BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

HDL nyelvek: VHDL

Fehér Béla
Szántó Péter, Lazányi János, Raikovich Tamás
BME MIT
FPGA laboratórium

Strukturális elemek

- **Entity: interfész megadás**
- **Architecture: viselkedés, funkcionalitás leírása**
- **Configuration: Architecture/Entity választás**
- **Package: függvények, típus deklarációk**
- **Library: lefordított VHDL objektumok gyűjteménye**

Entity

- **Interfész megadás a modul és a külvilág között**

```
entity hadder is
port (
  a, b:      in  std_logic;
  s, co:     out std_logic
);
end hadder;
```

Port típusok

- **in**
  - Bemenet, csak olvasható
- **out**
  - Kimenet, csak írható (!)
- **buffer**
  - Kimenet, de olvasható, csak egy meghajtó jel
- **inout**
  - Kétirányú port

Architecture

- **A terv implementációt tartalmazó blokk**
- **Egy Entity tartalmazhat több Architecture-t**
  - Entity portok elérhetők az összes Architecture-ből
- **Párhuzamosan végrehajtott utasításokat tartalmaz**

```
entity hadder is
port (
  a, b:       in  std_logic; ……..
);
end hadder;

architecture rtl of hadder is
……..
begin
……..
……..
end rtl;
```

Strukturális leírás

- **A VHDL támogatja a hierarchikus leírást, azaz a komponensek „példányosítását”**
  - Komponens deklaráció
  - Példányosítás (egyedi névvel)
  - Port hozzárendelés
    - név szerint
    - sorrend szerint

Komponens deklaráció

- **Példányosítani kívánt komponensek deklarációja**

```
architecture rtl of adder is
.........
component hadder
port (
  a, b:      in  std_logic;
  s, co:     out std_logic
);
end component;

.........
.........
.........
```

Példányosítás (1)

```
port(a, b, ci   : in  std_logic;
     so, co     : out std_logic);
……
……
architecture struct of adder is
--komponens deklaráció
signal c0, c1, s0 : std_logic;

begin

hadder_0: hadder
port map(a, b, s0, c0);

hader_1: hadder
port map(s0, c0, so, c1);

or_0: or2
port map(c0, c1, co);
```

Példányosítás (2)

```
port(a, b, ci    : in  std_logic;
     so, co      : out std_logic);
……
……
architecture struct of adder is
--komponens deklaráció
signal c0, c1, s0 : std_logic;

begin

hadder_0: hadder
port map(a=>a, b=>b, s=>s0, co=>c0);

hader_1: hadder
port map(a=>s0, b=>c0, s=>so, co=>c1);

or_0: or2
port map(a=>c0, b=>c1, o=>co);
```

Példányosítás (3)

```
port(a, b, ci    : in  std_logic;
     so, co      : out std_logic);
……
……
architecture struct of adder is
-- NINCS komponens deklaráció!!!
signal c0, c1, s0 : std_logic;

begin

hadder_0: entity work.hadder(rtl)
port map(a=>a, b=>b, s=>s0, co=>c0);

hader_1: entity work.hadder(rtl)
port map(a=>s0, b=>c0, s=>so, co=>c1);

or_0: entity work.or2(rtl)
port map(a=>c0, b=>c1, o=>co);
```

Package

- **Package tartalmazhat:**
  - konstans deklarációk
  - típus deklarációk
  - komponensek
  - szubrutinok
- **Felhasználás Entity-ben:**

```
package USER_PACK is
.....
end USER_PACK;
```

```
use work.USER_PACK.all;
```

Package példa

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package USER_PACK is

function MIN (a: integer; b: integer) return integer;

end USER_PACK;

package body USER_PACK is

function MIN( a: integer; b: integer) return integer is
begin
   if (a<b) then return a;
   else return b;
   end if;
end MIN;

end USER_PACK;
```

Library

- **Minden lefordított objektum Library-be kerül**
  - package, entity, architecture, configuration
- **Saját objektumok: work library**
- **Standard library-k**
- **Felhasználás:**

```
library IEEE;
use IEEE.std_logic_1164.all;
```

Megjegyzések, konstansok

- **Megjegyzések**
  - -- egy soros
  - Nincs több soros
- **Konstansok**
  - 1 bites: ‘0’, ‘1’
  - Vektor
    - Bináris: „00010010”
    - Hexadecimális: x”AB” (a konstans és a változó szélessége ugyanaz!!)
    - CONV_STD_LOGIC_VECTOR(*konstans, bitszám*)

Adattípusok

- **Standard adattípusok**
  - boolean (true, false)
  - bit (0, 1)
  - integer (platform függő, min. 32 bit)
  - real (lebegőpontos, nem szintetizálható)
  - time
- **Library-kben előre definiált adattípusok**
  - szintézisre: std_logic_1164
- **Saját adattípusok**

Bit, integer

- **Bit típus (csak 0, 1)**

```
signal sb  : bit;
signal sbv : bit_vector(3 downto 0);

sb  <= '0';
sbv <= „0000"; v. sbv <= x"A";
```

- **Integer**

```
signal si  : integer range 0 to 15;

si <= 2;
```

- **Szintézis után ugyanaz (lehet) az eredmény**

Vektor változók

- **Konkatenálás**
  - &: csak kifejezés jobb oldalán
- **Csoportosítás**
  - (, , ,): kifejezés mindkét oldalán
- **Kiválasztás**

```
signal a2, b2, c2, d2: bit_vector(1 downto 0);
signal a4, b4, c4: bit_vector(3 downto 0);

a4 <= a2 & b2;
(c2, d2) <= „0101”;
d2 <= b4(2 downto 1);
c4 <= (0=>’1’, others=>’0’);
```

Time típus

- **Mértékegység: fs, ps, ns, us, ms, sec, min, hr**
- **Szimulációhoz, modellezéshez használható**
  - végrehajtás késleltetése
  - fizikai késleltetések modellezése
- **Szorozható és osztható (eredmény: ‘time’):**
  - integer
  - real

Time példa

```
constant PERIOD : time := 5 ns;

begin

process
begin
   wait for PERIOD;
.........
   wait for 2*PERIOD;
.........
   wait for 3.5*PERIOD;
.........
end process;

CLK <= not CLK after PERIOD/2;
```

IEEE std_logic

- **std_ulogic, std_ulogic_vector, std_logic, std_logic_vector**
- **értékek:**
  - ` U `, nem inicializált
  - ` X `, ismeretlen
  - ` 0 `,
  - ` 1 `,
  - ` Z `, nagy impedanciás
  - ` W `, ismeretlen (gyenge meghajtás)
  - ` L `, gyenge 0
  - ` H `, gyenge 1
  - ` - `, don`t care

std_logic/std_ulogic

- **Resolved/Unresolved**

```
a <= '1';
a <= b;
```

- **Eredmény:**
  - std_logic: a = 1
  - std_ulogic: ERROR
- **Elvileg, de szintézer függő!**
- **Szimuláció alapállapot: U (undefined)**
- **Ipari szabvány: std_logic**

Aritmetikai műveletek

- **std_logic_arith package**
  - signed és unsigned adattípus

```
signal a : signed(7 downto 0);
```

  - NEM tud műveletet végezni std_logic_vector típuson

```
c <= signed(a)*signed(b);
```

  - signed/unsigned típusokon nem lehet pl. logikai műveleteket végezni
- **std_logic_signed és std_logic_unsigned package**
  - std_logic package kiterjesztése
  - NEM használható egyszerre a kettő (egy file-ban)
- **Javasolt: std_logic_arith**
  - (de a „régebbi kódok” nagy része nem ezt használja)

IEEE signed package

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity mul is
port (
      op_a, op_b : in  std_logic_vector(17 downto 0);
      res    : out std_logic_vector(35 downto 0)
);
end mul;
architecture rtl of mul is
begin

res <= op_a * op_b;

end rtl;
```

