TSEA83 : Datorkonstruktion
Fö10

VHDL 3/3

LINKÖPINGS UNIVERSITET

Fö10 : Agenda

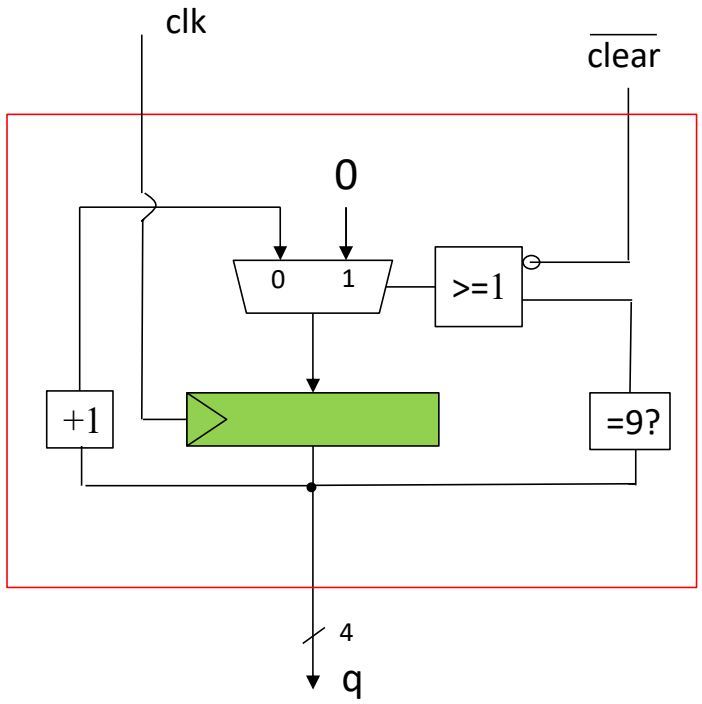
- Repetition
  buffer, record, loop
  kombinatoriska processer
- Varning
  latchar, hasard
- uprogCPU
  VHDL-kod för mikromaskin med hämtfas
- Minnen i FGPA
  Distributed RAM (LUT)
  Block-RAM
- 3-portars registerfil
- pipeCPU
  VHDL-kod för pipeline-CPU med instruktionshämtning
- VGA-labben
- Kravspec + Designspec

Repetition, sekvensnät

Buffer, record, loop, kombinatoriska processer

Angående buffer

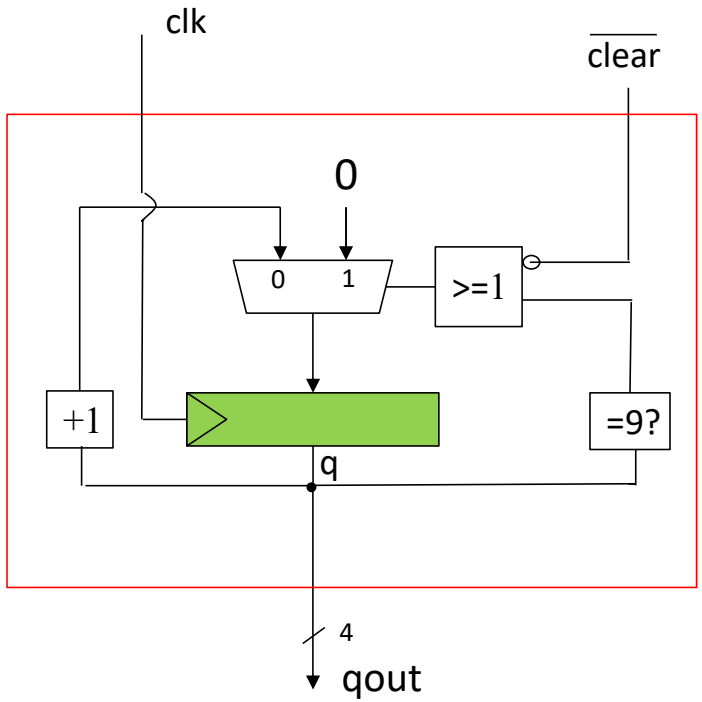
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter is
port(clk, clear: in std_logic;
     q: buffer unsigned(3 downto 0));
end counter;

architecture simple of counter is
begin
  process(clk)
  begin
    if rising_edge(clk) then
    if clear='0' then
         q <= "0000";
    elsif q=9 then
          q <= "0000";
       else
          q <= q + 1;
       end if;
    end if;
  end process;
end simple;
```



Istället för buffer

skapa en intern signal och använd out som vanligt

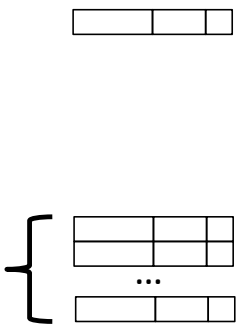
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter is
port(clk, clear: in std_logic;
     qout: out unsigned(3 downto 0));
end counter;

architecture simple of counter is
     signal q: unsigned(3 downto 0);
begin
  process(clk)
  begin
    if rising_edge(clk) then
    if clear='0' then
         q <= "0000";
    elsif q=9 then
           q <= "0000";
        else
           q <= q + 1;
       end if;
    end if;
  end process;

  qout <= q;
end simple;
```



Varför?
Modelsim ”gillar inte” buffer!

record

```
type controlword is record
     alu: unsigned(3 downto 0);
     tobus: unsigned(2 downto 0);
     halt: std_logic;
end record;
type styrminne is array(0 to 31) of controlword;
```

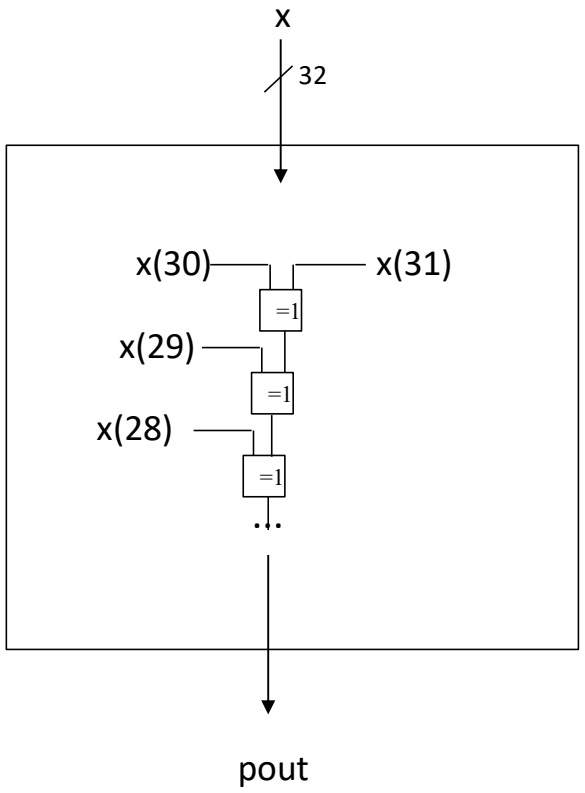



```
signal styr1, styr2: controlword;
signal mm: styrminne;
--
styr1.halt <= '0';
styr1.alu <= "1011";
styr1.tobus <= styr2.tobus;
--
mm(3) <= ("1011","111",'0');
```

Lite överkurs - loop

Vi har en buss x, med 32 ledningar. Vi vill bilda paritet mellan alla ledningarna. Loopen beskriver på ett kompakt sätt det kombinatoriska nätet!

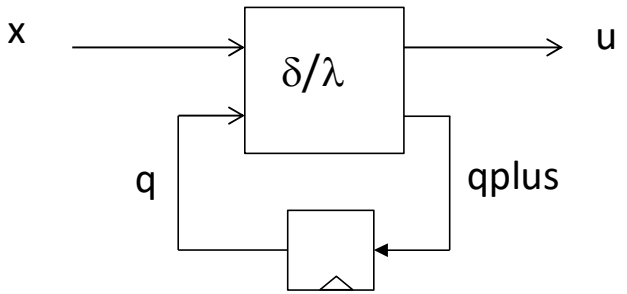
```
entity parity is
    port ( x : in  UNSIGNED (31 downto 0);
           pout : out  STD_LOGIC);
end entity;

architecture func of parity is
begin
  -- kombinatoriskt nät
  process(x)
    variable p: std_logic := '0';
  begin
    for i in 31 downto 0 loop
        p := p xor x(i);
    end loop;
    if p='1' then
      pout <= '1';
    else
      pout <= '0';
    end if;
  end process;

end architecture;
```



Kombinatoriska processer

Vi kan använda `process` för att göra kombinatorik
+ `if-` och case-saterna blir tillgängliga
- varning för latchar!

Exempel: δ/λ-nätet i ett sekvensnät




```
process(clk)
begin
 if rising_edge(clk)
   q <= qplus;
 end if;
end process;
```

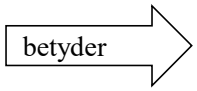
alla insignaler, ej clk

Default-värden
för alla utsignaler

```
process(x,q)
begin
 u <= "00";
 qplus <= idle;

 -- sats för delta/lambda
 -- nu räcker det att beskriva
 -- när det inte ska va default
 …
end;
```

Kombinatoriska processer lämpar sig inte alltid

```
process(b)
begin
 y <= '0';
 if b='1' then
   y <= '1';
 end if;
end process;
```




```
process(b)
begin
 if b='1' then
   y <= '1';
 else
   y <= '0';
 end if;
end process;
```


```
y <= b;
```

Det lämpar sig inte alltid med en kombinatorisk process.

Ett varningsord : Oönskade latchar

Vid **`select`**-sats och **`case`**-sats kräver VHDL att alla fall täcks!

Det är inte nödvändigt vid **`if`**-sats och **`when`**-sats!
Ibland är detta bra och ibland är det förskräckligt dåligt.
För de fall som inte täcks bibehålls föregående utsignal.

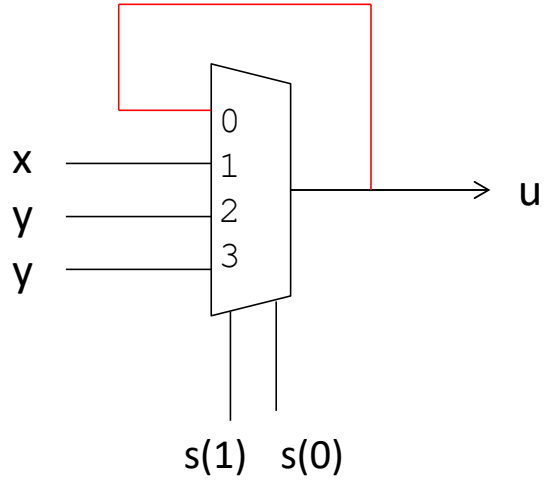
	Sekvensnät (inuti klockad process)	Kombinatorik?

Ofullst.	`if count='1' then`
	`  q <= q+1;`
	`end if;`
		`u <= y when s(1) = '1' else`
		`  x when s(0) = '1';`

Fullst.	`if count='1' then`
	`  q <= q+1;`
	`else`
	`  q <= q;`
	`end if;`
		`u <= y when s(1) = '1' else`
		`  x when s(0) = '1' else`
		`  '0' when others;`

Ett varningsord : Oönskade latchar

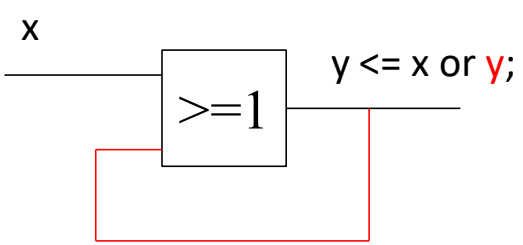
Latch = asynkront minneselement



s	u
1	x
0	x
2	y
0	y

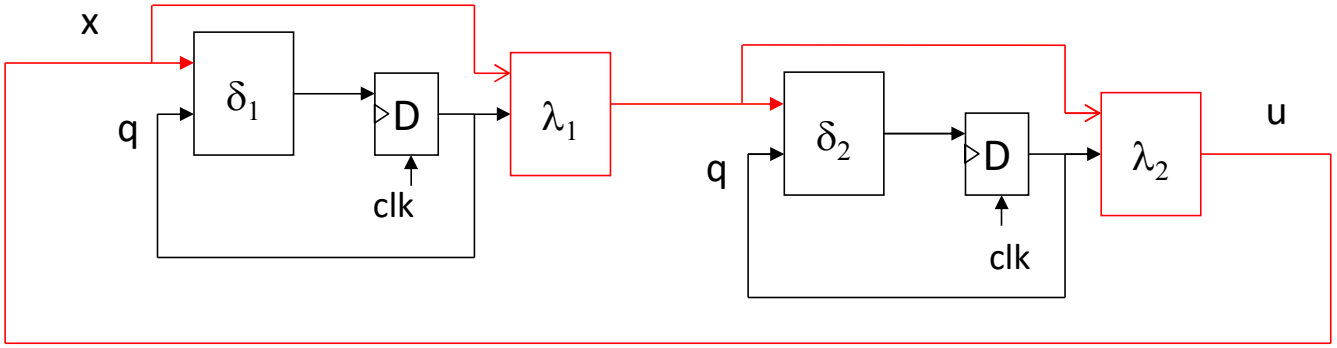
Ett varningsord : Oönskade latchar

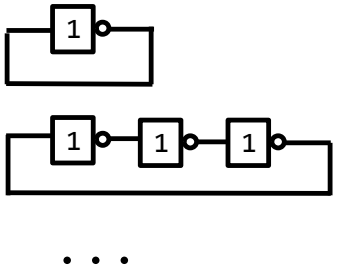
- Oönskat (oklockat) minneselement pga kombinatorisk loop
- Ihopkoppling av Mealynät kan ge kombinatorisk loop!
  Använd hellre Moore-nät!



