DIGITAL LOGIC WITH VHDL
(Fall 2013)
Unit 4

✓ *'Integer' DATA TYPE*

✓ *STRUCTURAL DESCRIPTION*

- Hierarchical design: port-map, for-generate, if-generate.
- Examples: Adders, comparators, multipliers, Look-up Tables, Barrel shifter

Instructor: Daniel Llamocca

✓ INTEGER Data Type

- A signal of type '*integer*' represents a binary number. But we do not specify the number of bits for the signal, only the range of decimal values (this can be very convenient).
- **Example**: 7-segment decoder. The BCD input is an integer from 0 to 9, requiring 4 bits (computed by the synthesizer).
- Drawback: the datatype 'integer' does not allow access to the individual bits (unlike '*std_logic_vector*')

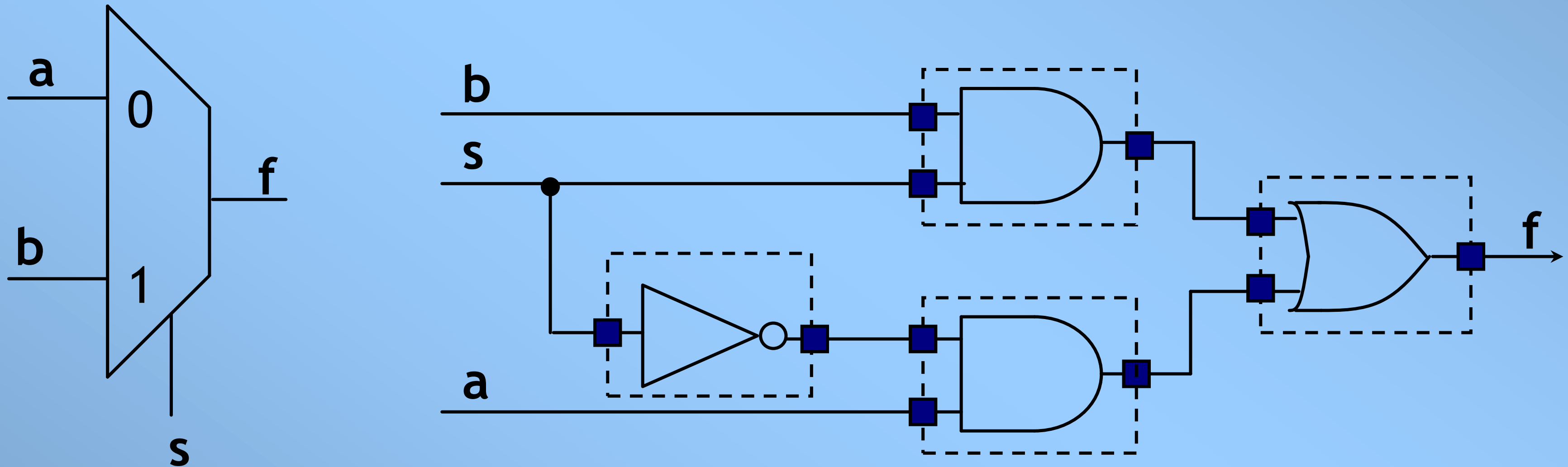
```
library ieee;
use ieee.std_logic_1164.all;

entity sevenseg is
 port ( bcd: in integer range 0 to 9;
        -- bcd: 0000 to 1001 -> 4 bits required
        sevenseg: out std_logic_vector (6 downto 0);
        EN: in std_logic_vector (3 downto 0));
end sevenseg;

architecture struct of sevenseg is
    signal leds: std_logic_vector (6 downto 0);
begin
-- |  a  | b   |  c  |  d  |  e  |  f  |  g  |
-- |leds6|leds5|leds4|leds3|leds2|leds1|leds0|
    with bcd select
       leds <= "1111110" when 0,
               "0110000" when 1,
               "1101101" when 2,
               "1111001" when 3,
               "0110011" when 4,
               "1011011" when 5,
               "1011111" when 6,
               "1110000" when 7,
               "1111111" when 8,
               "1111011" when 9;
    -- Nexys3: LEDs are active low.
    --   Each 7-seg display has an active-low enable
    EN <= "0111";
    sevenseg <= not(leds);
end struct;
```

