Digital Systems - 8

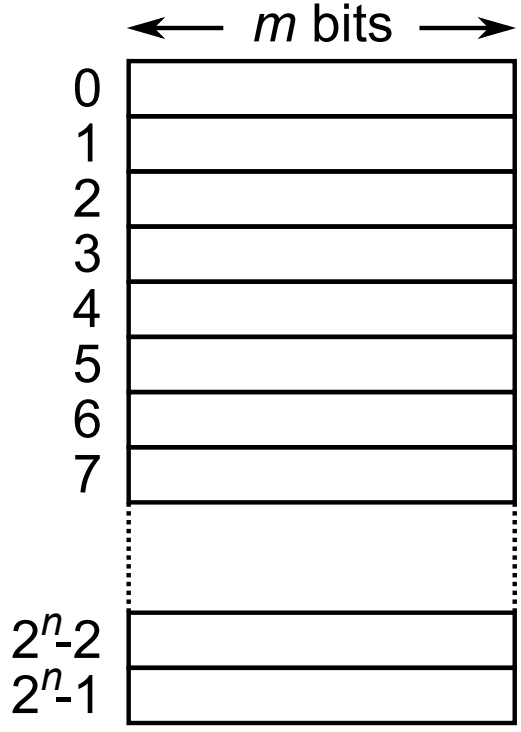
Pere Palà - Alexis López

iTIC http://itic.cat

March 2016

Memories

- A memory : array of storage locations
  - Unique address
  - Storing m bits of data
  - Similar to a collection of registers.
- Address
  - Unsigned coding
  - Address bits: $n \rightarrow 2^n$ locations



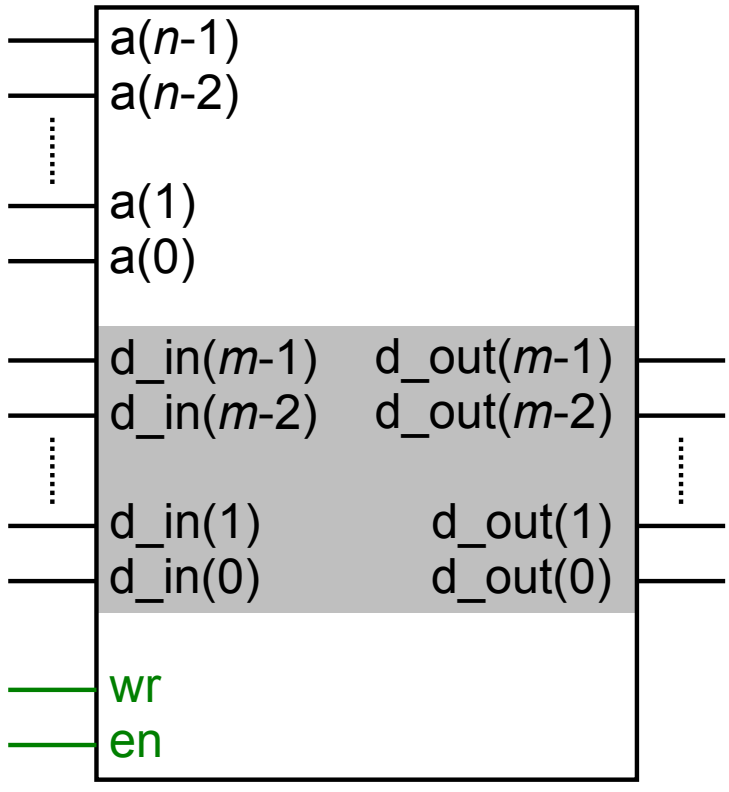
Memory Units

- Use power of 2 multipliers
  - Kilo (K): $2^{10} = 1\ 024 \approx 10^3$
  - Mega (M): $2^{20} = 1\ 048\ 576 \approx 10^6$
  - Giga (G): $2^{30} = 1\ 073\ 741\ 824 \approx 10^9$
- Different memory organizations
- Example: 1 Mbit memory
  - 128K $\times$ 8-bit
  - 64K $\times$ 16-bit
  - 32K $\times$ 32-bit
  - 16K $\times$ 64-bit

Elementary Memory

- Address Inputs
  - a($n-1..0$)
- Data In
  - d_in($m-1..0$)
- Data Out
  - d_out($m-1..0$)
- Control Signals
  - en (enable)
  - wr (write)

- To write: `wr<='1'; en<='1';`
- To read: `wr<='0'; en<='1';`
- Idle: `d_out <=Z when en='0';`
- ... or `d_out <= d_out_prev`



Elementary Memory in VHDL

```
type   RAM_16x8 is array(0 to 15)
                     of std_logic_vector(7 downto 0);
signal data          : RAM_16x8;
signal d_in, d_out   : std_logic_vector(7 downto 0);
signal wr, en        : std_logic;
signal addr          : unsigned(3 downto 0);
```

