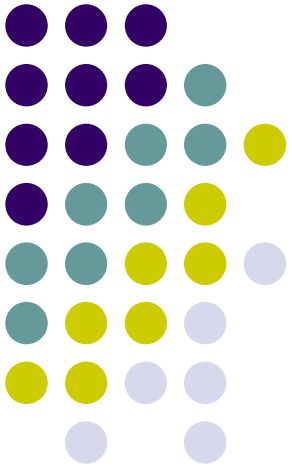
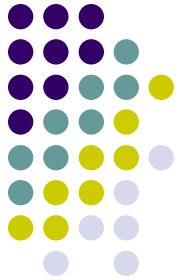
Introduction to VHDL
-component hierarchy

COMP311
Tony McGregor
G.1.05
tonym@cs.waikato.ac.nz

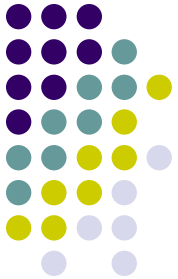


Hierarchy



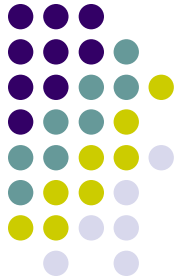
- VHDL is designed to support hierarchical design
  - done by including detailed design into a higher level design
- Build a nand gate from an `and' gate and a `not' gate
- Build a 4-bit (ripple) adder from 4 one bit adders
- Build a CPU from memory, data path, ALU and registers

AND Gate



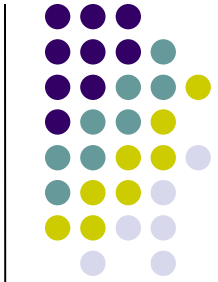
```
library ieee;
use ieee.std_logic_1164.all;
entity and_gate is
     port (
      and_a   : in  std_logic;
      and_b   : in  std_logic;
      and_out : out std_logic
     );
end and_gate;
architecture rtl of and_gate is
begin
  and_out <= and_a and and_b;
end rtl
```

NOT gate in VHDL



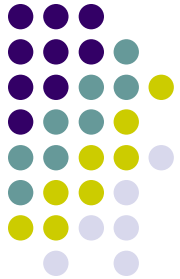
?

NOT gate in VHDL



```
library ieee;
use ieee.std_logic_1164.all;
entity not_gate is
     port (
      not_in   : in std_logic;
      not_out  : in std_logic
     );
end not_gate;
architecture structural of not_gate is
begin
  not_out <= not not_in;
end structural
```

