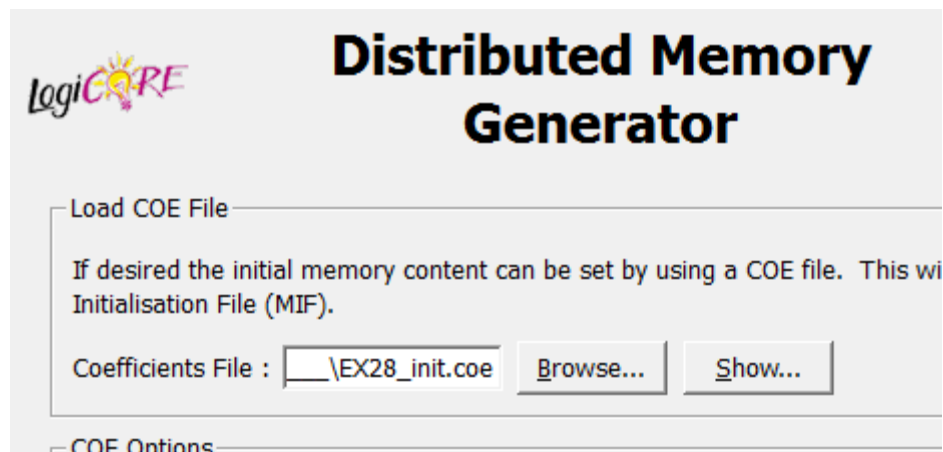
The screenshot shows the 'Distributed Memory Generator' configuration window. At the top left is the 'LogiCORE' logo. The title 'Distributed Memory Generator' is centered at the top. Below the title, there is a 'Component Name' field containing 'dist_rom16'. Underneath is an 'Options' section with two input fields: 'Depth' set to '16' with a range of '16..65536', and 'Data Width' set to '8' with a range of '1..1024'. Below the options is a 'Memory Type' section with five radio button options: 'ROM' (selected), 'Dual Port RAM', 'Simple Dual Port RAM', 'Single Port RAM', and 'SRL16-based Memory'.

Állítsuk be a memória mélységét 16-ra, az adat szélességet pedig 8-ra.

A memória típusok közül válasszuk ki a ROM-ot.

MIF fájl hozzáadása a memóriamodulhoz

A generátor harmadik oldalán tallózzuk be a korábban létrehozott .ceo fájlunkat

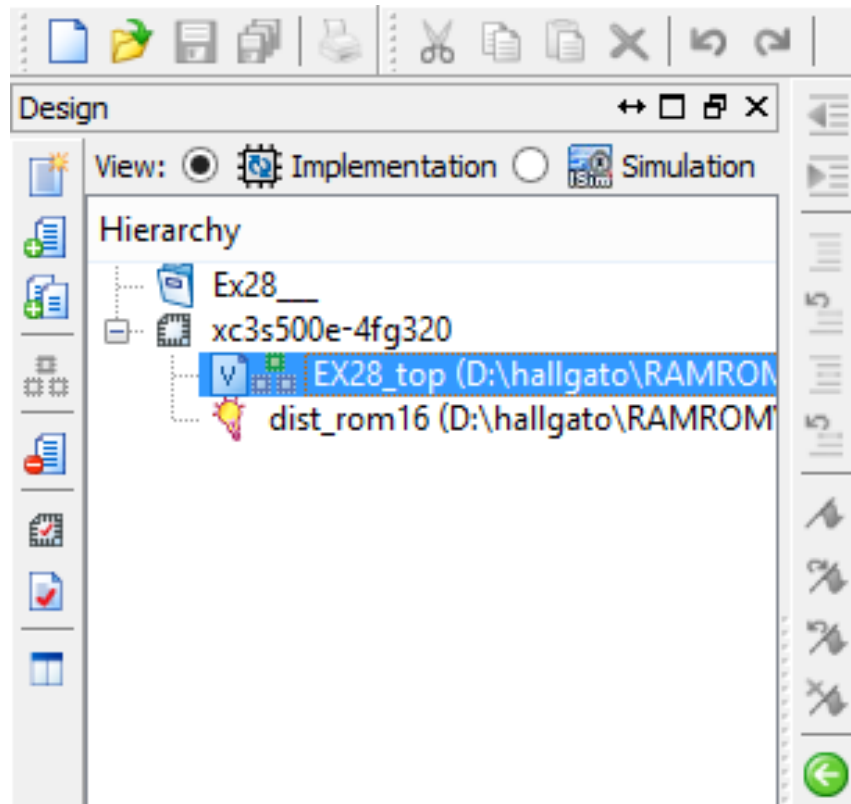


The screenshot shows the 'Distributed Memory Generator' interface. At the top left is the 'LogiCORE' logo. The main title is 'Distributed Memory Generator'. Below the title, there is a section titled 'Load COE File'. A text box contains the instruction: 'If desired the initial memory content can be set by using a COE file. This will be the Initialisation File (MIF)'. Below this text, there is a label 'Coefficients File :', followed by a text input field containing the path '\\EX28_init.coe'. To the right of the input field are two buttons: 'Browse...' and 'Show...'. At the bottom left, there is a checkbox labeled 'COE Options'.

