HOGESCHOOL ROTTERDAM

Hardware Programming HWP01 2020-2021

capturing a

FPGA

design with

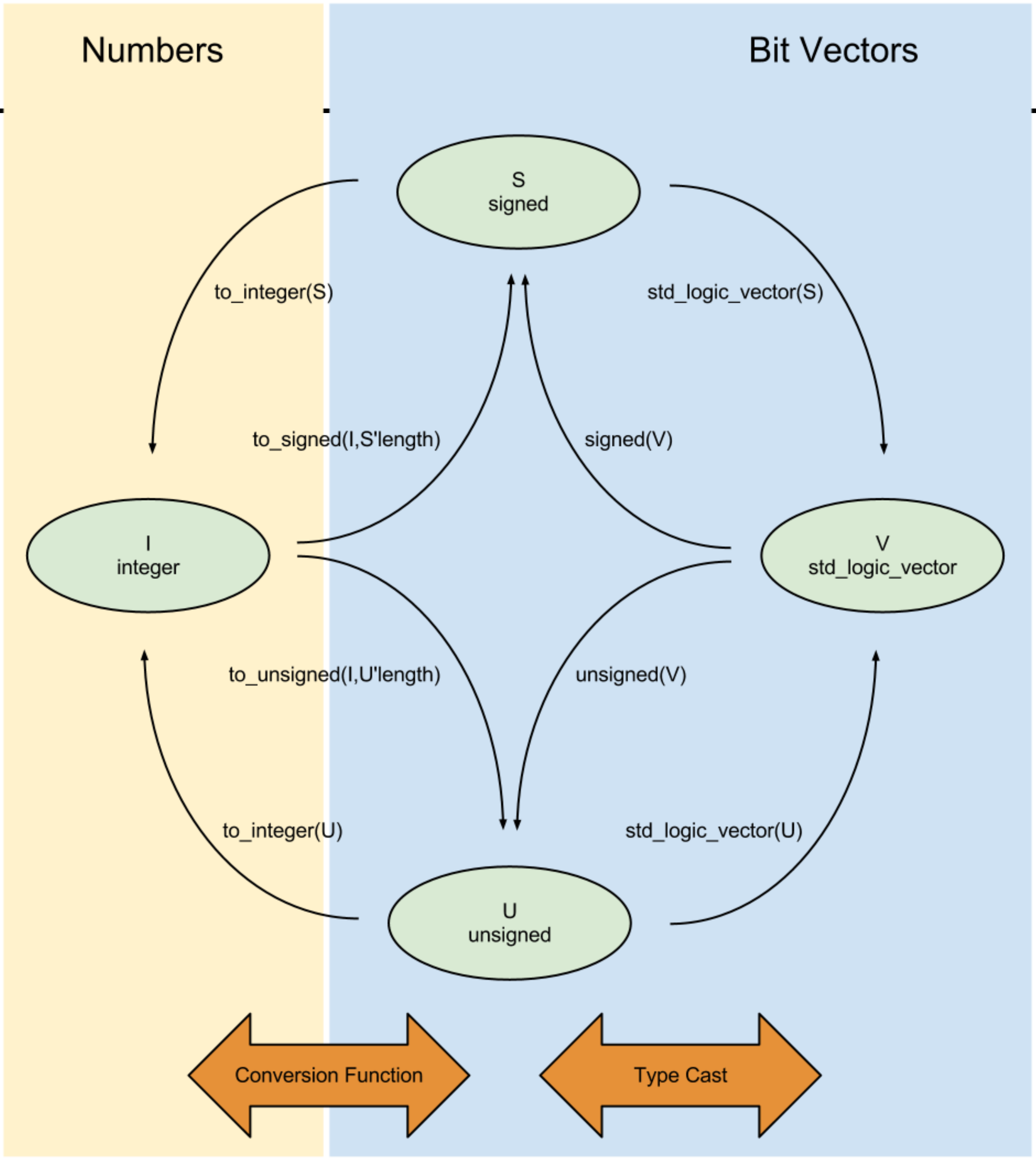
VHDL

Planning: theory

- First week
  - Introduction digital systems
  - Structured digital Design
  - RTL
- Second week
  - Introduction VHDL
  - Code structure and data types
  - Design verification
- **Third week**
  - Combinational versus sequential design
  - Concurrent and sequential code
  - Signals and variables
- Fourth week
  - Introduction to state machines
- Fifth week
  - Designing state machines
  - Advanced VHDL design

Agenda

- **Discussion of previous week**
- Combinational versus sequential design
- Concurrent and sequential code
- Signals versus variables