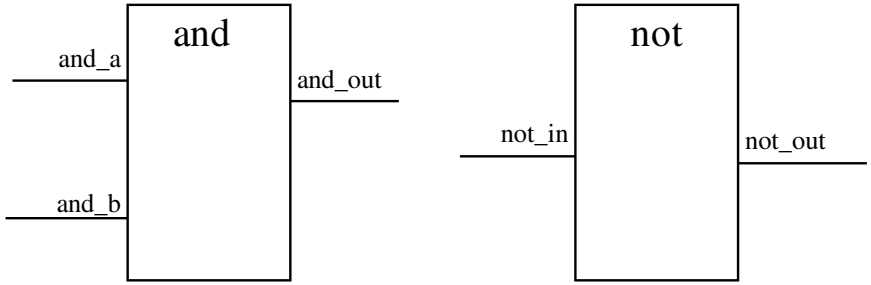
NAND gate

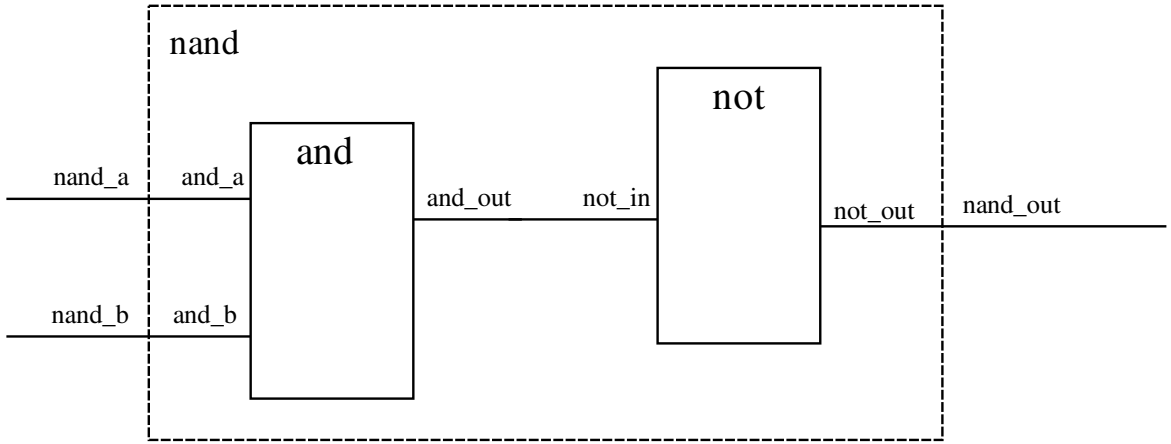


- build nand from not and and
- build a new entity that references not and and definitions.

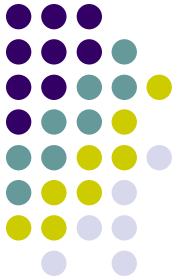
```
component and_gate
  port (
    and_a   : in  std_logic;
    and_b   : in  std_logic;
    and_out : out std_logic
);
```

NAND VHDL


and
and_a
and_b
and_out
not
not_in
not_out

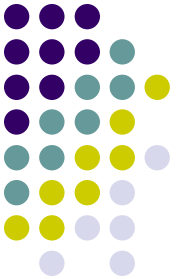

nand
and
not
nand_a
and_a
nand_b
and_b
and_out
not_in
not_out
nand_out

Nand from and and not



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

entity nand_gate is
   port (
     nand_a   : in  std_logic;
     nand_b   : in  std_logic;
     nand_out : out std_logic
     );
end entity nand_gate;

architecture rtl of nand_gate is
  component and_gate
    port (
     and_a  : in  std_logic;
     and_b  : in  std_logic;
     and_o  : out std_logic);
  end component and_gate;

  component not_gate
    port (
     not_in  : in  std_logic;
     not_out : out std_logic);
  end component not_gate;

  signal and_out_i : std_logic;

Begin

 ANDGATE: and_gate
   port map (
     and_a   => nand_a,
     and_b   => nand_b,
     and_out => and_out_i);

  NOTGATE: not_gate
     port map (
       not_in => and_out_i;
       not_out => nand_out);

end architecture rtl;
```

Alternative NAND gate syntax



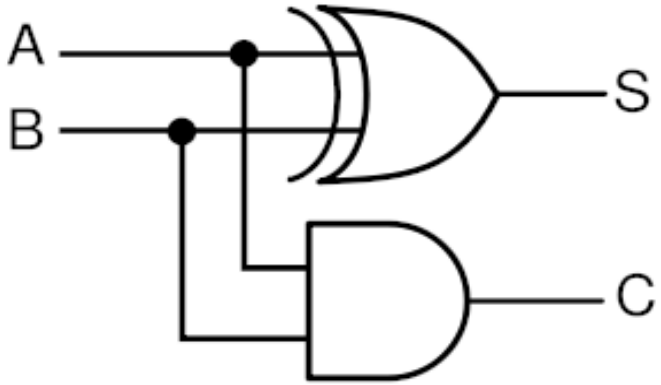
```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
  port (
    nand_a   : in  std_logic;
    nand_b   : in  std_logic;
    nand_out : out std_logic);
end entity nand_gate;

architecture rtl of nand_gate is

  signal and_out_i : std_logic;

begin  -- rtl
  ANDGATE: entity and_gate
     port map (
        and_a   => nand_a,
        and_b   => nand_b,
        and_out => and_out_i);

