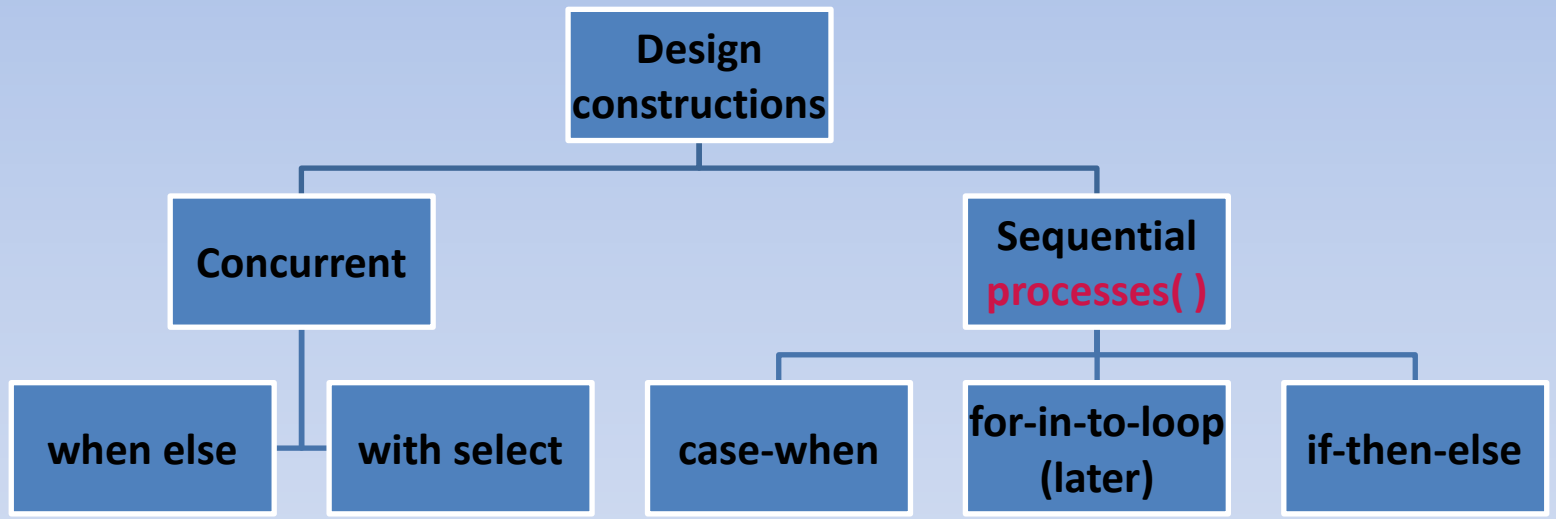
VHDL programmering för inbyggda system
Välkommen

- Introduktion till synkron process
- Asynkron och synkron reset (clear, initiering..)
- VHDL syntax för RAM och ROM

Version	Date	Responsible	Description
0.0	2011	LL&Mia	Preliminary version
1.0	2015	LL	Some bug fix

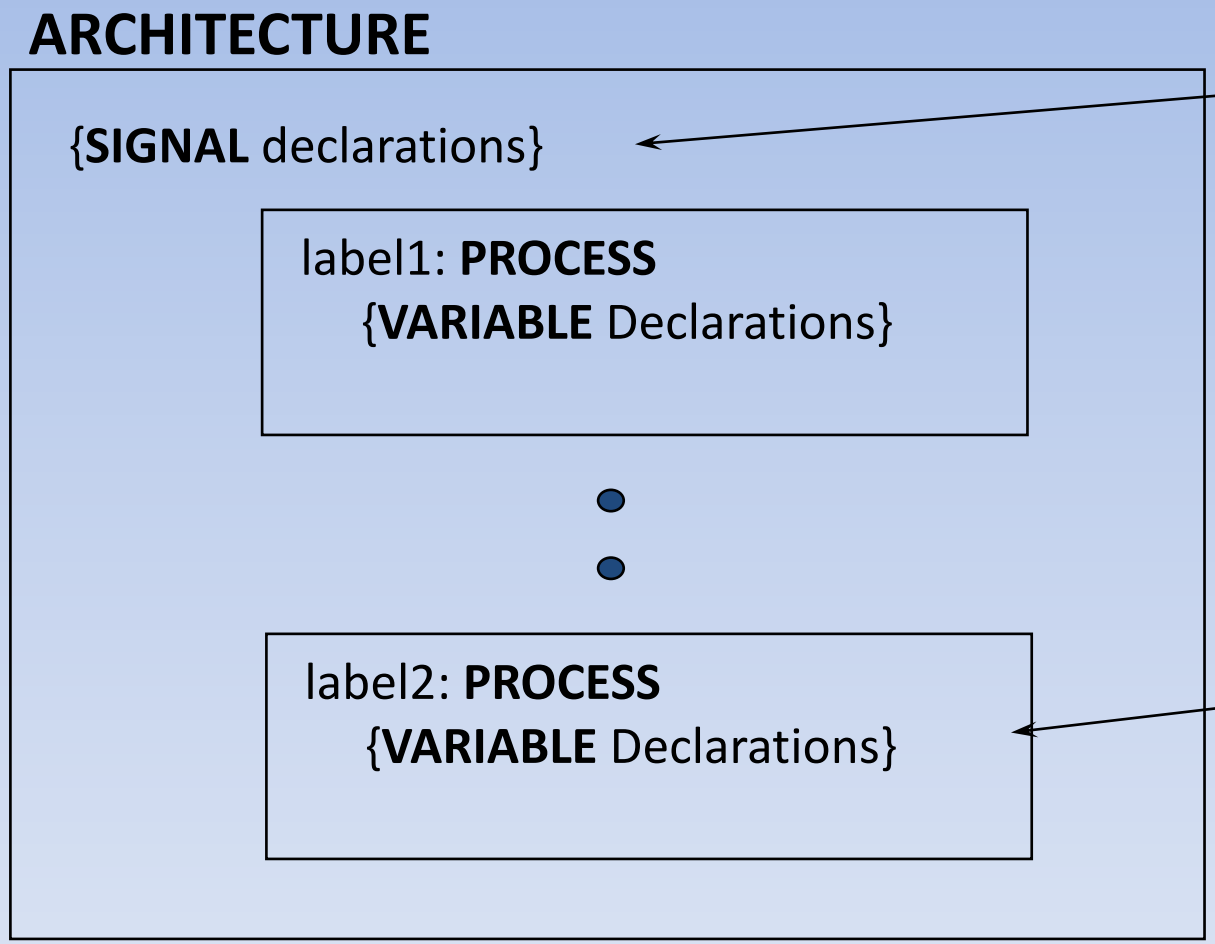
AGSTU

Design constructions



AGSTU

Signal and Variable Scope



Declared outside of the process statements (Visible to all process statements)