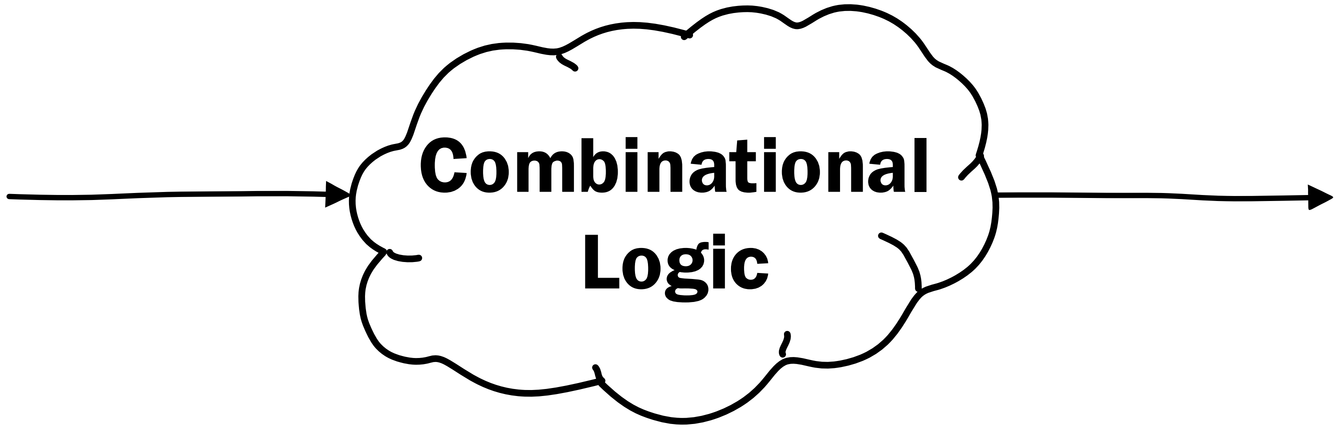
Zie ook:
TABEL:
H.3 ; blz 76

Agenda

- Discussion of previous week
- **Combinational versus sequential design**
- Concurrent and sequential code
- Signals versus variables

Combinational and sequential logic (1/2)

- Combinational: **no** memory

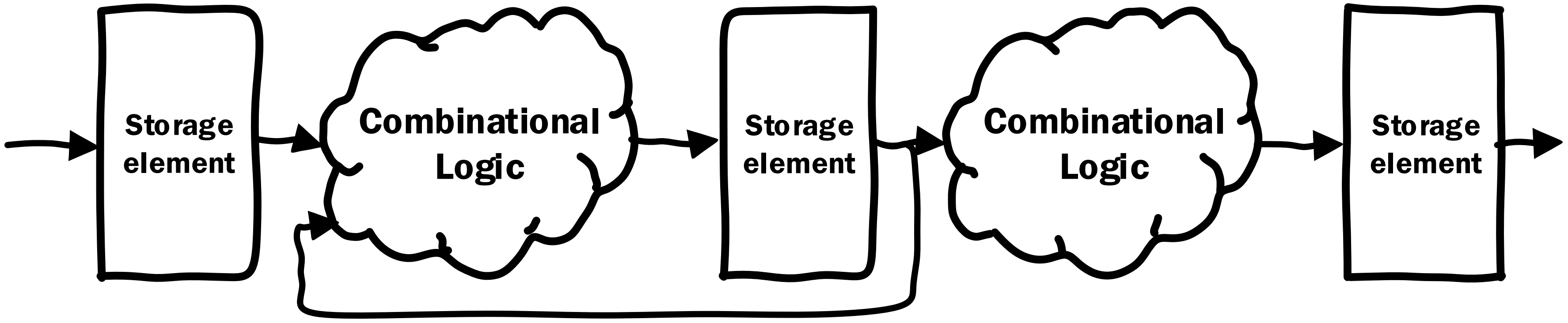


- The output is a function only of the current inputs.

- In any implementation there is a ***propagation delay***
  - For this course, we pretend the propagation delay is zero (except for one assignment).

Combinational and sequential logic (2/2)

- Sequential logic:



- The output of sequential logic, is ...

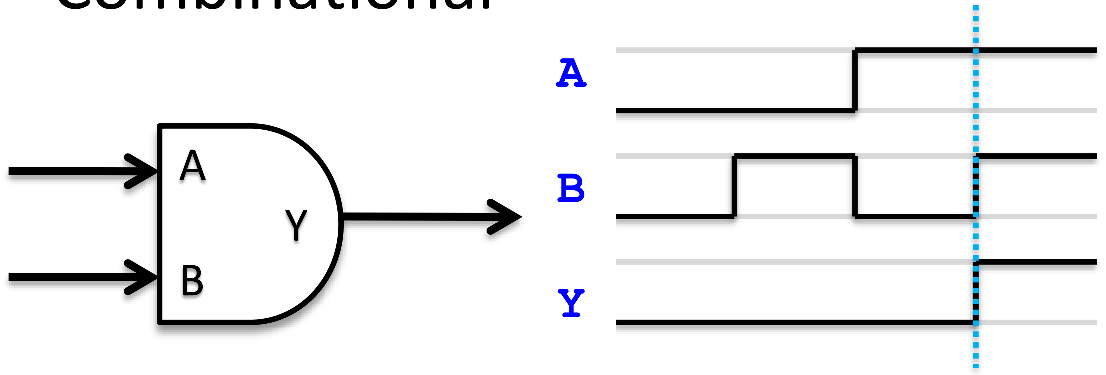
... a function of a *sequence* of operations ..