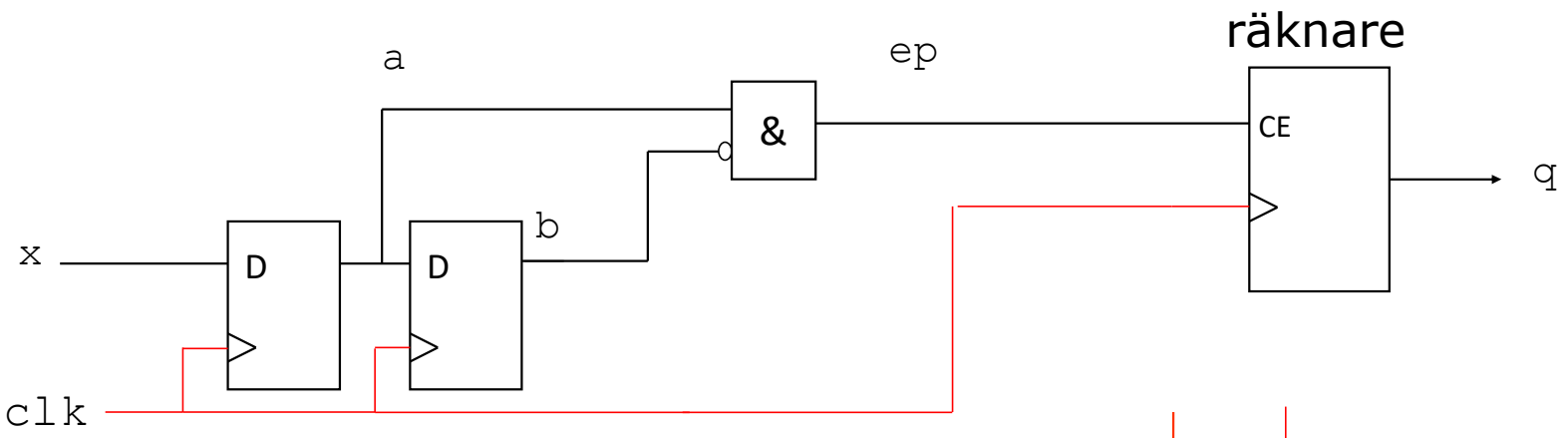
x	y
0	0
1	1
-	1

Kombinatorisk återkoppling kan även orsaka självsvängning, t ex om signalen inverteras ett ojämnt antal gånger.

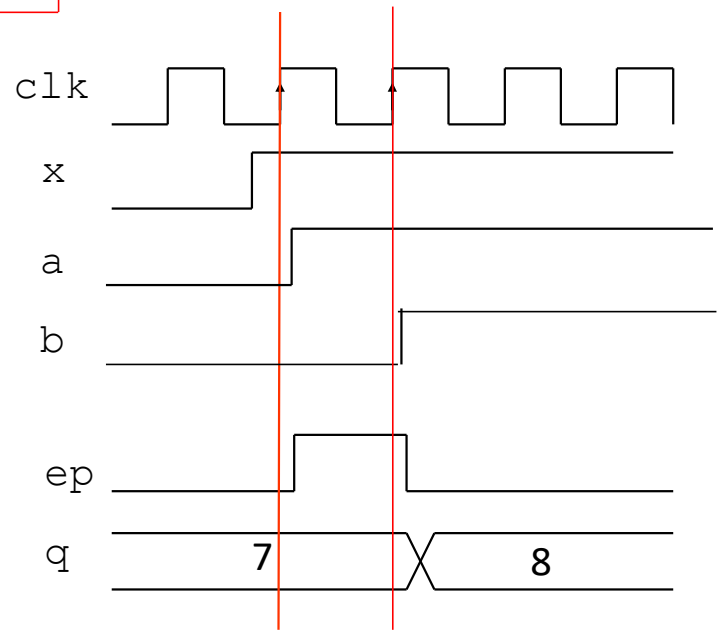




Synkronisering + enpulsning. Bra!



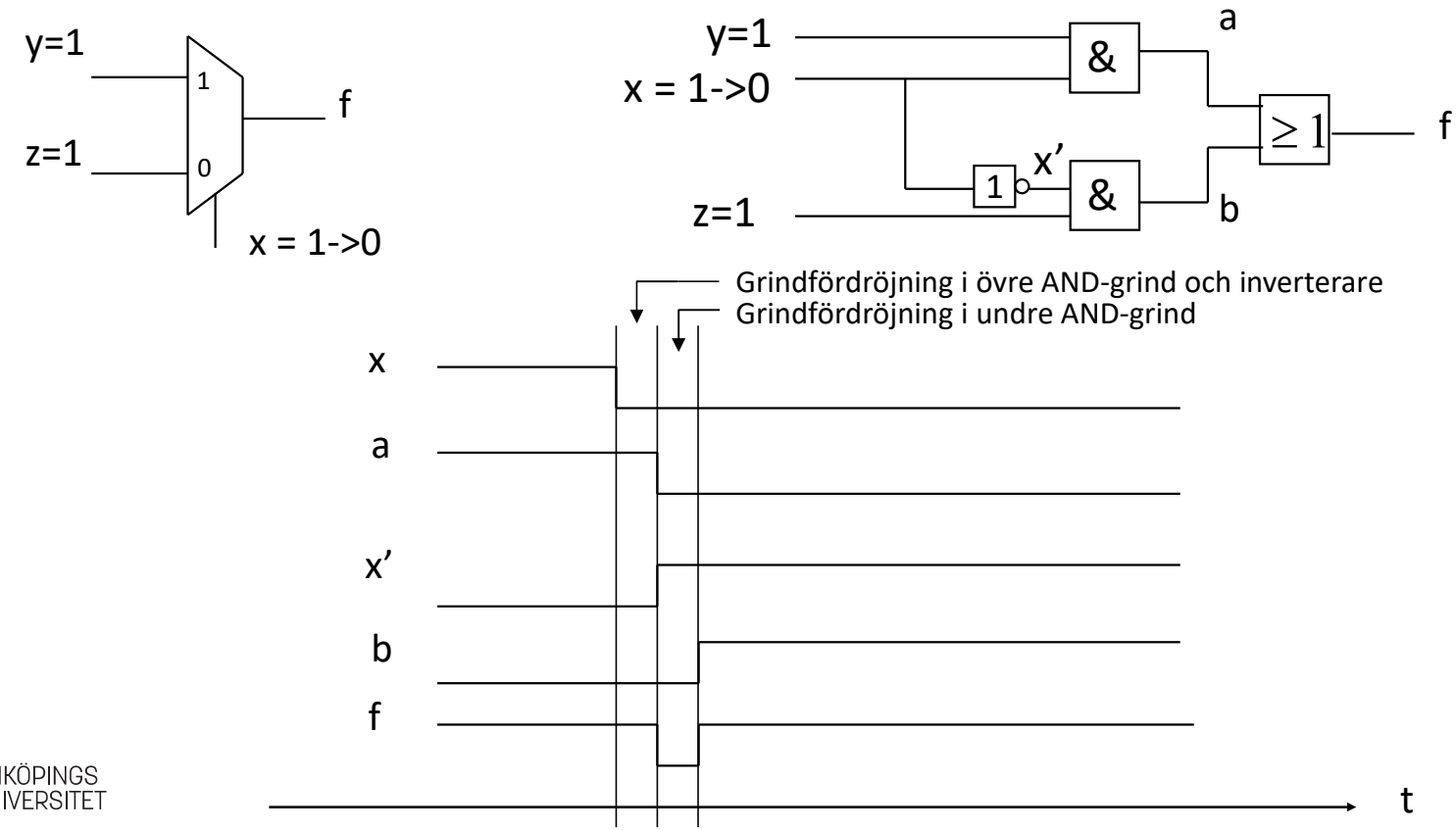

```
sep: process(clk)
begin
   if rising_edge(clk)   then
      a<=x;
      b<=a;
   end if;
end process sep;
-- till räknaren
ep <= a and not b;
```



Hasard

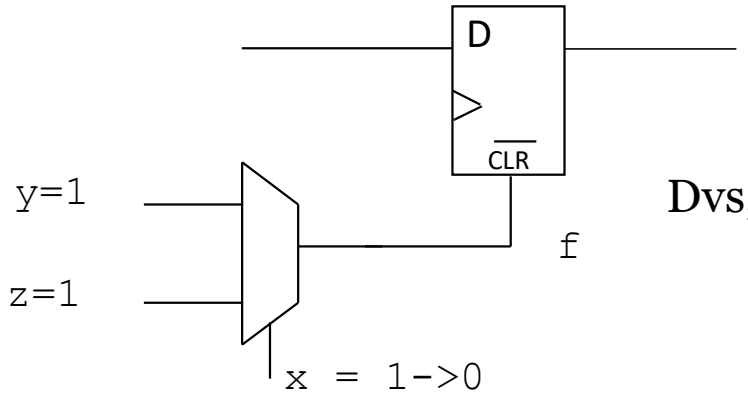
Def: Kortvariga värden på utgångarna från ett K-nät när någon insignal byter värde.

Exempel:



Hasard

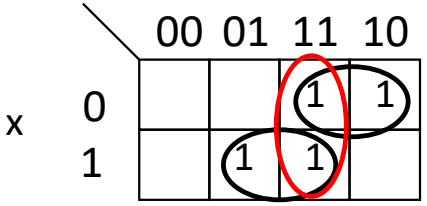
Om f kopplas till en asynkron ingång, så fungerar inte nätet (som det var tänkt) !



Dvs, varning för asynkron reset!

Studera övergången i ett KD,

`f = xz + x'y`



Hasarden kan elimineras

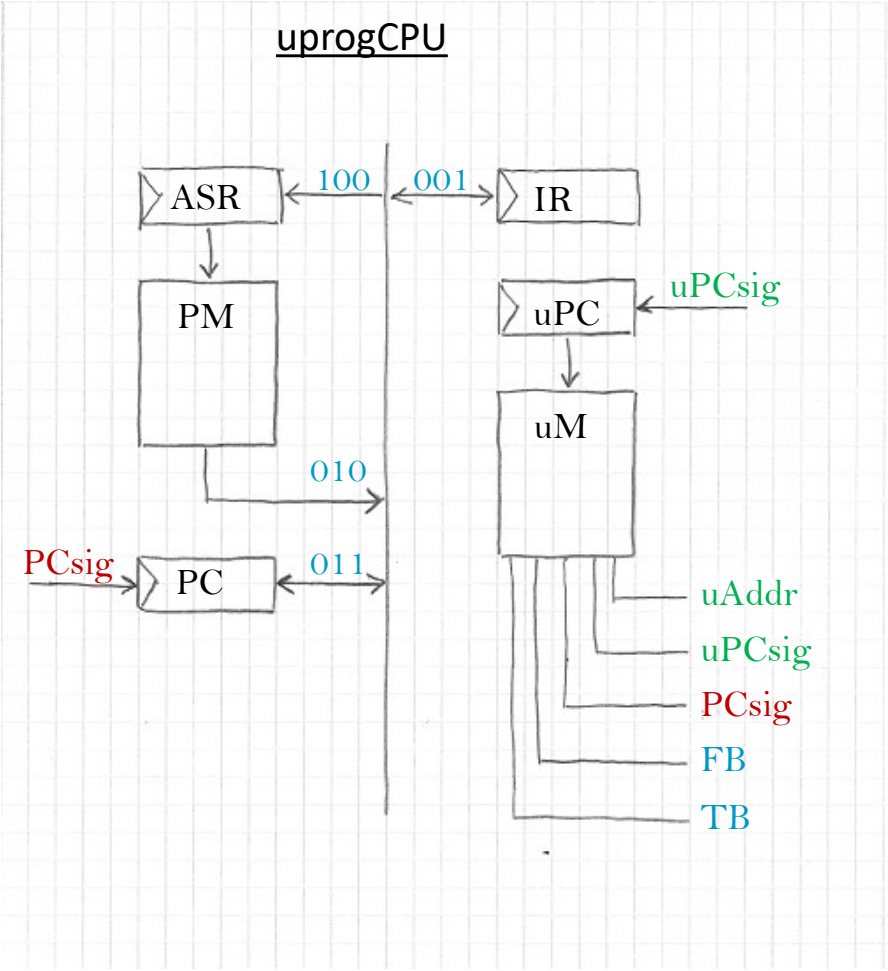
1) genom att lägga till termen yz,
   f = xz+x'y+yz
2) Synkronisera f
3) Undvika asynkrona ingångar

uprog CPU

VHDL-kod för mikromaskin med hämtfas

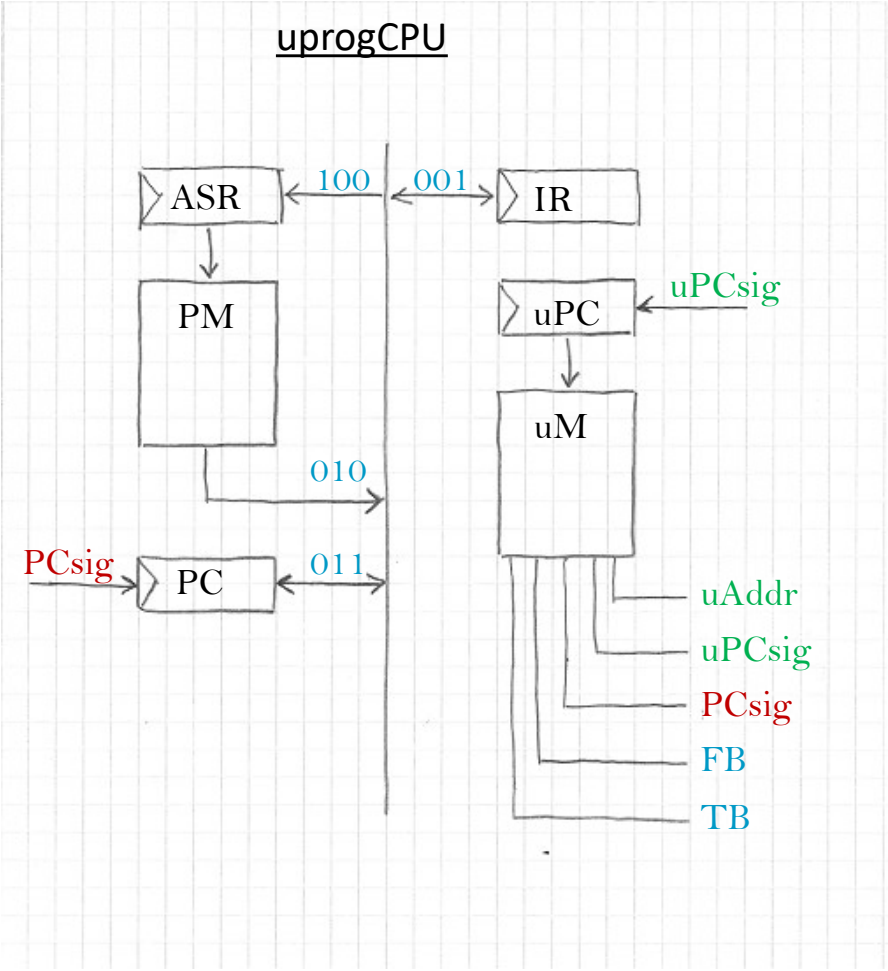
uprog CPU

Vi börja med en enkel skiss ...