✓STRUCTURAL DESCRIPTION

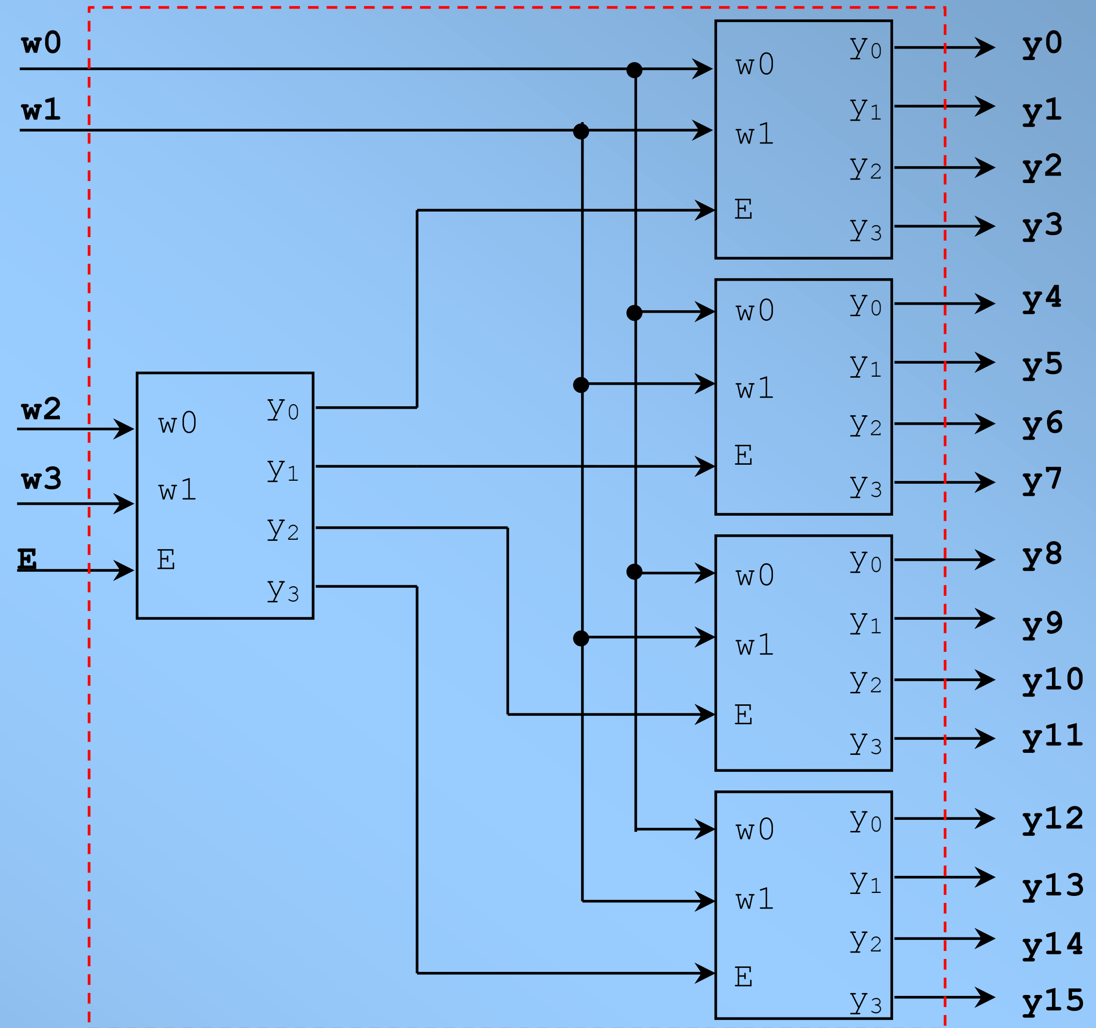
- It is the generalization of the Concurrent Description. The circuits are described via interconnection of its subcircuits. This subcircuits can be described in concurrent code and/or sequential code.
- **Example:** Multiplexor 2-to-1.