Signals get updated at the end of the delta cycle

Variables direct (delta cycle)

Declared inside the PROCESS statements (locally visible to the process statements)

AGSTU

IF-THEN Statements

```
PROCESS (sela, selb, a, b, c)
BEGIN
    IF sela='1' THEN
        q <= a;
    ELSIF selb='1' THEN
        q <= b;
    ELSE
        q <= c;
    END IF;
END PROCESS;
```


AGSTU

CASE Statement

```
PROCESS (sel, a, b, c, d)
BEGIN
    CASE sel IS
        WHEN "00" =>
            q <= a;
        WHEN "01" =>
            q <= b;
        WHEN "10" =>
            q <= c;
        WHEN OTHERS =>
            q <= d;
    END CASE;
END PROCESS;
```

AGSTU

Null – commando

Null do “nothing”

```
architecture VHDL_kod of VHDL_komp is
signal A:std_logic_vector(1 downto 0);
begin
        min_process: process(A)
        begin
        case A is
                when ”01” => q <= ’1’;
                when ”11” => q <= ’0’;
                when others => null;
        end case;
        end process;
end VHDL_kod;
```

AGSTU

Kombinatoriska processer

- Samtliga insignaler finnas i <sensitivity_list>.
- Samtliga utsignaler från processer ska alltid tilldelas ett värde varje gång den exekveras.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity VHDL_komp is
port(A,B: in std_logic;
      C: out std_logic);
end VHDL_komp;

architecture VHDL_kod of VHDL_komp is
begin
            min_proc: process(A,B)
            begin
                        C <= A and B;
            end process;
end VHDL_kod;
```

AGSTU

Equivalent Functions?? YES

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

ENTITY simp IS
      PORT (
            a, b : IN STD_LOGIC;
            y : OUT STD_LOGIC
      );
END ENTITY simp;

ARCHITECTURE logic OF simp IS
      SIGNAL c : STD_LOGIC;
BEGIN
      c <= a AND b;
      y <= c;
END ARCHITECTURE logic;
```

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

ENTITY simp_prc IS
      PORT (
            a,b : IN STD_LOGIC;
            y : OUT STD_LOGIC
      );
END ENTITY simp_prc;

ARCHITECTURE logic OF simp_prc IS
      SIGNAL c : STD_LOGIC;
BEGIN
      process1: PROCESS (a, b)
      BEGIN
            c <= a AND b;
      END PROCESS process1;
      process2: PROCESS (c)
      BEGIN
            y <= c;
      END PROCESS process2;
END ARCHITECTURE logic;
```

c AND **y** get executed and updated in parallel at the end of the process within one simulation cycle

AGSTU

Equivalent Functions?? NO

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

ENTITY simp IS
      PORT (
ENI
ARCHITECTURE logic OF simp IS
      SIGNAL c : STD_LOGIC;
BEGIN
      c <= a AND b;
      y <= c;
END ARCHITECTURE logic;
```

```
PROCESS (a, b)
c <= a AND b;
PROCESS (c)
y <= c;
```

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

ENTITY simp_prc IS
      PORT (
            a, b : IN STD_LOGIC;
EN
ARCHITECTURE logic OF simp_prc IS
      SIGNAL c: STD_LOGIC;
BEGIN
      PROCESS (a, b)
      BEGIN
            c <= a AND b;
            y <= c;
      END PROCESS;
END ARCHITECTURE logic;
```

New value of **c** not available for **y** until next process execution (requires another simulation cycle or transition on **a**/**b**)

C till next execution

C from last execution

C uppdates

Variable Assignment

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

ENTITY var IS
    PORT    (
        a, b : IN STD_LOGIC;
        y : OUT  STD_LOGIC
    );
END ENTITY var;

ARCHITECTURE logic OF var IS
BEGIN
    PROCESS (a, b)
        VARIABLE  c : STD_LOGIC;
    BEGIN
        c := a AND b;
        y <= c;
    END PROCESS;
END ARCHITECTURE logic;
```

Variable **c** updated immediately and new value is available for assigning to **y**

Variable declaration

Variable assignment

Variable is assigned to a signal to synthesize to a piece of hardware

AGSTU

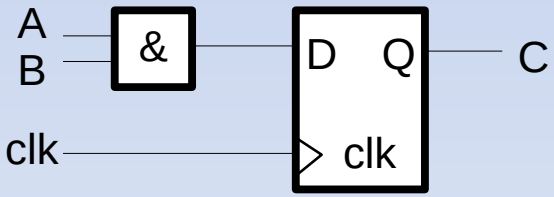
Synkrona processer

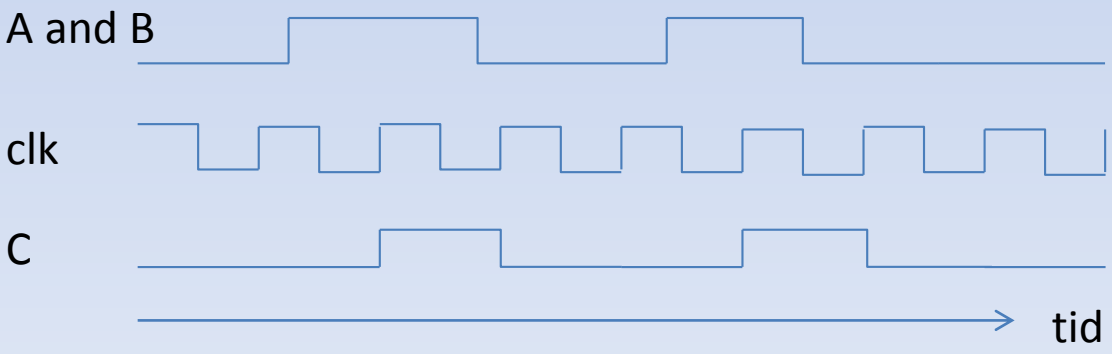
- Synkrona processer är klockade processer,
- Förenklar tidsanalysen av ett system,
- Klockan startar processen. Man kan få processen att starta på negativ, positiv eller på både negativ och positiv flank,
- Önskvärt är att hela systemet använder en av flankerna och helst även samma frekvens. Detta går inte alltid och det kommer att diskuteras senare i kursen.