... on current and/or *previous inputs*.

- Advantages?

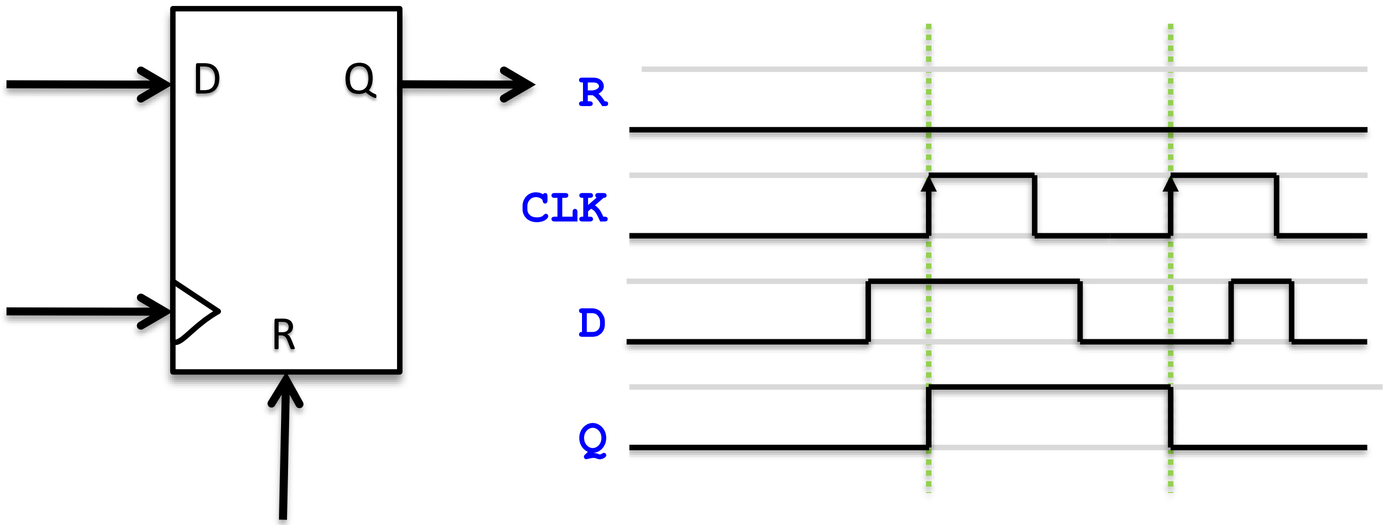
Combinational vs. sequential logic example

- Combinational



Updates instantly as input changes (in theory).

- Sequential



Updates only when CLK goes high. Until then it remembers the previous input (memory)

Combinational or sequential?

- Multiplexer vs. Gated-Multiplexer?
- Timer?
- Encoder (for example binary to 7-seg display)?
- Comparators?
- Adder?
- Multiplier?
- ALU?
- CPU?
- RAM?

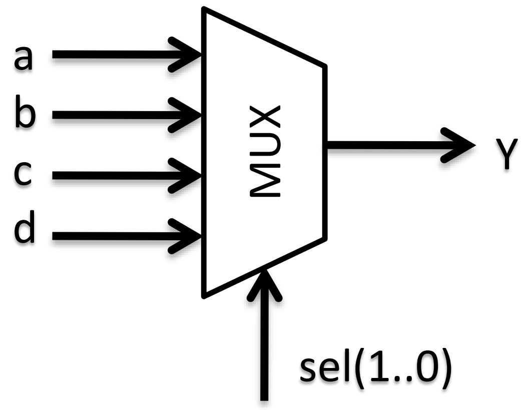
Agenda

- Discussion of previous week
- Combinational versus sequential design
- **Concurrent and sequential code**
- Signals versus variables

Concurrent Code

- Concurrent code is intended **only for combinational circuits**.
  - Often used for structural descriptions of a circuit.
- Outputs activated asynchronously, at any time.
- Statements for concurrent code:
  - **WHEN ... ELSE**
  - **WITH ... SELECT**
  - **GENERATE**
- Can be placed outside **PROCESS**, **FUNCTION** and **PROCEDURE**

WHEN/ELSE

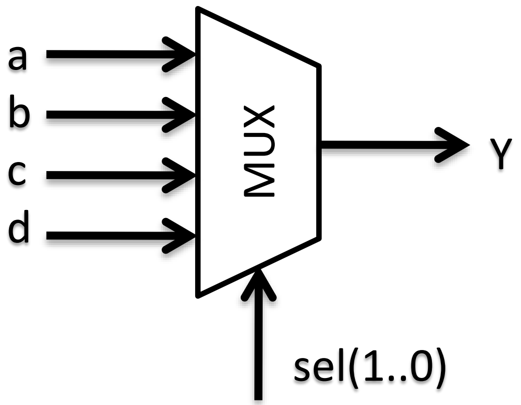


```
ARCHITECTURE mux1 OF mux IS
BEGIN
    y <= a WHEN sel="00" ELSE
         b WHEN sel="01" ELSE
         c WHEN sel="10" ELSE
         d;
END mux1;
```

- Read: when sel is equal to “00”, y obtains the value of a, else when sel is equal to “01”, y obtains the value of b, etc…
- Cover all combinations
- Let the synthesizer do the K-map for us!