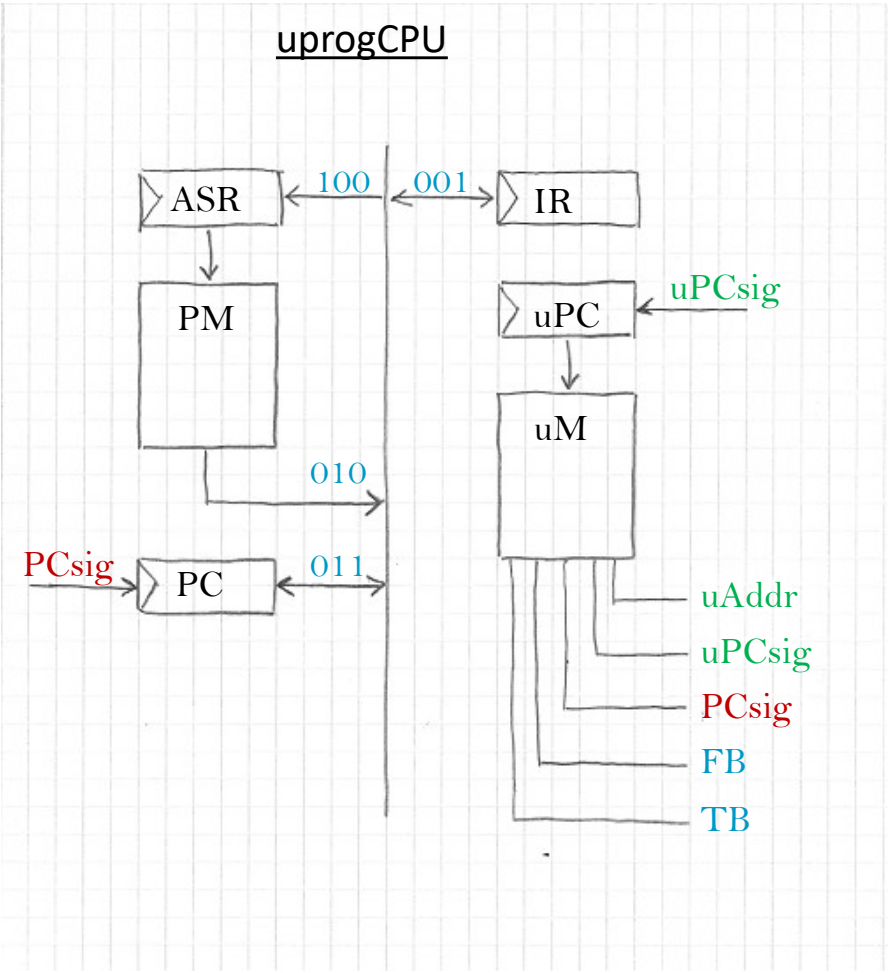
uprog CPU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- CPU interface
entity cpu is
  port(clk    : in std_logic;
       rst    : in std_logic);
end entity;

architecture func of cpu is

  -- micro Memory component
  component uMem
    port(uAddr     : in unsigned(5 downto 0);
         uData     : out unsigned(15 downto 0));
  end component;

  -- program Memory component
  component pMem
    port(pAddr     : in unsigned(15 downto 0);
         pData     : out unsigned(15 downto 0));
  end component;
```



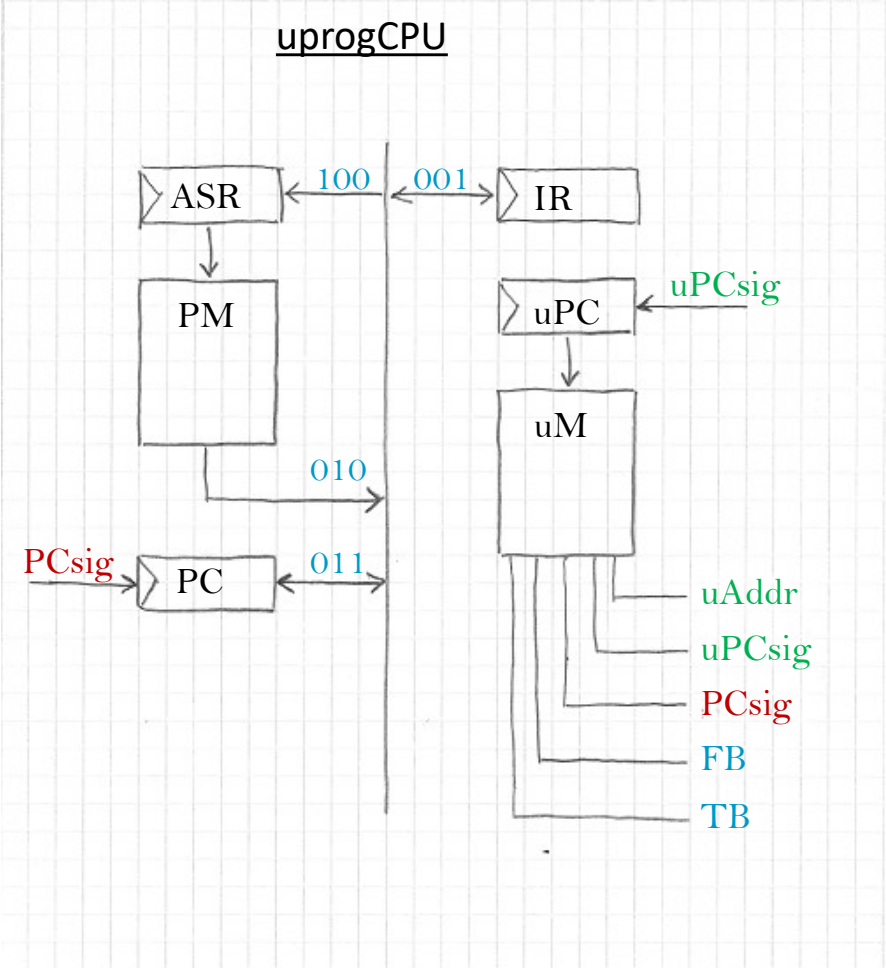
uprog CPU

```
-- micro Memory signals
signal uM     : unsigned(15 downto 0); -- micro Memory output
alias TB      : unsigned(2 downto 0) is uM(13 downto 11);
alias FB      : unsigned(2 downto 0) is uM(10 downto 8);
alias PCsig   : std_logic is uM(7);     -- (0:PC=PC, 1:PC++)
alias uPCsig  : std_logic is uM(6);     -- (0:uPC++, 1:uPC=uAddr)
alias uAddr   : unsigned(5 downto 0) is uM(5 downto 0);

-- program memory signals
signal PM     : unsigned(15 downto 0); -- Program Memory output

-- local registers
signal uPC    : unsigned(5 downto 0);  -- micro Program Counter
signal PC     : unsigned(15 downto 0); -- Program Counter
signal IR     : unsigned(15 downto 0); -- Instruction Register
signal ASR    : unsigned(15 downto 0); -- Address Register

-- local combinatorials
signal DATA_BUS : unsigned(15 downto 0); -- Data Bus
```



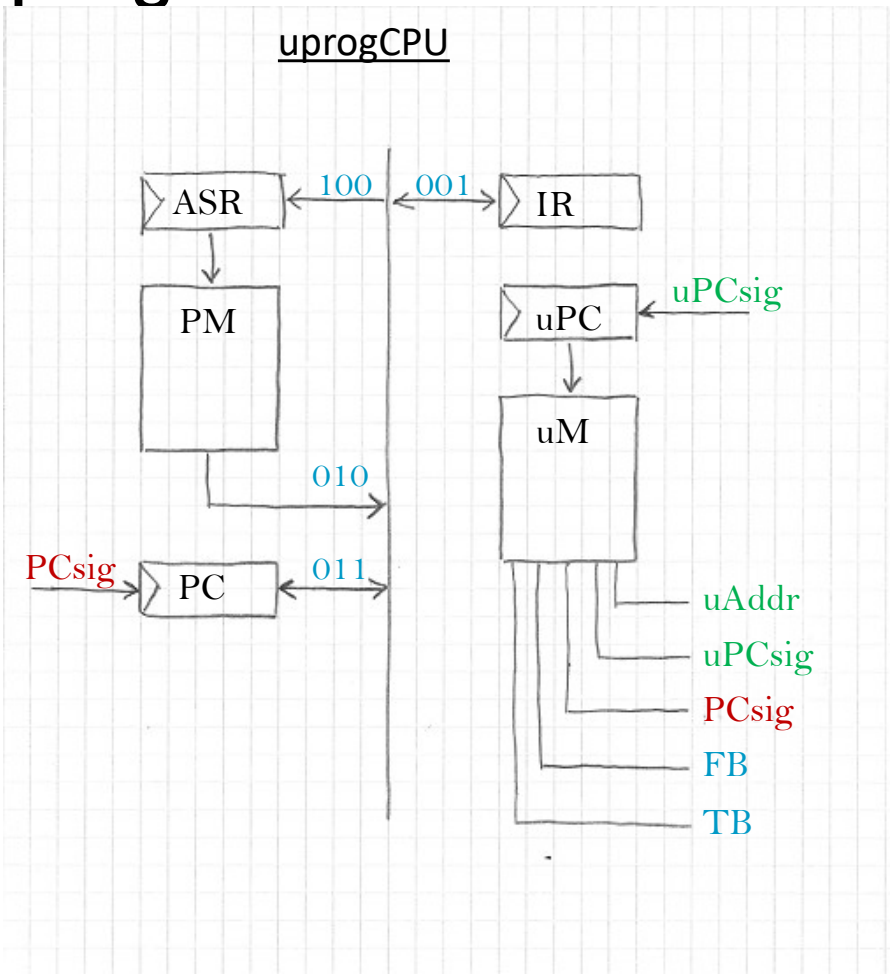
```
begin
-- mPC : micro Program Counter
process(clk)
begin
  if rising_edge(clk) then
    if (rst = '1') then
      uPC <= (others => '0');
    elsif (uPCsig = '1') then
      uPC <= uAddr;
    else
      uPC <= uPC + 1;
    end if;
  end if;
end process;

-- IR : Instruction Register
process(clk)
begin
  if rising_edge(clk) then
    if (rst = '1') then
        IR <= (others => '0');
    elsif (FB = "001") then
        IR <= DATA_BUS;
    end if;
  end if;
end process;
```

uprog CPU



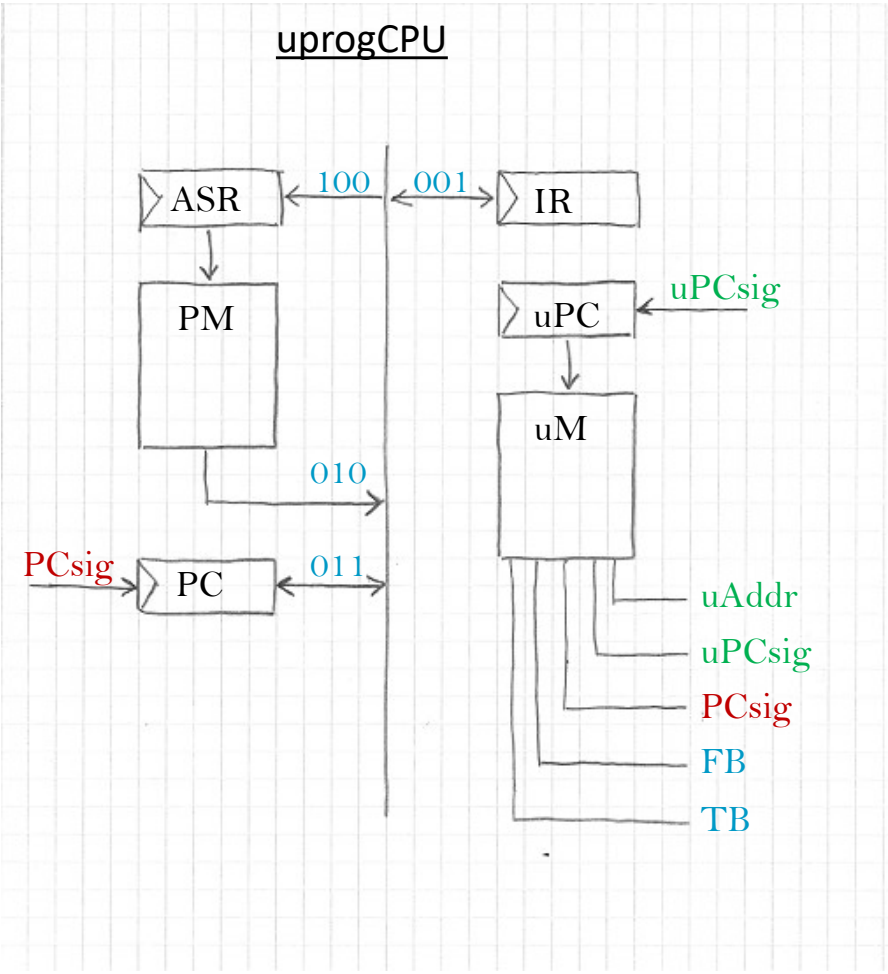
```
-- PC : Program Counter
process(clk)
begin
  if rising_edge(clk) then
    if (rst = '1') then
      PC <= (others => '0');
    elsif (FB = "011") then
      PC <= DATA_BUS;
    elsif (PCsig = '1') then
      PC <= PC + 1;
    end if;
  end if;
end process;