AGSTU

Synkrona processer
(exempel)

```
architecture VHDL_kod of VHDL_komp is
begin
        min_proc: process(clk)
        begin
        if rising_edge(clk) then
                C <= A and B;
        end if;
        end process;
end VHDL_kod;
```





AGSTU

Klockbeskrivningar i synkrona processer

Det finns flera olika sätt att beskriva klockan i en synkron process.

Alt. 1:

```
process(clk)
  begin
    if clk'event = '1' and clk = '1' then
            --funktion
    end if;
end process;
```

Alt. 2:

```
process(clk)
  begin
    if rising_edge(clk) then
            --funktion
    end if;
end process;
```

Alt. 3:

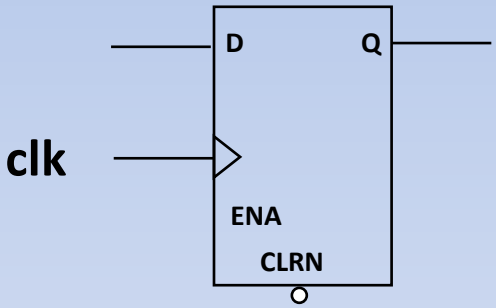
```
process
  begin
    wait until clk = '1';
            --funktion
end process;
```

Alt. 4:

```
process
  begin
    wait until rising_edge(clk);
            --funktion
end process;
```

AGSTU

How Many Registers?



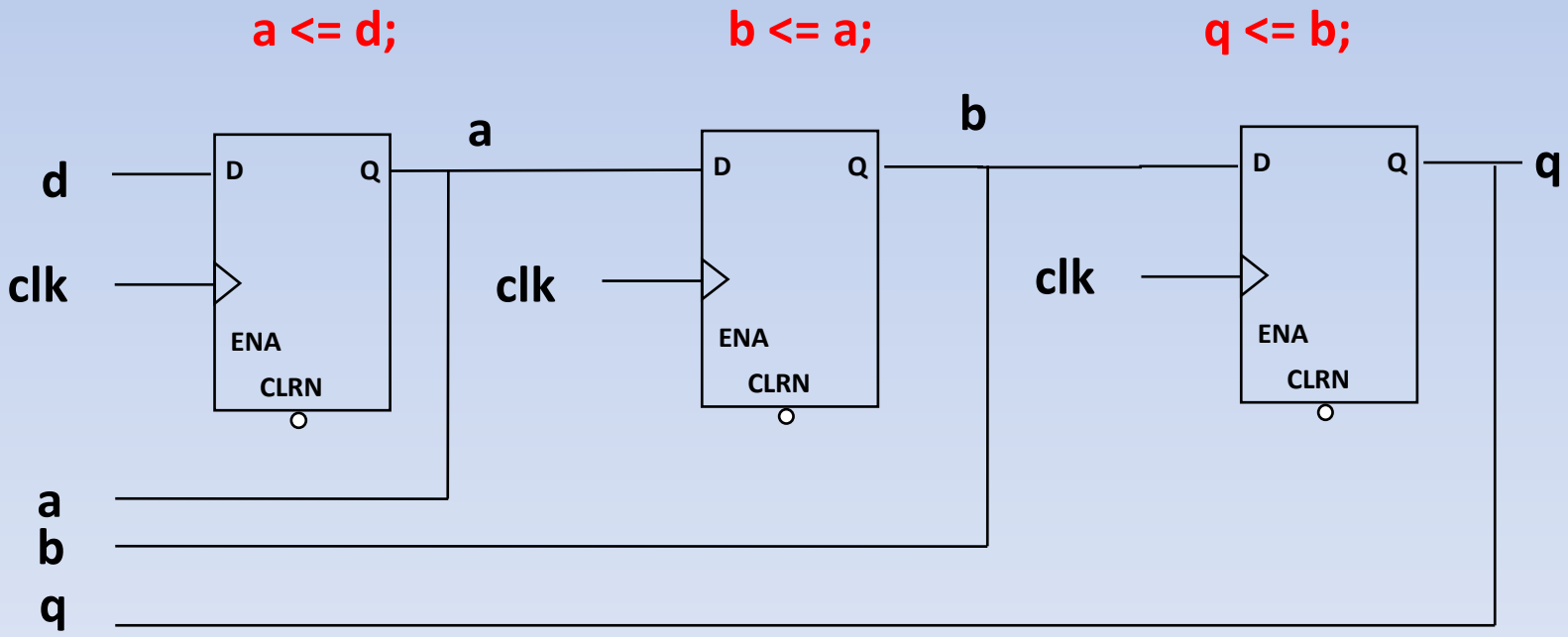

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg1 IS
        PORT (      d       : in STD_LOGIC;
                    clk     : in STD_LOGIC;
                    q       : out STD_LOGIC);
END reg1;

ARCHITECTURE reg1 OF reg1 IS
SIGNAL a, b : STD_LOGIC;
BEGIN
        PROCESS (clk)
        BEGIN
                IF rising_edge(clk) THEN
                        a <= d;
                        b <= a;
                        q <= b;
                END IF;
        END PROCESS;
END reg1;
```

$a(t) <= d(t-t_{clk})$;
$b(t) <= a(t-t_{clk})$;
$q(t) <= b(t-t_{clk})$;

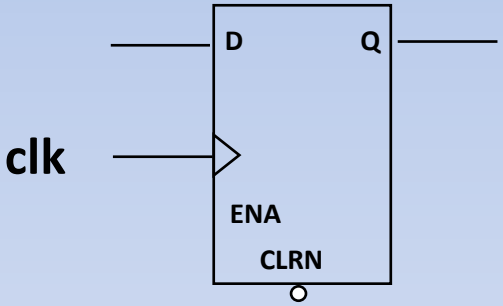
AGSTU

How Many Registers?

- Signal assignments inside the IF-THEN statement that hecks the Clock Condition Infer Registers



How Many Registers?