A Show... gombbal meg tudjuk tekinteni a betöltött fájl tartalmát. Itt láthatjuk ha a fájl nem lett megfelelően létrehozva.

Memória modul generálása

Ha kész vagyunk a paraméterek beállításával kattintsunk a Generate gombra, majd várjuk meg amíg a modul elkészül.

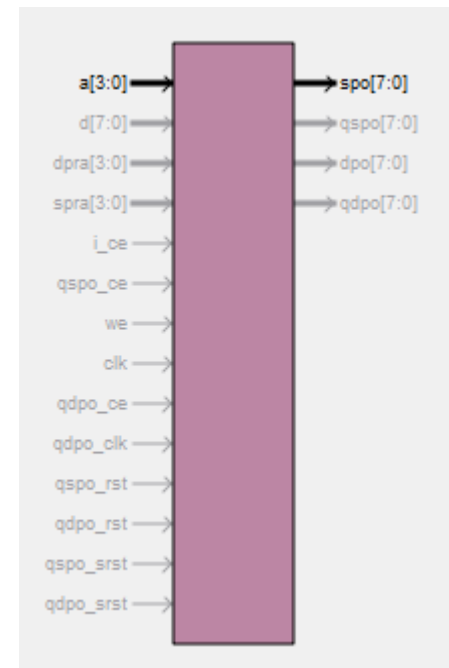


Az új modul meghívása a top modulban

Lépünk a top modulra, majd hívjuk meg a modult és kössük össze a modul ki és bemeneteit a top modulban létrehozott inputokkal és outputtal.

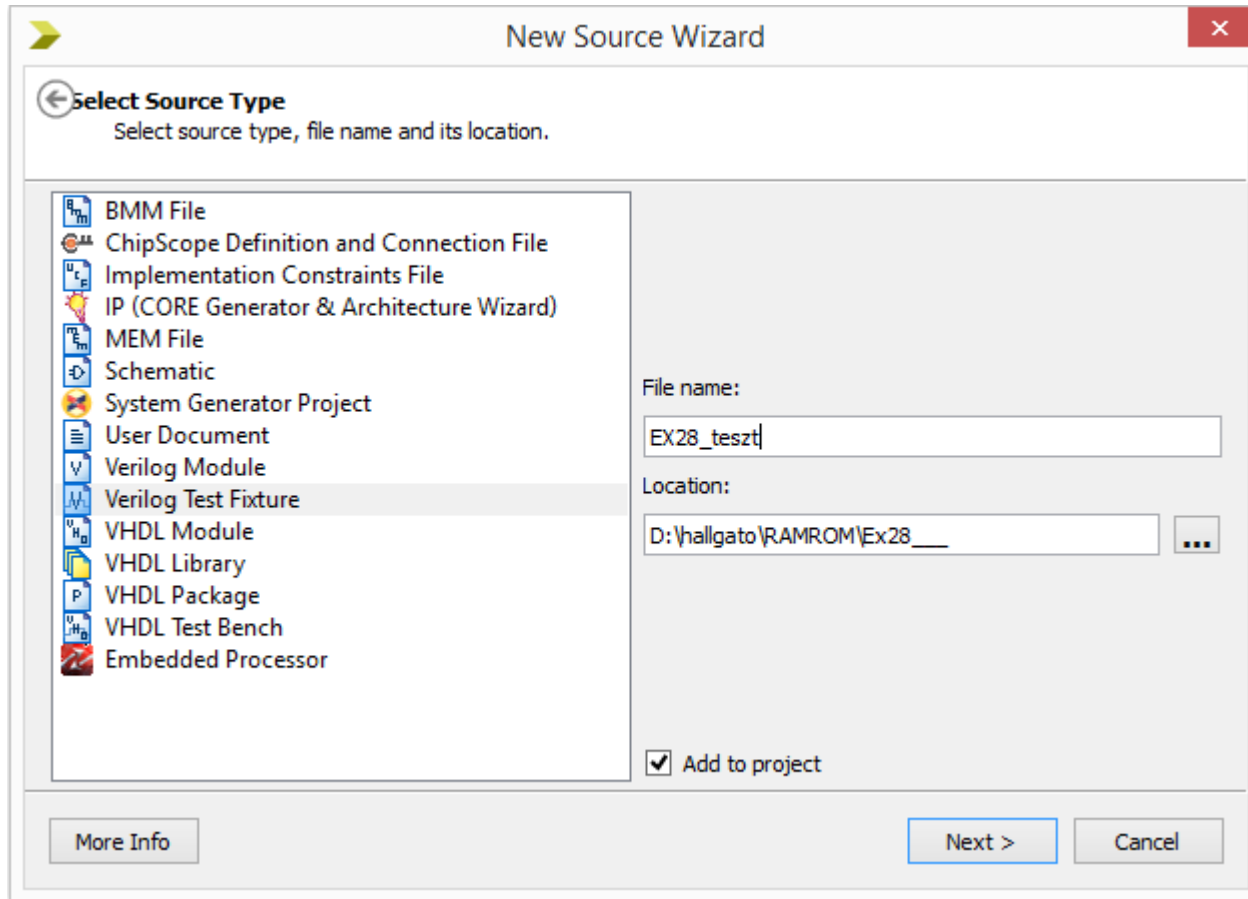
```
20 ///////////////////////////////////////////////////////////////////
21 module EX28_top(
22     input clk,
23     input [3 : 0] addr,
24     output [7 : 0] spo
25 );
26
27 dist_rom16 MEM(
28     .a(addr), // input [3 : 0] a
29     .spo(spo) // output [7 : 0] spo
30 );
31
32 endmodule
33
```

