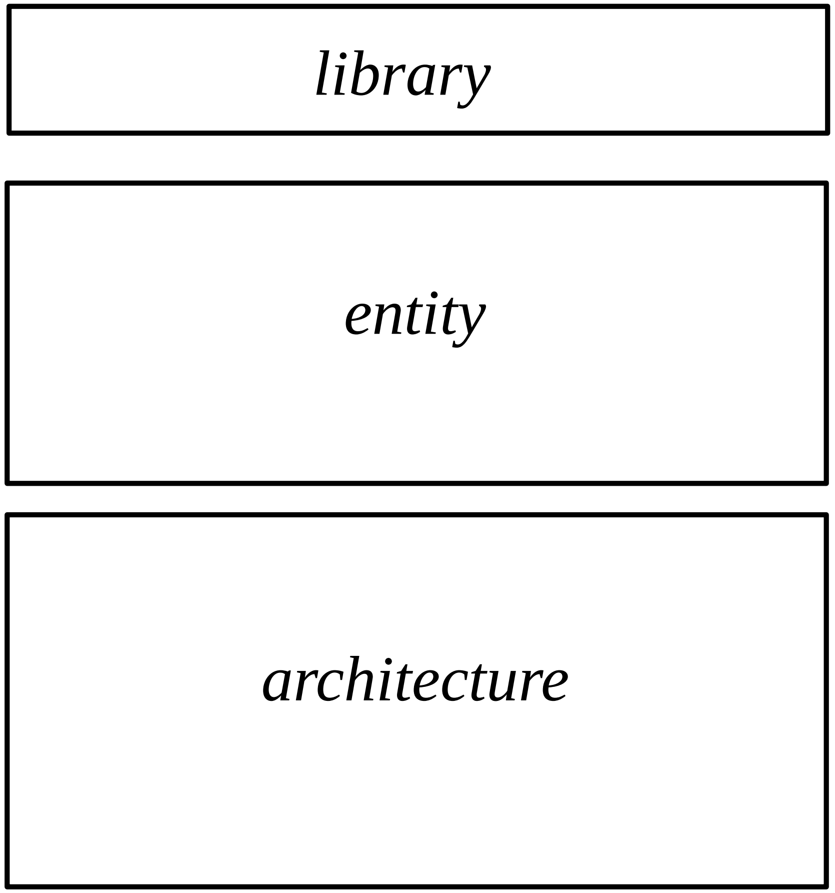
Turun yliopisto
University of Turku

HDL Based Design ME620138

Libraries and Packages

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity half_adder is
port(
        x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;

architecture myadder of half_adder is
begin
      sum <= x xor y;
      carry <= x and y;
end myadder;
```



Concurrency

- Concurrent statements are evaluated at the same time; thus, the order of these statements doesn’t matter
- Sequential/behavioural statements are executed in sequence

```
architecture dataflow of SR_flipflop is
begin

  gate_1 : q <= s_n nand q_n;
  gate_2 : q_n <= r_n nand q;

end architecture dataflow;
```

```
architecture checking of SR_flipflop is
begin

    set_reset : process (S, R) is
    begin
      assert S nand R;
      if S then
        Q <= '1';
      end if;
      if R then
        Q <= '0';
      end if;
    end process set_reset;