```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg1 IS
        PORT ( d        : in STD_LOGIC;
                clk     : in STD_LOGIC;
                q       : out STD_LOGIC);
END reg1;
ARCHITECTURE reg1 OF reg1 IS
SIGNAL a, b : STD_LOGIC;
BEGIN
        PROCESS (clk)
        BEGIN
                IF rising_edge(clk)  THEN
                        a <= d;
                        b <= a;
                END IF;
        END PROCESS;
        q <= b;
END reg1;
```

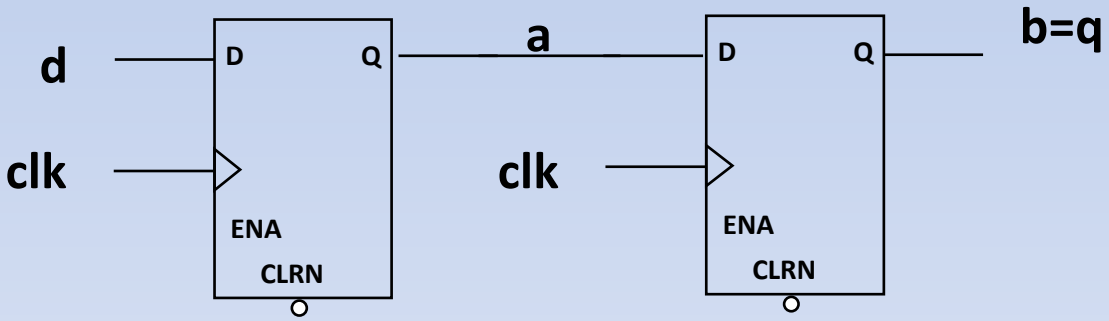
Signal Assignment Moved

$a(t) \le d(t-t_{clk})$;
$b(t) \le a(t-t_{clk})$;
$q(t) \le b(t)$;

AGSTU

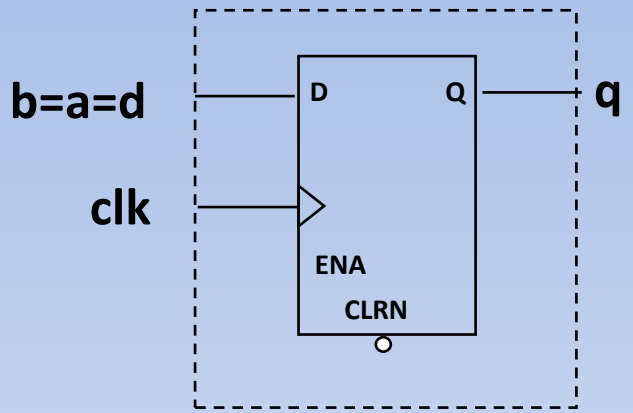
How Many Registers?

- B to Q assignment is no longer edge-sensitive because it is not inside the If-then statement that checks the clock condition



How Many Registers?

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg1 IS
        PORT ( d        : in STD_LOGIC;
               clk      : in STD_LOGIC;
               q        : out STD_LOGIC);
END reg1;

ARCHITECTURE reg1 OF reg1 IS
BEGIN
        PROCESS (clk)
        VARIABLE a, b : STD_LOGIC;
        BEGIN
                IF rising_edge(clk) THEN
                        a := d;
                        b := a;
                        q <= b;
                END IF;
        END PROCESS;
END reg1;
```



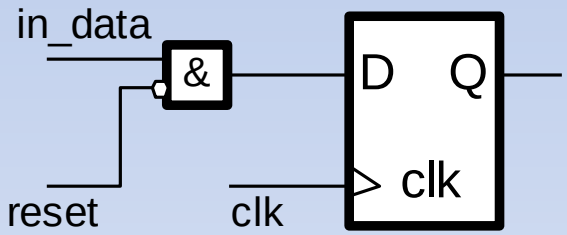
Signals Changed to Variables

$q(t) <= b(t-t_{clk})$;

Synkron reset_n

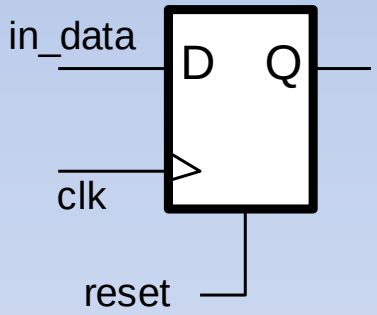
Reset signalen används för att nollställa vipporna i en process.

```
architecture RTL of RTL_Entity is
begin
        min_process: process(clk)
        begin
          if rising_edge(clk) then
           if reset_n = '0' then
                --reset
           else
                --function
           end if;
          end if;
        end process;
end RTL;
```



Asynkron reset

```
architecture RTL of RTL_Entity is
begin
        min_process: process(clk, reset_n)
        begin
          if reset_n = '0' then
                  ……
          elsif  rising_edge(clk) then
                          --funktion
          end if;
        end process;
end V_k;
```

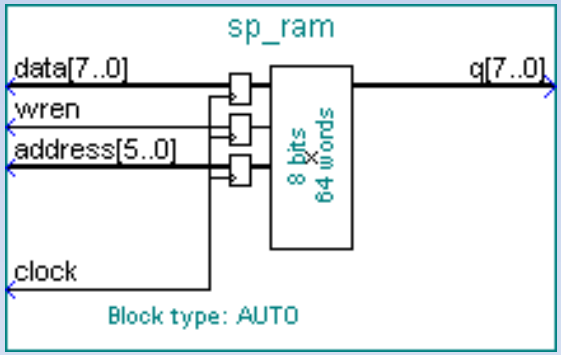


Vi skriver processer på detta sättet

Memory

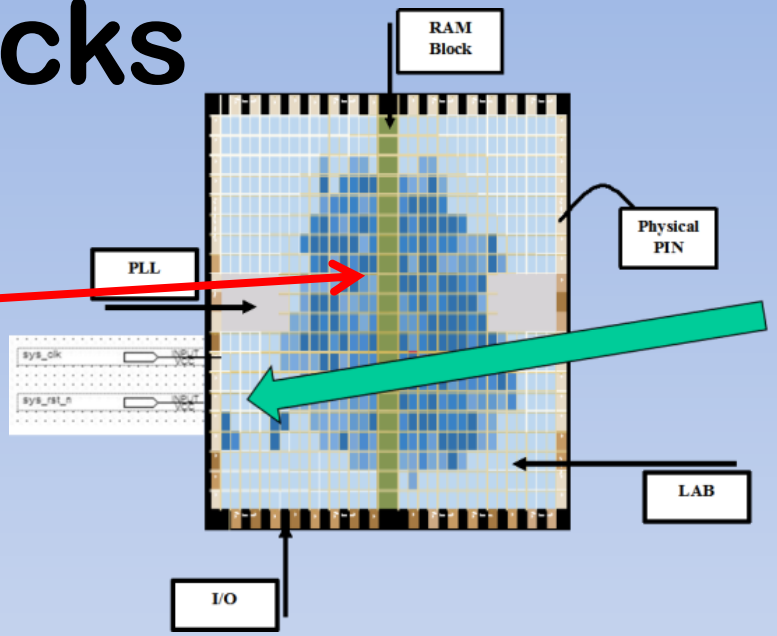
- Synthesis tools have different capabilities for recognizing memories in the FPGA (Block RAM)
  - ELSE register in LE (Expansive)
- Synthesis tools are sensitive to certain coding styles in order to recognize memories
- Must declare an array data type to hold memory values
- **Recommendation: Read Quartus II Handbook, Volume 1, for more information on inferring memories and read during write behavior**