WITH/SELECT

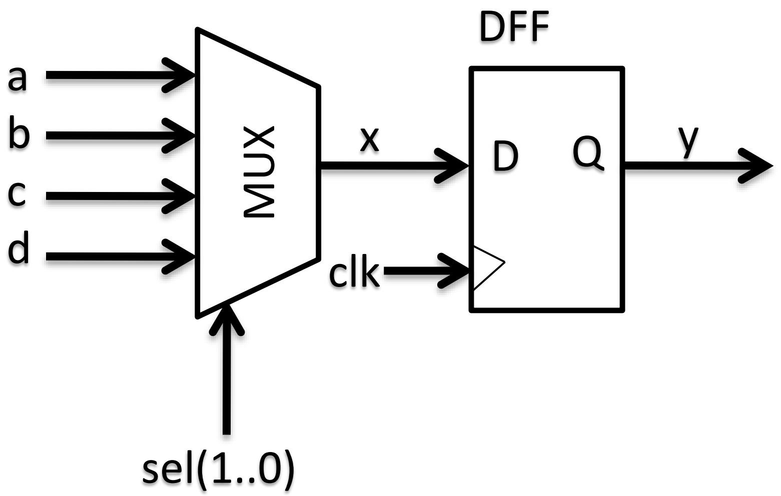
- **WITH** ... **SELECT** is used very often
- Read: depending on sel, y becomes a when sel is 00, y becomes b when sel is 01, etc…
- Note the usage of the **OTHERS** keyword here to cover *all* possibilities



```
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
   y <= a WHEN "00",    -- Use "," not ";"
        b WHEN "01",
        c WHEN "10",
        d WHEN OTHERS;
END mux2;
```