Tömbök

- **Előre definiált méretű**

```
type array0 is array (15 downto 0) of
                        std_logic_vector(7 downto 0);

signal m : array0;
```

- **Deklarációkor állítható méret**

```
type array0 is array (natural range <>) of
                        std_logic_vector(7 downto 0);

signal m : array0(15 downto 0);
```

- **Index típus lehet bármi, de tipikusan csak az integer szintetizálható**

Record

- **~struktúra létrehozása**

```
type date_r is
record
   year  : std_logic_vector(15 downto 0);
   month : std_logic_vector( 3 downto 0);
   day   : std_logic_vector( 4 downto 0);
end record;

signal date0, date1, date2 : date_r;

date0 <= (x"0000", „0000", „00000");
date1.year <= x"0000";
date2 <= date1;
```

- **record eleme lehet record**

Egyedi típusok

- **Tipikusan állapotgépekhez**
  - jobban olvasható kód
  - limitált értékkészlet

```
type STATE_TYPE is (RED, GREEN, BLUE);
```

- **Értékadáskor, vizsgálatkor csak a definiált értékek használhatók**
- **Szintézerek tipikusan automatikus leképzést (definíció - bitminta) valósítanak meg**
  - De rögzíthető

Logikai operátorok

- **Prioritás: not; and, or, nand, xor, xnor**
- **Vektor változókon bitenként hajtódnak végre**
  - Minden operandusnak ugyanolyan típusúnak és méretűnek kell lennie

Relációs operátorok

- <; <=; =; /=; >=; >
- **Minden standard adattípusra definiált**
- **std_logic_vector**
  - std_logic_arith
  - std_logic_signed/std_logic_unsigned package
- **Előjeles változók!**

Aritmetikai operátorok

- **+; -; *; **; /; mod; abs; rem**
- **Előre definiált integer változókra**
- **std_logic_vector**
  - std_logic_arith
  - std_logic_signed/std_logic_unsigned package
- **NEM mindegyik szintetizálható**
  - tipikusan: +, -, *

Shift operátorok

- **sll, srr**
  - logikai shiftelés
- **sla, sra**
  - aritmetikai shiftelés
  - az utolsó bit shiftelődik be
  - általában megkötésekkel szintetizálható
- **rol, ror**
  - rotálás
  - a „kieső” bit shiftelődik be
  - általában megkötésekkel szintetizálható

Konkurrens utasítások

- **Architecture-ön belüli egyszerű értékadások**
- **Egymással és a process-ekkel párhuzamos kiértékelés**
- **Signal típusú változónak adhatnak értéket**
- **A kifejezés bal oldala azonnal kiértékelődik ha a jobb oldali változók megváltoznak**

When

- **Feltételes értékadás**
  - ÉRTÉK: konstans vagy signal
- **When – Else:** ~IF - ELSE szerkezet

```
CÉL <= ÉRTÉK_0 when FELTÉTEL_0 else
       ÉRTÉK_1 when FELTÉTEL_1 else
       .............
       ÉRTÉK_N;
```

- **With – When:** ~CASE szerkezet

```
with VÁLTOZÓ select
  CÉL <= ÉRTÉK_0 when KONSTANS_0,
         ÉRTÉK_1 when KONSTANS_1 | KONSTANS_2,
         .........
         ÉRTÉK_N when others;
```

When (példa)

```
res <= „001” when (din=1) else
       „010” when (din=2) else
       „011” when (din=4) else
       „100” when (din=8) else
       „000”;
```

```
with din select
   res <= „001” when 1,
   res <= „010” when 2,
   res <= „011” when 4,
   res <= „100” when 8,
   res <= „000” when others;
```

Process

- **Architecture-ön belül használható**
- **A process-ek egymással (és a process-en kívüli értékadásokkal) párhuzamosan hajtódnak végre**
- **A végrehajtást triggereli**
  - érzékenységi lista
  - wait szerkezet
- **Írható változók:**
  - signal
  - variable (lokális változó)
  - (shared variable)

Process

- **Érzékenységi lista:**
  - a process végrehajtása ennek teljesülésekor történik
  - szintérzékeny, élérzékeny

```
process(a,b)
begin
   c <= a or b;
end process;
```

```
process(clk)
begin
if (clk'event and clk='1') then
   c <= a or b;
end if;
end process;
```

```
process(clk)
begin
if rising_edge(clk) then
   c <= a or b;
end if;
end process;
```

Szekvenciális kifejezések

- **If szerkezet (csak process-ben)**

```
if FELTÉTEL then
.........
elsif FELTÉTEL then
.........
else
.........
end if;
```

- **Case szerkezet (csak process-ben)**

```
case KIFEJEZÉS is
when ÉRTÉK_0 => .........
when ÉRTÉK_1 => .........
.............
when others  => .........
end case;
```

For ciklus (1)

- **Ciklus változó implicit deklarálva**
- **Ciklusok száma fix**
- **Variable: „szekvenciális” végrehajtás, szintetizálható**
  - könnyű (túl) komplex logikát leírni !!
  - „Gondolkozz HW fejjel” !!
- **Pl.: 2:4 dekóder**

```
process(a)
begin
b <= „0000”;
for I in 0 to 3 loop
   if (a=I) then
      b(I) <= ‘1’;
   end if;
end loop;
end process;
```

For ciklus (2)

- **Hexa → decimális átalakítás**
- **Osztás, modulo képzés nem mindig szintetizálható**

```
dec_10 <= hex_num/10;
dec_1 <= hex_num mod 10;
```

- **Kis változószélességre: ROM, VHDL kóddal számolva**

```
process(hex_num)
variable dec_v : std_logic_vector(3 downto 0);
begin
dec_v := (others=>'0');
for I in 1 to 15 loop
   if (hex_num = I) then
      dec_v := CONV_STD_LOGIC_VECTOR((I/10), 4);
   end if;
   dec_1 <= dec_v;
end loop;
end process;
```

While ciklus

- **Feltételtől függő számú ciklus végrehajtás**
  - feltétel függhet változótól
- **Tipikusan NEM szintetizálható**
- **Szimulációban hasznos**

```
while FELTÉTEL loop
.........
.........
end loop;
```

Wait

- **Wait felfüggeszti a process végrehajtását a feltétel teljesüléséig**
- **NEM használható érzékenységi listával együtt**
- **Feltétel:**
  - adott ideig (szimuláció)
  - jel értékétől függően

```
process(clk)
begin
if (clk'event and clk='1') then
   Q <= D;
end if;
end prcess;
```

```
process
begin
   Q <= D;
wait until clk='1';
end process;
```