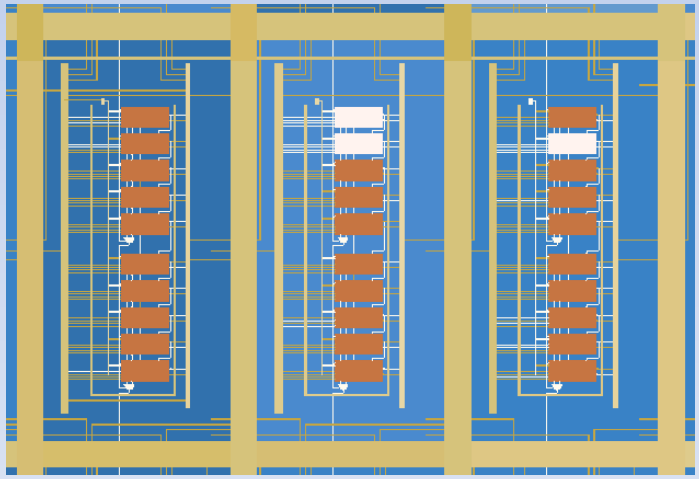
Altera recommends using synchronous memory blocks for Altera designs. Because memory blocks in the newest devices from Altera are synchronous.

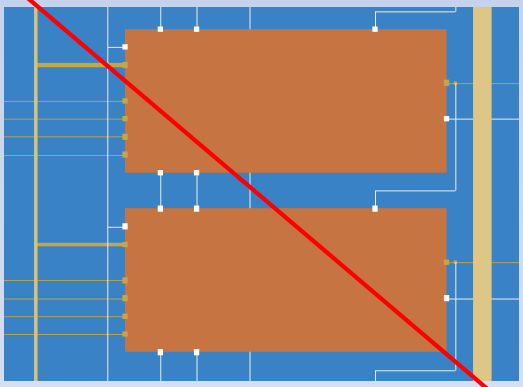


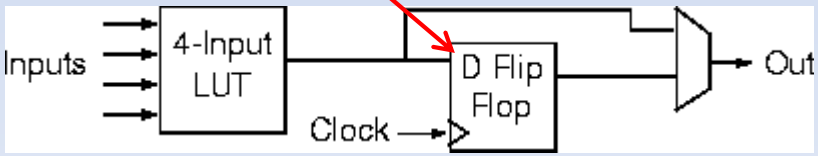
RAM Blocks

- Synthesis tools have different capabilities for recognizing memories in the FPGA (Block RAM)
  - ELSE register in LE (Expansive)









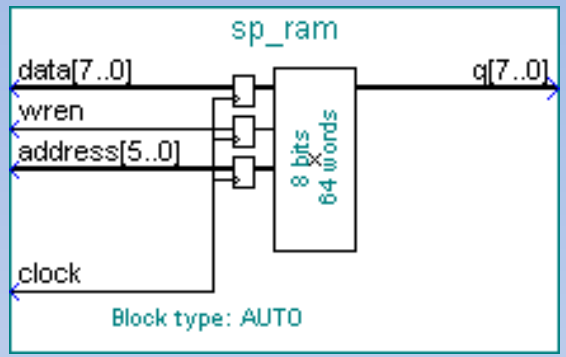
Single-Port Memory (1)

```
ARCHITECTURE logic OF sp_ram IS

TYPE mem_type IS ARRAY (0 TO 63) OF
                    std_logic_vector (7 DOWNTO 0);

SIGNAL mem: mem_type;

BEGIN

PROCESS (clock) BEGIN
    IF rising_edge(clock) THEN
        IF (wren = '1') THEN
            mem(conv_integer(address)) <= data;
        END IF;
    END IF;
END PROCESS;

q <= mem(conv_integer(address));

END ARCHITECTURE logic;
```

- *Code describes a **64 x 8 RAM** with synchronous write & **asynchronous read***
- ***Cannot be implemented in Altera embedded RAM due to asynchronous read***
  - *Uses general logic and registers*
- ***conv_integer** is a function found in the std_logic_unsigned or signed package*
  - *Use TO_INTEGER if using numeric_std package*

Single-Port Memory (2)

```
ARCHITECTURE logic OF sp_ram IS

TYPE mem_type IS ARRAY (0 TO 63) OF
                    std_logic_vector (7 DOWNTO 0);

SIGNAL mem: mem_type;

BEGIN

PROCESS (clock) BEGIN
      IF rising_edge(clock) THEN
            IF (wren = '1') THEN
                  mem(conv_integer(address)) <= data;
            END IF;
            q <= mem(conv_integer(address));
      END IF;
END PROCESS;

END ARCHITECTURE logic;
```

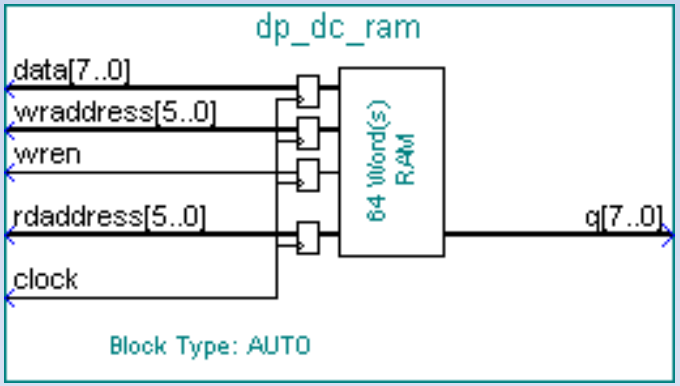


- *Code describes a **64 x 8 RAM** with synchronous write & synchronous read*
- *Old data read-during-write behaviour*
  - *Memory read in same process/cycle as memory write*
  - *Check target architecture for support as unsupported features built using LUTs/registers*