- This case is trivial, since the interconnection is realised via logic operators, but nevertheless it is an example of structural description.

✓STRUCTURAL DESCRIPTION

- **Example:** 4-to-16 decoder
- We can describe this decoder in an structured way based on 2-to-4 decoders.
- However, we can also describe the 4-to-16 decoder using the with-select statement.

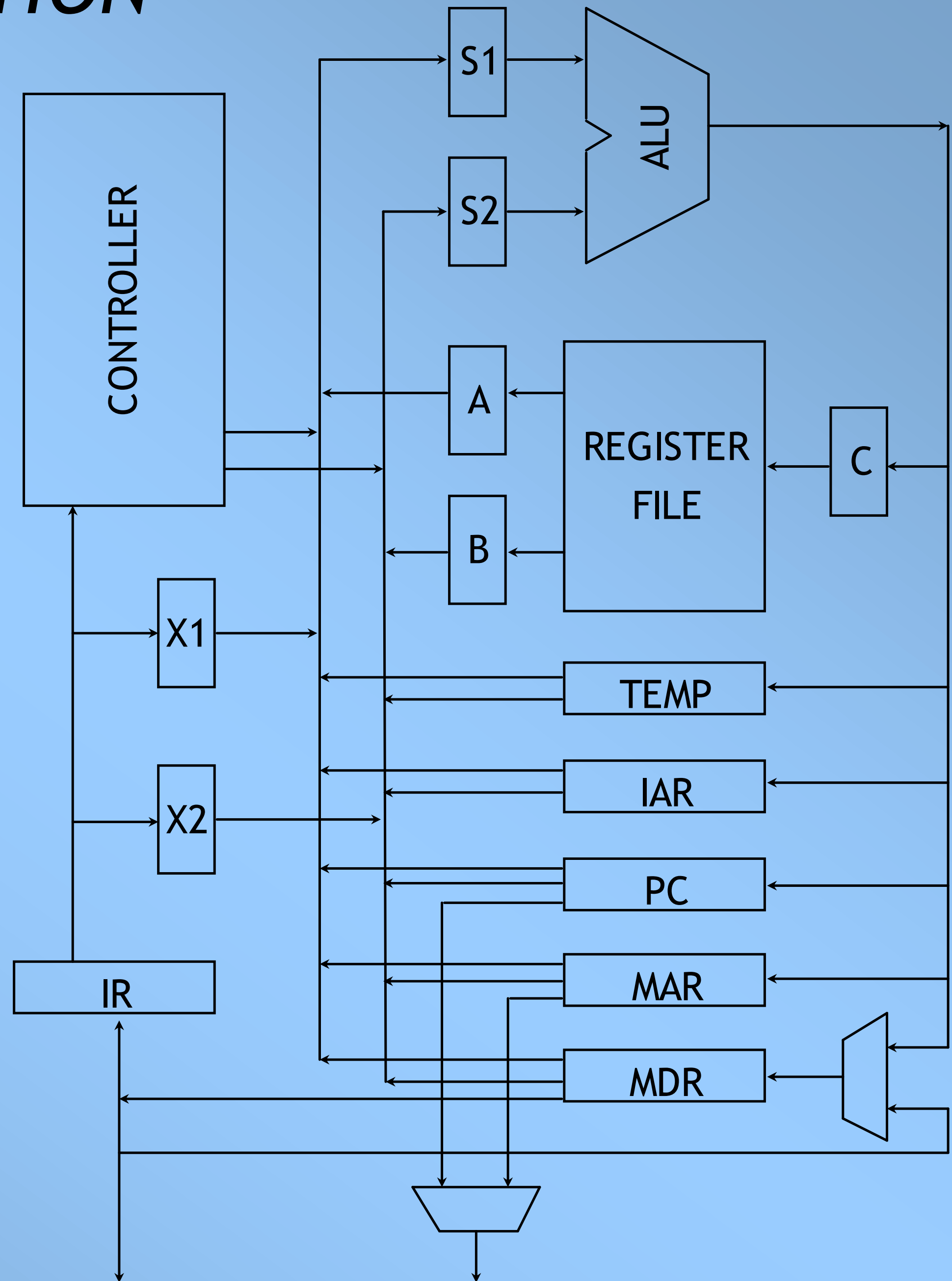


✓STRUCTURAL DESCRIPTION

- **Example:** DLX Processor

- In this type of systems, it is best to describe each component first, then assemble them to make the large system.

- We do not need to see such large system to realise the importance of the Structural Description.



✓ *STRUCTURAL DESCRIPTION*

- Many systems can be described entirely in one single block: we can use the behavioral description, and/or concurrent statements (with-select, when-else).
- However, it is advisable not to abuse of this technique since it makes: i) the code less readable, ii) the circuit verification process more cumbersome, and iii) circuits improvements less evident.
- The structural description allows for a hierarchical design: we can 'see' the entire circuit as the pieces it is made of, then identify critical points and/or propose improvements on each piece.
- It is always convenient to have basic building blocks from which we can build more complex circuits. This also allows building block (or sub-system) to be re-used in a different circuit.

✓ STRUCTURAL DESCRIPTION

- **Example:** 4-bit add/sub for numbers in 2's complement
- The circuit can be described in one single block. However, it is best to describe the Full Adder as a block in a separate file (`full_add.vhd`), then use as many full adders to build the 4-bit adder.