end architecture checking;
```

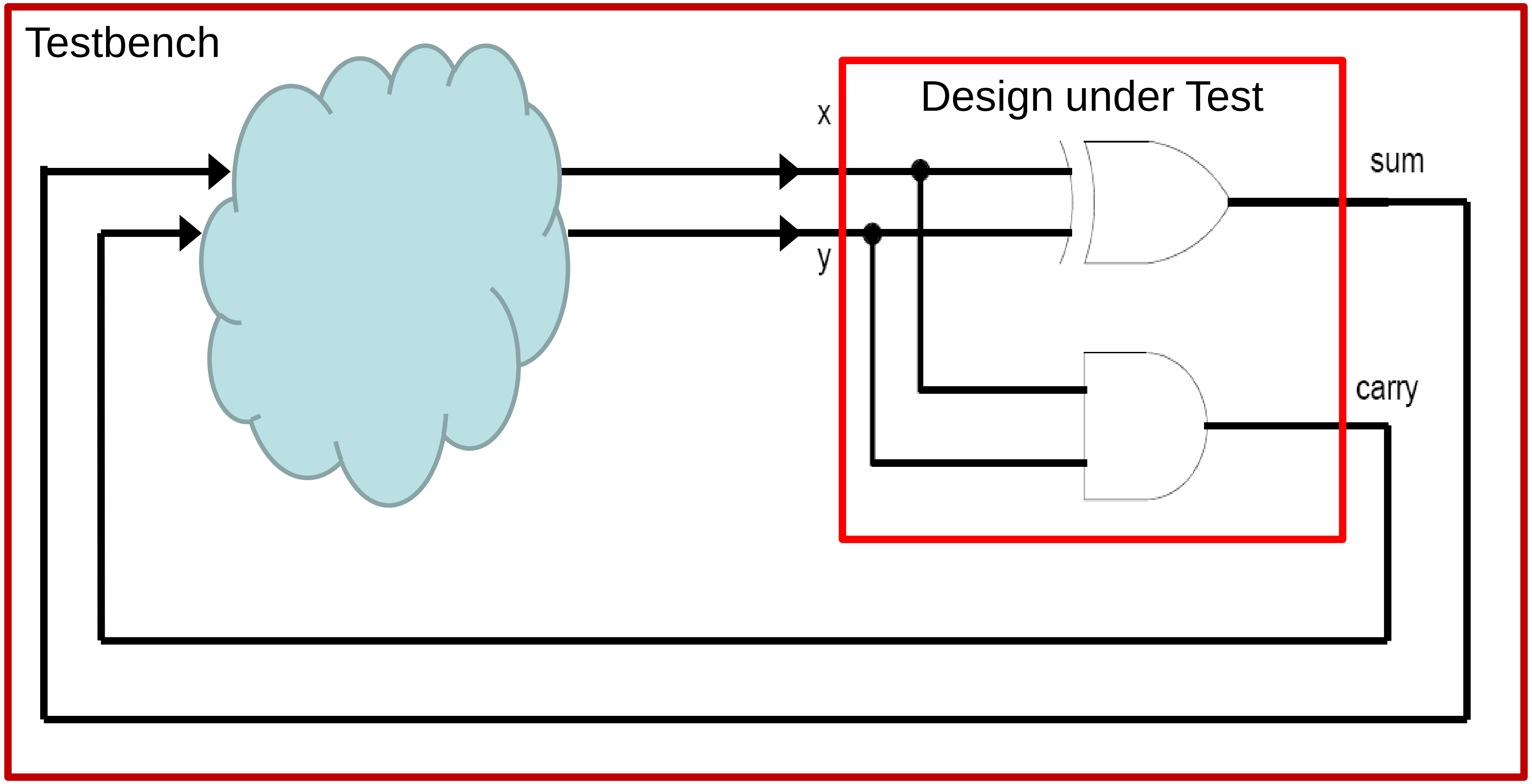
Difference between Variables and Signals

- Variables are sequential statements
  - That is, they are used in processes
- Signals are concurrent statements

- **The value of a variable is updated immediately whereas the value of a signal is updated after a delay**
  - Very important to remember

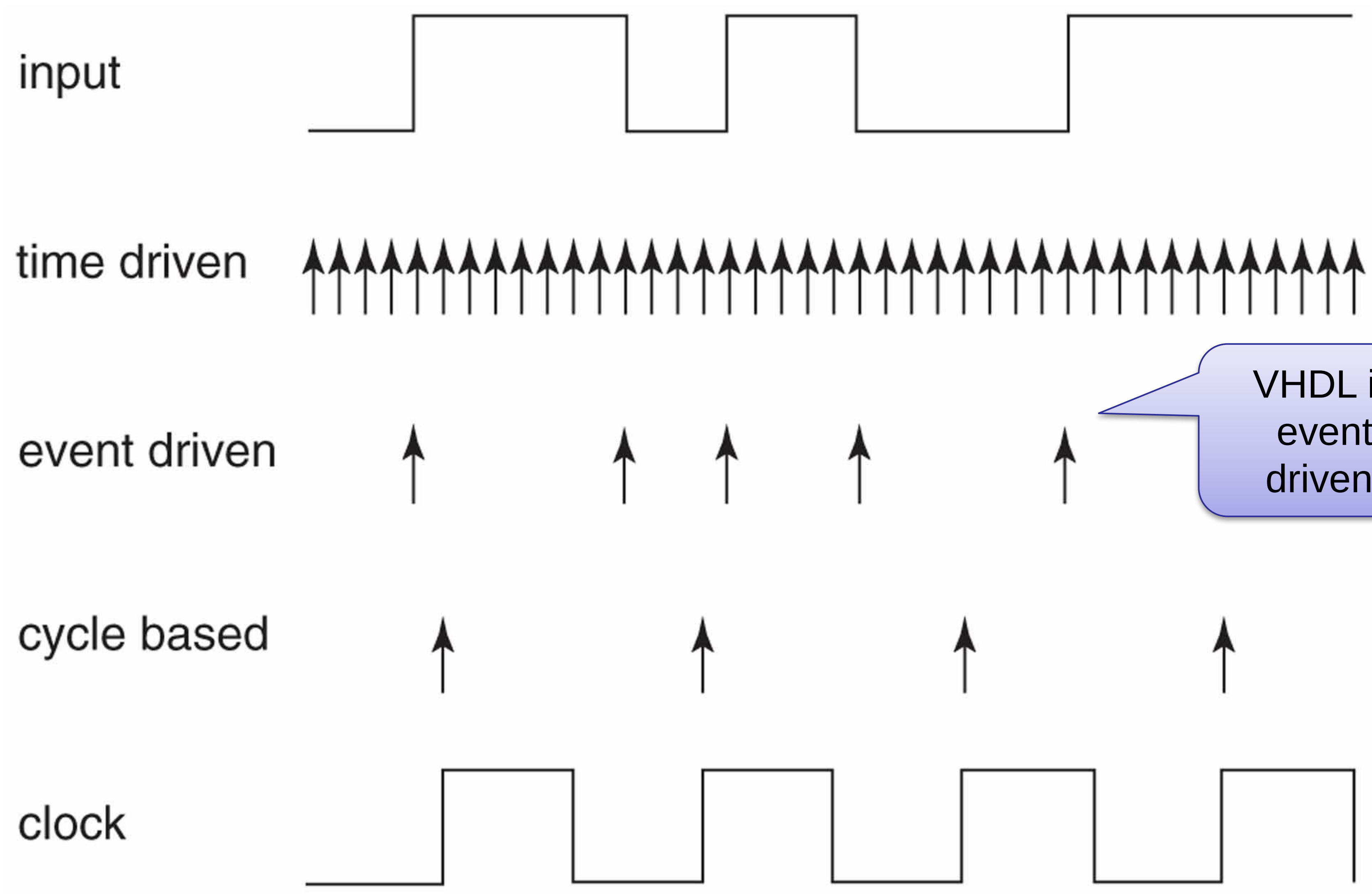
Testbench

- Testbench generates input signals for Device under Test (DUT) and observes DUT's response



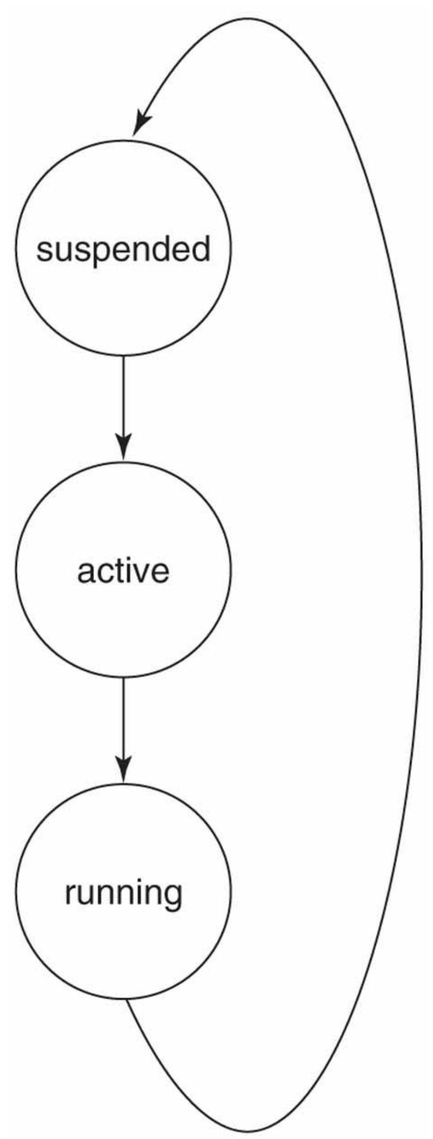
SIMULATION

Simulation Types



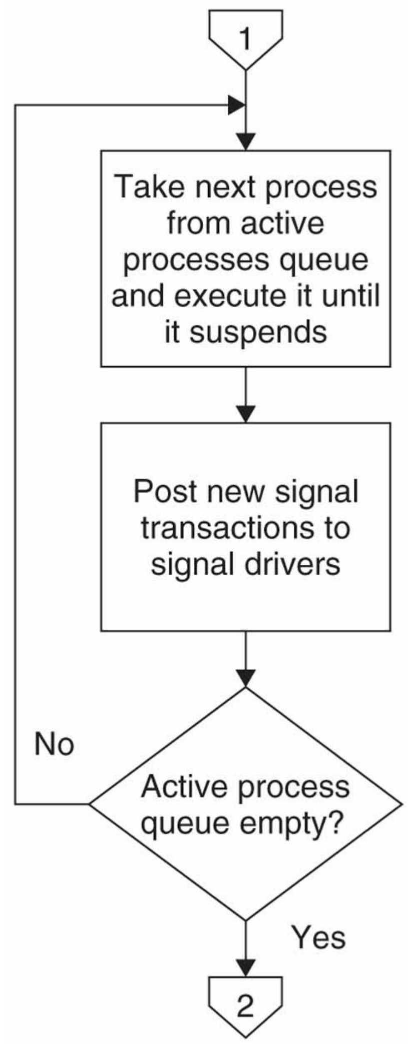
VHDL is event drivent

States of a Simulation Process



- **Suspended**: simulation process is not running or active
- **Active**: Simulation process is in the active processes queue waiting to be executed
- **Running**: Simulator is executing the simulation process

Simulation


1
Take next process from active processes queue and execute it until it suspends
Post new signal transactions to signal drivers
Active process queue empty?
No
Yes
2

- Simulation cycle consists of *signal update* and *process execution* (the figure on the left) phases
- The order in which the active processes are executed is not important

Simulation Cycle – in detail

1. The simulation time is first advanced to the next time at which a transaction on a signal has been scheduled
2. All the transactions scheduled for that time are performed
   - This may cause some events to occur on some signals
3. All processes that are sensitive to those events are resumed and are allowed to continue until they reach a wait statement and suspend
   - Again, the processes usually execute signal assignments to schedule further transactions on signals
4. When all the processes have suspended again, the simulation cycle is repeated

- When there are no further scheduled transactions, the simulation ends

Signals versus Variables

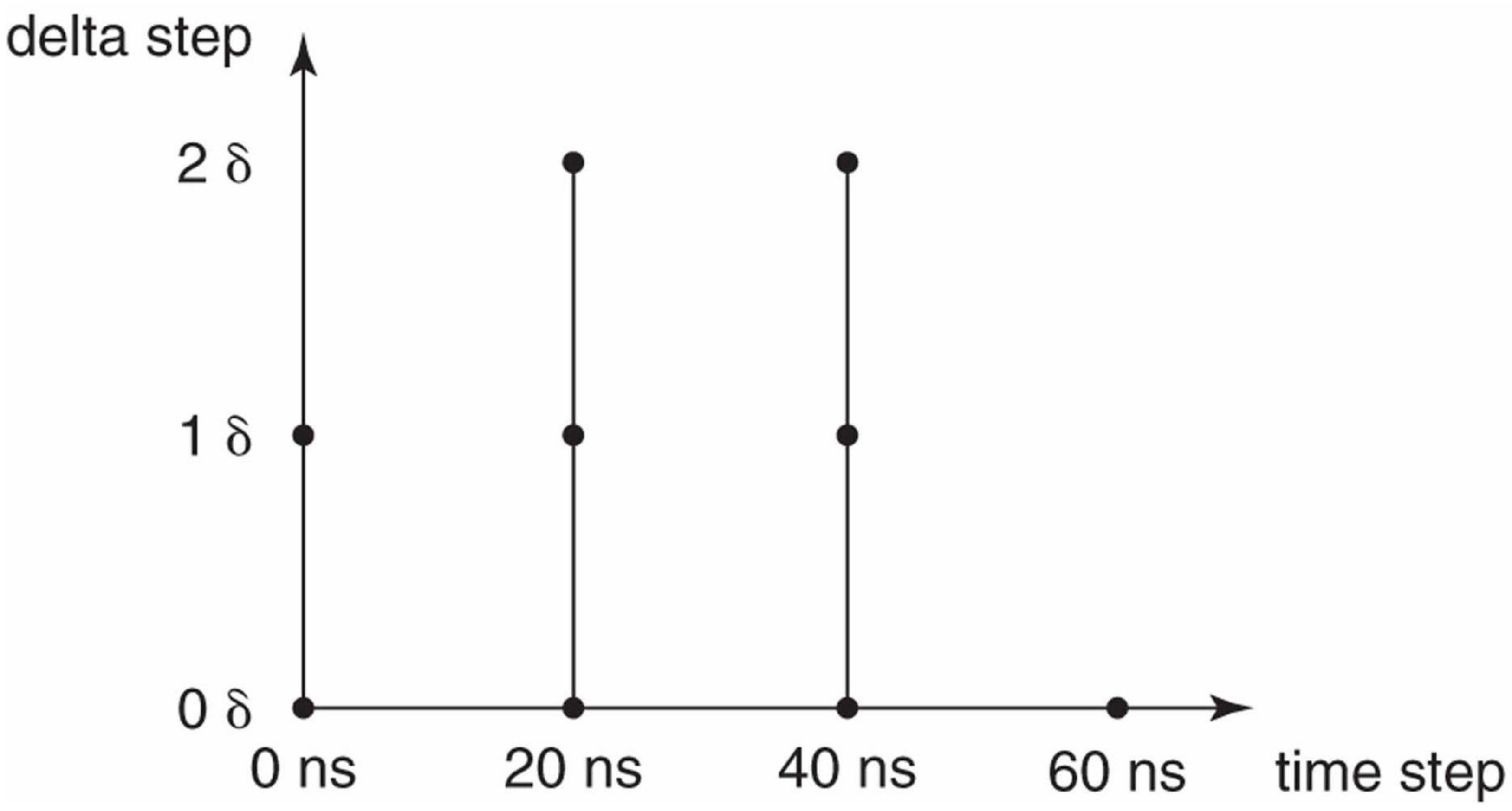
- Variable's value is updated immedately during the current simulation cycle
- Signal's value is not updated during the current simulation cycle
  - The new value will not take effect until the update phase of the next simulation cycle
  - Without any delay information the new value is updated after *a delta delay*

Delta Delay

- The signal value does not change as soon as the signal assignment statement is executed
- The process does not see the effect of the assignment until the next time it resumes, even if this is at the same simulation time