Simple Dual-Port, Single-Clock Memory

```
ARCHITECTURE logic OF sdp_ram IS

TYPE mem_type IS ARRAY (63 DOWNTO 0) OF
                        std_logic_vector (7 DOWNTO 0);

SIGNAL mem: mem_type;

BEGIN

    PROCESS (clock) BEGIN
        IF rising_edge(clock) THEN
            IF (wren = '1') THEN
                mem(conv_integer(wraddress)) <= data;
            END IF;
            q <= mem(conv_integer(rdaddress));
        END IF;
    END PROCESS;

END ARCHITECTURE logic;
```

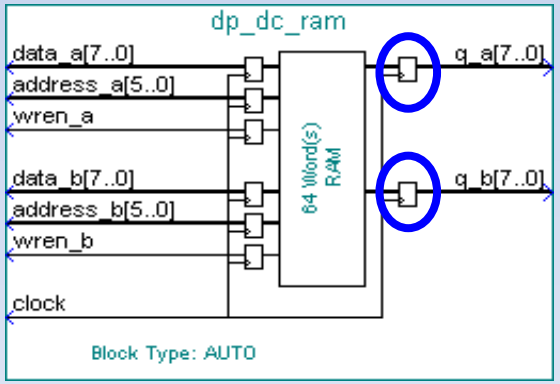
- *Code describes a **simple dual-port** (separate read & write addresses) 64 x 8 RAM with single clock*
- *Code implies old data read-during-write behaviour*
  - *New data support in simple dual-port requires additional RAM bypass logic*



True Dual-Port, Dual-Clock Memory

```
ARCHITECTURE logic OF dp_dc_ram IS
    TYPE mem_type IS ARRAY (63 DOWNTO 0) OF
                        std_logic_vector (7 DOWNTO 0);
    SIGNAL mem: mem_type;
    SIGNAL addr_reg_a, addr_reg_b :
                        std_logic_vector (7 DOWNTO 0);
BEGIN
    PROCESS (clock_a) BEGIN
        IF rising_edge(clock_a) THEN
            IF (wren_a = '1') THEN
                mem(conv_integer(address_a)) <= data_a;
            END IF;
            addr_reg_a <= address_a;
        END IF;
        q_a <= mem(conv_integer(addr_reg_a));
    END PROCESS;

    PROCESS (clock_b) BEGIN
        IF rising_edge(clock_b) THEN
            IF (wren_b = '1') THEN
                mem(conv_integer(address_b)) <= data_b;
            END IF;
            addr_reg_b <= address_b;
        END IF;
        q_b <= mem(conv_integer(addr_reg_b));
    END PROCESS;
END ARCHITECTURE logic;
```

- *Code describes a **true dual-port** (two individual addresses) 64 x 8 RAM*
- *May not be supported in all synthesis tools*
- *New data same-port read-during-write behaviour shown*
  - *Mixed port behaviour undefined with multiple clocks*



Initializing Memory Contents Using Files

```
ARCHITECTURE logic OF sp_ram IS

TYPE mem_type IS ARRAY (0 TO 63) OF
std_logic_vector (7 DOWNTO 0);

SIGNAL mem: mem_type;
ATTRIBUTE ram_init_file : STRING;
ATTRIBUTE ram_init_file OF mem : SIGNAL IS
            "init_file_name.hex";

BEGIN

PROCESS (clock) BEGIN
      IF rising_edge(clock) THEN
            IF (we = '1') THEN
                  mem(conv_integer(address)) <= data;
            END IF;
            q <= mem(conv_integer(address));
      END IF;
END PROCESS;

END ARCHITECTURE logic;
```

- *Use VHDL attribute to assign initial contents to inferred memory*
- *Store initialization data as **.HEX** or **.MIF***
- *Contents of initialization file downloaded into FPGA during configuration*

Unsupported Control Signals

- e.g. Clearing RAM contents with reset

```
BEGIN

        PROCESS (clock, reset)
        BEGIN
                IF reset = '1' THEN
                        mem(conv_integer(address)) <=
                                                (OTHERS => '0');
                ELSIF rising_edge(clock) THEN
                        IF (we = '1') THEN
                                mem(conv_integer(address)) <= data;
                        END IF;
                END IF;
        END PROCESS;

        q <= mem(conv_integer(address));

END ARCHITECTURE logic;
```

- ***Memory content cannot be cleared with reset***
- *Recommendations*
  1. *Avoid reset checking in RAM read or write processes*

Inferred ROM (Constant)

```
ARCHITECTURE logic OF rom16x7 IS
TYPE rom_type IS ARRAY (0 TO 15) OF
        STD_LOGIC_VECTOR (6 DOWNTO 0);
CONSTANT rom : rom_type := (
                        "0111111",
                        "0011000",
                        "1101101",
                        "1111100",
                        "1011010",
                        "1110110",
                        "1110111",
                        "0011100",
                        "1111111",
                        "1111110",
                        "1011111",
                        "1110011",
                        OTHERS => "0000000"
                         );
```

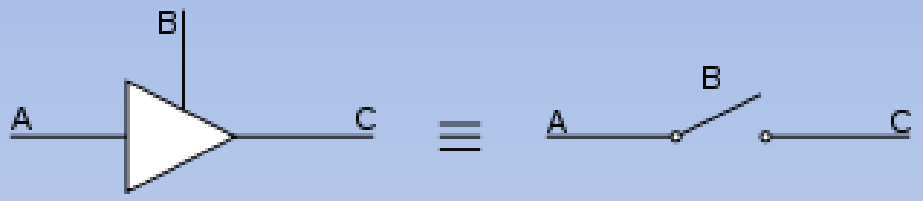
```
BEGIN
PROCESS (clock)
BEGIN
    IF rising_edge (clock) THEN
        qa <= rom(CONV_INTEGER(addr_a));
        qb <= rom(CONV_INTEGER(addr_b));
    END IF;
END PROCESS;
END ARCHITECTURE logic;
```