A modul be és kimenetei a generálás során megjelenő ábrán jól láthatók.



Teszt modul létrehozása

Adjunk hozzá a projektben lévő top modulhoz egy új Verilog Test Fixture fájlt.



Teszt modul beállítása

```
initial begin
    // Initialize Inputs
    clk = 0;
    addr = -1;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here

end

always@(*)
begin
    #50
    clk <= ~clk;
end

always@(clk)
begin

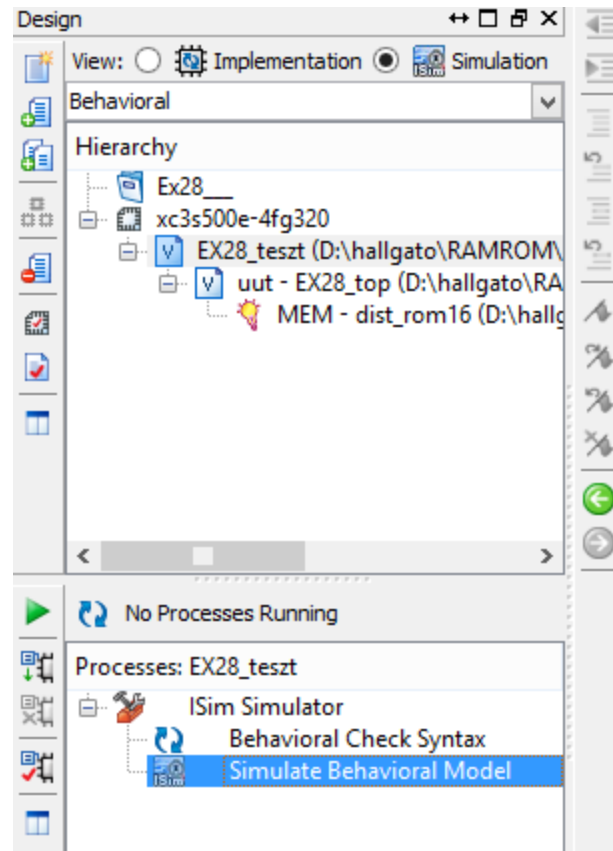
    addr = addr + 1;
end
```

A teszt fájl végén állítsuk be az órajelet.

Hogy a szimulációban jól látszódjon, hogy a különböző címeken milyen értékek tárolódnak, növeljük a cím változó értékét órajelenként.