-- ASR : Address Register
process(clk)
begin
  if rising_edge(clk) then
    if (rst = '1') then
      ASR <= (others => '0');
    elsif (FB = "100") then
      ASR <= DATA_BUS;
    end if;
  end if;
end process;
```

uprog CPU



uprog CPU

```
 -- micro memory component connection
  U0 : uMem port map(uAddr=>uPC, uData=>uM);

  -- program memory component connection
  U1 : pMem port map(pAddr=>ASR, pData=>PM);

  DATA_BUS <= IR when (TB = "001") else
              PM when (TB = "010") else
              PC when (TB = "011") else
              ASR when (TB = "100") else
              (others => '0');

end architecture;
```

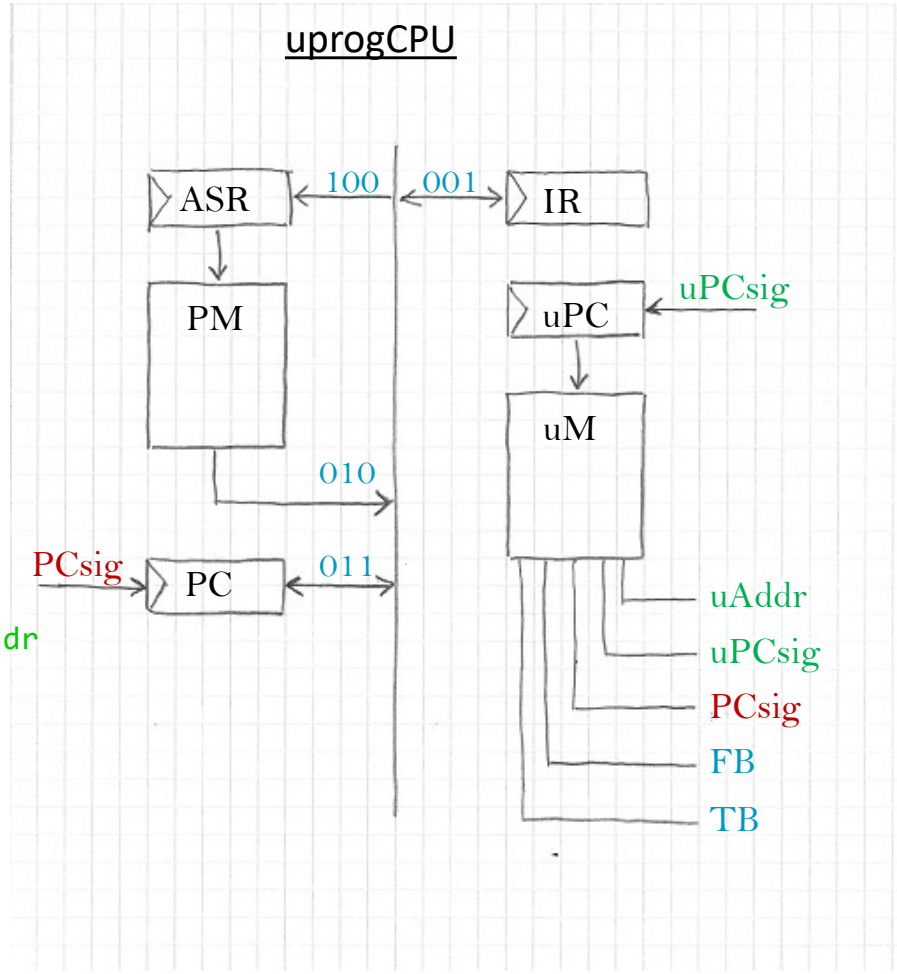


Makefile

```
...
proj.%: S=uprogCPU.vhd uMem.vhd pMem.vhd
proj.%: T=uprogCPU_tb.vhd
proj.%: U=Nexys3_Master.ucf
...
```

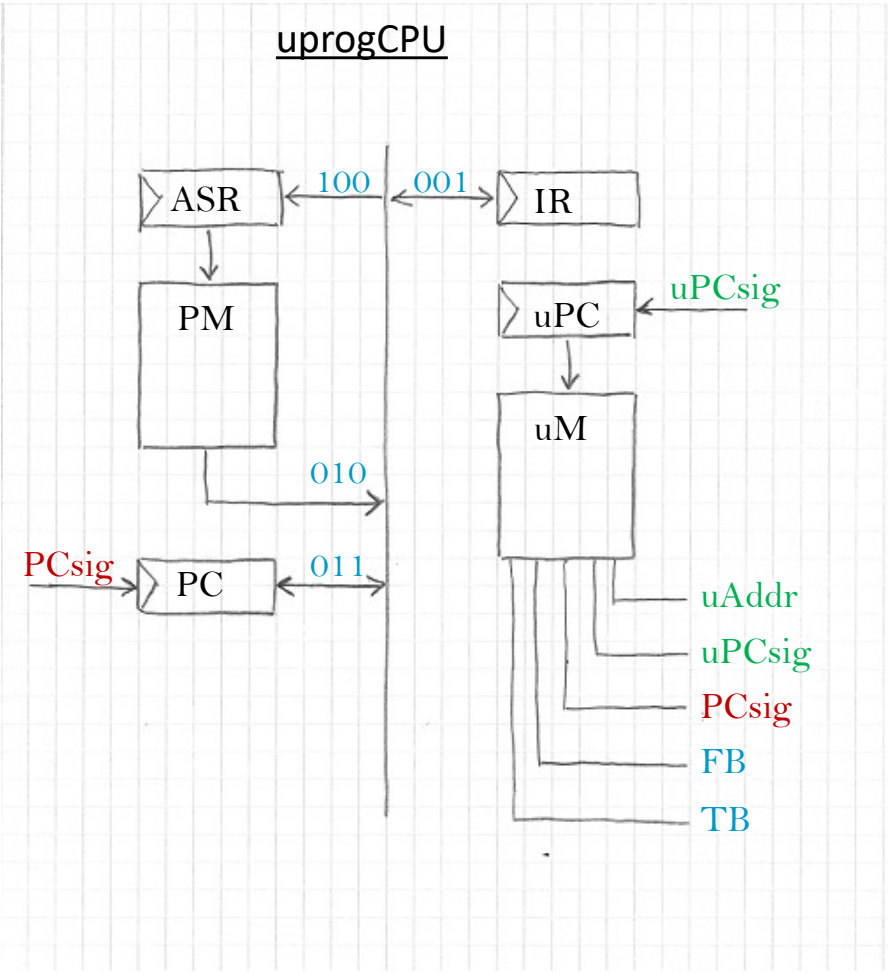
uMem.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity uMem is
  port(uAddr       : in unsigned(5 downto 0);
       uData       : out unsigned(15 downto 0));
end entity;

architecture func of uMem is
-- micro Memory
type u_mem_t is array (0 to 15) of unsigned(15 downto 0);
constant u_mem_c : u_mem_t :=
          --ALU_TB_FB_PC_uPC_uAddr
         (b"00_011_100_0_0_000000",      -- ASR:=PC
          b"00_010_001_1_1_000000",      -- IR:=PM, PC:=PC+1, uPC:=uAddr
          b"00_000_000_0_0_000000",
                  . . .
          b"00_000_000_0_0_000000");

signal u_mem : u_mem_t := u_mem_c;

begin
  uData <= u_mem(to_integer(uAddr));
end architecture;
```

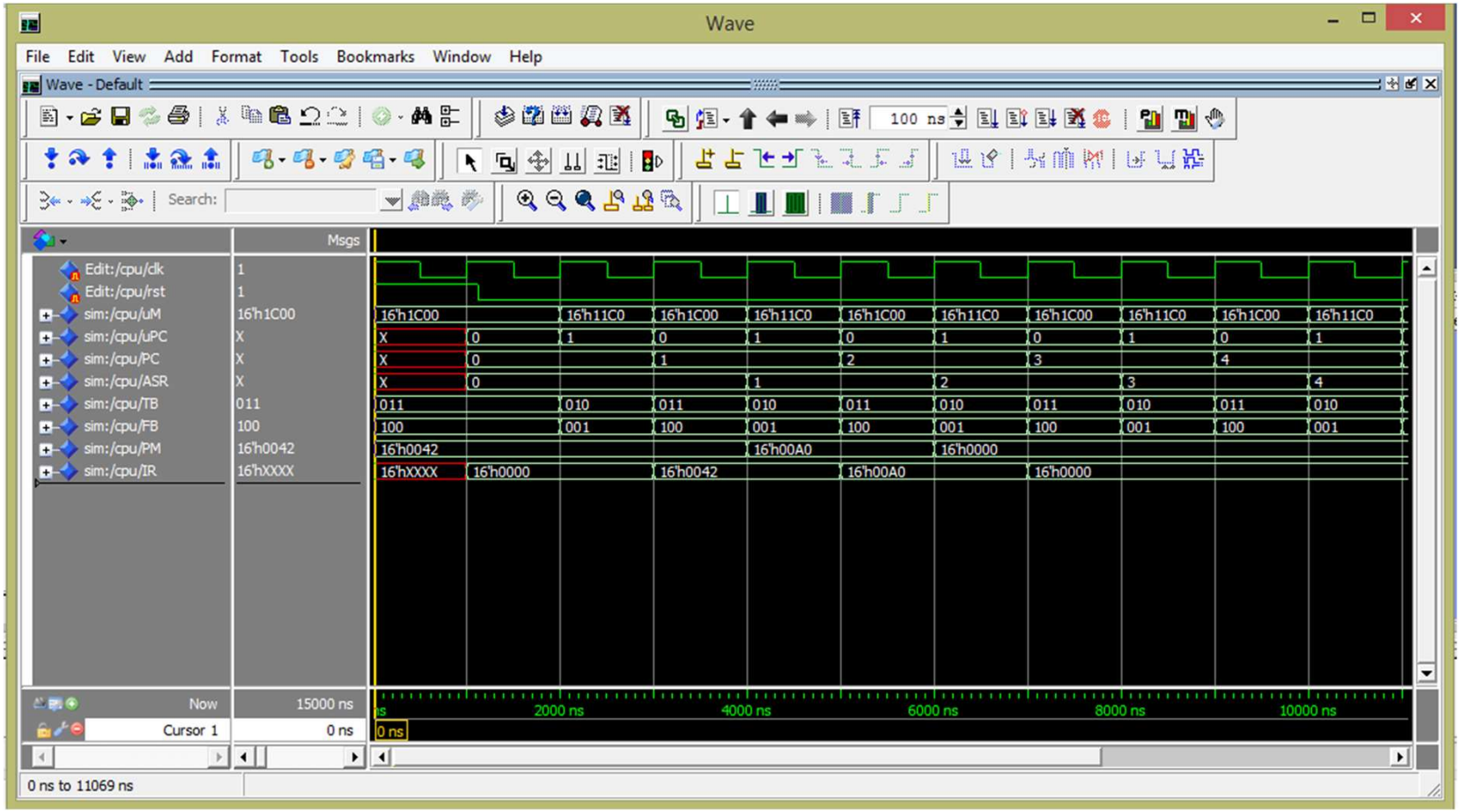


pMem.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pMem is
  port(pAddr       : in unsigned(15 downto 0);
       pData       : out unsigned(15 downto 0));
end entity;

Architecture func of pMem is
-- program Memory
type p_mem_t is array (0 to 15) of unsigned(15 downto 0);
constant p_mem_c : p_mem_t :=
         (x"0042",      --
          x”00A0",      --
          x”0000",--
          x”0000",
            . . .
          x"0000");

signal p_mem : p_mem_t := p_mem_c;