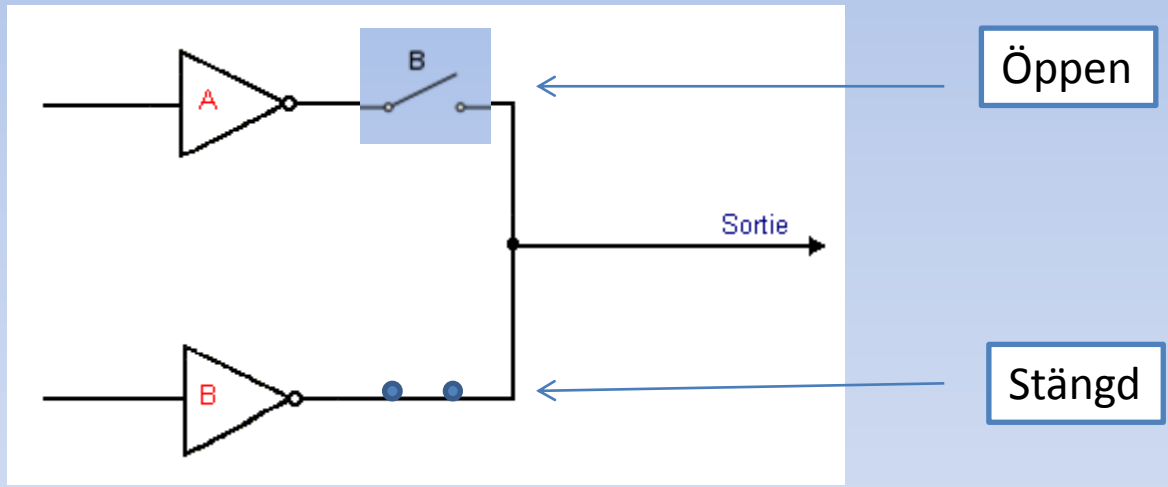
- *Needs 1 constant value for each ROM address*
- *Example shows dual-port access*
- *May place type & constant declaration in package for re-use*
- *Alternate: Create and use initialization function routine (see RAM example)*

Three-state

Tabelle komplett

A	B	C
0	0	0
1	0	1
0	1	Z
1	1	Z

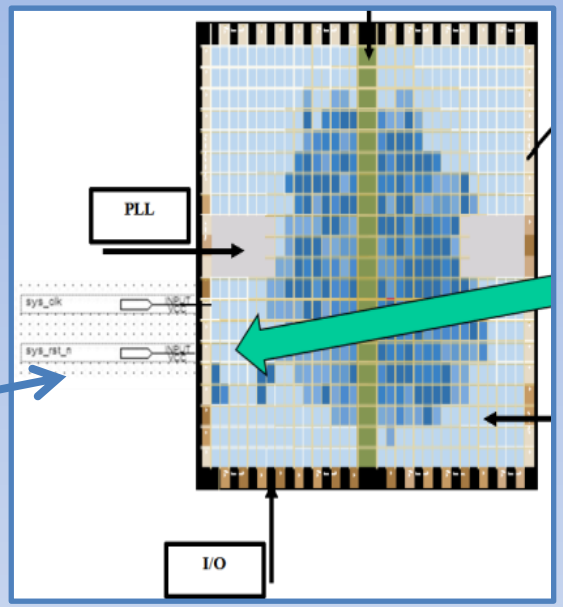




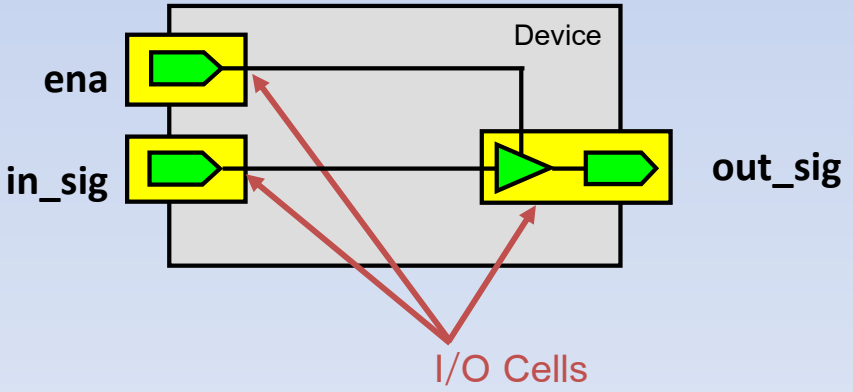
- http://en.wikipedia.org/wiki/Three-state_logic

Three-state

- IEEE defines ‘Z’ value in STD_LOGIC package
  - Simulation: Behaves like high-impedance state
  - Synthesis: Converted to three-state buffers
- Altera devices have three-state buffers only in I/O cells




```
ARCHITECTURE behavior OF tri1 IS
BEGIN
        out_sig <= in_sig WHEN ena = ‘1’ ELSE ‘Z’;
END ARCHITECTURE behavior;
```



AGSTU

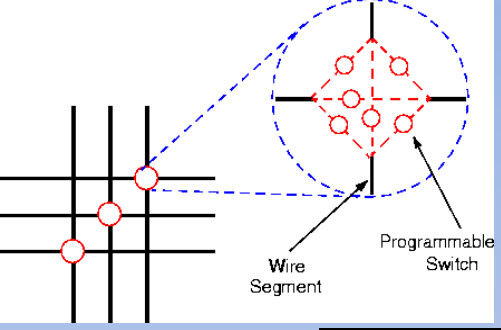
RAM_ROM uppgift

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

entity demo_3a is
                port
                (
                clk             : in std_logic;                 -- CLOCK_50
                -- RAM
                addr_ram        : in std_logic_vector(3 downto 0);
                data_ram        : in std_logic_vector(3 downto 0);
                we_ram          : in std_logic;
                q_ram           : out std_logic_vector(3 downto 0);
                -- ROM
                addr_rom        : in std_logic_vector(2 downto 0);
                q_rom           : out std_logic_vector(2 downto 0) );
end entity;

ARCHITECTURE Block_RAM_ROM_FPGA OF demo_3a IS
--------------------RAM --------------------
TYPE mem_type IS ARRAY (0 TO 15) OF std_logic_vector (3 DOWNTO 0);
SIGNAL RAM_mem: mem_type;

---------- ROM ----------------------
TYPE rom_type IS ARRAY (0 TO 3**2 - 1) OF STD_LOGIC_VECTOR (2 DOWNTO 0);
CONSTANT ROM_mem : rom_type := (
"111", -- adr 0
"000",
"101",
"100",
"010",
"110",
"100", -- adr 6
OTHERS => "000");
```

```
---------- RAM ----------------------------------
BEGIN
RAM: PROCESS (clk)

BEGIN
IF rising_edge(clk) THEN
  IF (we_ram = '1') THEN
    RAM_mem(conv_integer(addr_ram)) <= data_ram;
  END IF;
q_ram <= RAM_mem(conv_integer(addr_ram));
END IF;