```
s <= '1';
...
if s then ...
```

Delta Cycle



To understand Delta Cycle, you should understand the simulation cycle in VHDL

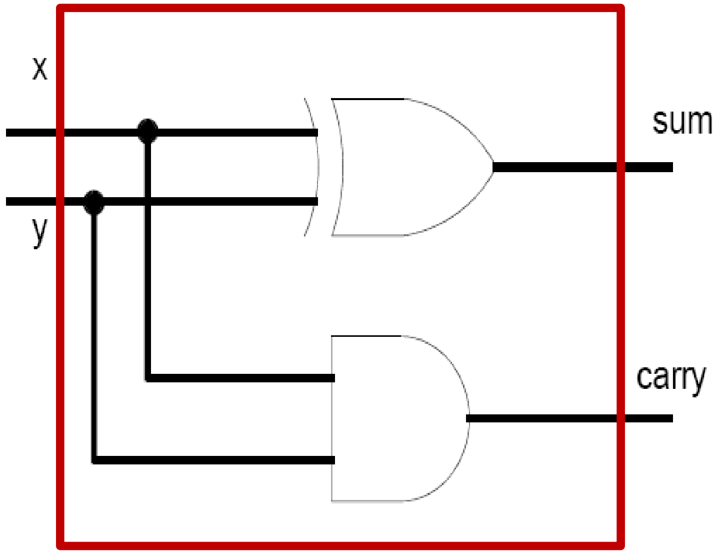
signal update phase is followed by a process execution phase

Turun yliopisto
University of Turku

TIMING

An Example Module

```
entity half_adder is
port(
        x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;

architecture myadder of half_adder is
begin
      sum <= x xor y;
      carry <= x and y;
end myadder;
```



Timing

- To postpone the update of a signal value, use an **after** clause to define the time period

```
Dout <= not Din after 10 ns;
```

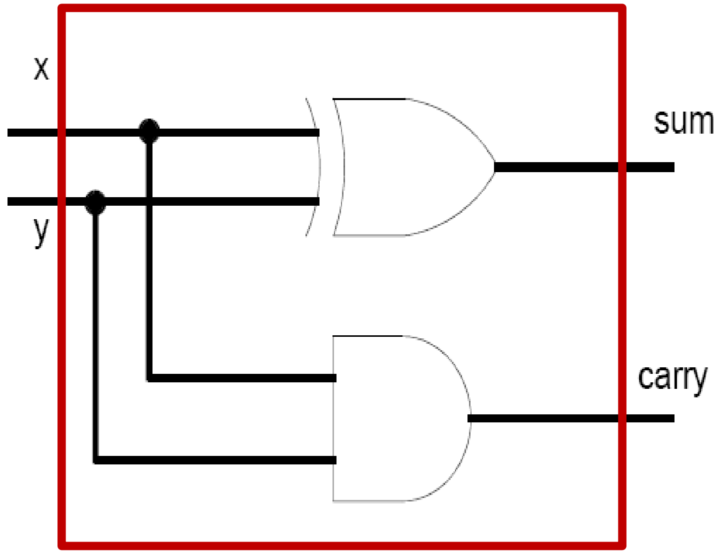
- The time period is added to current simulation time
- You can create a wave form using the **after** clause:

```
Dout <= ‘0’ after 10 ns, ‘1’ after 20 ns, ‘0’ after 30 ns;
```

Turun yliopisto
University of Turku

An Example Module

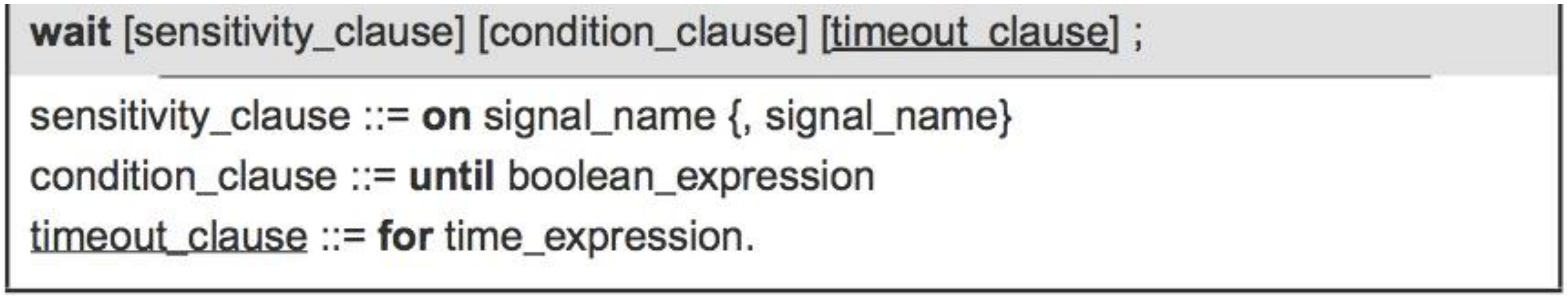
```
entity half_adder is
port(
        x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;