  NOTGATE: entity not_gate
     port map (
        not_in  => and_out_i,
        not_out => nand_out);
end architecture rtl;
```

Revision:
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;
```

Half Adder

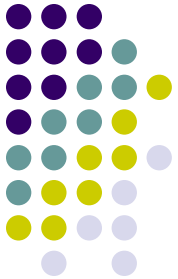
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



S <= A xor B

C <= A and B

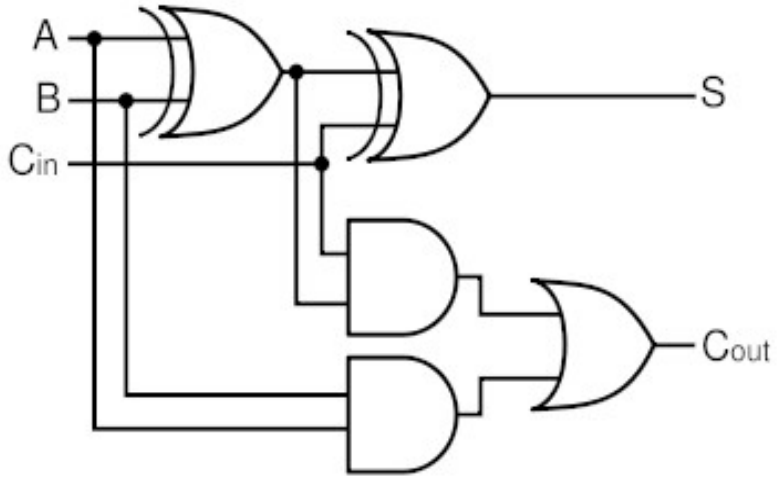
Half Adder VHDL



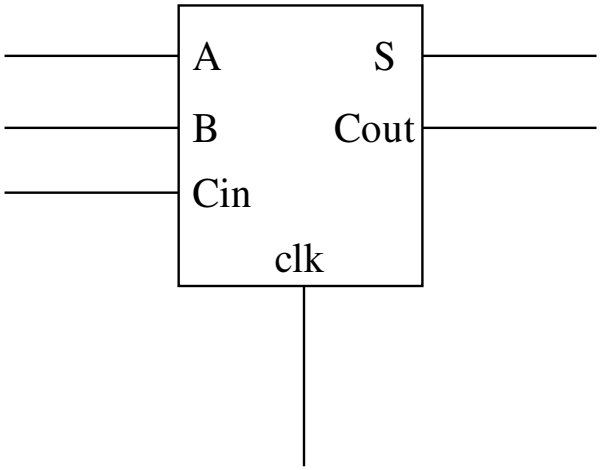
```
library ieee;
use ieee.std_logic_1164.all;
entity half is
     port (
      a   : in std_logic;
      b   : in std_logic;
      s   : out std_logic;
      c   : out std_logic
     );
architecture rtl of half is
begin
  s <= a xor b;
  c <= a and b;
end rtl
```

Full Adder

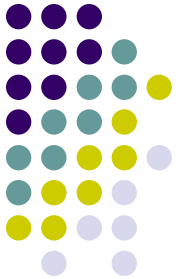
Cin	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



S <= (A xor B) xor Cin
Cout <= ((A xor B) and Cin) or (A and B)