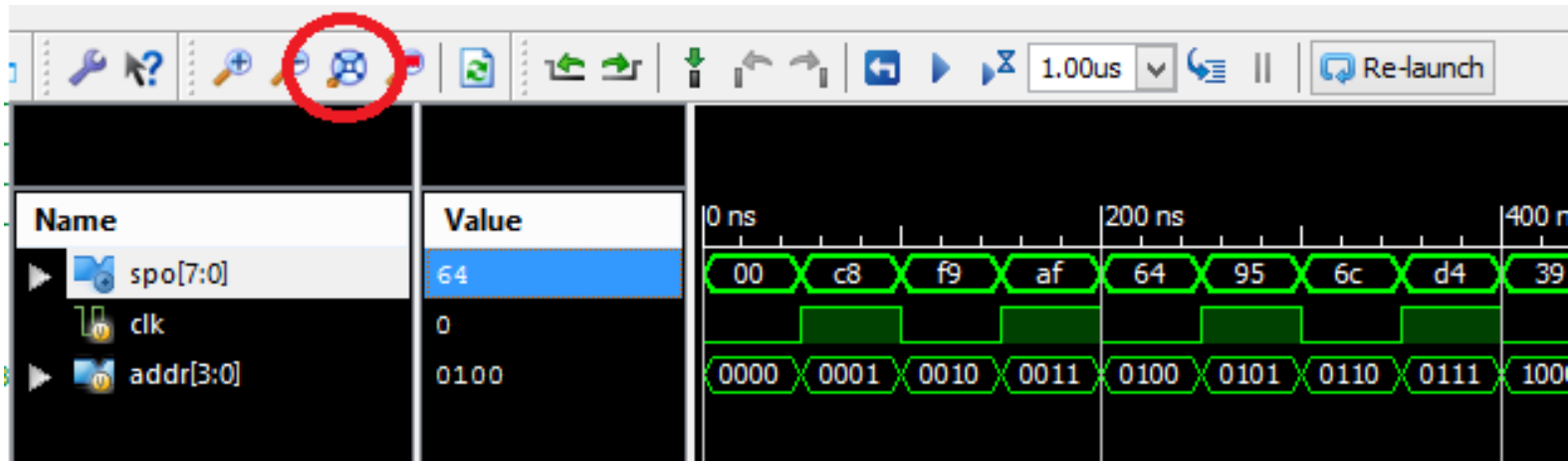
Szimuláció futtatása

Lépünk a Simulation fülre, majd futtassuk az EX28_teszt modult.



Szimuláció eredménye

Az output sorra jobb gombbal kattintva, a Radix menüből válasszuk ki a Hexadecimal lehetőséget. (Használjuk a nagyítót)



Az eredmény összehasonlítható a korábban létrehozott .coe fájl tartalmával. A memóriában valóban az inicializáláskor megadott értékek találhatók.

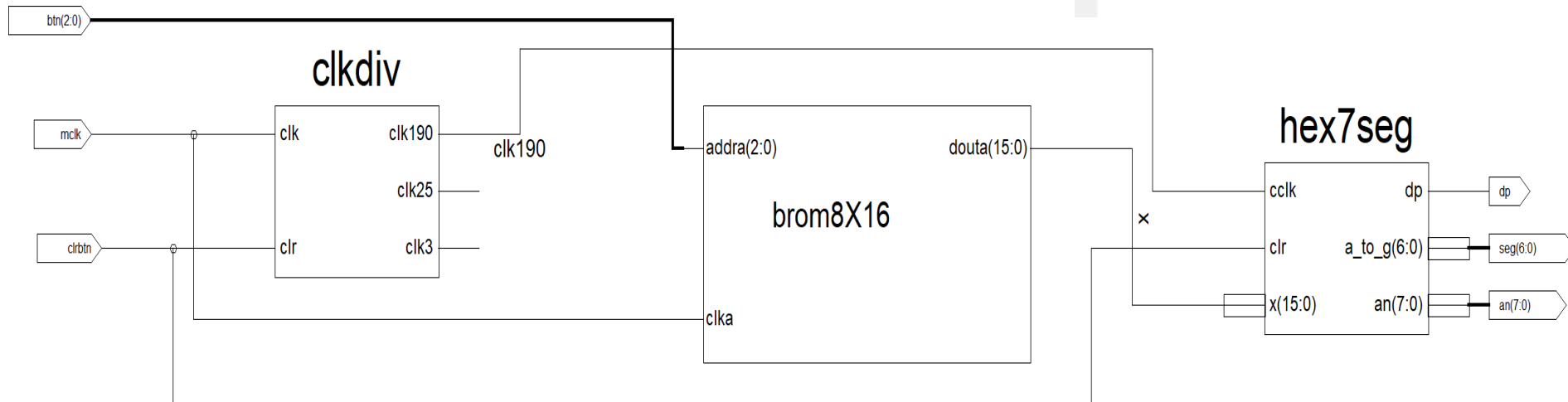
Program fordítása, betöltése

- Adjunk hozzá a projekthez egy .ucf fájlt majd rendeljük hozzá a szükséges I/O eszközöket.
- Le is tölthetünk egy előre elkészített teljes UCF fájlt, ilyenkor a nem szükséges sorokat rakjuk megjegyzésbe, vagy állítsuk be hogy a fordító ne vegye figyelembe a nem használt sorokat.
- Fordítsuk a kódot, majd töltsük be az FPGA-ba.
- Az UCF-ben beállított switch-eken memóriacímeket beállítva, a ledeken megjelennek binárisan az adott címhez tartozó értékek.

Lab11_3 BLOCK RAM

1. Hozzunk létre egy új projektet, a nexys4 kártya paramétereivel
2. A projekthez adjunk hozzá egy új verilog modult például „EX30_top” néven
3. Hozzuk létre a szükséges input és output változókat

```
////////////////////////////////////  
module EX30_top(  
  input wire mclk ,  
  input wire [3:0] btn ,  
  input wire clrbtn ,  
  output wire [6:0] seg ,  
  output wire dp ,  
  output wire [7:0] an  
);
```



Memory Initialisation file (MIF) létrehozása

A memória modul létrehozásához szükségünk lesz egy MIF fájlra. Ezzel tudjuk inicializálni a memóriánkat.