architecture myadder of half_adder is
begin
      sum <= x xor y after 10 ns;
      carry <= x and y after 10 ns;
end myadder;
```



Turun yliopisto
University of Turku

Waiting

- Wait statement is used in processes to wait on signal changes or a time period



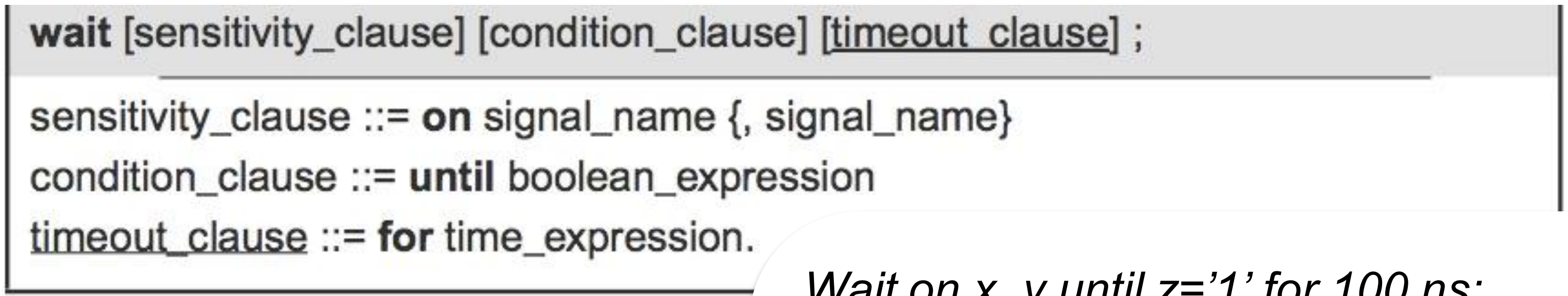
wait [sensitivity_clause] [condition_clause] [timeout_clause] ;

sensitivity_clause ::= **on** signal_name {, signal_name}
condition_clause ::= **until** boolean_expression
timeout_clause ::= **for** time_expression.

- With **on**, you can specify signal to which the process responds
- With **until**, you can stall the process until a condition is met
- With **for**, you can stall the process a time period

Waiting

- Wait statement is used in processes to wait on signal changes or a time period



wait [sensitivity_clause] [condition_clause] [timeout_clause] ;

sensitivity_clause ::= **on** signal_name {, signal_name}
condition_clause ::= **until** boolean_expression
timeout_clause ::= **for** time_expression.

wait on clk **until** reset = ’1’;

wait until x **for** 10 ns;

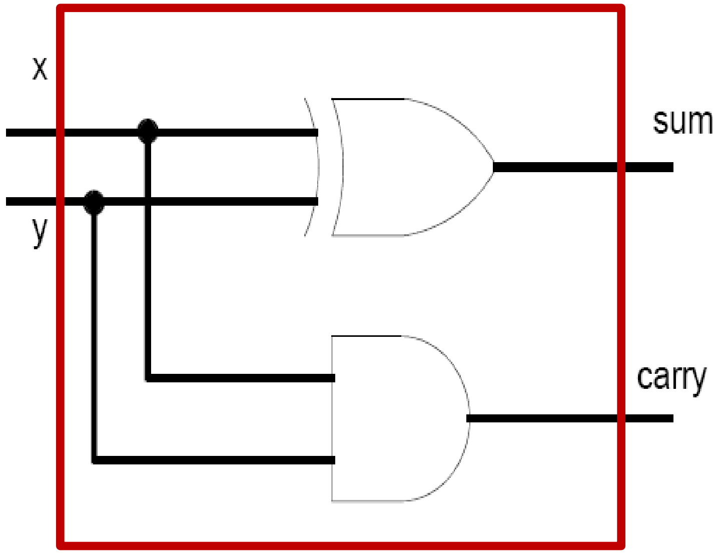
wait;

Wait on x, y until z=’1’ for 100 ns;

The execution of the process is suspended for a maximum of 100 ns. **If an event occurs on x or y prior to 100 ns, the condition z is evaluated.** If z is true (that is z=’1’), when the event on x or y occurs, the process will resume at that time; otherwise, syspension continues

An Example Module

```
entity half_adder is
port(
        x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;

architecture myadder of half_adder is
begin
      process is
      begin
              sum <= x xor y after 10 ns;
              carry <= x and y after 10 ns;
      wait on x, y;
      end process;
end myadder;
```

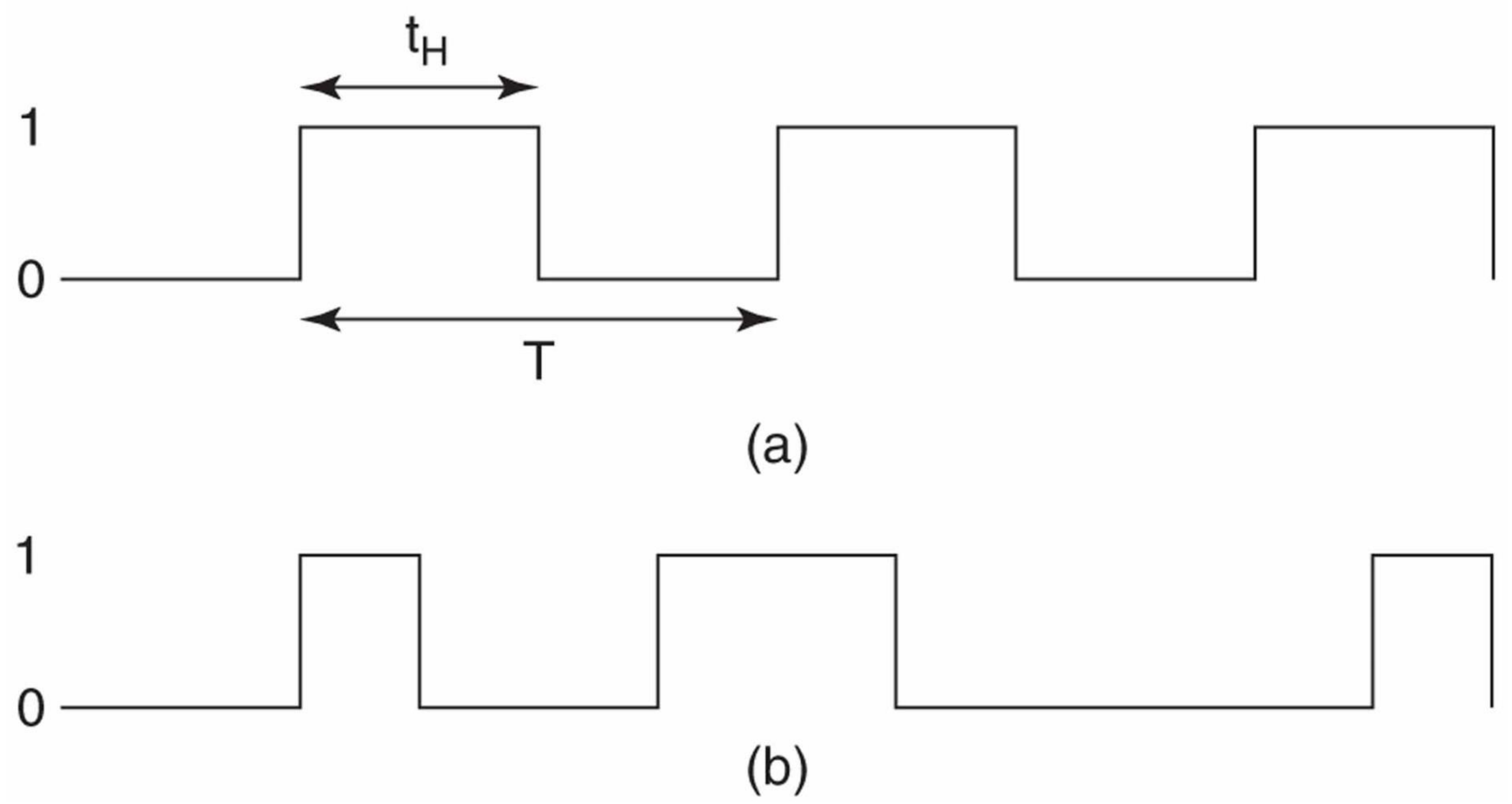


SYNCHRONOUS DESIGN

Turun yliopisto
University of Turku

Clock Signal

- Clock signal is a sequence of pulses



Turun yliopisto
University of Turku

Clocks

- Free running clock definitions:

```
signal  clock : std_ulogic;
. . .
clock_gen : process is
begin
      clk <= ‘1’;
      wait for 10 ns;
      clk <= ‘0’;
      wait for 10 ns;
end process clock_gen;
```

```
signal  clock : std_ulogic := '1';
. . .
clock <= not clock after 5ns;
```

- Limited number of clock cycles:

```
signal  clock : std_ulogic;
. . .
process
  begin
    for i in 1 to num_clockcycles loop
      clock <= not clock;
      wait for 10 ns;
      clock <= not clock;
      wait for 10 ns;
      end loop;
      wait;
  end process;
```

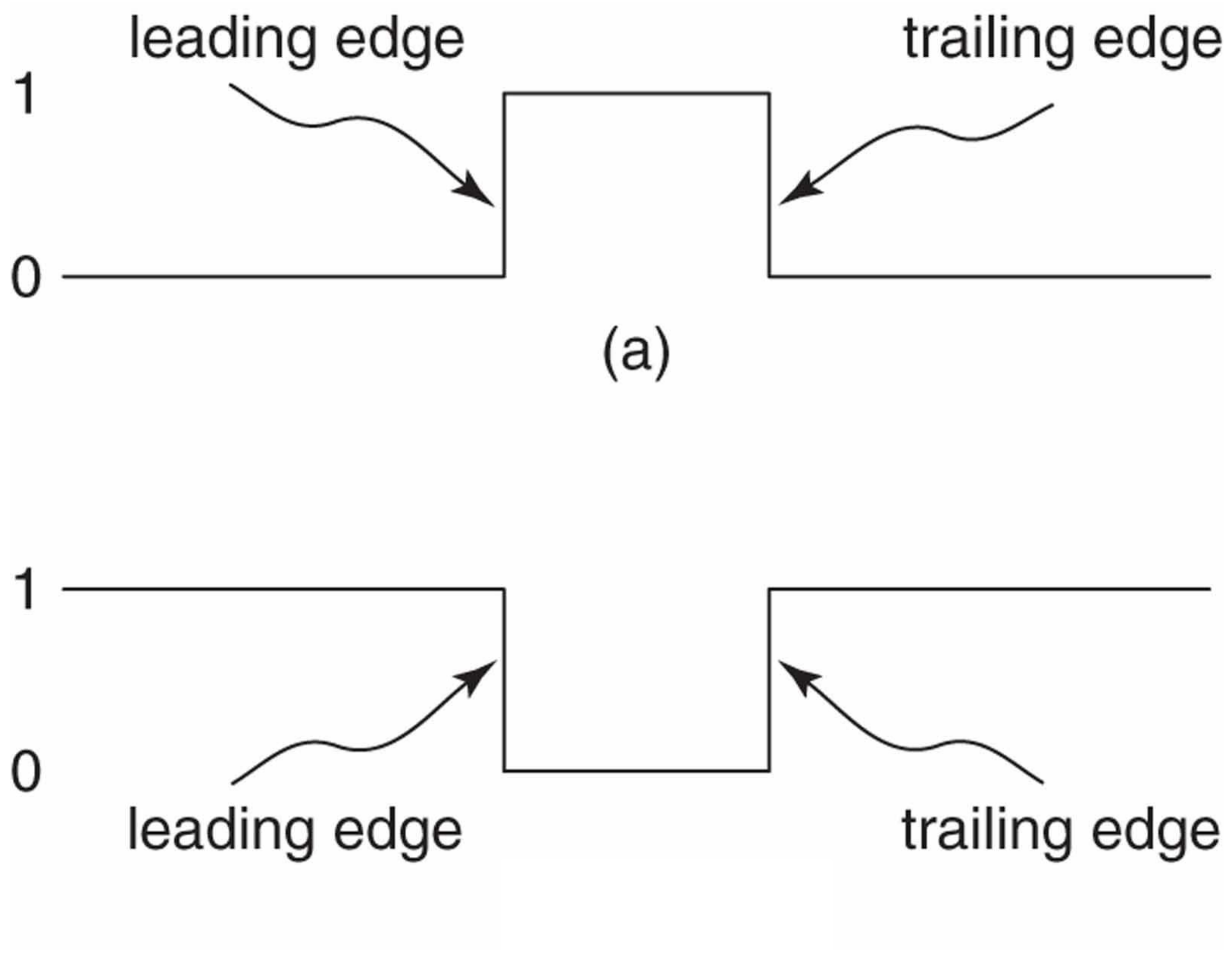
Reset Signal

- Reset signal sets the system in a predefined state

```
signal  clock : std_ulogic := '1';
signal rst: std_logic;
. . .
rst <= '0', '1' after 10 ns, '0' after 100 ns;
clock <= not clock after 5ns;
```

LATCHES AND FLIP-FLOPS

Signal Pulse



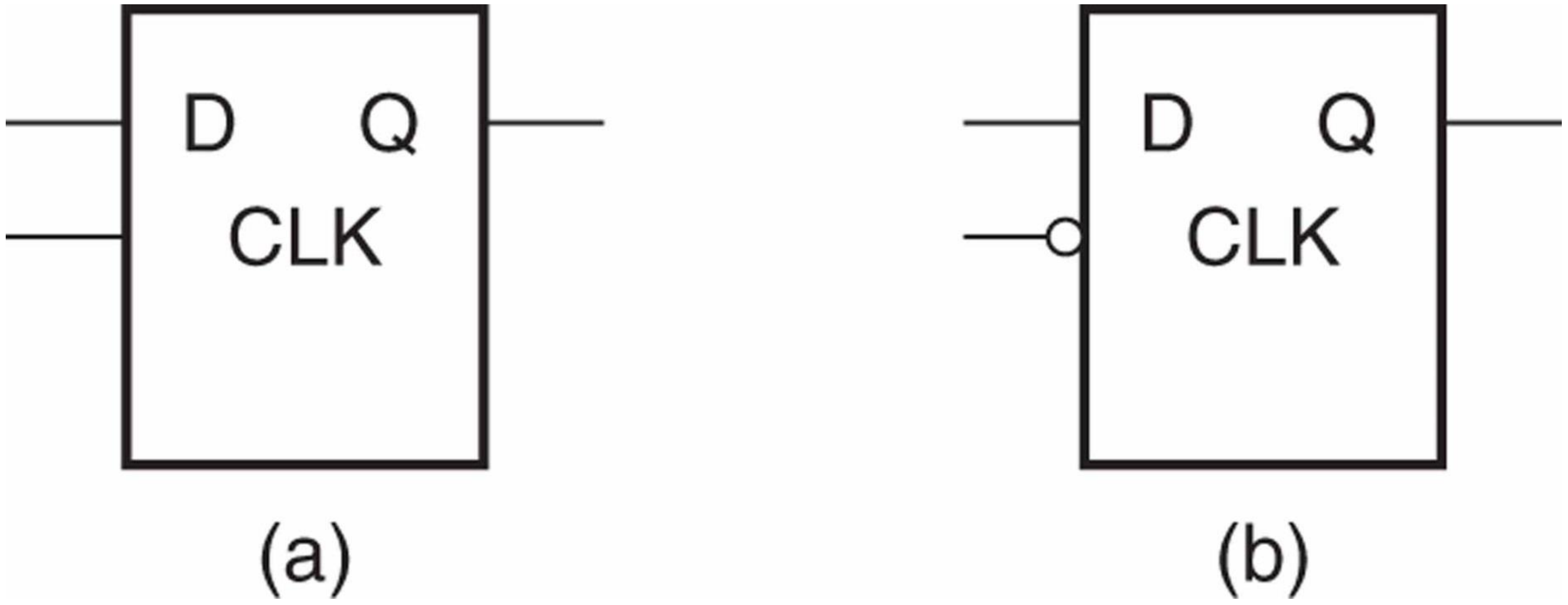
Function table for a positive-level D latch.

D	CLK	Q_{t+1}
0	0	Q_t
0	1	0
1	0	Q_t
1	1	1

Function table for a positive-edge-triggered D flip-flop.

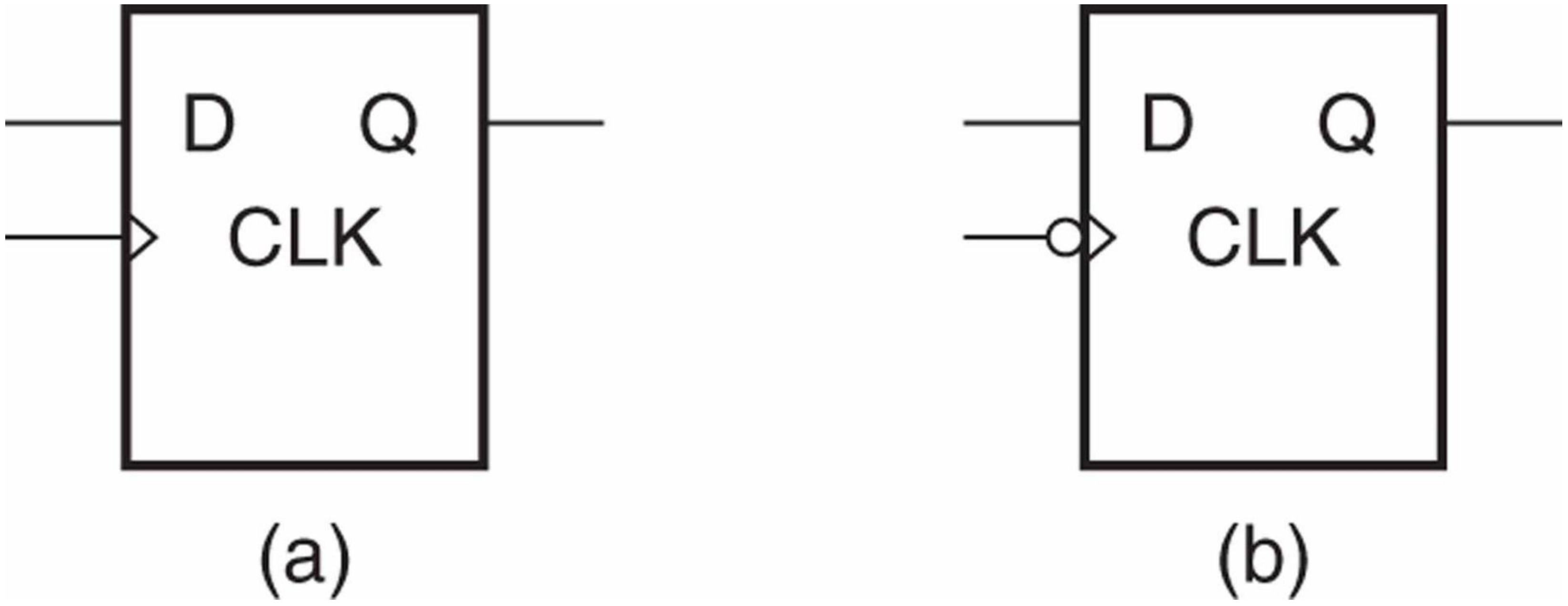
D	CLK	Q_{t+1}
0	↑	0
1	↑	1
X	0	Q_t
X	1	Q_t

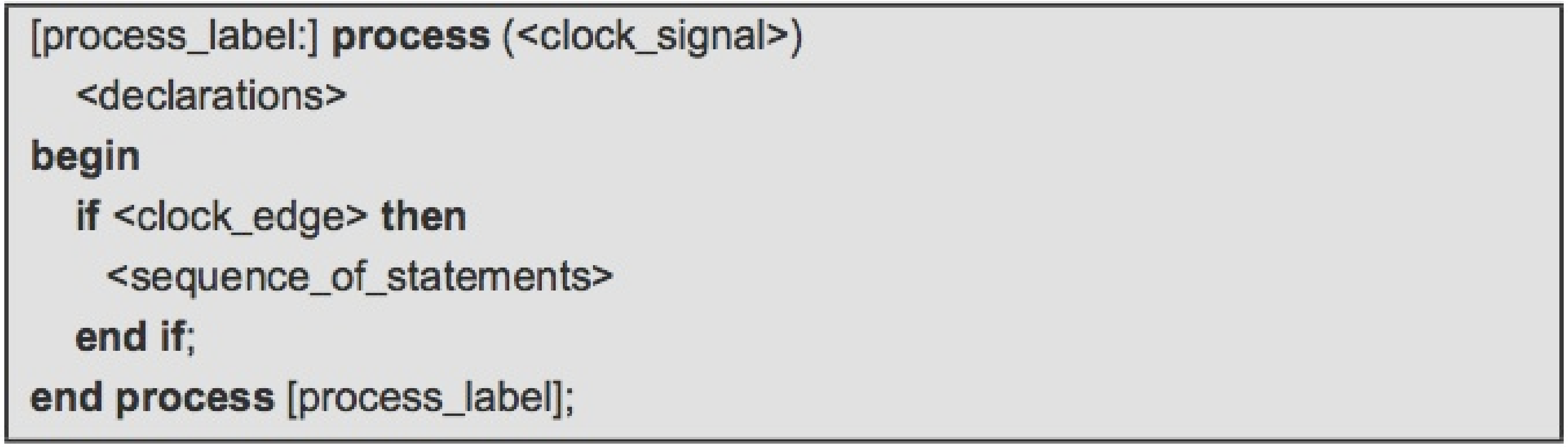
D Latch