begin
  pData <= p_mem(to_integer(pAddr));
end architecture;
```



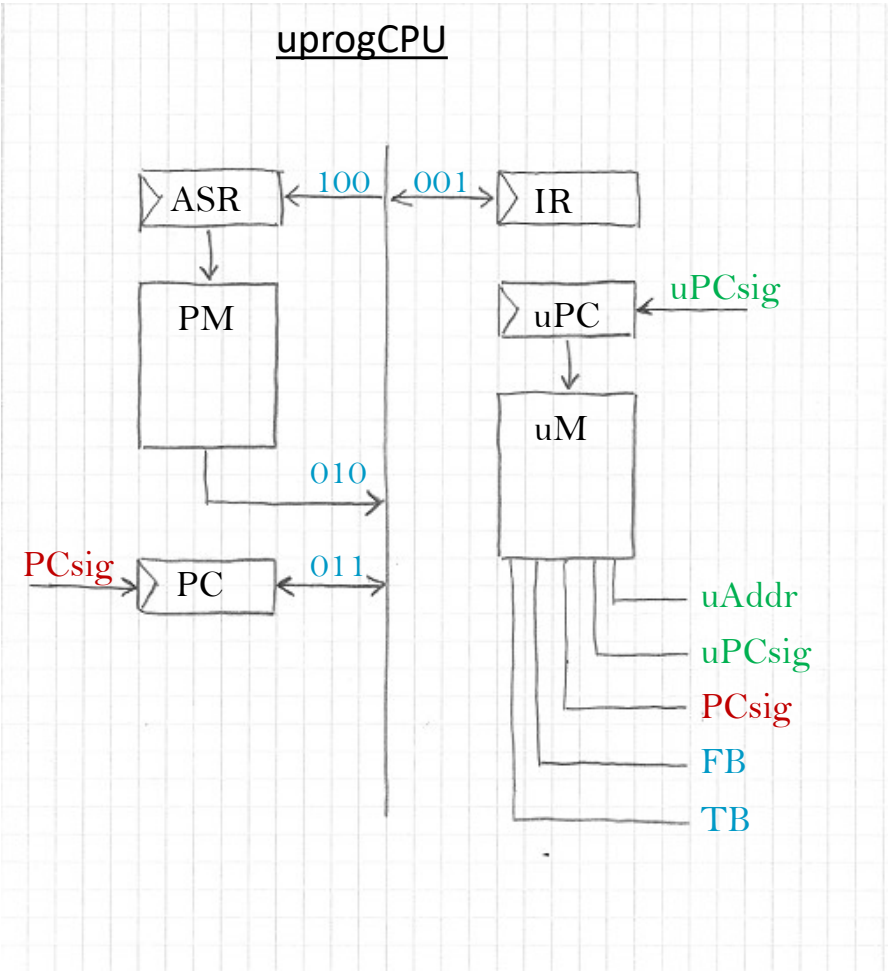
Datorkonstruktion uprog CPU simulering med Modelsim



LINKÖPINGS UNIVERSITET

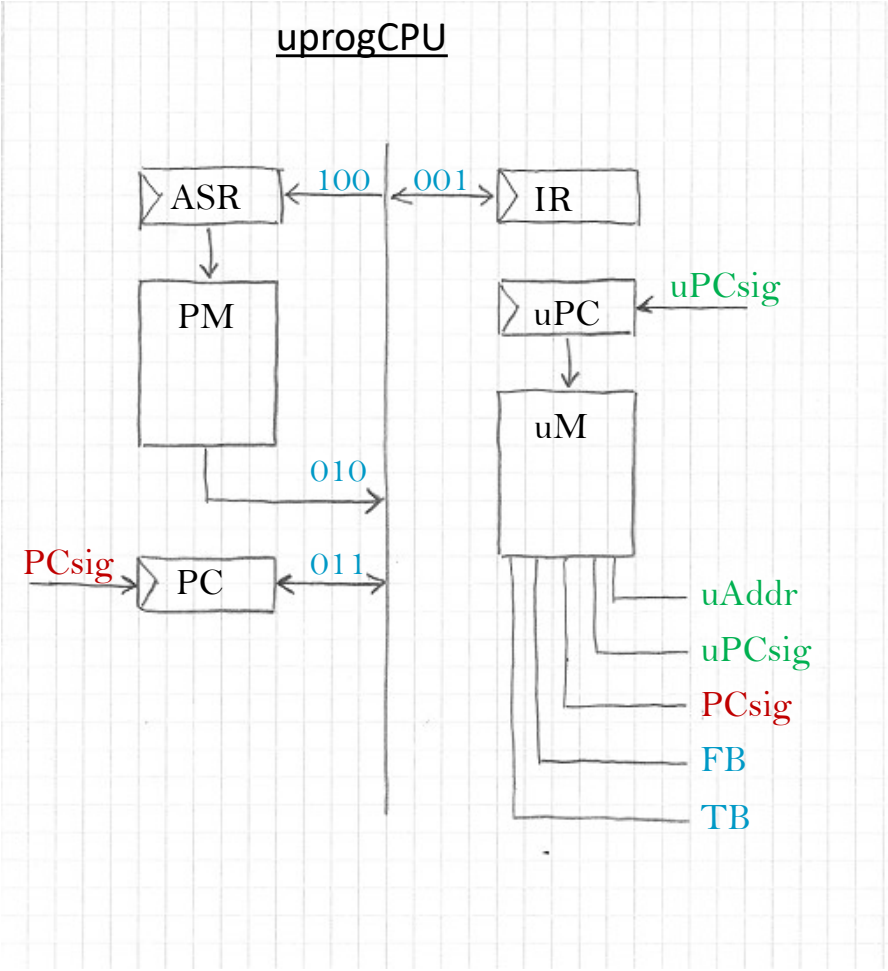
Datorkonstruktion uprogCPU_tb.vhd testbänken

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY uprogCPU_tb IS
END uprogCPU_tb;

ARCHITECTURE func OF uprogCPU_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT uprogCPU
    PORT(
         clk : IN  std_logic;
         rst : IN  std_logic
        );
    END COMPONENT;

   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';

   -- Clock period definitions
   constant clk_period : time := 1 us;
```



uprogCPU_tb.vhd testbänken

```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: uprogCPU PORT MAP (
          clk => clk,
          rst => rst
        );

   -- Clock process definitions
   clk_process :process
   begin
     clk <= '0';
     wait for clk_period/2;
     clk <= '1';
     wait for clk_period/2;
   end process;

   rst <= '1', '0' after 1.5 us;


END;
```



Minnen i FPGA

Distributed RAM
BlockRAM

Minnen i FPGA

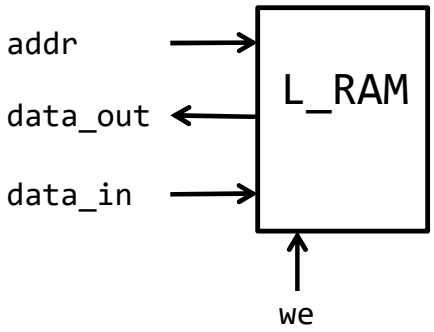
- Distributed RAM (LUT) : passar till K1, K2, registerfil, programminne, mikrominne (ROM)
  - Kombinatorisk läsning
  - Klockad skrivning

- Block RAM: passar till bildminne, (programminne), ...
  - Klockad läsning
  - Klockad skrivning

Man kan påverka vilken minnestyp det blir med sin VHDL-kod, men i båda fallen rekommenderas att syntesverktyget får avgöra

LUT-RAM deklareras i separat fil

```
entity L_RAM is
port(clk  : in std_logic;
     -- port
     we       : in std_logic; -- write enable
     data_in  : in std_logic_vector(7 downto 0);
     data_out : out std_logic_vector(7 downto 0);
     addr     : in unsigned(10 downto 0));
end entity;

architecture func of L_RAM is
 -- RAM type
type ram_t is array (0 to 2047) of std_logic_vector(7 downto 0);
 -- RAM init : address 0 = x"1F", other addresses = 0
signal lram : ram_t := (0 => x"1F", others => (others => '0'));

process(clk)
begin
  if rising_edge(clk) then
    if (we = '1') then
      lram(to_integer(addr)) <= data_in;
    end if;
  end if;
end process;

data_out <= lram(to_integer(addr));

end architecture;
```

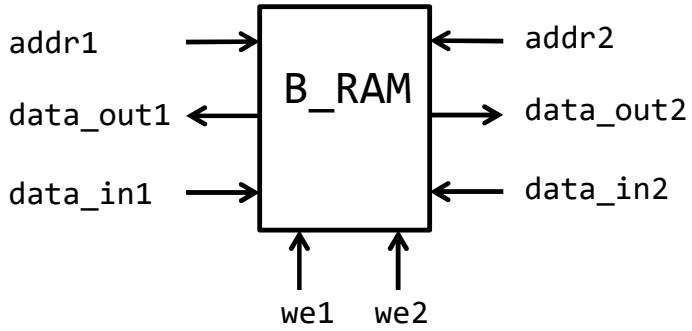


-"En-ports-RAM"
-Synkron skrivning
-ASynkron läsning

Access av minnet, ska **ENDAST** förekomma inne i RAM-komponenten. Detta för att undvika multipla instanser av minnet.

Block-RAM deklareras i separat fil

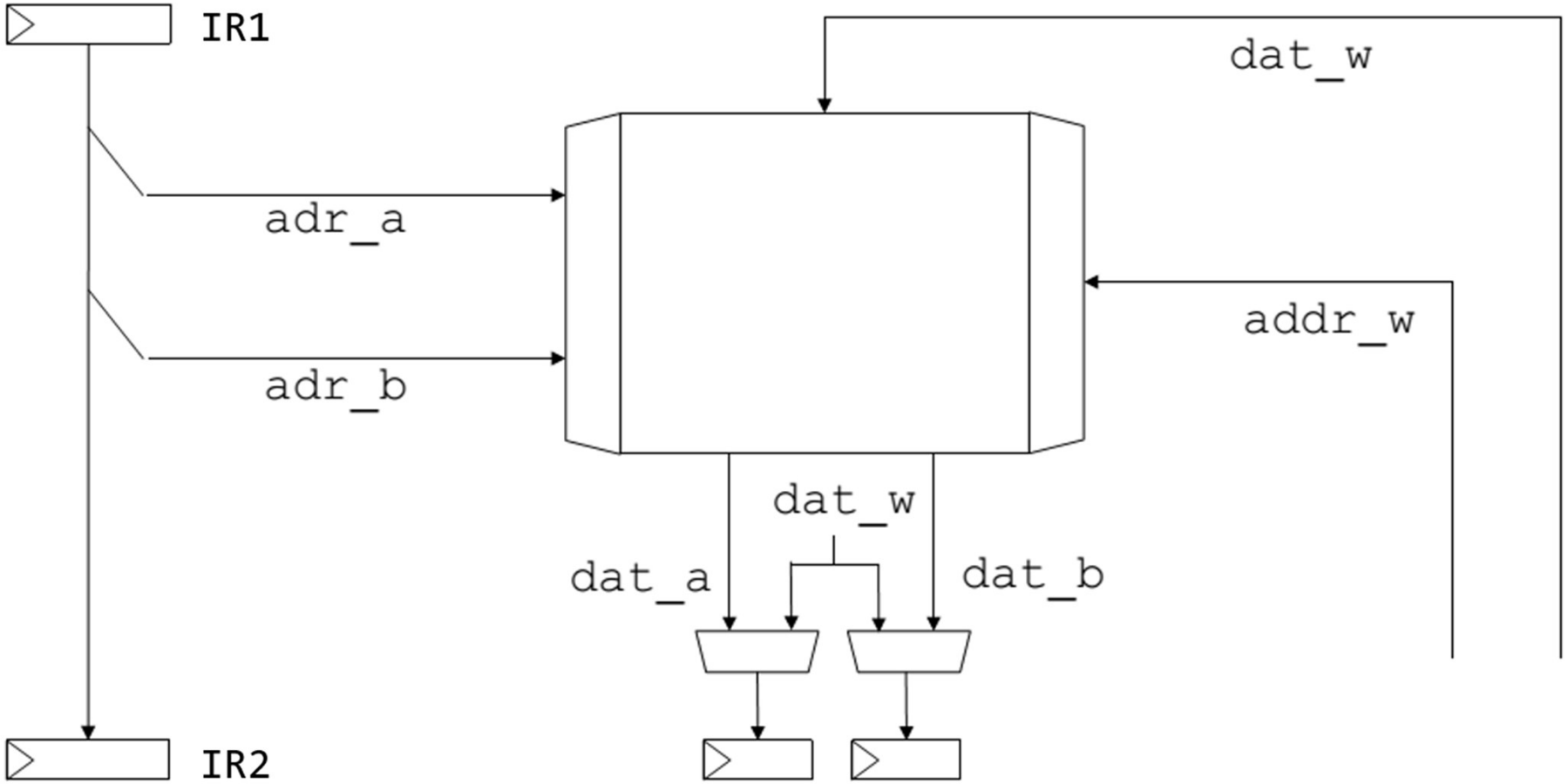
```
entity B_RAM is
port(clk  : in std_logic;
     -- port 1
     we1      : in std_logic;
     data_in1 : in std_logic_vector(7 downto 0);
     data_out1: out std_logic_vector(7 downto 0);
     addr1    : in unsigned(10 downto 0);
     -- port 2
     we2      : in std_logic;
     data_in2 : in std_logic_vector(7 downto 0);
     data_out2: out std_logic_vector(7 downto 0);
     addr2    : in unsigned(10 downto 0));
end entity;

architecture func of B_RAM is
 -- RAM type
type ram_t is array (0 to 2047) of std_logic_vector(7 downto 0);
 -- RAM init : address 0 = x"1F", other addresses = 0
signal bram : ram_t := (0 => x"1F", others => (others => '0'));

process(clk)
begin
  if rising_edge(clk) then
    if (we1 = '1') then
      bram(to_integer(addr1)) <= data_in1;
    end if;
    data_out1 <= bram(to_integer(addr1));

    if (we2 = '1') then
      bram(to_integer(addr2)) <= data_in2;
    end if;
    data_out2 <= bram(to_integer(addr2));
  end if;
end process;

end architecture;
```



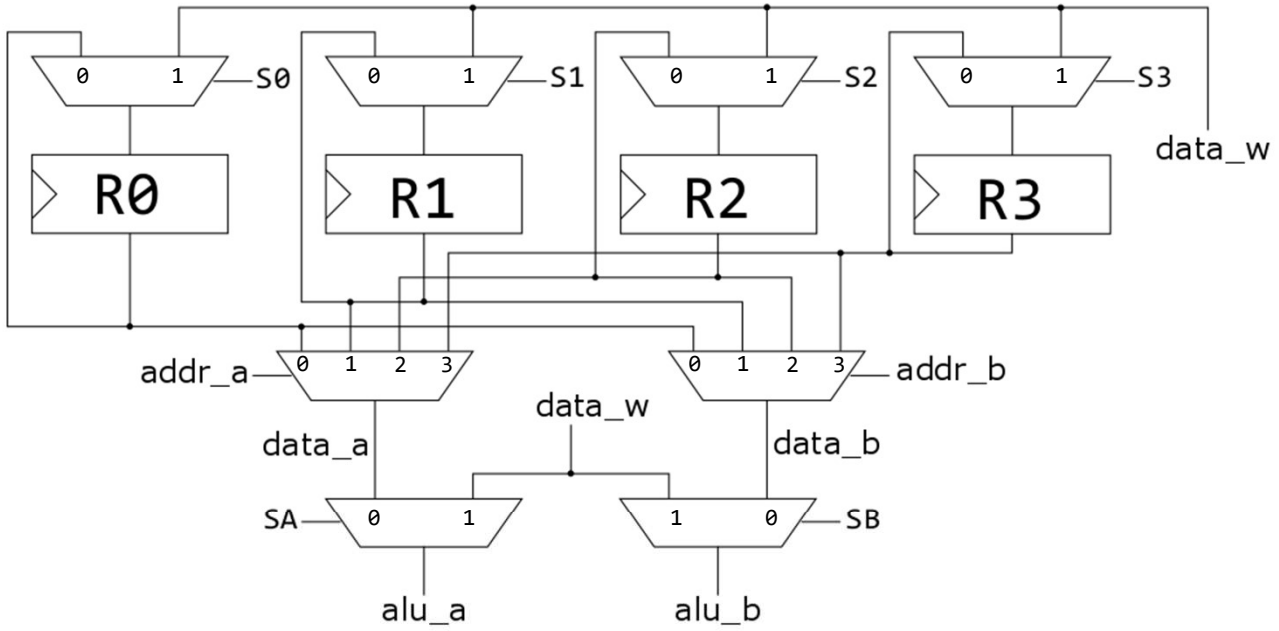
-"Två-ports-RAM"
-Synkron skrivning
-Synkron läsning

Access av minnet, ska **ENDAST** förekomma inne i RAM-komponenten. Detta för att undvika multipla instanser av minnet.