Egy szövegszerkesztő programmal (Pl: Notepad++) hozzunk létre egy fájlt majd írjuk bele a következőt:

Listing 4.7 Example30.coe

```
; Example 30 Initialization file for a 8x16 block ROM
memory_initialization_radix = 16;
memory_initialization_vector =
0000 1111 2222 3333
4444 5555 6666 7777;
```

A memóriának személyes adatokat kell tartalmaznia:

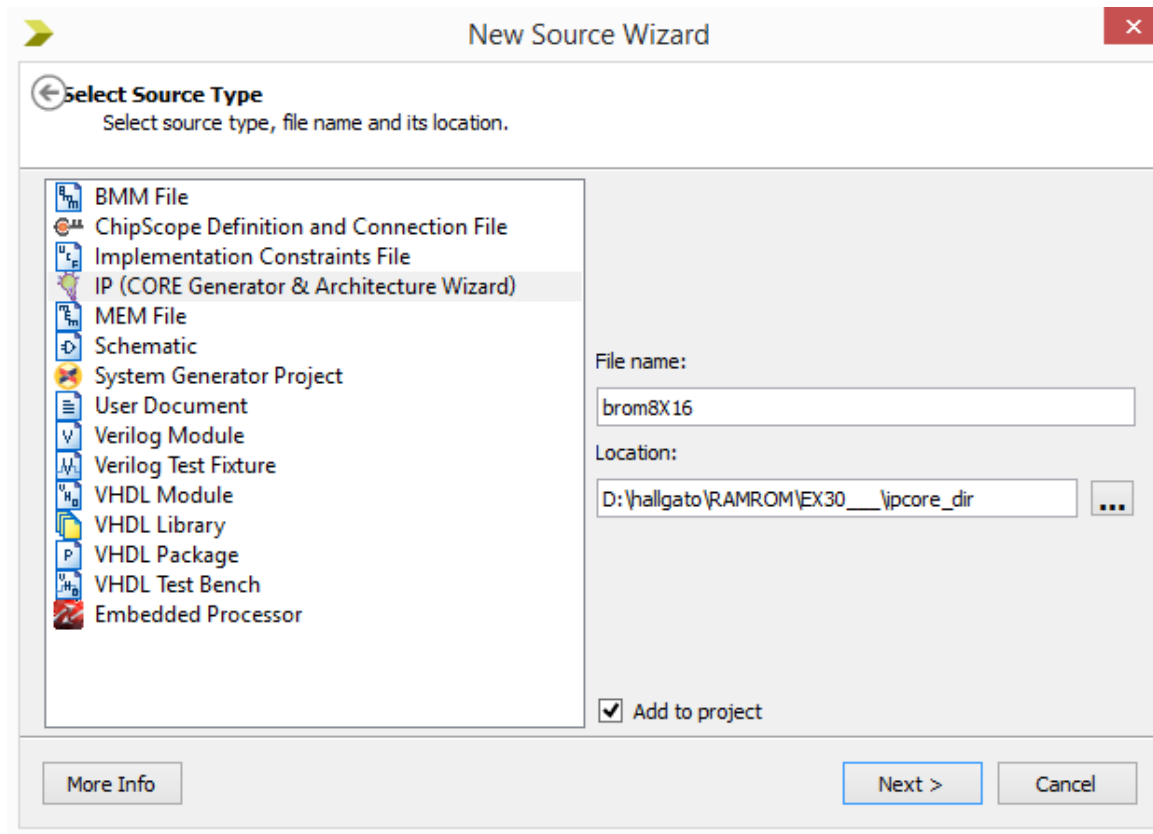
Neptun kód (6 byte) + MI (2 byte) + év + hónap + nap (4 + 2+ 2 byte)

karaktereinek az ASCII kód hexadecimális értékeit <https://cs.smu.ca/~porter/csc/ref/ascii.html>

Például: Neptun kód: A1B2C3MI20210526 => 4131 4232 4333 4D49 3230 3231 3035 3236;

Mentsük el a fájlt a projekt könyvtárba, pl EX30_init.coe néven. FONTOS, hogy a fájl kiterjesztése .coe legyen!