```
[process_label:] **process** (<clock_signal>, <input_signals>)
    <declarations>
begin
    if <clock_level> then
      <sequence_of_statements>
    end if;
end process [process_label];
```

D Flip-flop

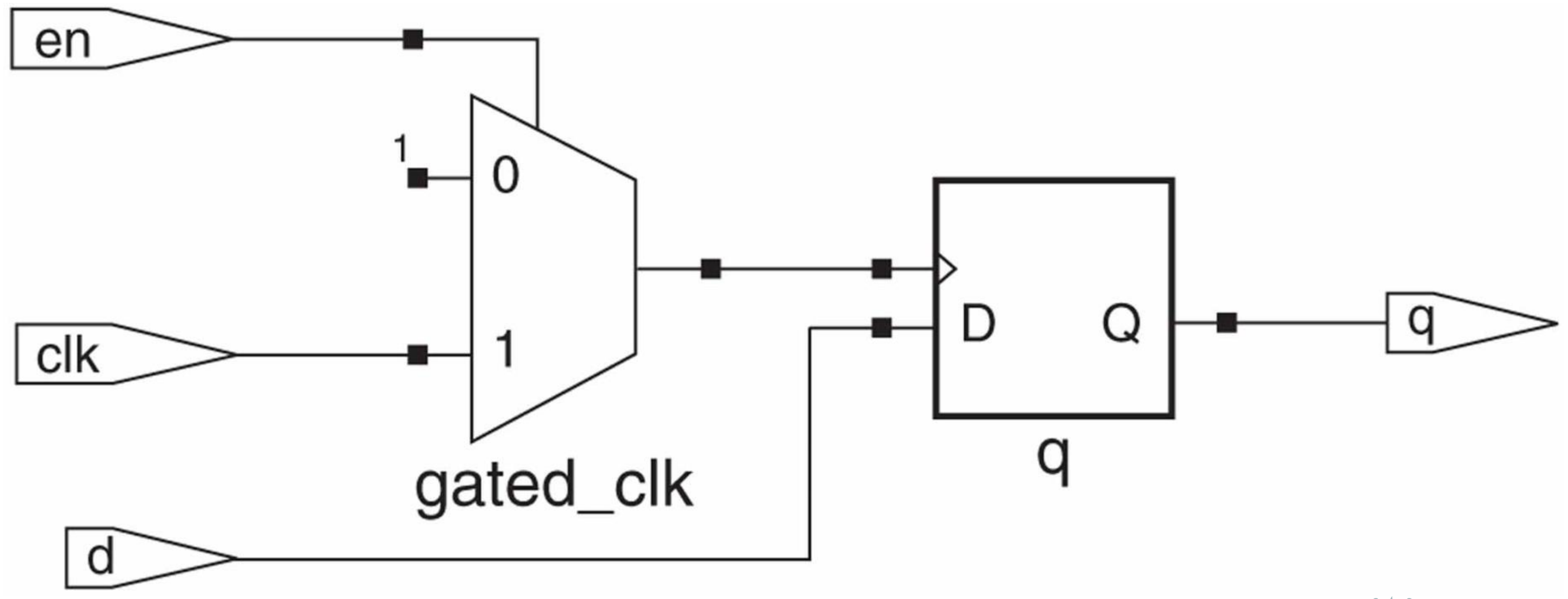




```
[process_label:] process (<clock_signal>)
    <declarations>
begin
    if <clock_edge> then
      <sequence_of_statements>
    end if;
end process [process_label];
```

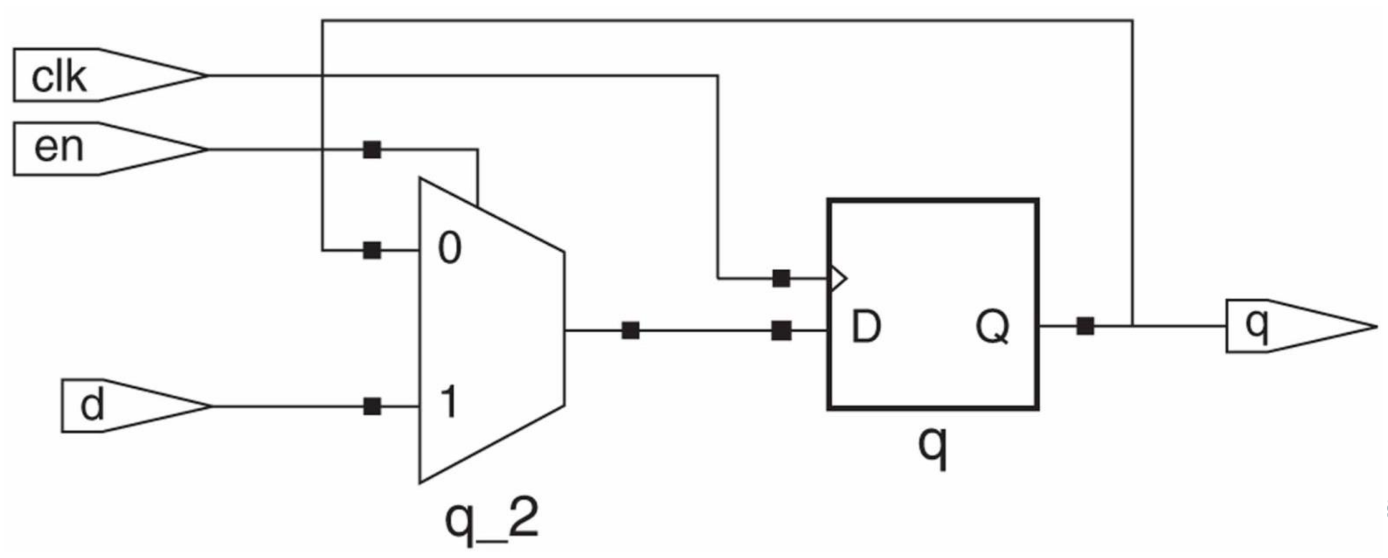
Gated Flip-Flop