Registerfil i pipelinad dator "3-ports-minne"

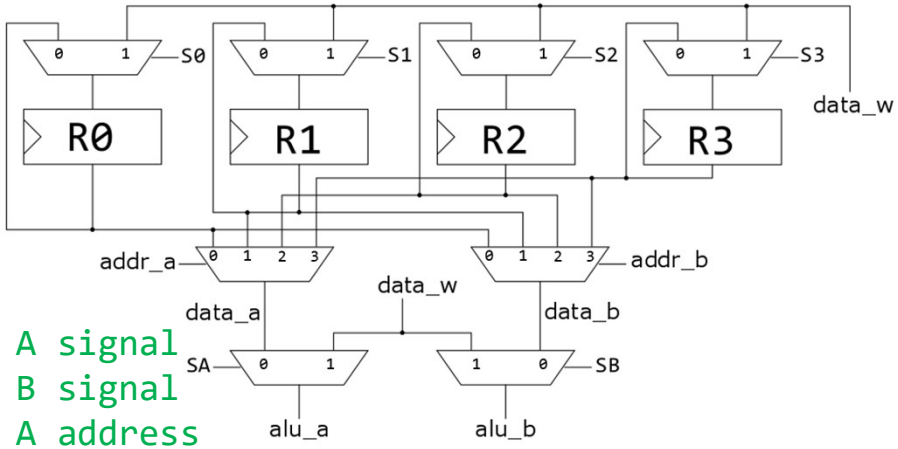


Registerfil i pipelinad dator "3-ports-minne"




```
S0 : (addr_w = "00") and (w_d = '1')
S1 : (addr_w = "01") and (w_d = '1')
S2 : (addr_w = "10") and (w_d = '1')
S3 : (addr_w = "11") and (w_d = '1')

SA : (addr_a = addr_w) and (r_a = '1') and (w_d = '1')
SB : (addr_b = addr_w) and (r_b = '1') and (w_d = '1')
```

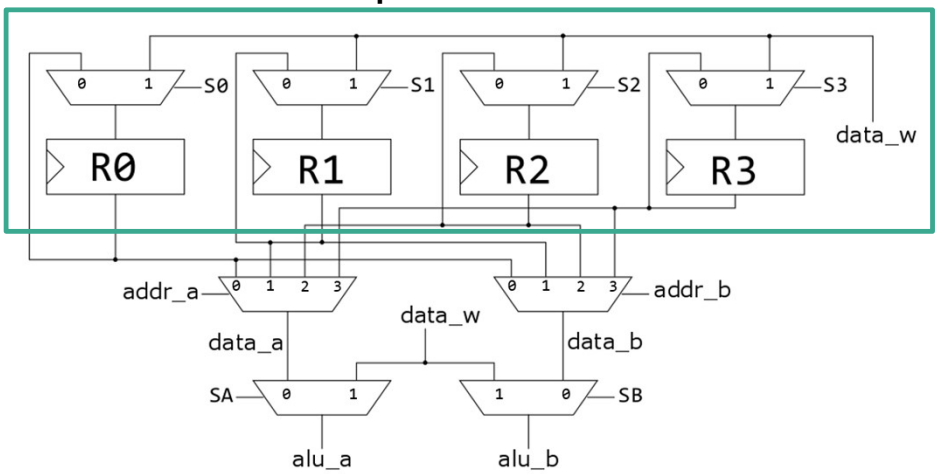
RF.vhd : Registerfil i pipelinad dator "3-ports-minne"




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity register_file is
  port(clk : in std_logic;
       r_a : in std_logic; -- read source register A signal
       r_b : in std_logic; -- read source register B signal
       addr_a : in unsigned(1 downto 0); -- source A address
       addr_b : in unsigned(1 downto 0); -- source B address
       w_d : in std_logic; -- write destination register signal
       addr_w : in unsigned(1 downto 0); -- destination address
       data_w : in unsigned(7 downto 0); -- destination data
       alu_a : out unsigned(7 downto 0); -- alu A data
       alu_b : out unsigned(7 downto 0)  -- alu B data
      )
end entity;
```

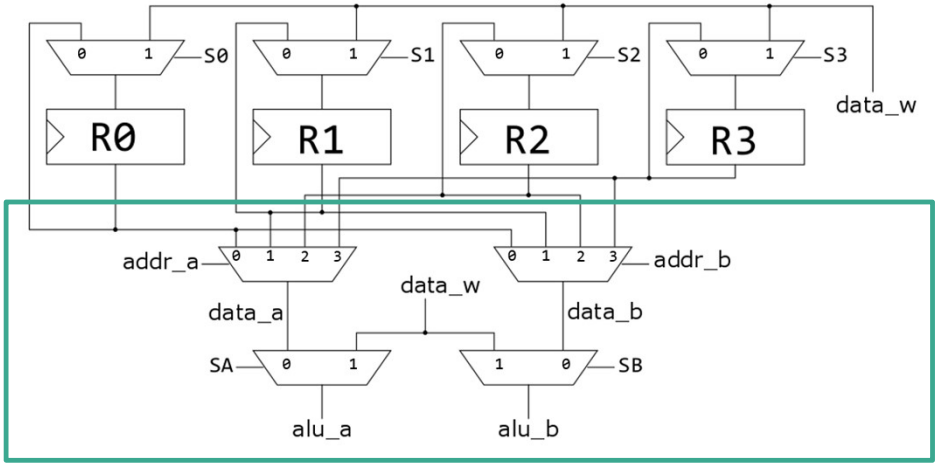
RF.vhd : Registerfil i pipelinad dator "3-ports-minne"

```
architecture func of register_file is

signal R0, R1, R2, R3 : unsigned(7 downto 0);
signal data_a, data_b : unsigned(7 downto 0);

begin

process(clk)
begin
  if rising_edge(clk) then
    if (w_d = '1') then
      case addr_w is
        when "00" => R0 <= data_w;
        when "01" => R1 <= data_w;
        when "10" => R2 <= data_w;
        when "11" => R3 <= data_w;
      end case;
    end if;
  end if;
end process;
```



RF.vhd : Registerfil i pipelinad dator "3-ports-minne"

```
data_a <= R0 when (addr_a = "00") else
          R1 when (addr_a = "01") else
          R2 when (addr_a = "10") else
          R3;

data_b <= R0 when (addr_b = "00") else
          R1 when (addr_b = "01") else
          R2 when (addr_b = "10") else
          R3;

alu_a <= data_w when ((addr_a = addr_w) and
                      (w_d = '1') and
                      (r_a = '1')) else
         data_a;

alu_b <= data_w when ((addr_b = addr_w) and
                      (w_d = '1') and
                      (r_b = '1')) else
         data_b;

end architecture;
```

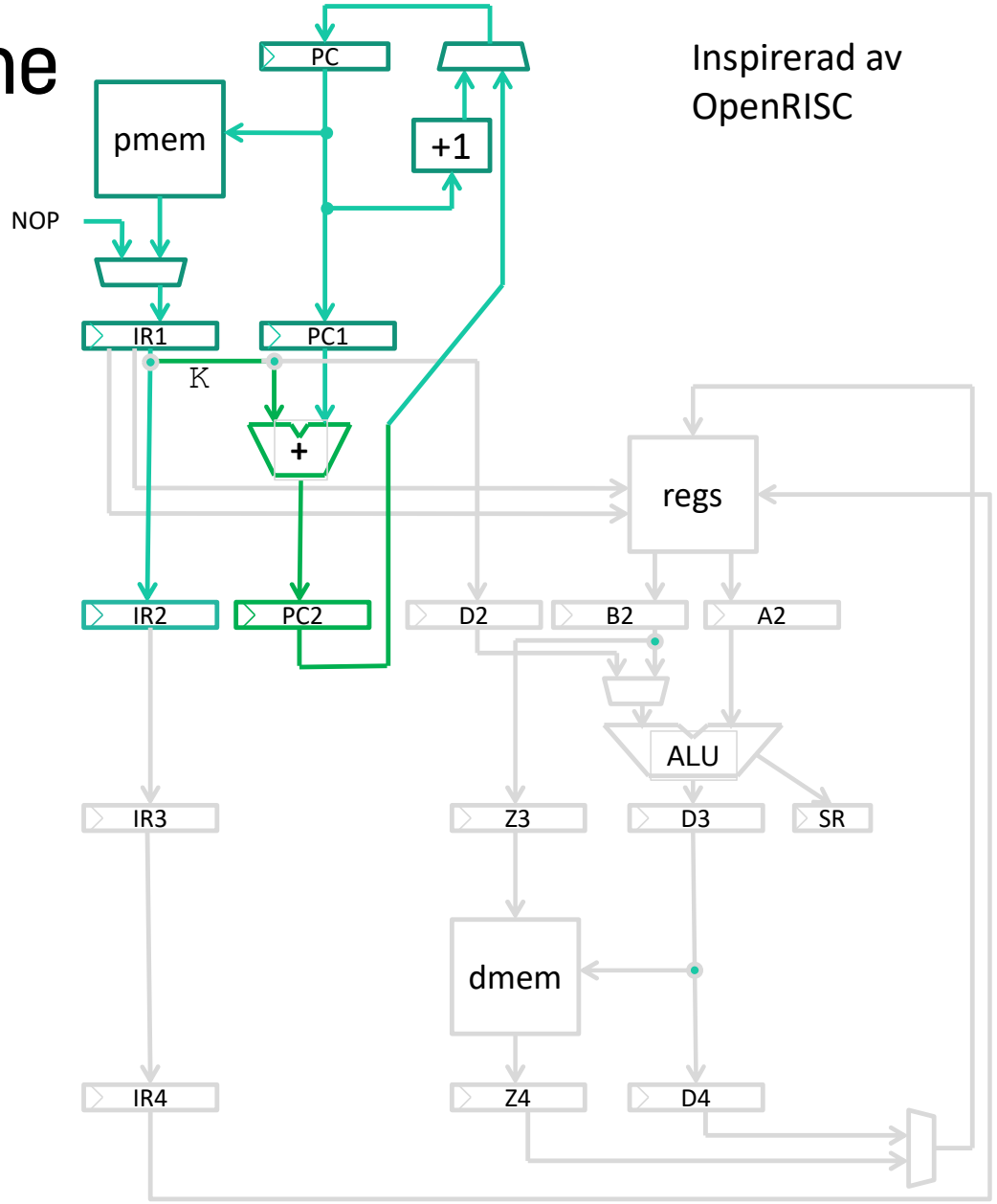


pipelinad CPU

VHDL-kod för pipeline-CPU med instruktionshämtning

Klassisk 5-stegs pipeline

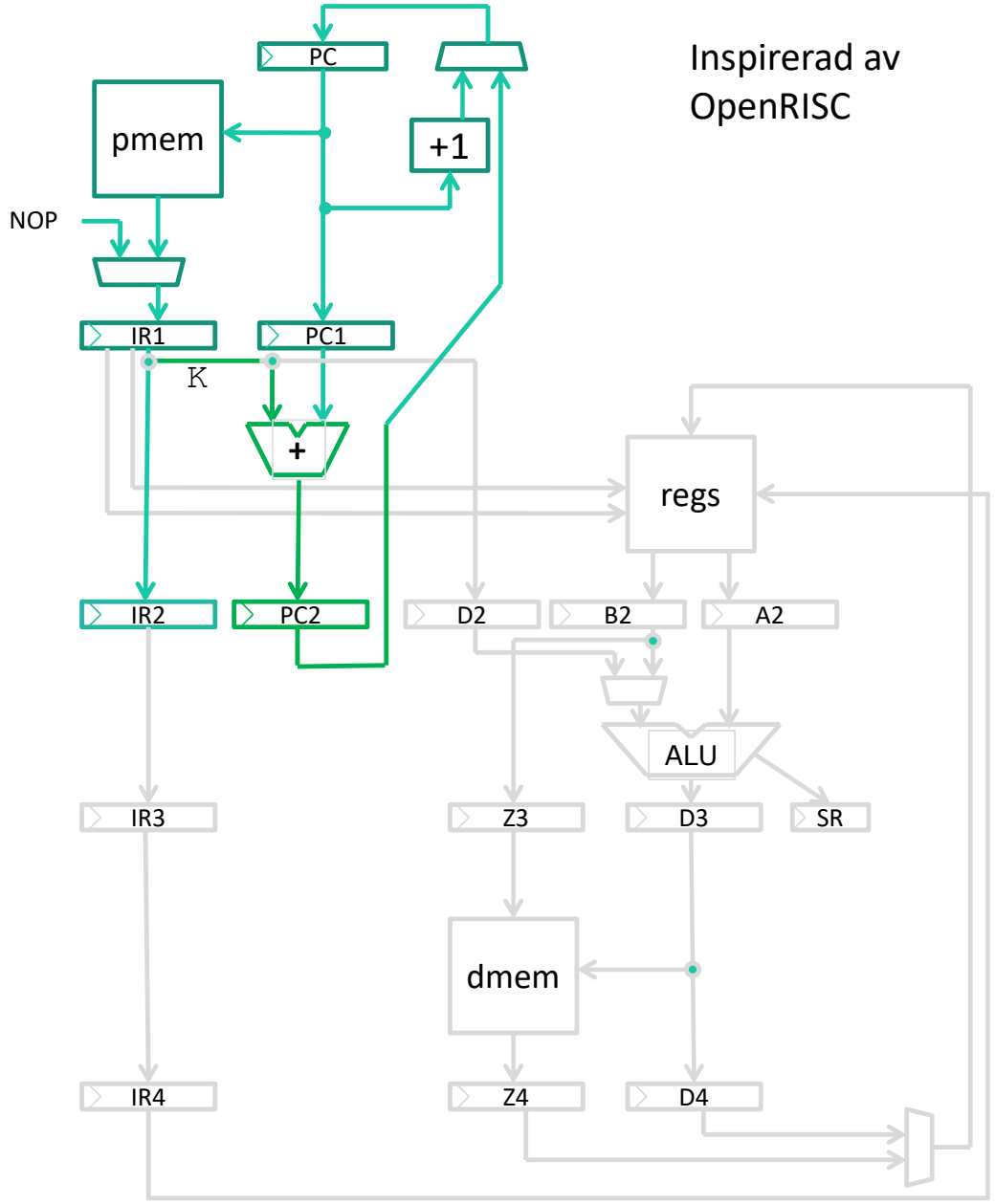
- **IF**: instruction fetch
  hämta instr och ny PC
- **RR**: register read
  läs reg/beräkna hopp
- **EXE**: execute
  kör ALU
- **MEM**: read/write dmem
  läs/skriv/ingenting
- **WB**: write back register
  skriv reg/ingenting



Inspirerad av OpenRISC

pipeCPU.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- CPU interface
entity pipeCPU is
    port(
      clk     : in std_logic;
      rst     : in std_logic
    );
end entity;
```