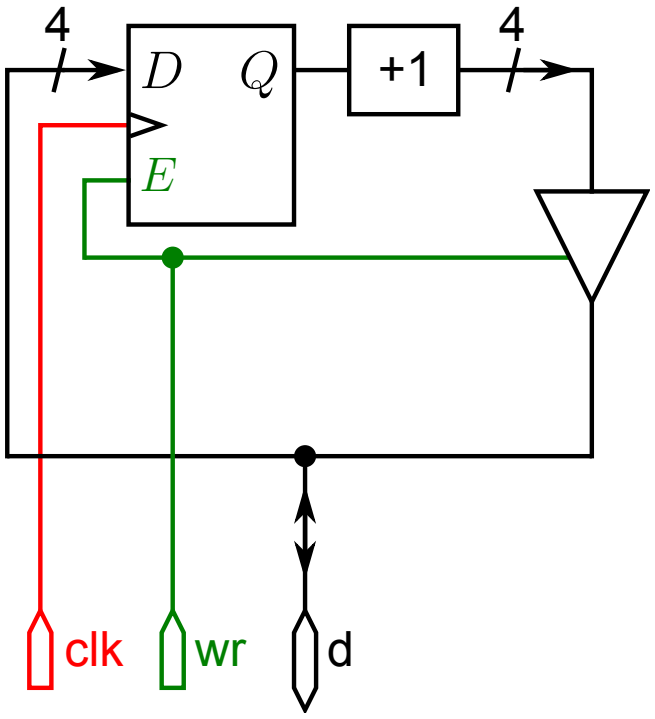
```
process (addr,wr,en)
begin
  if en='1' then
     if wr='1' then
        data(to_integer(addr)) <= d_in;
     end if;
     d_out <= data(to_integer(addr));
  else
     d_out <= (others=>'Z');
  end if;
end process;
```

Memory Types

- Random-Access Memory (RAM)
  - Read and Write
  - Volatile: Stores data only if power is maintained
  - Static RAM (SRAM)
    - Not clocked: Asynchronous SRAM
    - Clocked: Synchronous SRAM (SSRAM)
  - Dynamic RAM (DRAM)
- Read-Only Memory (ROM)
  - Non Volatile
  - Can Read only
  - Synchronous or Asynchronous
  - Combinational function
- Flash RAM
  - Non volatile
  - Read (fast) and write (slow)
  - Limited number of write (erase) cycles

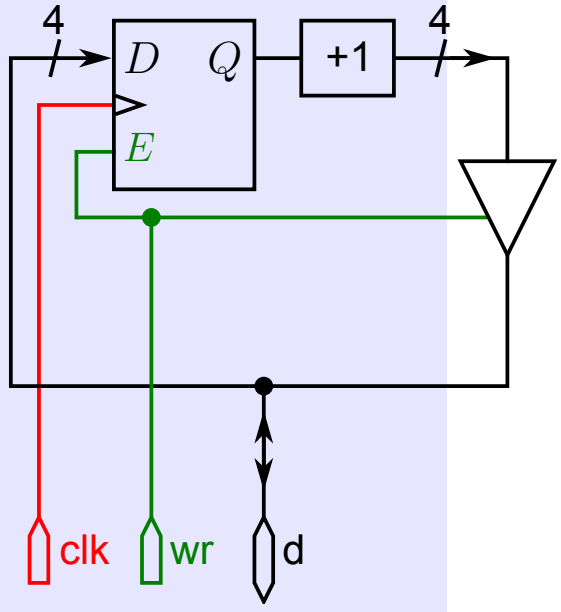
Bidirectional Ports

- Reduce number of pins
  - Board level: less pins / traces
  - Infrequent in-chip
- Master / Slave
  - Master activates control signals
  - Slave acts accordingly
  - Use of 'Z': no conflict with other driver
- Master is writing on *d*
  - Output buffer: writes 'Z'
  - Input port: senses *d*
- Master is reading
  - Output port: writes data
  - Input port: senses data (ignored)



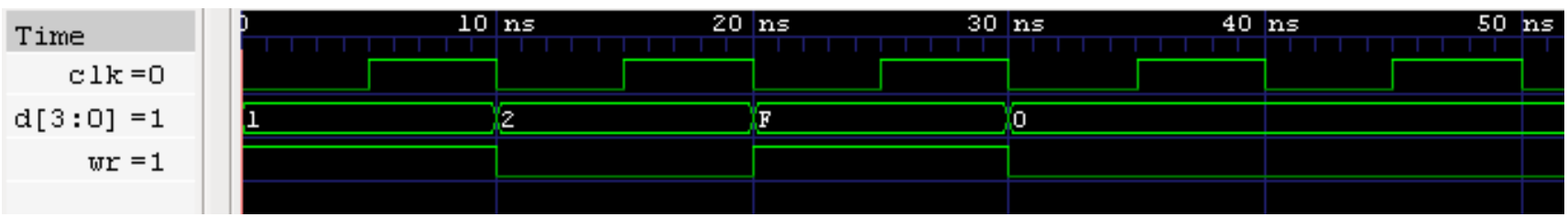
Bidirectional Ports in VHDL

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

entity bidir_inc is
  port( d       : inout std_logic_vector(3 downto 0);
        clk,wr  : in std_logic);
end bidir_inc;