Gated Multiplexer

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY gated_mux IS
   PORT (a, b, c, d, clk: IN  STD_LOGIC;
         sel        : IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
         x, y       : OUT STD_LOGIC);
END gated_mux;
--------------------------------------------
ARCHITECTURE behavioral OF gated_mux IS
BEGIN
      WITH sel SELECT
            x <= a WHEN "00",    -- Use "," not ";"
            b WHEN "01",
            c WHEN "10",
            d WHEN OTHERS;
PROCESS (clk)
BEGIN
      IF rising_edge(clk) THEN
      y <= x;
END PROCESS
END ARCHITECTURE
```



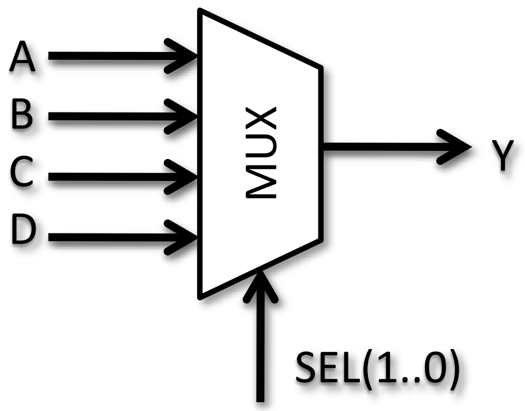
Sequential Code

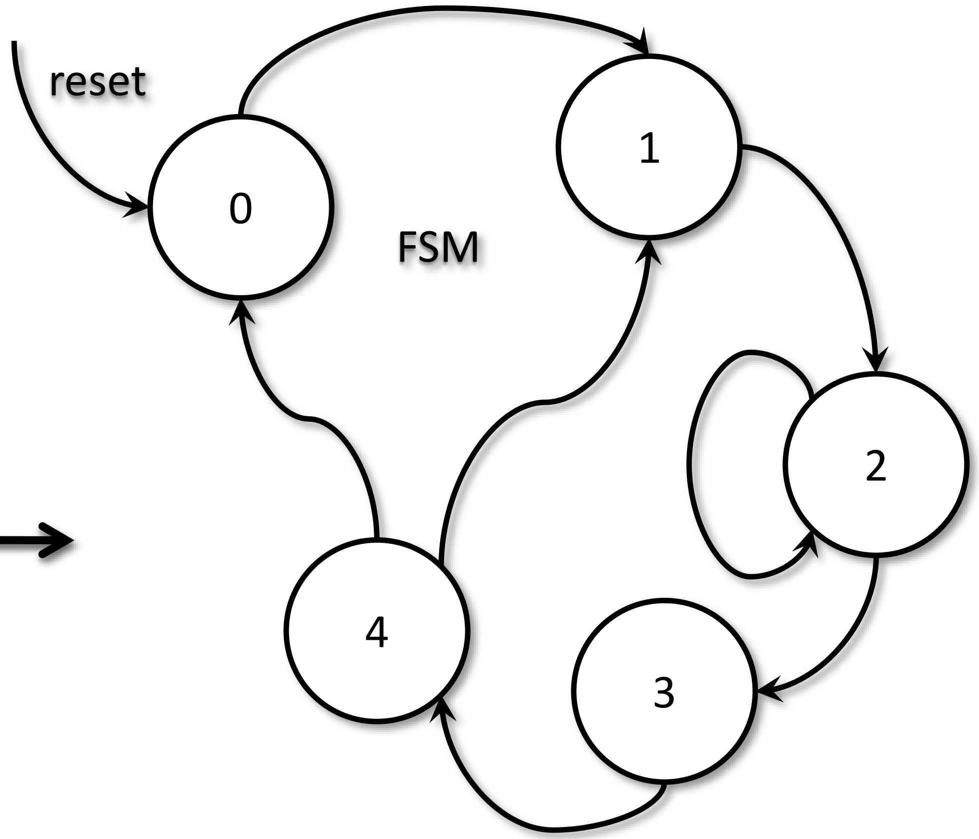
- Statements for sequential code:
  - **IF**
  - **WAIT**
  - **FOR** x **IN** 0 **TO** 10 **LOOP**
  - **CASE**

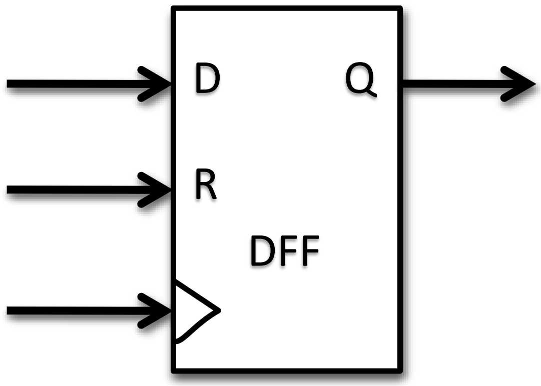
- Sequential code can be used to design both **sequential** ***and*** **combinational** circuits

- Code within a **PROCESS**, **FUNCTION** or **PROCEDURE** is sequential.

Sequential Code Circuit Examples







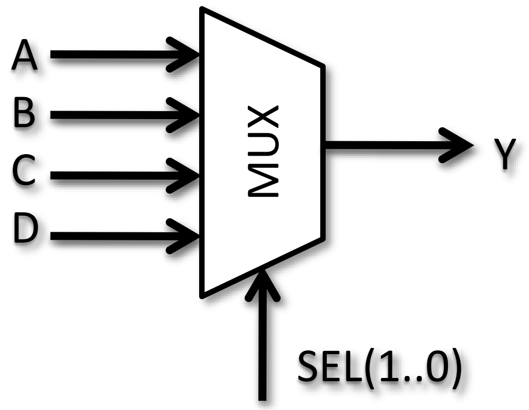
Sequential Code

- **PROCESS** is a *sequential* section, located in the **ARCHITECTURE**

- Note that multiple processes are allowed. They are *concurrent* to each other.

- *ONLY* support for the following *sequential* statements:
  - **IF**
  - **WAIT**
  - **LOOP**
  - **CASE**

Multiplexer with sequential code



- A **PROCESS** has a **sensitivity list**
- The outputs of the **PROCESS** get updated if the value of an object in the list changes

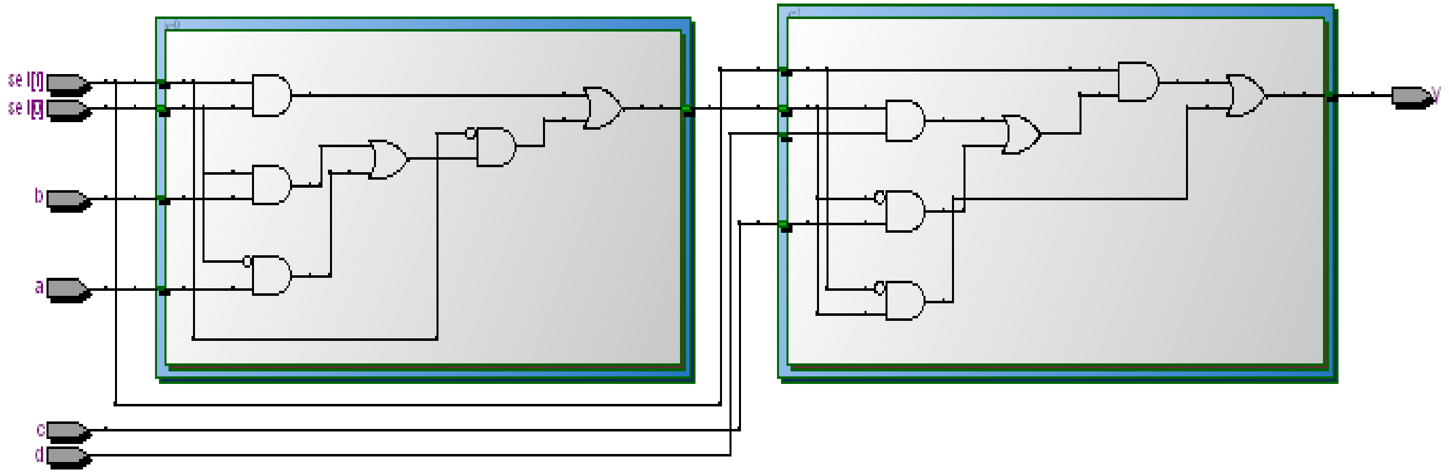
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY seq_code_mux IS
   PORT (a, b, c, d: IN  STD_LOGIC;
         sel       : IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
         y         : OUT STD_LOGIC);
END seq_code_mux;
---------------------------------------------
ARCHITECTURE seq_code_impl OF seq_code_mux IS
BEGIN
         PROCESS(sel,a,b,c,d)
         BEGIN
                  IF    sel = "00" THEN
                      y <= a;
                  ELSIF sel = "01" THEN
                      y <= b;
                  ELSIF sel = "10" THEN
                      y <= c;
                  ELSE
                      y <= d;
                  END IF;
         END PROCESS;
END seq_code_impl;
```