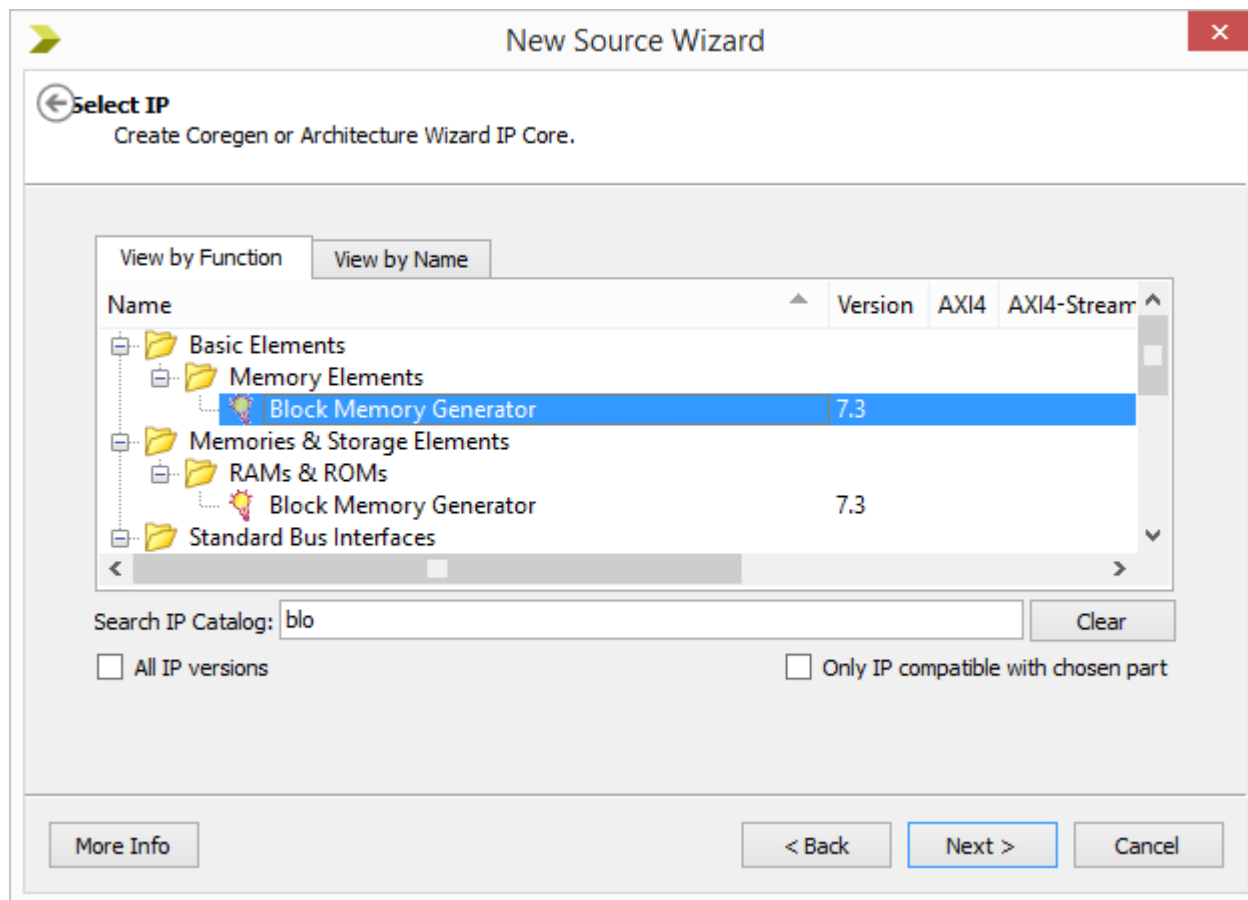
Core Generator használata



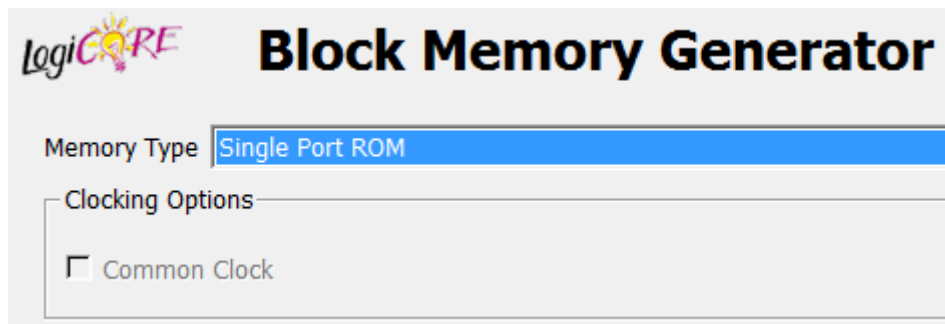
1. Adjunk hozzá egy új forrást a projekthez, majd a „New Source Wizard” ablakban válasszuk az IP Core forrás típust.
2. Adjunk nevet az új forrásnak (pl „brom8x16”) majd kattintsunk a Next > gombra.

Block RAM/ROM létrehozása

A megjelenő ablakban keressük meg a *Block Memory Generator-t*. (Használjuk a keresőmezőt)

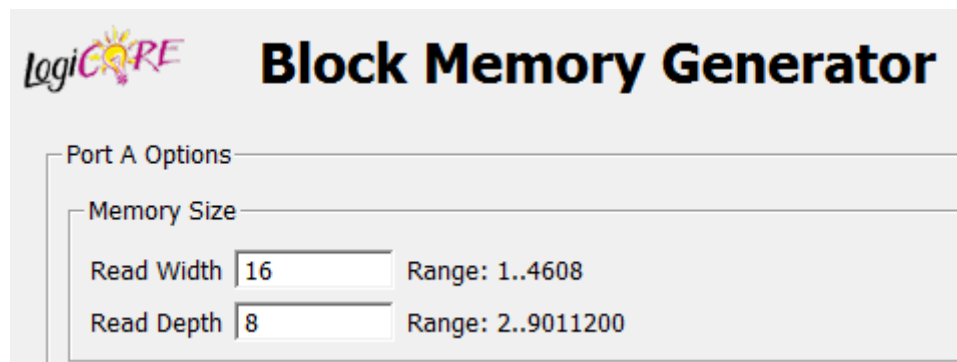


A Memória paramétereinek beállítása



The screenshot shows the 'LogiCORE Block Memory Generator' interface. The 'Memory Type' dropdown menu is open, and 'Single Port ROM' is selected. Below this, there is a 'Clocking Options' section with a checkbox for 'Common Clock' which is currently unchecked.