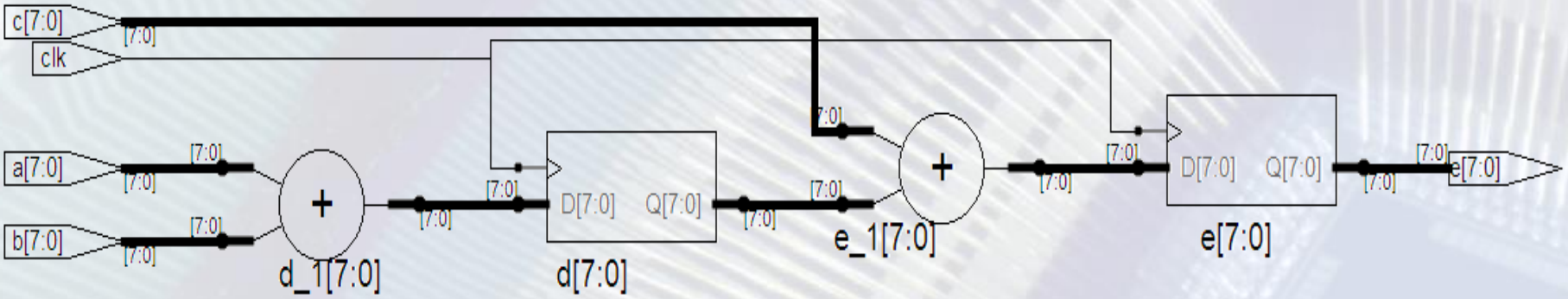
Variable

- **Process-en belül lokális (kivéve shared)**
- **Azonnali értékadás (signal: a process lefutása „után”)**
- **Értéke átadható signal-nak**
- **Értékét tartja a következő process „futásig”**
- **Pl.: túlcsordulás detekció:**

```
process(clk)
variable ovl : std_logic;
begin
   ovl := not accu(34);
   for IB in 35 to 46 loop
      ovl := ovl or not accu(IB);
   end loop;
   ovl := ovl and accu(47);
   accu_ovl_n <= ovl;
end process;
```

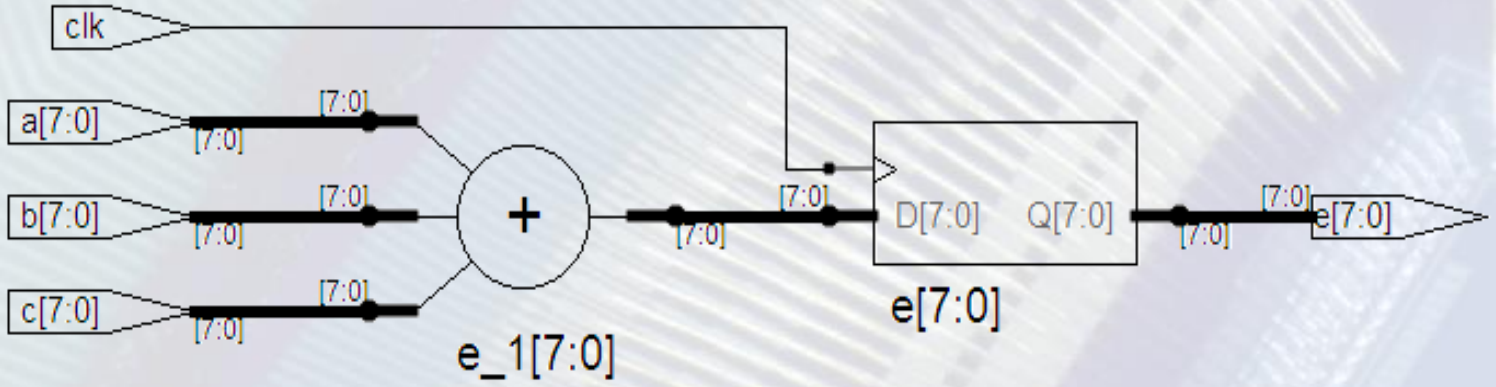
Variable – signal (1)

```
process(clk)
begin
if rising_edge(clk) then
   d <= a + b;
   e <= d + c;
end if;
end process;
```



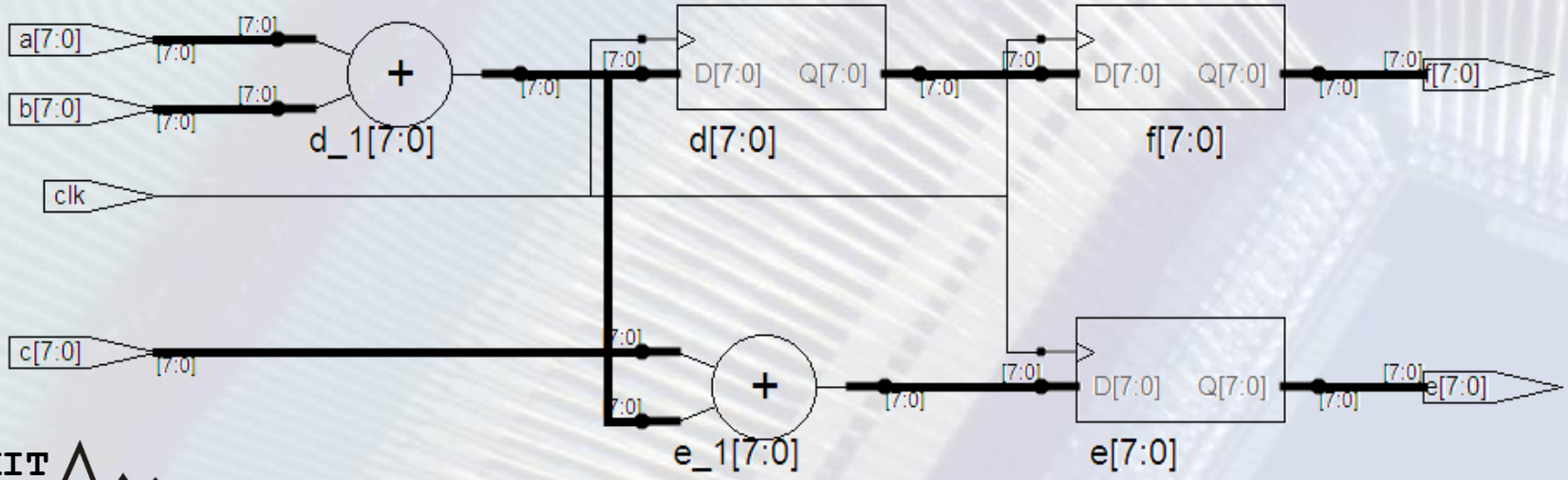
Variable – signal (2)

```
process(clk)
variable d : std_logic_vector(7 downto 0);
begin
if rising_edge(clk) then
begin
   d := a + b;
   e <= d + c;
end if;
end process;
```



Variable – signal (3)

```
process(clk)
variable d : std_logic_vector(7 downto 0);
begin
if rising_edge(clk) then
   f <= d;
   d := a + b; -- ÉRTKADÁS SORREND!!!
   e <= d + c;
end if;
end process;
```



Paraméterezhető modulok (1)

Entity deklaráció

```
entity adder
generic(
   DWIDTH : integer
);
port(
   a, b : in  std_logic_vector(DWIDTH-1 downto 0);
   c    : out std_logic_vector(DWIDTH downto 0)
);
end adder;
architecture rtl of adder is
begin

c <= ('0' & a) + ('0' & b);

end rtl;
```

Paraméterezhető modulok (2)

Példányosítás

```
.........
.........
adder_16: entity work.adder(rtl)
generic map(
   DWIDTH => 16
)
port map(
   a => in_data0,
   b => in_data1,
   c => out_data0
);
......
......
```

Generate

- **Architecture-en belül**
- **Struktúra ismétlése, tartalmazhat:**
  - Konkurrens értékadás
  - Process
  - Példányosítás

Generate példa

- **Pl.: cím dekóder**

```
signal cs_int : std_logic_vector(15 downto 0);

GEN_CS_INT:
for ICS in 0 to 15 generate
   cs_int(ICS) <= '1' when (cs='1' and addr=ICS) else '0';
end generate;
```