```
gated_clk <= clk when en = ‘1’ else ‘1’;

process (gated_clk) begin
    if rising_edge(gated_clk) then
    . . .
    end if;
end process;
```



Enabled Flip-Flop

process (clk) **begin**
    if rising_edge(clk) **then**
      If en = ’1’ **then**
          q <= d ;
      else
          q <= q;
      end if;
    end if;
end process;
	process (clk) **begin**
	    if rising_edge(clk) **then**
	      If en = ’1’ **then**
	          q <= d ;
	      end if;
	    end if;
	end process;



True and complement outputs for Flip-Flop

```
architecture dataflow is
    signal qsig : std_logic;
begin
    process (clk) begin
          if rising_edge(clk) then
              qsig <= d ;
          end if;
      end if;
    end process;
    q <= qsig;
    qnot <= not qsig;
end architecture;
```

Turun yliopisto
University of Turku

Sync and Async Reset Signal

```vhdl
process (clk)
begin
    if rising_edge(clk) then
            if rst = '1' then
                    dout <= 0;
            else
                    dout <= d ;
            end if;
    end if;
end process;
```

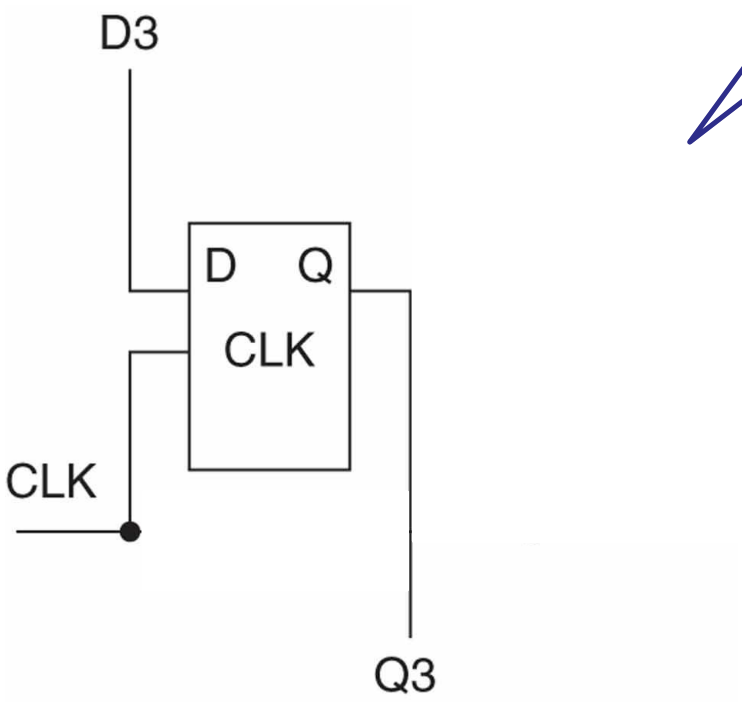
```vhdl
process (clk,rst)
begin
    if rst = '1' then
            dout <= 0;
    elsif rising_edge(clk) then
            dout <= d ;
    end if;
end process;
```

MEMORIES

Register

```
if rising_edge(clk) then
        if rst= '0' then
                Q3 <= '0'
        else
                Q3 <= D3;
        end if;
end if;
```

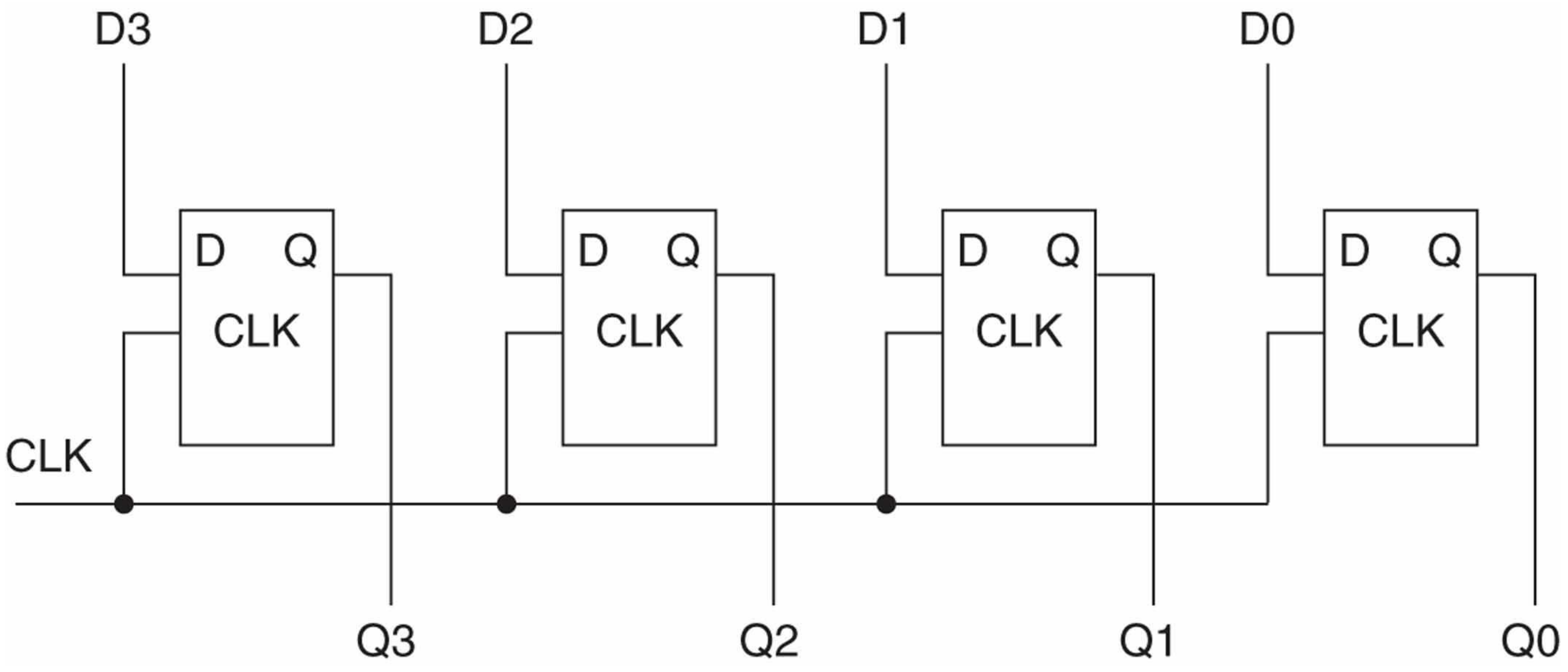
How to change the code to create this register bank?



Register

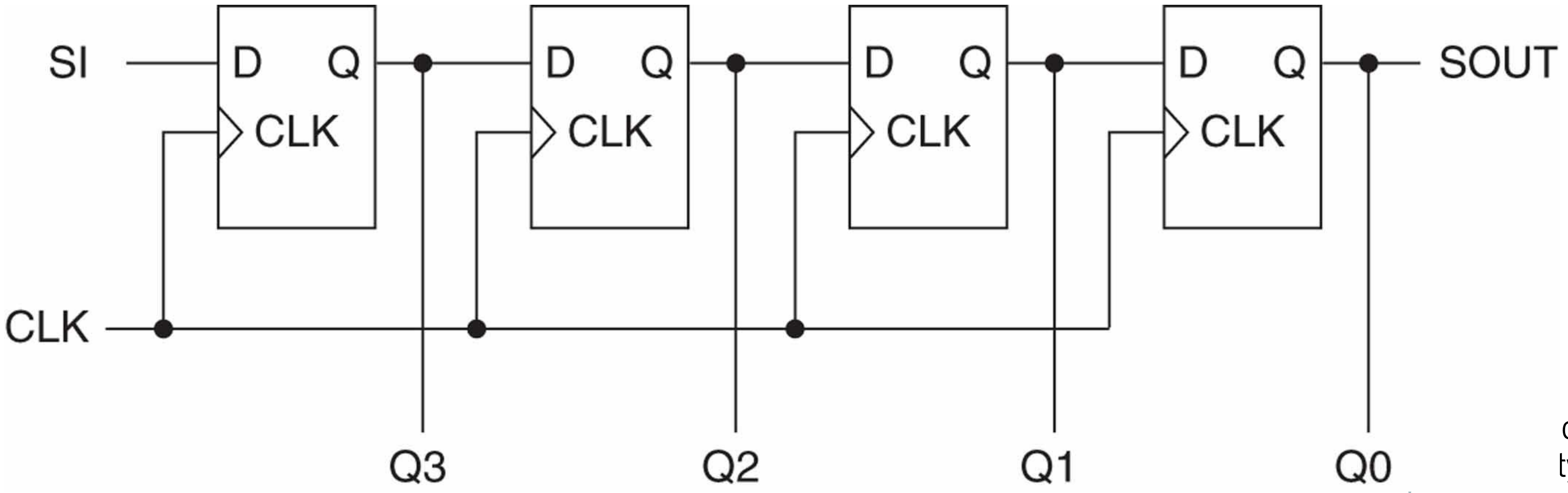
```
if rising_edge(clk) then
        if rst= '0' then
                Q3 <= '0'
        else
                Q3 <= D3;
        end if;
end if;
```