A 2. oldalon válasszuk ki a „Single Port ROM” típust.

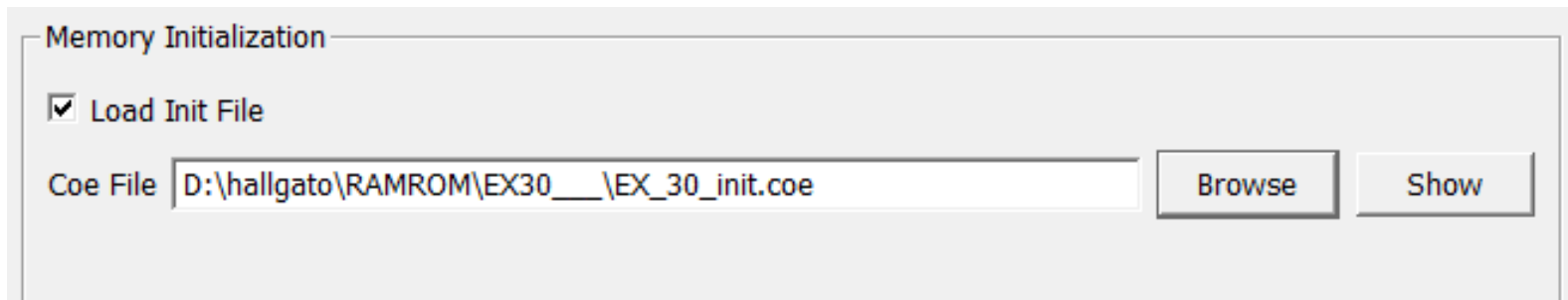


The screenshot shows the 'LogiCORE Block Memory Generator' interface. The 'Port A Options' section is expanded, showing 'Memory Size' settings. The 'Read Width' is set to 16 (Range: 1..4608) and the 'Read Depth' is set to 8 (Range: 2..9011200).

A 3. oldalon állítsuk be a memória méreteit.

MIF fájl hozzáadása a memóriamodulhoz

A generátor 4. oldalán pipáljuk be a Load Init File checkbox-ot, majd tallózzuk be a korábban létrehozott .coe fájlunkat



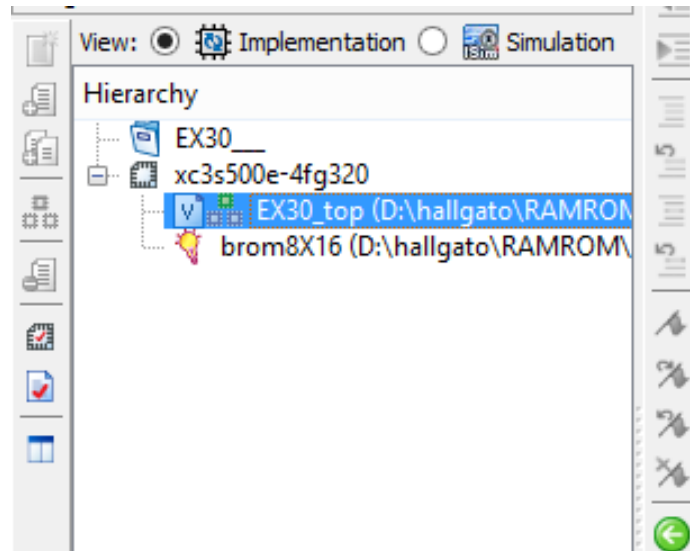
Memory Initialization

Load Init File

Coe File

Memória modul legenerálása

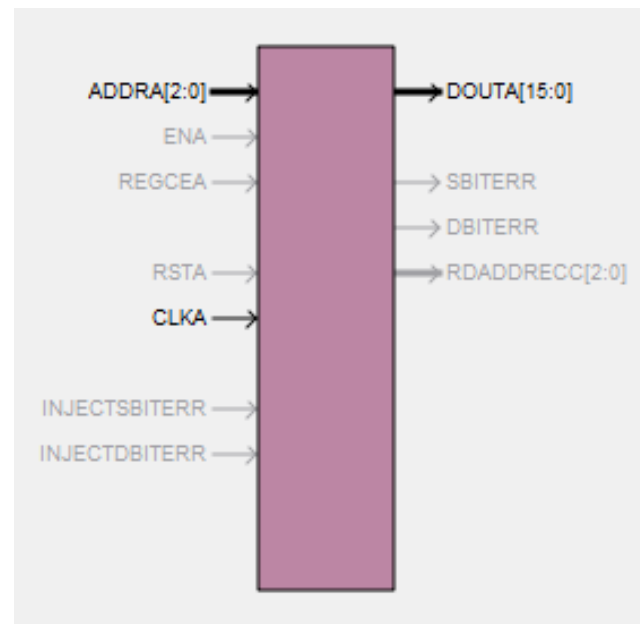
Ha kész vagyunk a paraméterek beállításával kattintsunk a Generate gombra, majd várjuk meg amíg a modul elkészül.