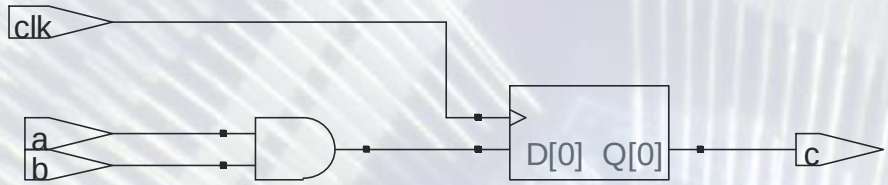
- **Eredmény:**
  - 16 db komparátor + 16 db ÉS kapu

Process – Flip Flop

- **Flip Flop: élérzékeny D tároló**

```
process (clk)
begin
if (clk'event and clk='1') then
   c <= a and b;
end if;
end process;
```

```
process (clk)
begin
if rising_edge(clk) then
   c <= a and b;
end if;
end process;
```



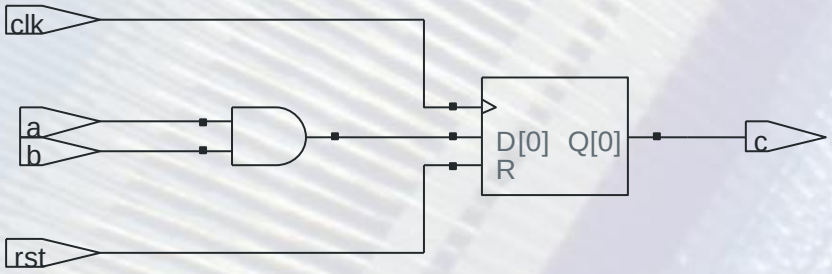
Process – Flip Flop

Szinkron reset

```
process(clk)
begin
if (clk'event and clk='1') then
   if (rst='1')
      c <= '0';
   else
      c <= a and b;
   end if;
end if;
end process;
```

Aszinkron reset

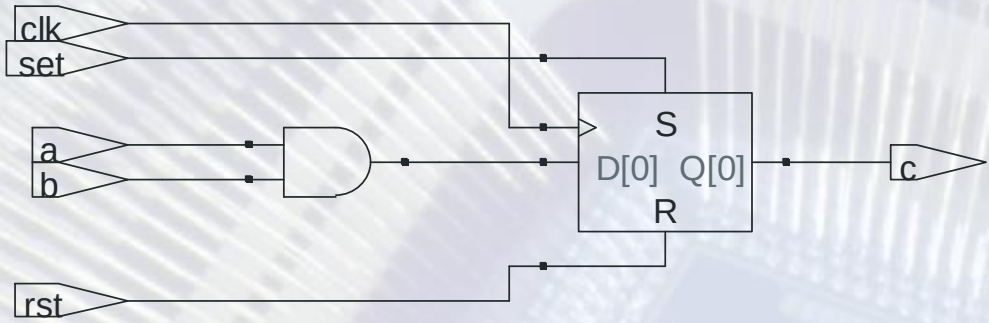
```
process(clk, rst)
begin
if (rst='1')
   c <= '0';
elsif (clk'event and clk='1') then
   c <= a and b;
end if;
end process;
```



Process – Flip Flop

- **Xilinx FPGA-kban a FF egy CLK bemenettel, két alaphelyzet beállító jellel és egy CE órajel engedélyező bemenettel rendelkezik.**
  - Szinkron vezérlés: Minden jel kiértékelése szinkron, ebben az esetben érvényesítés az órajel aktív élénél

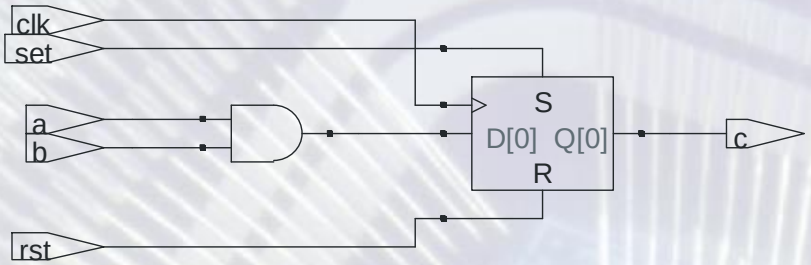
```
process(clk)
begin
if (clk'event and clk='1') then
   if (rst='1') then
      c <= '0';
   elsif (set='1') then
      c <= '1';
   else
      c <= a and b;
   end if;
end process;
```



Process – Flip Flop

- **Xilinx FPGA-kban a FF egy CLK bemenettel, két alaphelyzet beállító jellel és egy CE órajel engedélyező bemenettel rendelkezik.**
  - Aszinkron vezérlés: A vezérlőjelek változása azonnal érvényre jut, prioritás a felírás sorrendjében

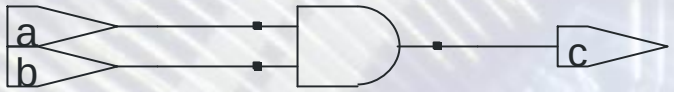
```
process(clk, rst, set)
begin
if (rst='1') then
   c <= '0';
elsif (set='1') then
   c <= '1';
elsif (clk'event and clk='1') then
   c <= a and b;
end if;
end process;
```



Process – kombinációs logikához

- **Szemléletesen:**
  - A process eseményvezérelt
  - A bemenőjelek bármely változása ilyen esemény
  - Ennek hatására az eljárás lefut, a kimenet kiértékelődik

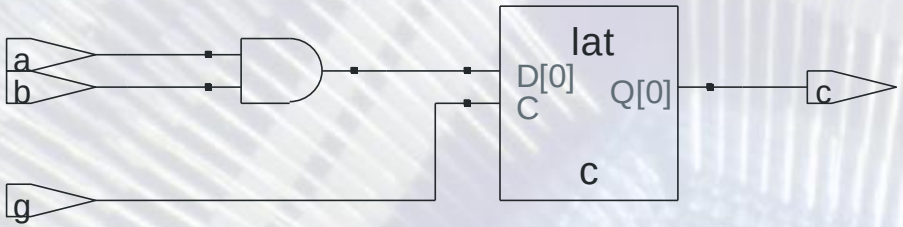
```
process (a, b)
begin
   c <= a and b;
end process;
```



Process – 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.

```
process (g, a, b)
begin
if (g='1') then
   c <= a and b;
end if;
end process;
```



Process – latch hiba

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

```
process(sel,in0, in1, in2)
begin
case sel is
   when „00”=> r <= in0;
   when „01”=> r <= in1;
   when „10”=> r <= in2;
end case;
end process;
```

```
process(sel,in0, in1, in2)
begin
if (sel=0) then
   r <= in0;
elsif (sel=1) then
   r <= in1;
elsif (sel=2) then
   r <= in2;
end if;
end process;
```

Egysz. értékadás – latch hiba

```
r <= in0 when (sel=0) else
     in1 when (sel=1) else
     in2 when (sel=2);
```

```
with sel select
   r <= in0 when 0,
   r <= in1 when 1,
   r <= in2 when 2;
```