Multiplexer with CASE

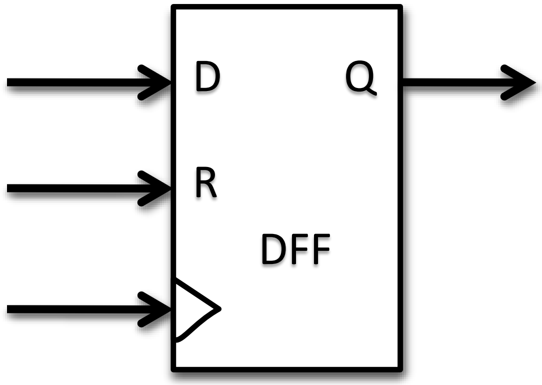
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY seq_code_mux IS
   PORT (a, b, c, d: IN  STD_LOGIC;
         sel       : IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
         y         : OUT STD_LOGIC);
END seq_code_mux;
---------------------------------------------
ARCHITECTURE seq_code_impl OF seq_code_mux IS
BEGIN
         PROCESS(sel,a,b,c,d)
         BEGIN
                  CASE sel IS
                           WHEN "00" =>
                                    y <= a;
                           WHEN "01" =>
                                    y <= b;
                           WHEN "10" =>
                                    y <= c;
                           WHEN OTHERS =>
                                    y <= d;
                  END CASE;
         END PROCESS;
END seq_code_impl;
```

Tech. map of mux with sequential code

Sequential Code can make Combinational Circuits



D Flip-Flop

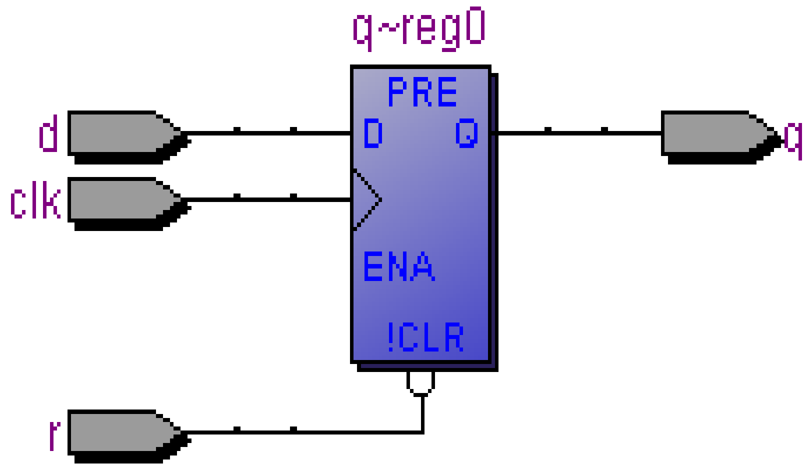


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------
ENTITY dff_example IS
   PORT (
            clk: IN  STD_LOGIC;
            d,r: IN  STD_LOGIC;
            q  : OUT STD_LOGIC
         );
END dff_example;
------------------------------------------
ARCHITECTURE dff_implementation OF dff_example IS
BEGIN
    PROCESS(clk, r )
    BEGIN
        IF r = '1' THEN
            q <= '0';
        ELSIF RISING_EDGE(clk) THEN
            q <= d;
        END IF;
    END PROCESS;
END dff_implementation;
```

We only want to update Q when the CLK goes high, so only the CLK & R are in the sensitivity list

Technology Map of DFF

```
Sequential circuits can only be written with
sequential code
```