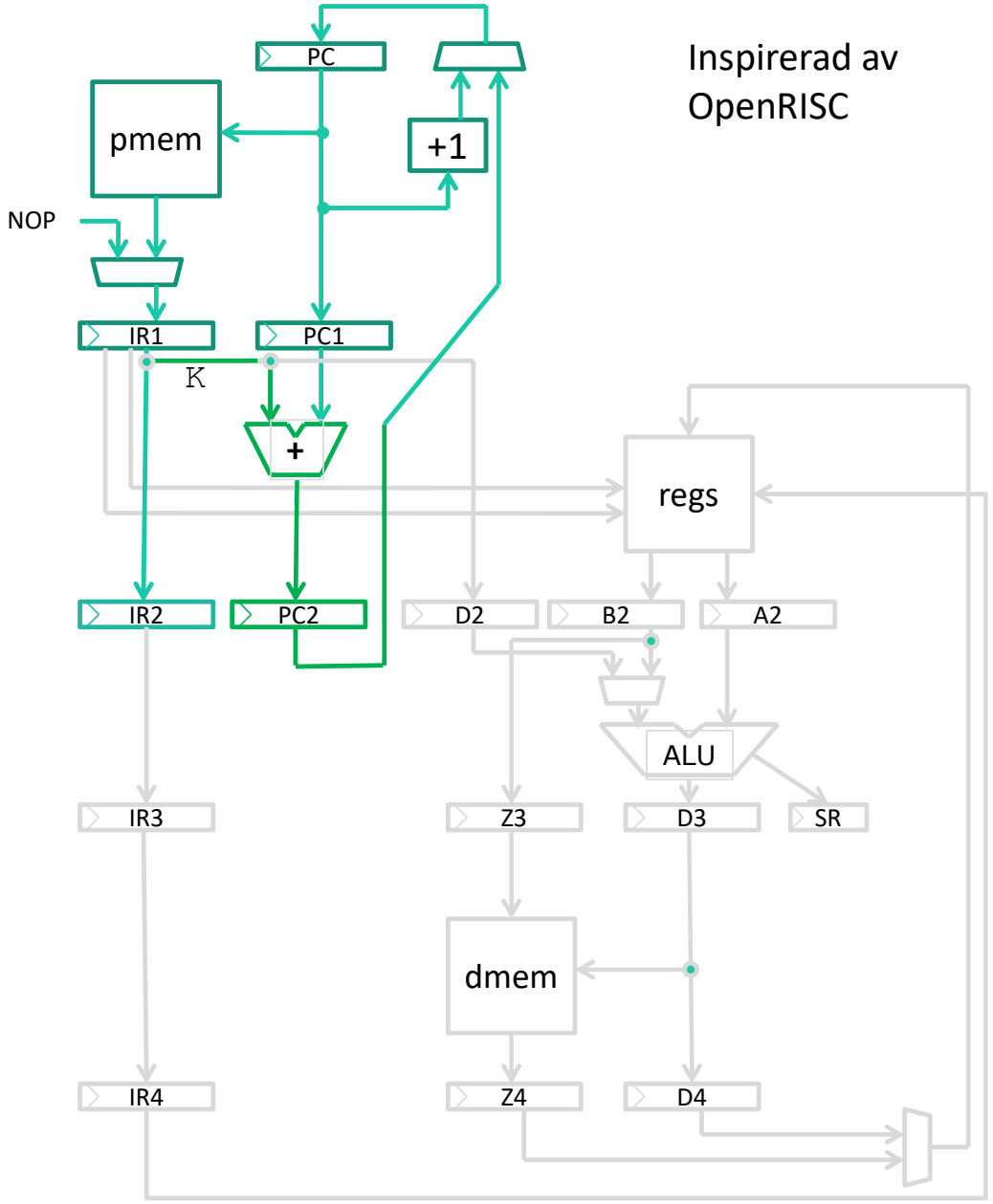


Inspirerad av OpenRISC

Datorkonstruktion

pipeCPU.vhd

```
architecture func of pipeCPU is

signal IR1 : unsigned(31 downto 0);
alias IR1_op : unsigned(5 downto 0) is IR1(31 downto 26);
alias IR1_d : unsigned(4 downto 0) is IR1(25 downto 21);
alias IR1_a : unsigned(4 downto 0) is IR1(20 downto 16);
alias IR1_b : unsigned(4 downto 0) is IR1(15 downto 11);
alias IR1_c : unsigned(10 downto 0) is IR1(10 downto 0);

signal IR2 : unsigned(31 downto 0);
alias IR2_op : ...
alias ...

signal PC, PC1, PC2 : unsigned(10 downto 0);

signal PMdata_out : unsigned(31 downto 0);
signal pm_addr : unsigned(8 downto 0);

constant iNOP : unsigned(5 downto 0) := "000000";
constant iJ   : unsigned(5 downto 0) := "010101";
constant iBF  : unsigned(5 downto 0) := "000100";
```

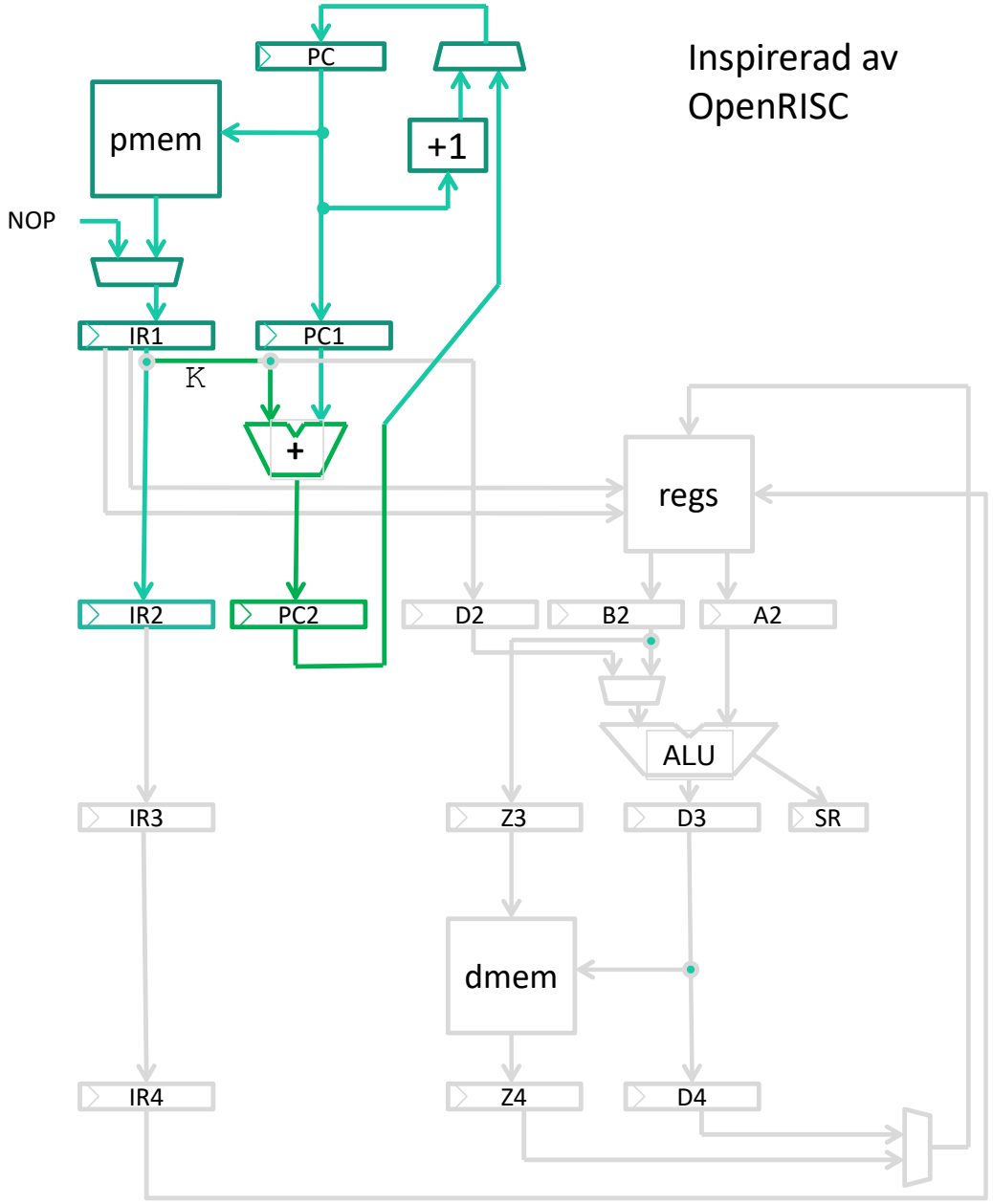


Inspirerad av OpenRISC

LINKÖPINGS UNIVERSITET

pipeCPU.vhd

```
component PM_comp is
  port(addr : in unsigned(8 downto 0);
       data_out : out unsigned(31 downto 0));
end component;

begin

  U1 : PM_comp port map(
       addr => pm_addr,
       data_out => PMdata_out
  );
```



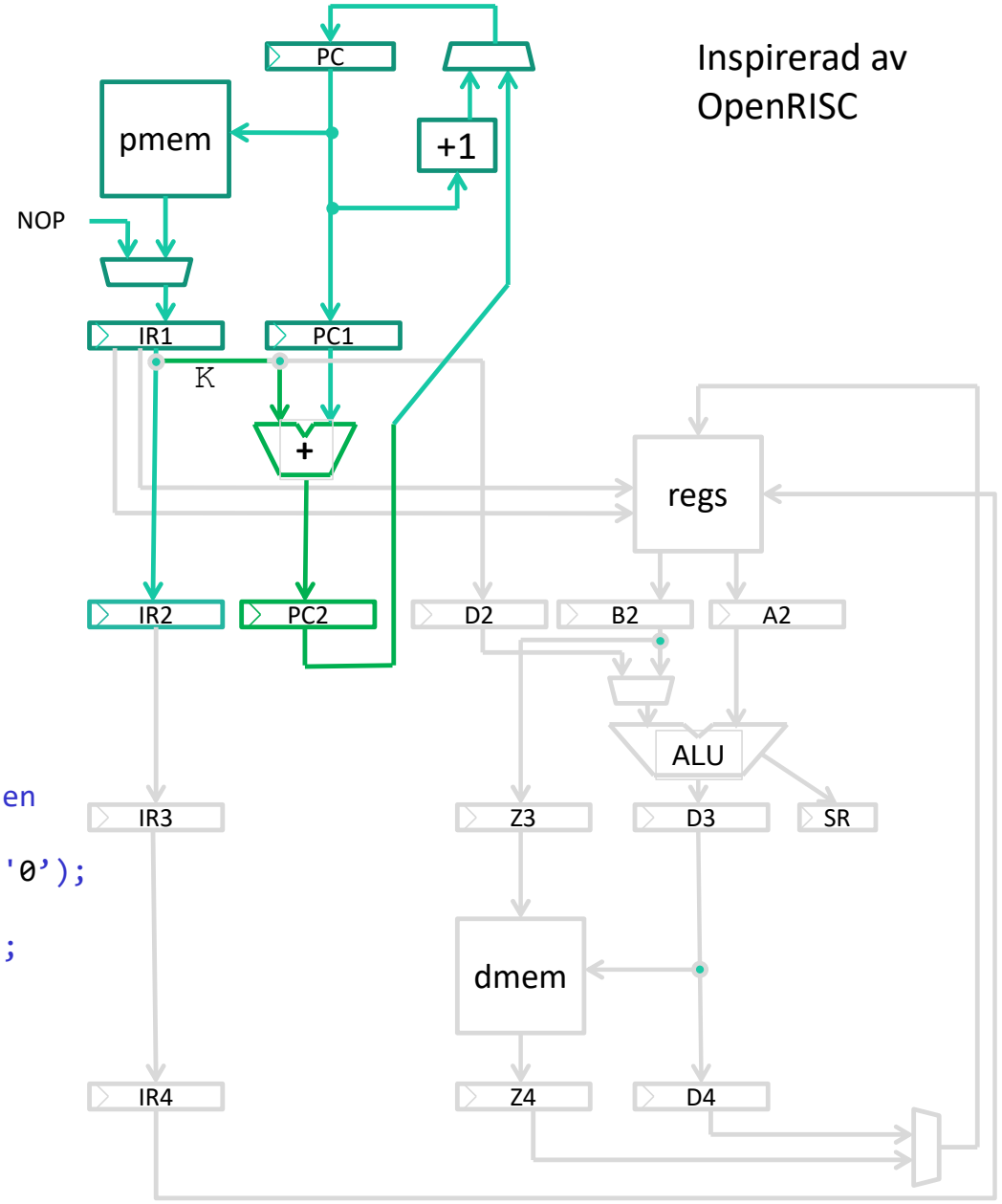
Inspirerad av OpenRISC

pipeCPU.vhd

```
process(clk)
begin
  if rising_edge(clk) then
    if (rst='1') then
      PC <= (others => '0');
    elsif (IR2_op = iJ) then
      PC <= PC2;
    else
      PC <= PC + 1;
    end if;
  end if;
end process;

pm_addr <= PC(8 downto 0);
```

```
process(clk)
begin
  if rising_edge(clk) then
    if (rst='1') then
      PC1 <= (others => '0');
    else
      PC1 <= PC;
    end if;
  end if;
end process;
```