Process – helyes

- **Helyes kód**

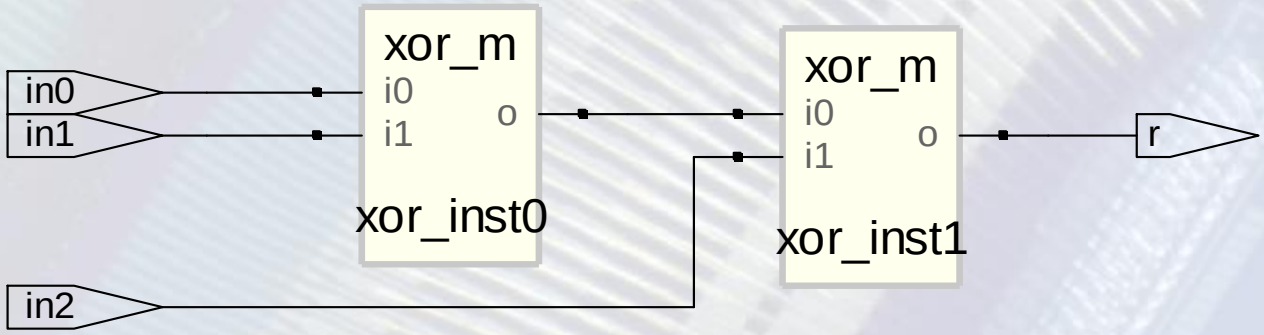
```
process(sel,in0, in1, in2)
begin
case sel is
   when „00”    => r <= in0;
   when „01”    => r <= in1;
   when „10”    => r <= in2;
   when others => r <= „X”;
end case;
end process;
```

```
process(sel,in0, in1, in2)
begin
if (sel=0) then
   r <= in0;
elsif (sel=1) then
   r <= in1;
else
   r <= in2;
end if;
end process;
```

Strukturális leírás

- **Hierarchia felépítése: modulok összekapcsolása**

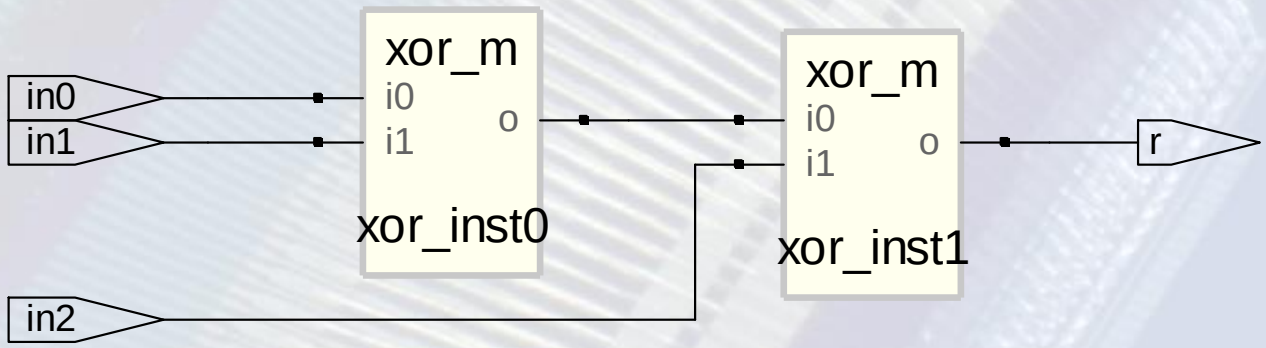
```
signal xor0 : std_logic;
.............
begin
.............
xor_inst0 entity work.xor_m(rtl)
   port map(i0=>in0, i1=>in1, o=>xor0);
xor_inst1 entity work.xor_m(rtl)
   port map(i0=>xor0, i1=>in2, o=>r);
.............
```



Strukturális leírás - generate

- **Hierarchia felépítése: modulok összekapcsolása**

```
in_bus0(0) <= in0;
in_bus1    <= in2 & in1;

GEN_INST:
for I in 0 to 1 generate
   xor_inst entity work.xor_m(rtl)
      port map(i0=>in_bus0(I), i1=>in_bus1(I), o=>in_bus0(I+1));
end generate;

r <= in_bus0(2);
```



Generate példa

```
GEN_PPC_TO_PHY:
for IB in 0 to 4 generate
  ddro: entity work.ddr_ff_o(SYN)
  port map
  (
    aclr          => '0',
    datain_h      => tx_data_phy(IB),
    datain_l      => tx_data_phy(5+IB),
    outclock      => tx_clk_phy,
    dataout       => GETXD_PIN(IB)
  );
end generate;
```

Példa – MUX (1.)

- **Különböző leírási stílusok a 2:1 multiplexerre**

```
process(sel,in0, in1)
begin
case sel is
   when '0'   => r <= in0;
   when '1'   => r <= in1;
end case;
end process;
```

```
process(sel,in0, in1)
begin
if (sel=1) then
   r <= in1;
else
   r <= in0;
end if;
end process;
```

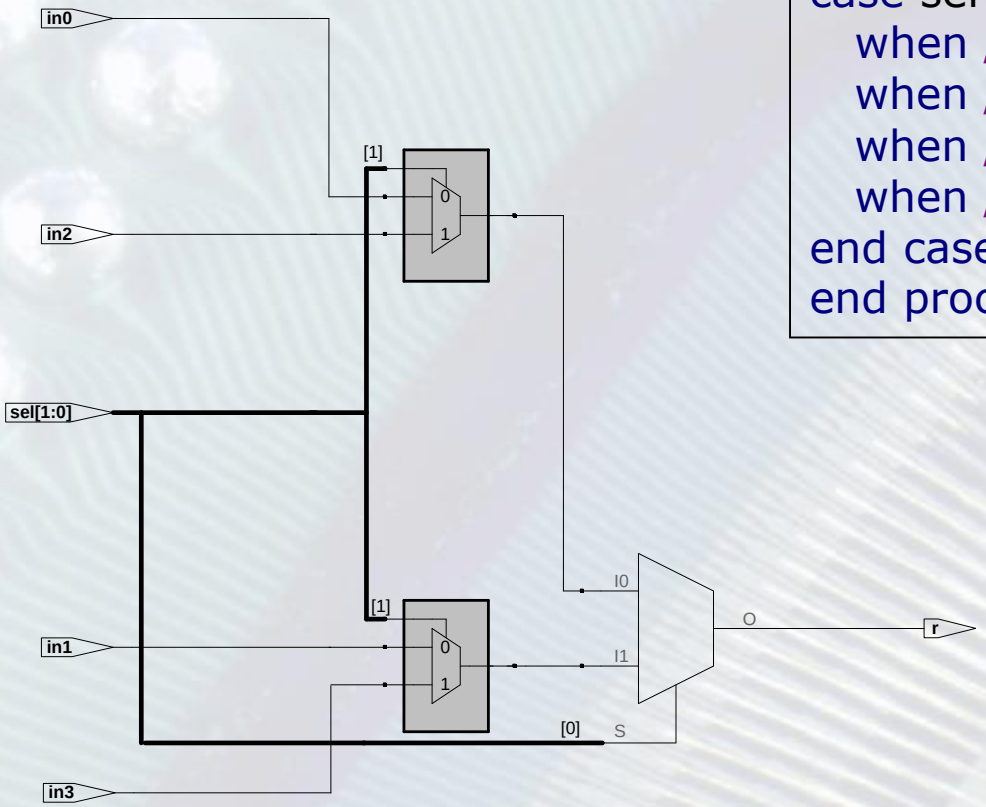
```
in_b <= in1 & in0;
r <= in_b(CONV_INTEGER(sel));
```

```
r <= in1 when (sel='1') else
      in0;
```

Példa – MUX (2.)