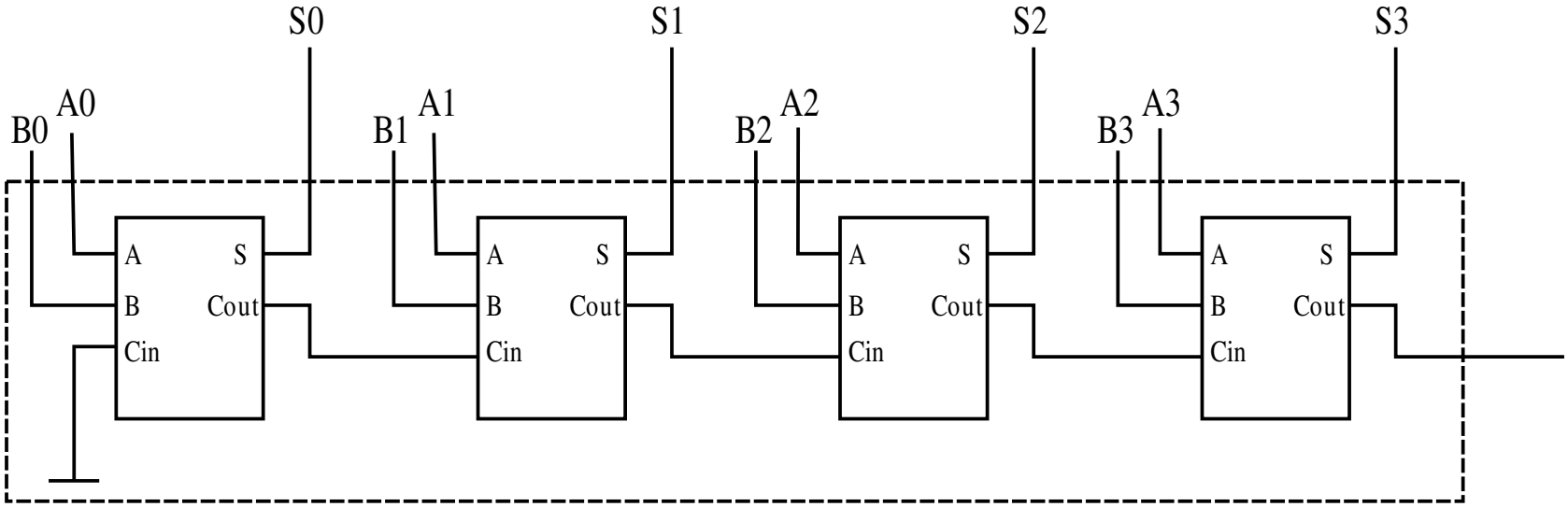


Full Adder VHDL

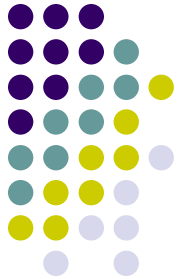


```
library ieee;
use ieee.std_logic_1164.all;
entity full is
     port (
      a    : in  std_logic;
      b    : in  std_logic;
      s    : out std_logic;
      cin  : in  std_logic;
      cout : out std_logic
     );
end full;
architecture rtl of full is
begin
  s    <= (a xor b) xor cin;
  cout <= ((a xor b) and cin) or (a and b);
end rtl
```

4-bit adder

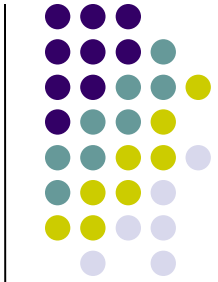

S0
S1
S2
S3
B0
A0
B1
A1
B2
A2
B3
A3
A
S
B
Cout
Cin
A
S
B
Cout
Cin
A
S
B
Cout
Cin
A
S
B
Cout
Cin

4-bit adder VHDL



```
entity add4 is
  port (
    a : in  std_logic_vector(3 downto 0);
    b : in  std_logic_vector(3 downto 0);
    o : out std_logic_vector(3 downto 0);
    carry : out std_logic);
end entity add4;
```

4-bit adder VHDL



```
architecture rtl of add4 is
signal c : std_logic_vector(2 downto 0);
  begin
    bit0: entity full
    port map(a => a(0), b => b(0), s => s(0),
             cin => '0', cout => c(0));

    bit1: entity full
    port map(a => a(1), b => b(1), s => s(1),
             cin => c(0), cout => c(1));

    bit2: entity full
    port map(a => a(2), b => b(2), s => s(2),
             cin => c(1), cout => c(2));

    bit3: entity full
    port map(a => a(3), b => b(3), s => s(3),
             cin => c(2), cout => carry);
end architecture rtl;
```