architecture arch of bidir_inc is
  signal q, q_plus : std_logic_vector(3 downto 0);
begin
  process (clk)
  begin
    if rising_edge(clk) then
       if wr='1' then
          q<=d;
       end if;
    end if;
  end process;
  q_plus <= std_logic_vector(unsigned(q)+1);
  d      <= (others=>'Z') when wr='1' else
            q_plus;
end arch;
```



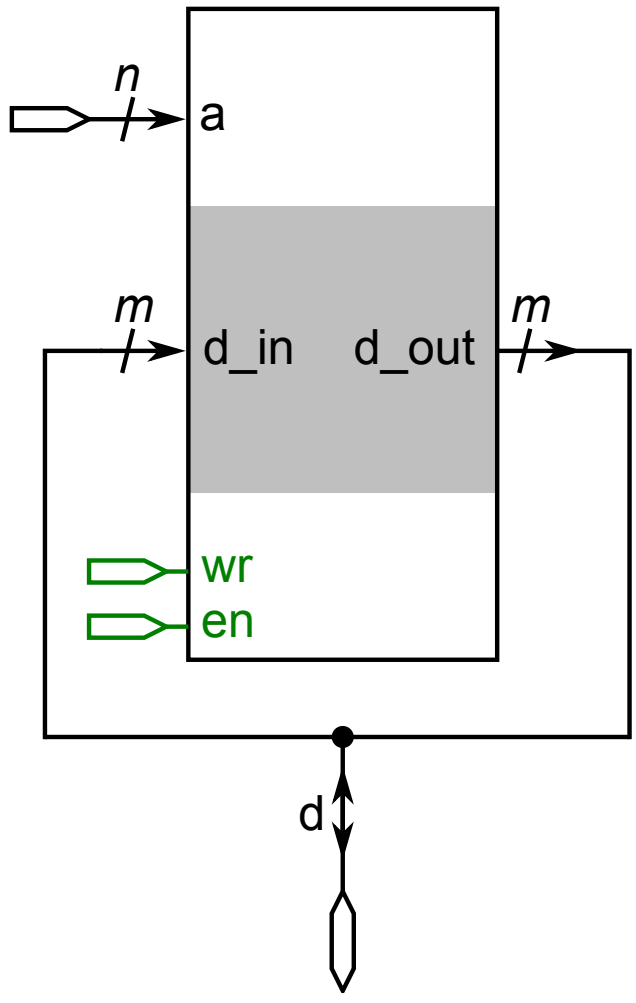
Testbench

```
sig_gen: process
begin                    --@0 ns
  wr<='1';               --Master writes -> Slave reads
  d<=x"1";               --Master drives data, slave drives Z
  wait for 10 ns;        --@10 ns
  wr<='0';               --Slave writes -> Master reads
  d<=(others=>'Z');      --Master drives Z, slave drives data
  wait for 10 ns;        --@20 ns
  wr<='1';
  d<=x"F";
  wait for 10 ns;        --@30 ns
  wr<='0';
  d<=(others=>'Z');
```



Memory with Tristate Ports

- Writing
  - Output port: Z
  - Input port: senses *d*
- Reading
  - Output port: data
  - Input port: senses data (ignored)
- Bidirectional Port *d*



Asynchronous Memory with Bidirectional Bus in VHDL

```
entity bidir_RAM is
  port( d       : inout std_logic_vector(7 downto 0);
        wr, en  : in std_logic;
        addr    : in unsigned(3 downto 0));
end bidir_RAM;

architecture arch of bidir_RAM is
type RAM_16x8 is array(0 to 15)
                    of std_logic_vector(7 downto 0);
signal data : RAM_16x8 :=(others=>x"00"); --Initialization
signal mem_0,mem_1,mem_2 : std_logic_vector(7 downto 0);
begin
  process (addr,wr,en,d)
  begin
    if en='1' then
       if wr='1' then
          d <= (others=>'Z');
          data(to_integer(addr)) <= d;
       else d <= data(to_integer(addr));
       end if;
    else d <= (others=>'Z');
    end if;
  end process;
  mem_0 <= data(0); mem_1 <= data(1); mem_2 <= data(2);
end arch;
```

Testbench

```
sig_gen: process
begin                   --@  0ns
  en<='1'; addr<="0000"; wr<='1'; d<=x"A1";
  wait for 10 ns; --@ 10ns
                          wr<='0'; d<=(others=>'Z');
  wait for 10 ns; --@ 20 ns        ^^^^^^^^^^^^^^^^ write Z!
  en<='0';
  wait for 10 ns; --@ 30 ns
  en<='1'; addr<="0010";
  wait for 10 ns; --40 ns
                          wr<='1'; d<=x"B2";
  wait for 10 ns; --50 ns
           addr<="0001";          d<=x"C3";
  wait for 10 ns; --60 ns
  wait;
end process sig_gen;
```