- **4:1 multiplexer**

```
process(sel, in0, in1, in2, in3)
begin
case sel is
   when „00” => r <= in0;
   when „01” => r <= in1;
   when „10” => r <= in2;
   when „11” => r <= in3;
end case;
end process;
```



Ciklus példa

```
process(lad, ip2bus_data_ar0)
variable ip2bus_data_v : std_logic_vector(31 downto 0);
begin
ip2bus_data_v := ip2bus_data_ar0(31 downto 0);
for IW in 1 to AR0_REGS-1 loop
   if (lad(LOG2_CEIL(AR0_REGS)+1 downto 2)=IW) then
      ip2bus_data_v := ip2bus_data_ar0(IW*32+31 downto IW*32);
   end if;
end loop;
din <= ip2bus_data_v;
end process;
```

Példa – 1 bites összeadó

```
xor_0: entity work.xor3_m(rtl)
   port map (i0=>a, i1=>b, i2=>cin, o=>s);
and_0: entity work.and2_m(rtl)
   port map (i0=>a, i1=>b, o=>a0);
and_1: entity work.and2_m(rtl)
   port map (i0=>a, i1=>cin, o=>a1);
and_2: entity work.and2_m(rtl)
   port map (i0=>b, i1=>cin, o=>a2);
or_0: entity work.or3_m(rtl)
   port map (i0=>a0, i1=>a1, i2=>a2, o=>cout);
```

```
s    <= a xor b xor cin;
cout <= (a and b) or (a and cin) or (b and cin);
```

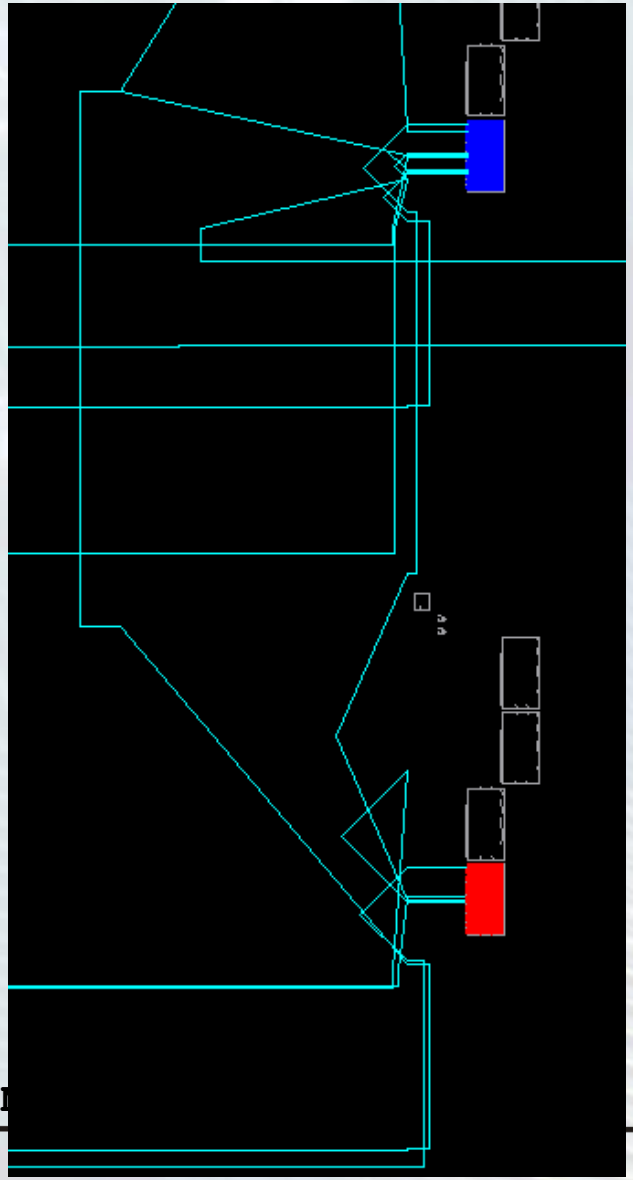
```
signal dbus : std_logic_vector(1 downto 0);
.................
dbus <= a + b + cin;
s    <= dbus(0);
cout <= dbus(1);
```

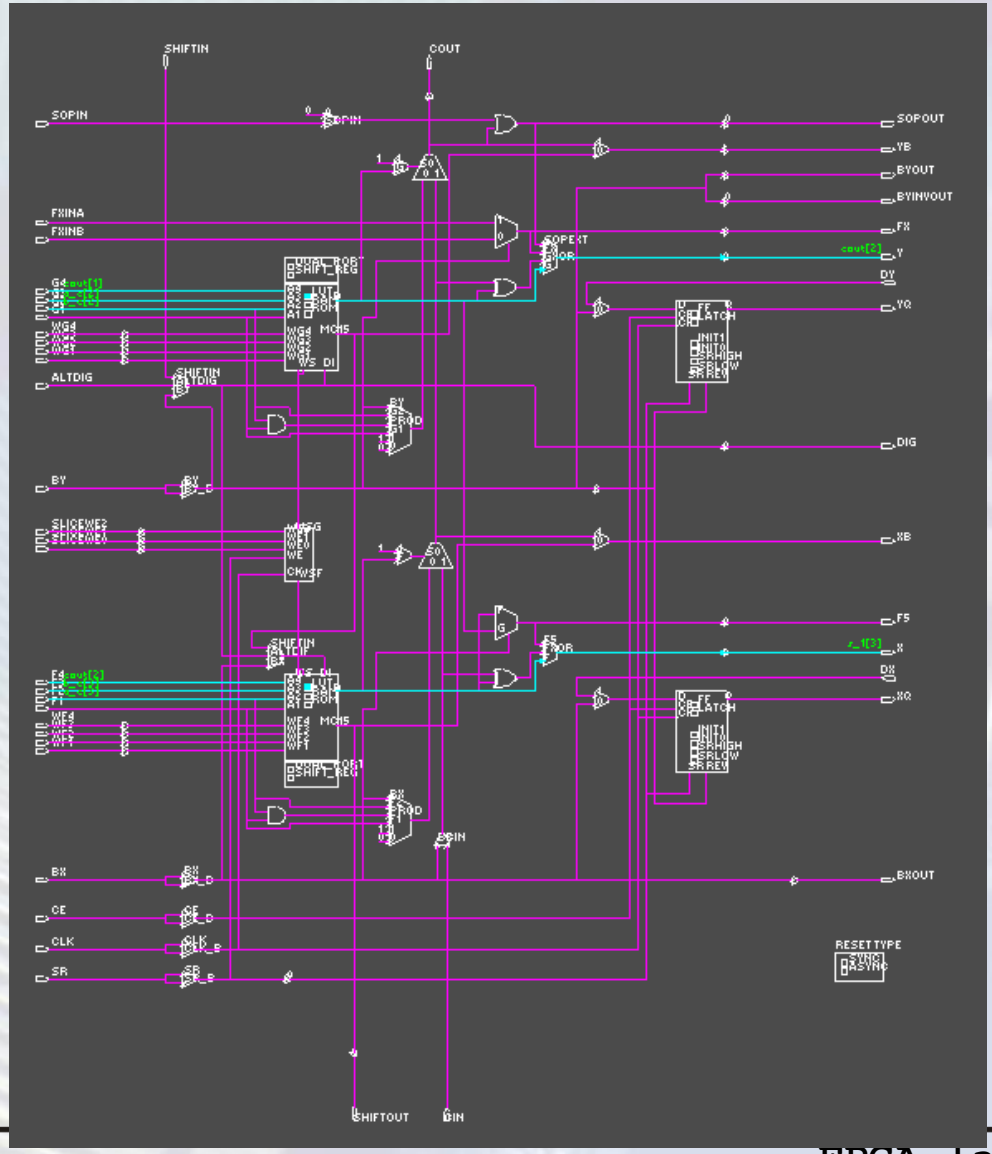
Példa – 4 bites összeadó

```
.........
signal cout : std_logic_vector(3 downto 0);
.........
add0: entity work.add1_full(rtl)
   port map (a=>a(0), b=>b(0), cin=>'0', cout=>cout(0), s=>s(0));
add1: entity work.add1_full(rtl)
   port map (a=>a(1), b=>b(1), cin=>cout(0), cout=>cout(1), s=>s(1));
add2: entity work.add1_full(rtl)
   port map (a=>a(2), b=>b(2), cin=>cout(1), cout=>cout(2), s=>s(2));
add3: entity work.add1_full(rtl)
   port map (a=>a(3), b=>b(3), cin=>cout(2), cout=>s(4), s=>s(3));
.........
```