END PROCESS;

---------------------------------- ROM ----------------------
ROM: PROCESS (clk)

BEGIN
IF rising_edge (clk) THEN
 q_rom <= rom_mem(conv_integer(addr_rom));
END IF;

END PROCESS;

END ARCHITECTURE Block_RAM_ROM_FPGA;
```

AGSTU

FPGA chip



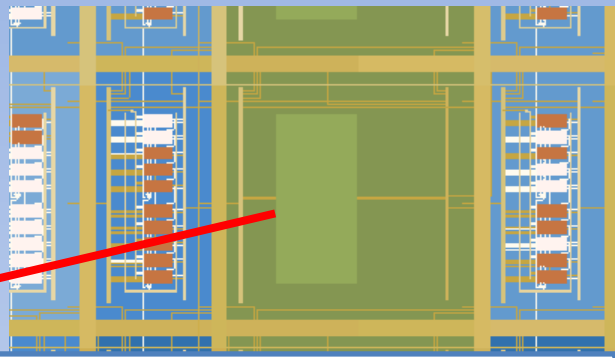
RAM Block

PLL

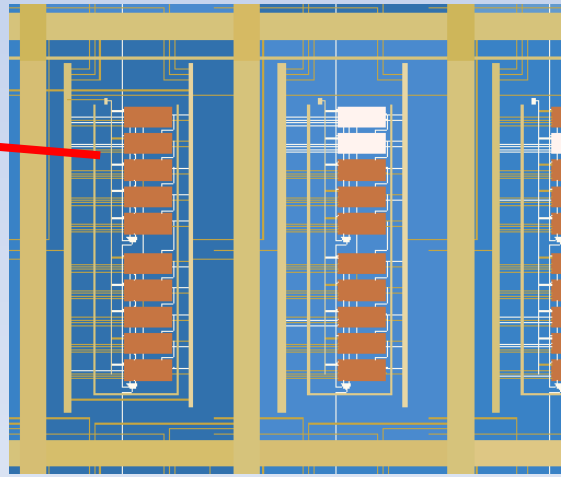
Physical PIN

LAB

I/O



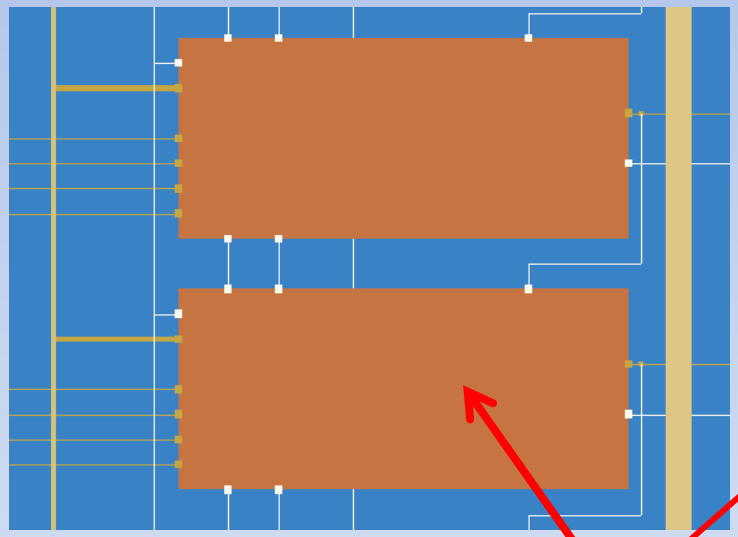
```
BEGIN
IF rising_edge(clk) THEN
  IF (we_ram = '1') THEN
    RAM_mem(conv_integer(addr_ram)) <= data_ram;
  END IF;
q_ram <= RAM_mem(conv_integer(addr_ram));
END IF;
```

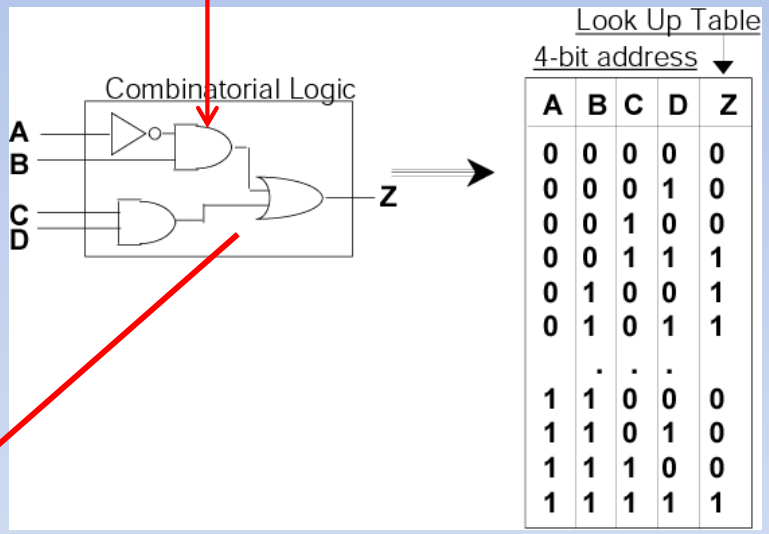


AGSTU

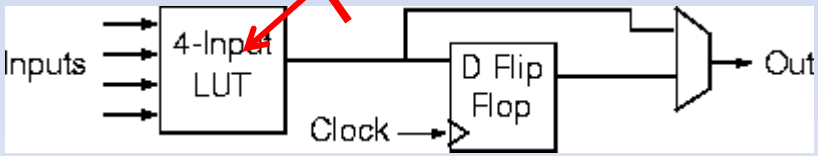
The Logic Elements

```
BEGIN
IF rising_edge (clk) THEN
 q_rom <= rom_mem(conv_integer(addr_rom));
END IF;
```





A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



AGSTU

SLUT

The language is very rich and it takes some time to be able to deal with its different types ..but i love it!

AGSTU
Arbete Genom STUdier
Utbildning

All rights reserved and Disclaim

- **All rights reserved.** No part of this document (PPT, Doc, film etc.) may be reproduced, in any form or by any means, without permission in writing from the publisher. Unless otherwise specified, all information (including software, designs and files) provided are copyrighted by AGSTU AB.
- **Disclaim**
  All the information (including hardware, software, designs, text and files) are provided "as is" and without any warranties expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. In no event should the author be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use or inability to use information (including text, software, designs and files) provided in this document.