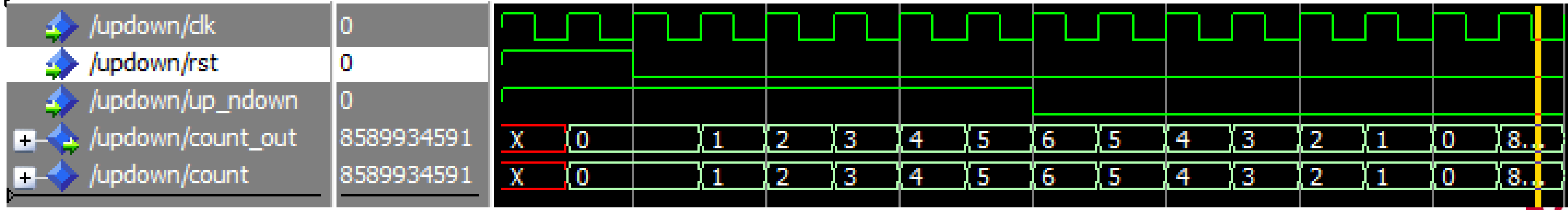


Up/down counter

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY updown IS
         GENERIC (
                  COUNTER_WIDTH : INTEGER := 31
         );

         PORT (
                  clk                 : IN STD_LOGIC;
                  rst                 : IN STD_LOGIC;

                  up_ndown  : IN STD_LOGIC;

                  count_out : OUT UNSIGNED(COUNTER_WIDTH DOWNTO 0)
         );
END updown;
```

Up/down counter

```
ARCHITECTURE arch OF updown IS
         SIGNAL count : UNSIGNED(COUNTER_WIDTH DOWNTO 0);
BEGIN
         PROCESS(clk)
         BEGIN
                  IF rising_edge(clk) THEN
                           IF rst = '1' THEN
                                    count <= (OTHERS => '0');
                           ELSE
                                    IF up_ndown = '1' THEN
                                             count <= count + 1;
                                    ELSE
                                             count <= count - 1;
                                    END IF;
                           END IF;
                  END IF;
         END PROCESS;

         count_out <= count;

END arch;
```



/updown/clk	0	
/updown/rst	0	
/updown/up_ndown	0	
/updown/count_out	8589934591	X 0 1 2 3 4 5 6 5 4 3 2 1 0 8..
/updown/count	8589934591	X 0 1 2 3 4 5 6 5 4 3 2 1 0 8..

WAIT

- In a **PROCESS** you can use the **WAIT** keyword

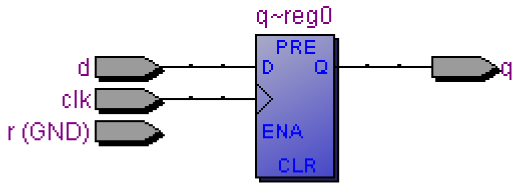
- A process CANNOT have both a sensitivity list and **WAIT** statements

- Three types:
  - **WAIT UNTIL** <signal>
  - **WAIT ON** sig1, sig2, ..., sign
  - **WAIT FOR** *time*

- **WAIT FOR** cannot be synthesized; only for simulation and test benches

D Flip-Flop with WAIT

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY dff_example IS
   PORT (
            clk: IN  STD_LOGIC;
            d,r: IN  STD_LOGIC;
         q   : OUT STD_LOGIC
            );
END dff_example;
---------------------------------------------
ARCHITECTURE dff_implementation OF dff_example IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL RISING_EDGE(clk);
          q <= d;
    END PROCESS;
END dff_implementation;
```



Not very handy without a reset ☹

Loops in VHDL

- **FOR** and **WHILE** loops
- **FOR** ... **LOOP** repeats until the upperbound is reached
- Static bounds; use constants to specify the upper bound of your loop

```
PROCESS(sel)
  ..
  BEGIN
  FOR i IN 0 to 5 LOOP
     x <= i;
  END LOOP;
END PROCESS;
```

Loops in VHDL

- **WHILE** ... **LOOP** is similar in structure as the **FOR** ... **LOOP**

- See page 161 of your book for some examples with loops

```
PROCESS(sel)
  ..
  BEGIN
  WHILE(i<10) LOOP
   ...do something...
  END LOOP;
END PROCESS;
```

Agenda

- Discussion of previous week
- Combinational versus sequential design
- Concurrent and sequential code
- **Signals versus variables**

Signals versus variables

- **SIGNAL** properties:
  - Can *ONLY* be declared outside a **PROCESS** but can be used within a **PROCESS**
  - Within sequential code the signal is not ‘updated immediately’ (at the end of the **PROCESS**)
  - Only a *single* assignment is allowed to a signal in the whole code (multiple assignments in **PROCESSES** are fine, but only the last one will be effective!)
- **VARIABLE** properties:
  - Can *ONLY* be declared inside a **PROCESS**
  - Is ‘updated immediately’ and can be used in the next line of code
  - *Multiple* assignments are not a problem

Signals and Variables in VHDL (1/2)