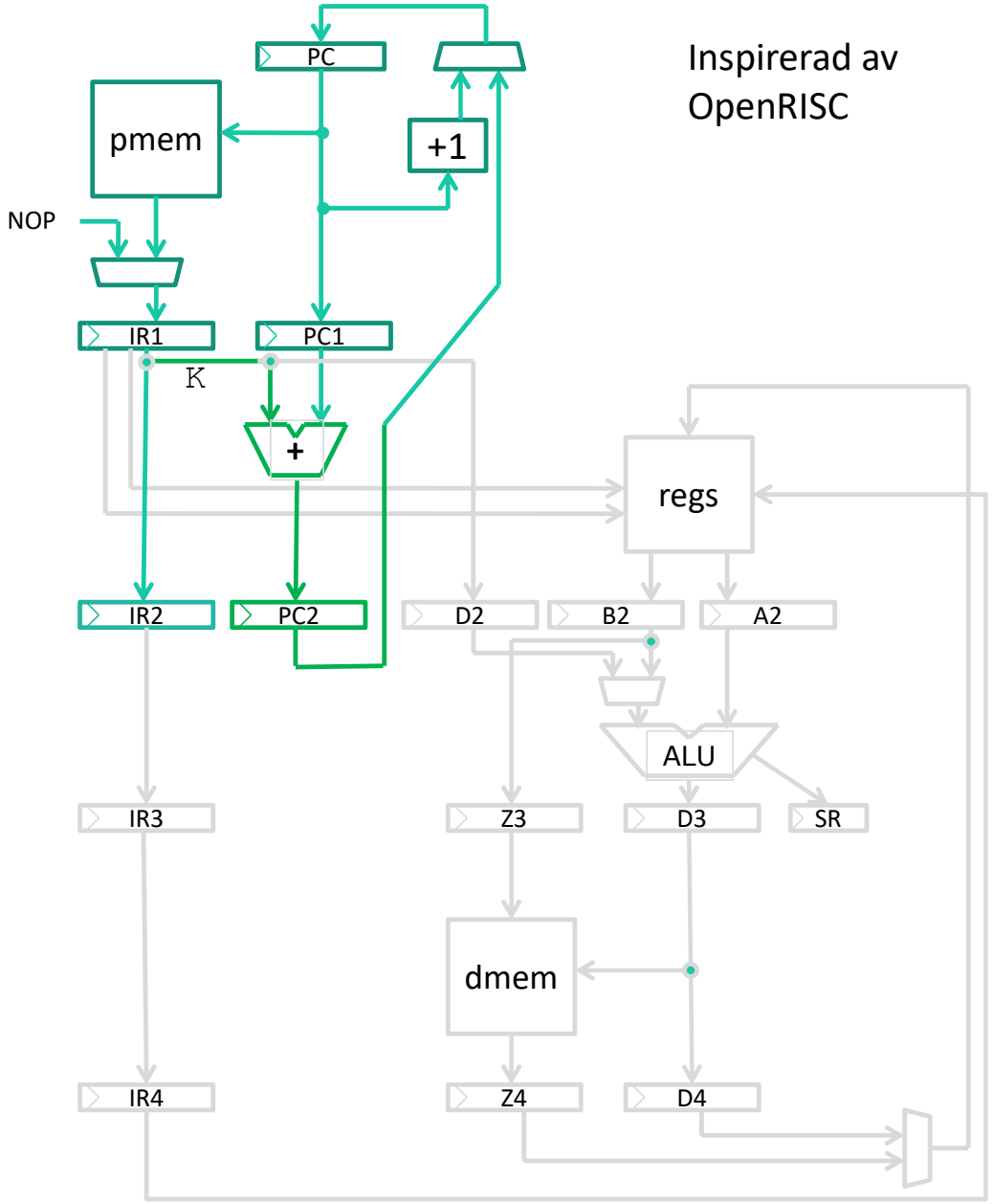
```
process(clk)
begin
  if rising_edge(clk) then
    if (rst='1') then
      PC2 <= (others => '0');
    else
      PC2 <= PC1 + IR1_c;
    end if;
  end if;
end process;
```



Inspirerad av OpenRISC

pipeCPU.vhd

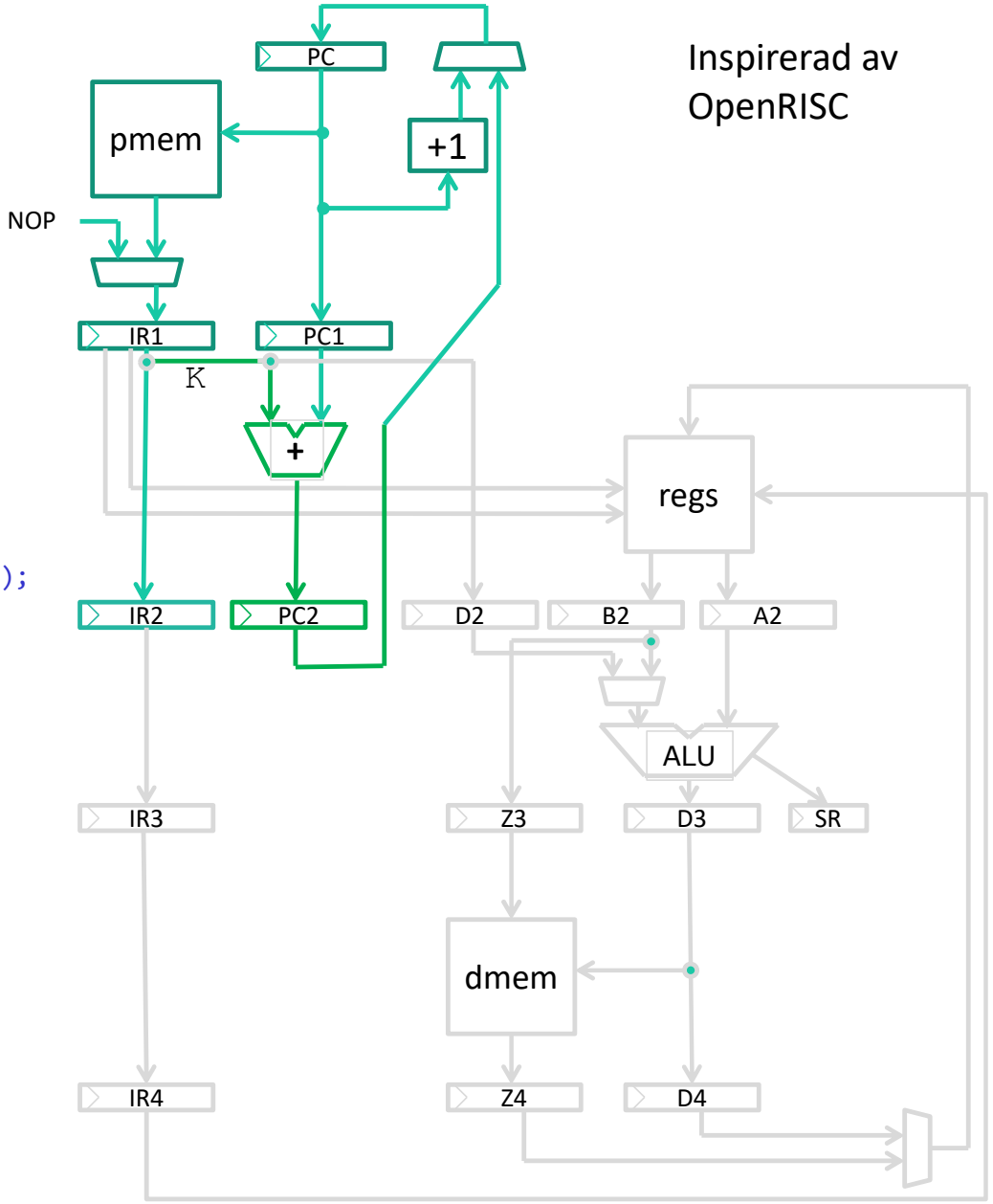
```
process(clk)
begin
  if rising_edge(clk) then
    if (rst='1') then
      IR1 <= (others => '0');
      IR1_op <= iNOP;
    elsif (IR2_op = iJ) then
      IR1_op <= iNOP;
    else
      IR1 <= PMdata_out(31 downto 0);
    end if;
  end if;
end process;


process(clk)
begin
  if rising_edge(clk) then
    if (rst='1') then
      IR2 <= (others => '0');
      IR2_op <= iNOP;
    else
      IR2 <= IR1;
    end if;
  end if;
end process;


end architecture;
```



Inspirerad av OpenRISC

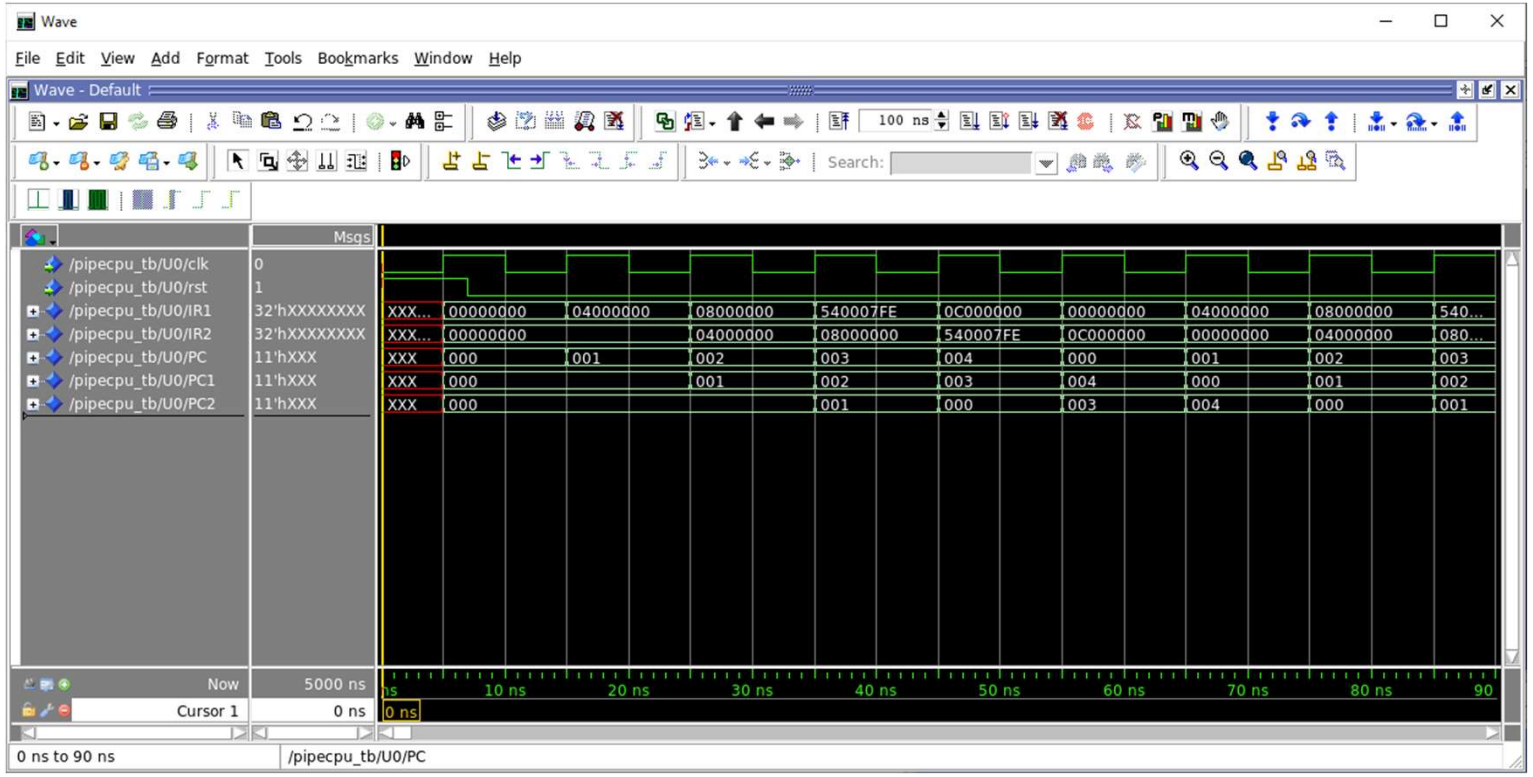
PM.vhd deklareras i separat fil

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

entity PM_comp is
  port(
    addr : in unsigned(8 downto 0);
    data_out : out unsigned(31 downto 0)
  );
end entity;

architecture func of PM_comp is

  type PM_t is array(0 to 511) of unsigned(31 downto 0);
  constant PM_c : PM_t := (
           X"04000000",          -- dummy
           X"08000000",          -- dummy
           X“540007FE",          -- J 0
           X"0C000000",          -- dummy
           X"10000000",          -- dummy

           others => (others => '0')
           );

  signal PM : PM_t := PM_c;

begin
  data_out <= PM(to_integer(addr));
end architecture;
```



Inspirerad av OpenRISC

Simulering


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/pipecpu_tb/U0/clk 0
/pipecpu_tb/U0/rst 1
/pipecpu_tb/U0/IR1 32'hXXXXXXXX XXX... 00000000 04000000 08000000 540007FE 0C000000 00000000 04000000 08000000 540...
/pipecpu_tb/U0/IR2 32'hXXXXXXXX XXX... 00000000 04000000 08000000 540007FE 0C000000 00000000 04000000 080...
/pipecpu_tb/U0/PC 11'hXXX XXX 000 001 002 003 004 000 001 002 003
/pipecpu_tb/U0/PC1 11'hXXX XXX 000 001 002 003 004 000 001 002
/pipecpu_tb/U0/PC2 11'hXXX XXX 000 001 000 003 004 000 001
Now 5000 ns
Cursor 1 0 ns
10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90
0 ns
0 ns to 90 ns
/pipecpu_tb/U0/PC

Datorkonstruktion Lab4 : VGA

- VGA-labben
  -Makefile
  -VHDL-filer
  -Nexys3.ucf
- Kravspec - påminnelse
- Designspec – hur ska datorn byggas?

Anders Nilsson

www.liu.se

LINKÖPINGS UNIVERSITET