Az új modul meghívása a top modulban

Lépünk a top modulra, majd hívjuk meg a modult és kössük össze a modul ki és bemeneteit a top modulban létrehozott inputokkal és outputtal.

```
wire [15:0] x;  
wire [2:0] addr;  
wire clk190;  
  
clkdiv U1 (.clk(mclk),  
  .clr (clrbtn),  
  .clk190(clk190)  
);  
  
hex7seg X2 (.x(x),  
  .clk(clk190),  
  .clr(clrbtn),  
  .a_to_g(seg),  
  .an(an),  
  .dp(dp)  
);  
  
brom8X16 U5 (  
  .addr(btn), // Bus [2 : 0]  
  .clka(mclk),  
  .douta(x) // Bus [15 : 0]  
);  
endmodule
```



A modul be és kimenetei a generálás során megjelenő ábrán jól láthatók.

clkdiv modul létrehozása

```
20 ///////////////////////////////////////////////////////////////////
21 module clkdiv (
22     input wire clk ,
23     input wire clr ,
24     output wire clk190 ,
25     output wire clk25 ,
26     output wire clk3
27 ) ;
28
29 reg [24:0] q;
30 // 25-bit counter
31 always @(posedge clk or posedge clr)
32 begin
33     if(clr == 1)
34         q <= 0;
35     else
36         q <= q + 1;
37 end
38 assign clk190 = q[17]; // 190 Hz
39 assign clk25 = q[0]; // 25 MHz
40 assign clk3 = q[23]; // 3 Hz
41 endmodule
42
```

Hex7seg modul létrehozása 1.

```
////////////////////////////////////  
module hex7seg (  
input wire [15:0] x ,  
input wire cclk ,  
input wire clr ,  
output reg [6:0] a_to_g ,  
output reg [7:0] an ,  
output wire dp  
);  
reg [1:0] s;  
reg [3:0] digit;  
wire [3:0] aen;  
assign dp = 1;  
// set aen[3:0] for leading blanks  
assign aen[3] = x[15] | x[14] | x[13] | x[12];  
assign aen[2] = x[15] | x[14] | x[13] | x[12] | x[11] | x[10] | x[9] | x[8] ;  
assign aen[1] = x[15] | x[14] | x[13] | x[12] | x[11] | x[10] | x[9] | x[8] | x[7] | x[6] | x[5] | x[4] ;  
assign aen[0] = 1; // digit 0 always on  
  
// Quad 4-to-1 MUX: mux44  
always @(*)  
case(s)  
0: digit = x [3:0] ;  
1: digit = x [7:4] ;  
2: digit = x [11:8];  
3: digit = x [15:12] ;  
default: digit = x[3:0];  
endcase  
// 7-segment decoder: hex7seg
```

Hex7seg modul létrehozása 2.

```
// 7-segment decoder: hex7seg
always @(*)
case(digit)
0: a_to_g = 7'b1000000;
1: a_to_g = 7'b11111001;
2: a_to_g = 7'b0100100;
3: a_to_g = 7'b0110000;
4: a_to_g = 7'b0011001;
5: a_to_g = 7'b0010010;
6: a_to_g = 7'b0000010;
7: a_to_g = 7'b11111000;
8: a_to_g = 7'b0000000;
9: a_to_g = 7'b0010000;
'hA: a_to_g = 7'b0001000;
'hb: a_to_g = 7'b1100000;
'hC: a_to_g = 7'b0110001;
'hd: a_to_g = 7'b1000010;
'hE: a_to_g = 7'b0110000;
'hF: a_to_g = 7'b01111000;
default: a_to_g = 7'b0000001; // 0
endcase
// Digit select
always @(cclk)
begin
an = 4'b1111;
if(aen[s] == 1)
an[s] = 0;
end
77 // 2-bit counter
78
79 always @(posedge cclk or posedge clr)
80     begin
81         if(clr == 1)
82             s <= 0;
83         else
84             s <= s + 1 ;
85     end
86
87 endmodule
```

Program fordítása, betöltése

- Ucf fájl létrehozása
- Fordítsuk a kódot, majd töltsük be az FPGA-ba.
- Az UCF-ben beállított switch-eken memóriacímeket beállítva, a hét szegmenses kijelzőn megjelennek az adott címhez tartozó értékek