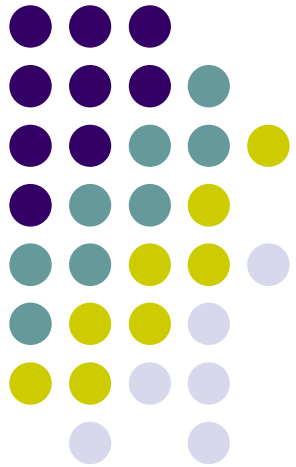
VHDL -sequential logic and arrays

COMP311

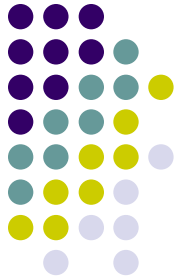
Tony McGregor

tonym@cs.waikato.ac.nz

G.1.05



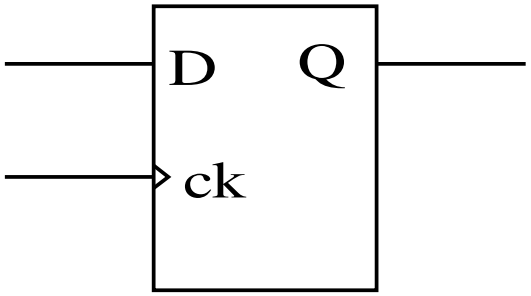
Sequential logic

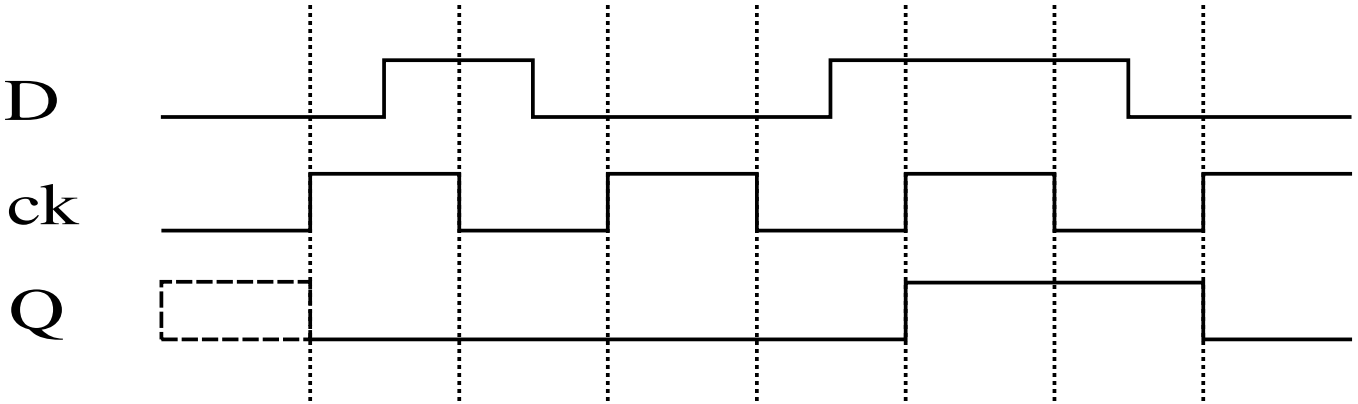


- Output is a function of the input, and the previous state.
- Circuit must have some ‘state’ or memory.
- Circuit must also have some sort of timing control mechanism.
  - We call this the ‘clock’.

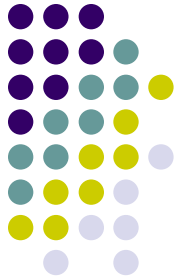
Memory elements

- We will consider positive edge-triggered flip-flops





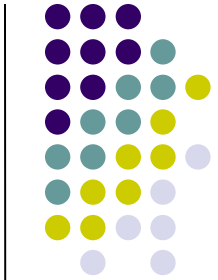
Sequential logic in VHDL



- The VHDL `process` statement introduces a sequence of serially executed statements
- It also introduces the notion of sensitivity

```
process (clk)
begin
   a;
   b;
   c
end process;
```

Conditional Execution

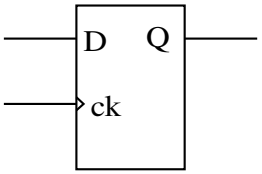


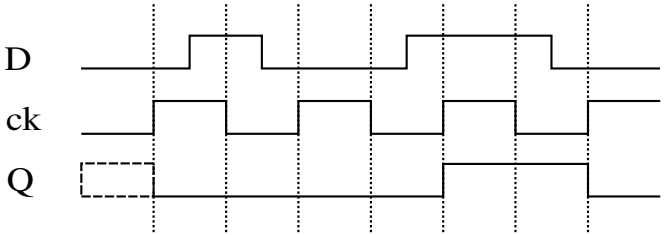
- VHDL has an if statement

```
if ( condition ) then
   statement(s);
elsif (condition ) then
   statement(s)
else
   statements(s)
end if
```