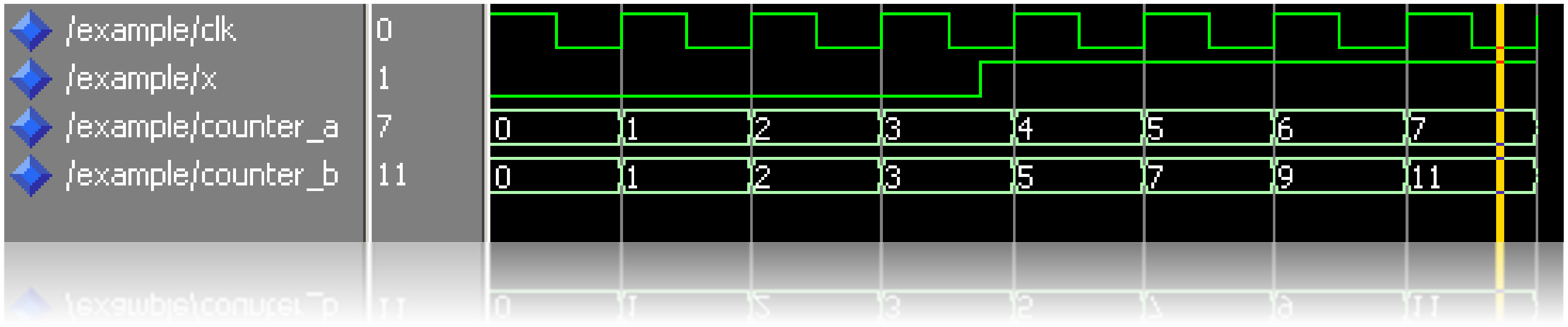
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity example is
        port (
                        clk:    in      std_logic;
                        x:      in      std_logic;

                        counter_a:      out integer range 0 to 15;
                        counter_b:      out integer range 0 to 15;
                );
end example;
```

Signals and Variables in VHDL (2/2)

```
architecture example of example is
        signal a: integer range 0 to 15 := 0;
begin
        process (clk)
                variable b: integer range 0 to 15 := 0;
        begin
                if rising_edge(clk) then
                        a <= a + 1;
                        b := b + 1;
                        if x = '1' then
                                a <= a + 1;
                                b := b + 1;
                        end if;
                end if;
                counter_b <= b;
        end process;
        counter_a <= a;
end example;
```

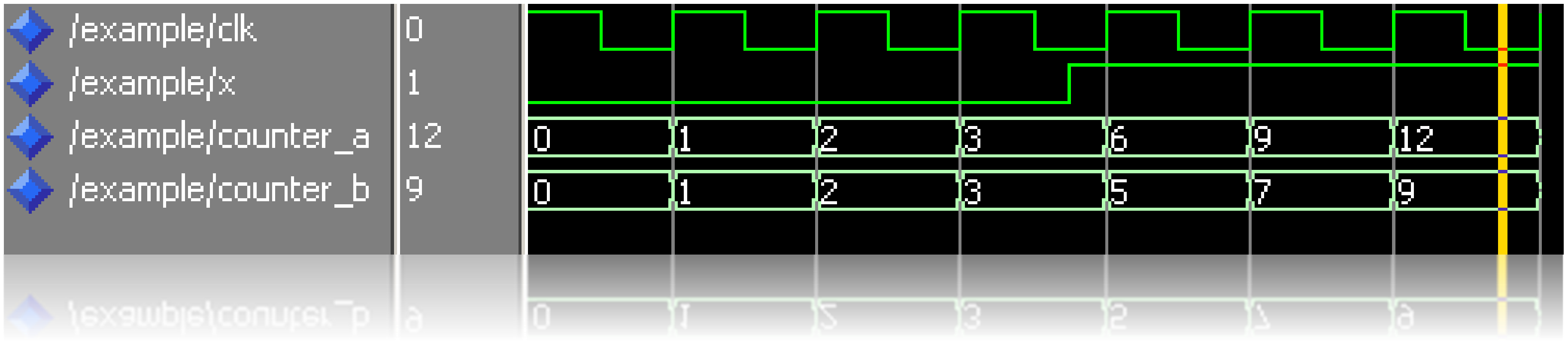
Variable can only be used inside the process!

Example



- When x is high
  - The variable counter is counted up twice
  - The signal counter is counted up only once
    - Only the last assignment is effective for a signal

Last signal assignment is only used



- Proof:
  - change a <= a + 1
  - to a <= a + 3
- Conclusion
  - Variables are useful when you have to do multiple assignments inside a process

```
if rising_edge(clk) then
        a <= a + 1;
        b := b + 1;
        if x = '1' then
                a <= a + 3;
                b := b + 1;
        end if;
end if;
```

Formally:
<= is a concurrent statement!
:= is a sequential statement!

Summary

- Combinational versus sequential design: no memory versus memory.
- In VHDL we can model combinational circuits with sequential statements
- Remember the differences between signals and variables

Homework

- Covered today:
  - Discussion of previous week
  - Combinatorial versus sequential design
  - Concurrent and sequential design
  - Signals versus variables

- Homework:
  - 4.4, 4.5, 5.1, 5.9, 6.3

- Next week:
  - Chapter 6 ‘Sequential Code’
  - Chapter 7 ‘SIGNAL and VARIABLE’
  - Chapter 9 ‘FUNCTION and PROCEDURE’