```
if rising_edge(clk) then
        if rst= '0' then
                Q <= "0000"
        else
                Q <= D;
        end if;
end if;
```



Shift Register with Parallel Output

```
if rising_edge(clk) then
        If rst= ’0’ then
              SOUT <= ”0000”
        else
              Q(0) <= Q(1);
              Q(1) <= Q(2);
              Q(2) <= Q(3);
              Q(3) <= SI;
        end if;
end if;
SOUT <= Q(0);
```

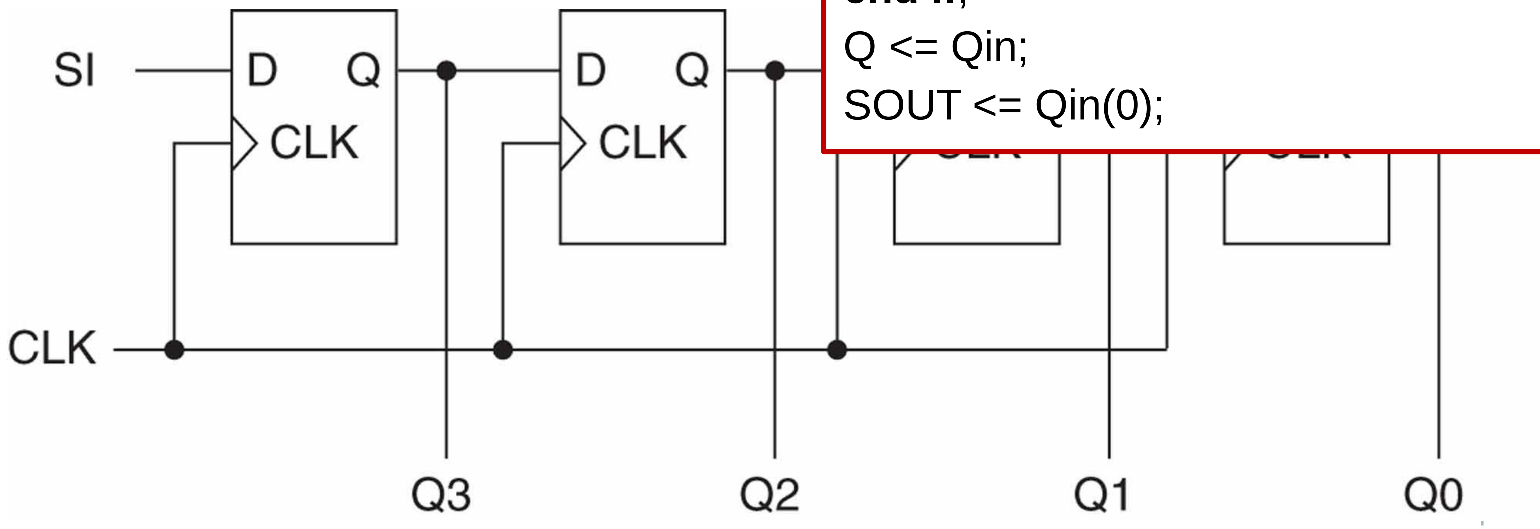


opisto
ty of Turku

Shift Register with Parallel Output

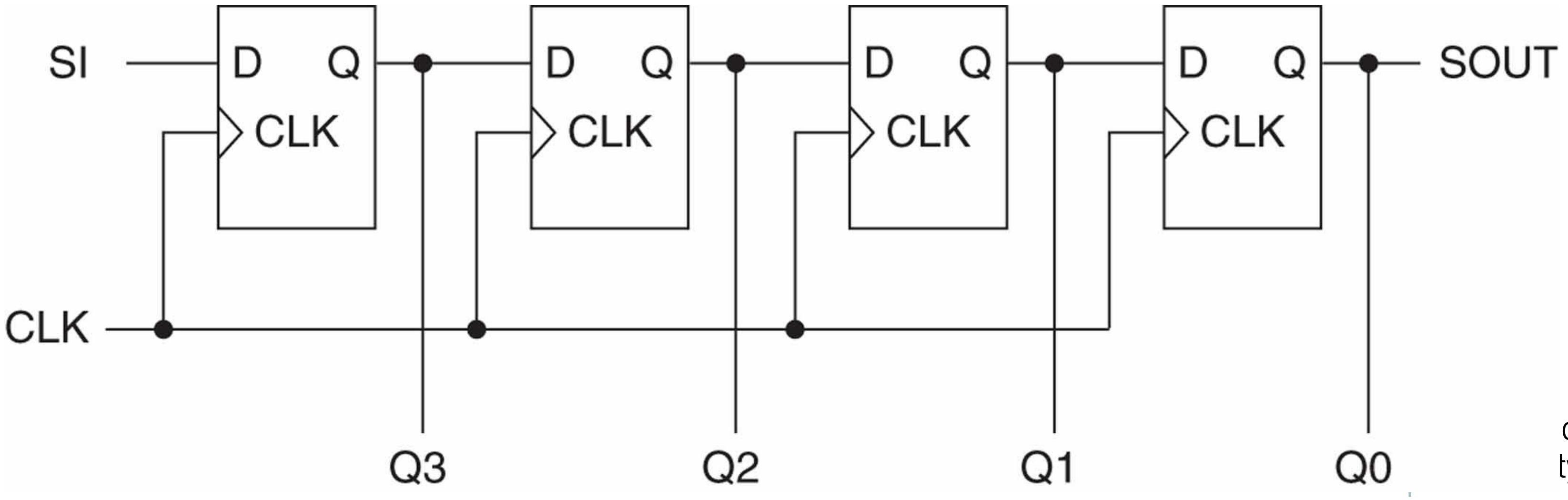
```
if rising_edge(clk) then
      If rst= '0' then
           SOUT <= "0000"
      else
           Q(0) <= Q(1);
           Q(1) <= Q(2);
           Q(2) <= Q(3);
           Q(3) <= SI;
      end if;
end if;
SOUT <= Q(0);
```

```
variable Qin : std_logic_vector(3 downto 0);

if rising_edge(clk) then
     If rst= '0' then
       Qin := "0000"
     else
       Qin(0) := Qin(1);
       Qin(1) := Qin(2);
       Qin(2) := Qin(3);
       Qin(3) := SI;
     end if;
end if;
Q <= Qin;
SOUT <= Qin(0);
```



Shift Register with Parallel Output

```
process (clk)
      variable Q : std_logic_vector(3 downto 0);
begin
if rising_edge(clk) then
      If rst= ’0’ then
         Q := ”0000”
      else
         Q := SI & Q(3 downto 1);
      end if;
      Qout <= Q;
end if;
```



opisto
ty of Turku