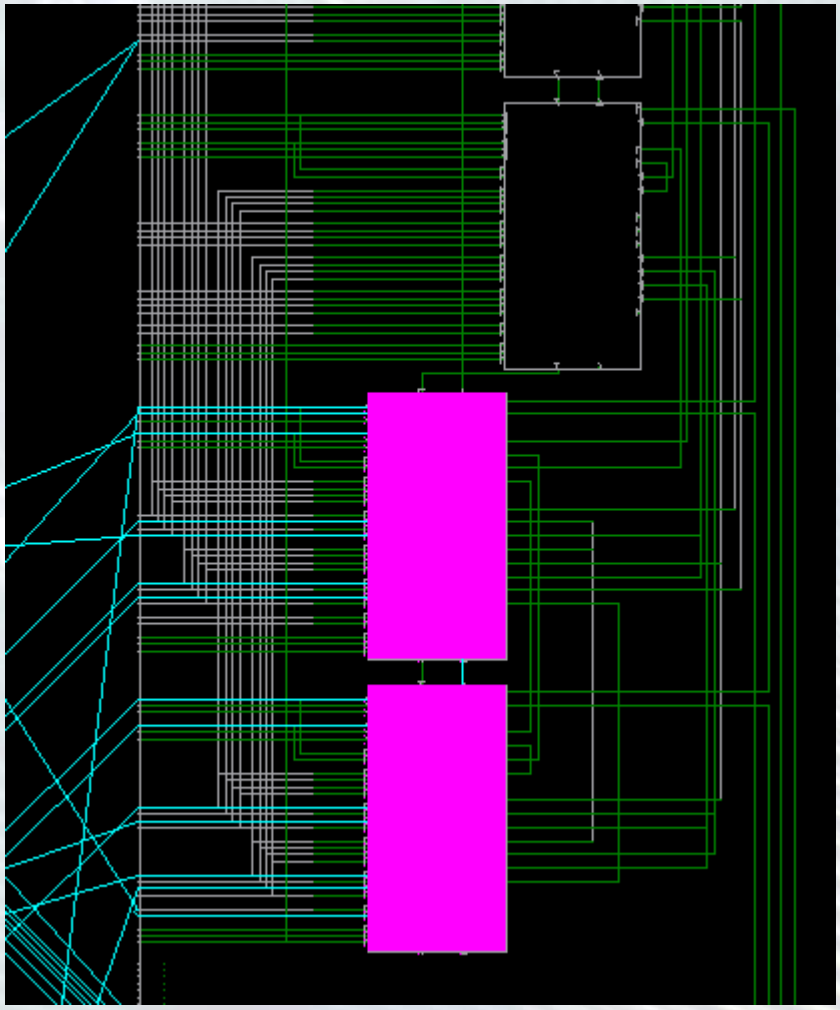
```
...
a,b : in  std_logic_vector(3 downto 0);
s   : out std_logic_vector(4 downto 0);
......

s <= ('0' & a) + ('0' & b);
......
```

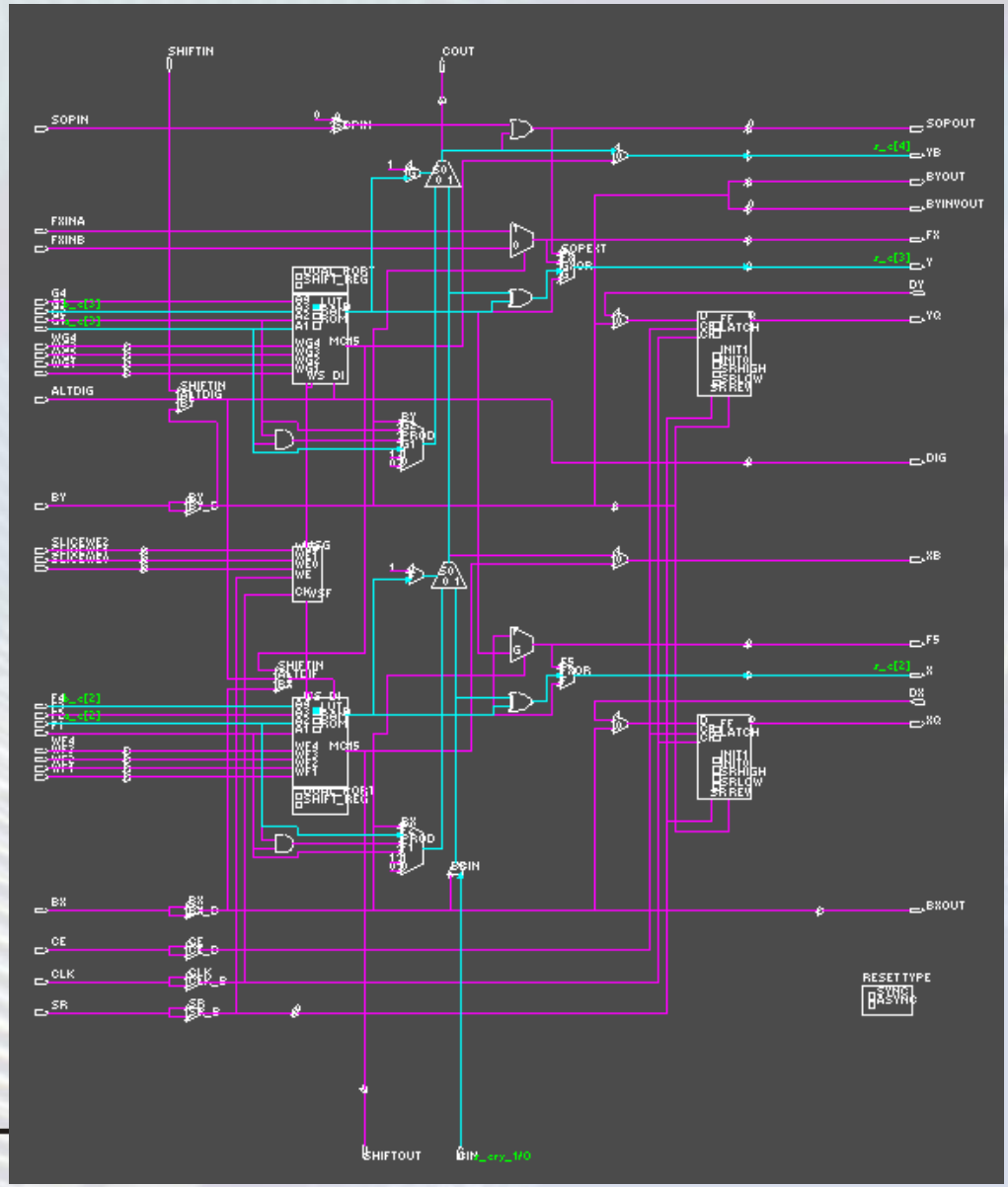
Példa – 4 bites összeadó, logikai op.