D Flip-flop in VHDL

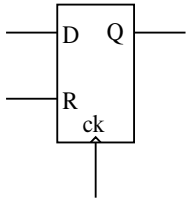
```
library ieee;
use ieee.std_logic_1164.all;

entity D_flipflop is
  port (
    clk : in std_logic;
    D   : in std_logic;
    Q   : out std_logic
    );
end D_flipflop;

architecture rtl of D_flipflop is
begin
  process (clk)
  begin
    if rising_edge(clk) then
      Q <= D;
    end if;
  end process;
end rtl;
```

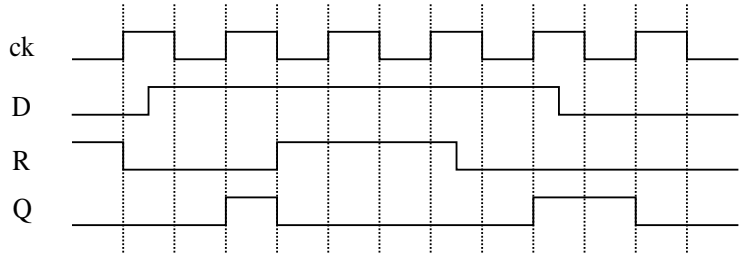




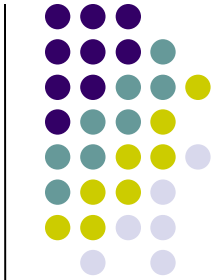
D Flip-flop with reset in VHDL

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

entity D_flipflop is
  port (
    reset, clk : in std_logic;
    D : in std_logic;
    Q : out std_logic
    );
end D_flipflop;

architecture rtl of D_flipflop is
begin
  process (reset, clk)
  begin
    if reset = '1' then
      Q <= '0';
    elsif rising_edge(clk) then
      Q <= D;
    end if;
  end process;
end rtl;
```



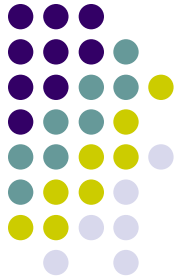


Arrays



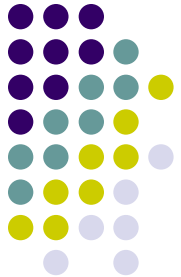
- VHDL provides support for arrays
- We will initially consider the std_logic_vector array provided by the IEEE library.
- We typically use std_logic_vector for representing a multi-bit signal (e.g. a bus).

Array Constructs



```
type std_logic_vector is array (31 downto 0) of std_logic;

type controller_state is (init, idle, active, stopped);
type state_counts is array (idle to active) of natural;
type state_counts is array(controller_state range idle to
                                    active) of natural;
type balance is array ( std_logic ) of natural;
type some_counts is array(controller_state range <>) of
                                    natural;

variable counters : state_counts;
variable my_counts : some_counts(init to idle);

type std_logic_vector is array (natural range <>) of
   std_logic;
```

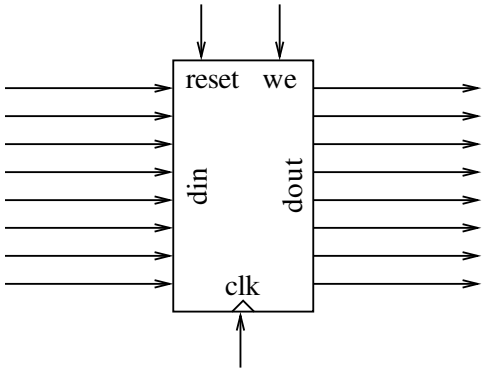
Array Aggregates



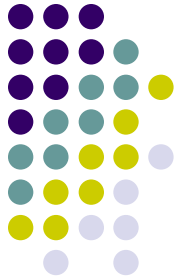
```
type point is array ( 1 to 3) of real;
constant origin : point := (0.0, 0.0, 0.0);
variable viewpoint : point := (10.0, 20.5, 0.0);
variable viewpoint : point := (1=>10.0, 2=>20.5, 3=>0.0);
variable viewpoint : point := (1=>10.0, 2=>20.5,
   others=>0.0);
```

8 bit register in VHDL

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

entity reg_8bit is
  port (
    reset, clk : in std_logic;
    we   : in std_logic;
    din  : in std_logic_vector(7 downto 0);
    dout : out std_logic_vector(7 downto 0);
    );
end reg_8bit;

architecture rtl of reg_8bit is
begin
  process (reset, clk)
  begin
    if reset = '1' then
      dout <= (others => '0');
    elsif rising_edge(clk) then
      if we = '1' then
        dout <= din;
      end if;
    end if;
  end process;
end rtl;
```

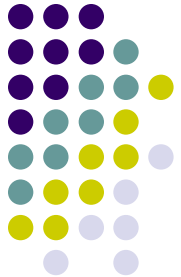


VHDL operators



- `<=` (Signal assignment)
- `+`, `-`, `*`, `/`, **`mod`**, **`rem`**, **`abs`**, `**`
  - Arithmetic operators
  - These may have varying degrees of synthesisability depending on the data type.
- `=` (Equality testing)
- `/=` (Inequality testing)
- `<` `>` `<=` `>=` (Comparisons)

VHDL operators cont…



- `and, or, not, xor` (Boolean operators)
- `&` (Array concatenation)
- There are others…