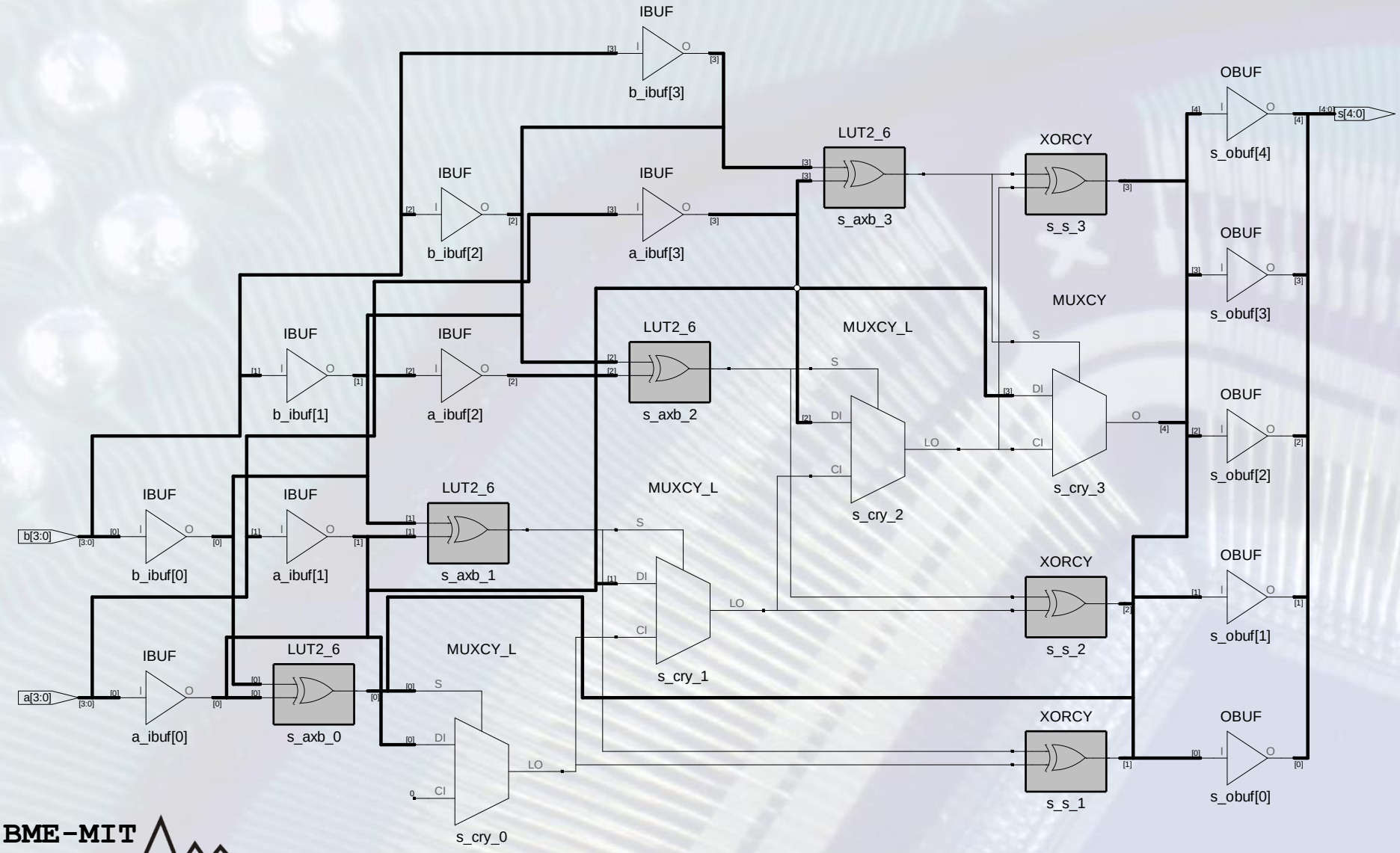


Példa – 4 bites összeadó, + operátor





Példa – 4 bites összeadó, +



Példa: Shift regiszter

- **16 bites shift regiszter,**
  - A LUT4 SRL16 soros shiftregiszter kihasználására

```
.....
clk, sh, din: in  std_logic;
dout:         out std_logic;
.....
signal shr: std_logic_vector(15 downto 0);
.....
process (clk)
begin
if (clk'event and clk='1') then
   if (sh='1') then
         shr <= shr(14 downto 0) & din;
   end if;
end if;
dout <= shr(15);
```

Példa: Számláló

- **Számláló minta leírás**
  - Szinkron, 8 bites
  - Szinkron RESET
  - Tölthető
  - Engedélyezhető
  - fel/le számláló
- **Megj:**
  - A CE nagyobb prioritású, mint a töltés, ez nem tipikus

```
.....
clk, rst, ce, load, dir : in std_logic;
din: in std_logic_vector(7 downto 0),
dout: out std_logic_vector(7 downto 0);
.....
signal cntr_reg : std_logic_vector(7 downto 0);
.....
process (clk)
begin
if (clk'event and clk='1') then
   if (rst='1') then
      cntr_reg <= (others=>'0');
   elsif (ce='1')
      if (load='1') then
         cntr_reg <= din;
      elsif (dir='1') then
         cntr_reg <= cntr_reg - 1;
      else
         cntr_reg <= cntr_reg + 1;
      end if;
    end if;
end if;
end process;

dout <= cntr_reg;
```

Redukciós operátor helyett…

```
GEN_OVL_N:
process(accu)
variable ovl     : std_logic;
begin
ovl := not accu (65);
for IB in 66 to 77 loop
       ovl := ovl or not accu(IB);
end loop;
ovl := ovl and accu (78);
accu_ovl_n <= ovl;
end process;
```

Példa

```
constant CONST_0 : std_logic_vector(127 downto 0) := x"00000000000000000000000000000000";
constant cnt1_1stage : integer := MAX(2, DIV_CEIL_2PWR(TAP, ADDER_INS));
constant cnt1_stages : integer := GET_1CNT_STAGES(TAP, ADDER_INS);
type array_TAPdIN_by_TAPb is array (cnt1_1stage-1 downto 0) of std_logic_vector(TAPb-1 downto 0);
type array_1cnt_stages   is array (cnt1_stages-1 downto 0) of array_TAPdIN_by_TAPb;
signal one_cntr : array_1cnt_stages;
GEN_1CNT_L0:
for IST in 0 to cnt1_stages-1 generate
  GEN_1cnt_0:
  if (IST = 0) generate
    process(clk)
    variable cntr_regs : array_TAPdIN_by_TAPb;
    begin
      if (clk'event and clk='1') then
        if (en='1') then
        for IR in 0 to cnt1_1stage-1 loop
          cntr_regs(IR) := CONST_0(TAPb-1 downto 0);
        end loop;
        for IB in 0 to TAP-1 loop
          cntr_regs(IB/ADDER_INS) := cntr_regs(IB/ADDER_INS) + data(IB);
        end loop;
        for IR in 0 to cnt1_1stage-1 loop
          one_cntr(0)(IR) <= cntr_regs(IR);
        end loop;
        end if;
      end if;
    end process;
  end generate;
```

Példa

```
GEN_1cnt_last:
  if ((IST = (cnt1_stages-1)) AND (IST /= 0)) generate
    process(clk)
    begin
    if (clk'event and clk='1') then
      if (en='1') then
        one_cntr(IST)(0) <= one_cntr(IST-1)(0) + one_cntr(IST-1)(1);
      end if;
    end if;
    end process;
  end generate;
  GEN_1cnt_others:
  if ((IST /= 0) AND (IST /= (cnt1_stages-1))) generate
    process(clk)
    begin
      if (clk'event and clk='1') then
        if (en='1') then
        for IR in 0 to DIV_CEIL(cnt1_1stage, (2**IST))-1 loop
          one_cntr(IST)(IR) <= one_cntr(IST-1)(2*IR) + one_cntr(IST-1)(2*IR+1);
        end loop;
        end if;
      end if;
    end process;
  end generate;
end generate;

res <= one_cntr(cnt1_stages-1)(0);
```