- The place where we use and connect as many full adders as desired, and possibly add extra circuitry is called the 'top file' (`my_addsub.vhd`). This creates a hierarchy of files in the VHDL project:

✓4-bit 2's complement Adder

- Full Adder: VHDL Description (`fulladd.vhd`):

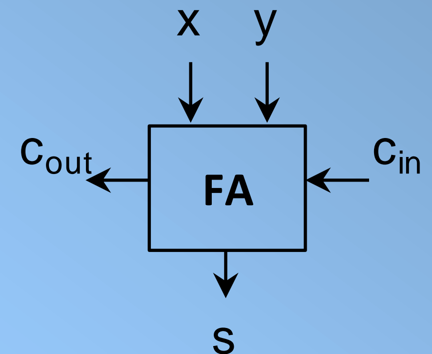
```
library ieee;
use ieee.std_logic_1164.all;

entity fulladd is
 port ( cin, x, y: in std_logic;
          s, cout: out std_logic);
end fulladd;

architecture struct of fulladd is
begin

   s <= x xor y xor cin;
   cout <= (x and y) or (x and cin) or (y and cin);

end struct;
```



✓4-bit 2's complement Adder

- Top file (`my_addsub.vhd`): We need 4 full adders block and extra logic circuitry.

In order to use the file '`fulladd.vhd`' into the top file, we need to declare it in the top file:

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

entity my_addsub is
 port ( addsub: in std_logic;
        x,y: in std_logic_vector(3 downto 0);
        s: out std_logic_vector(3 downto 0);
        cout, overflow: out std_logic);
end my_addsub;

architecture struct of my_addsub is
  component fulladd
     port ( cin, x, y: in std_logic;
            s, cout: out std_logic);
  end component;

  signal c: out std_logic_vector(4 downto 0);
  signal yt: out std_logic_vector(3 downto 0);

begin -- continued on next page
```

We copy what is in the entity of `full_add.vhd`

✓4-bit 2's complement Adder

- Here, we:
  - Insert the required extra circuitry (xor gates and I/O connections).
  - Instantiate the full adders and interconnect them (using the port map statement)

```
-- continuation from previous page
  c(0) <= addsub; cout <= c(4);
  overflow <= c(4) xor c(3);

  yt(0) <= y(0) xor addsub; yt(1) <= y(1) xor addsub;
  yt(2) <= y(2) xor addsub; yt(3) <= y(3) xor addsub;

  f0: fulladd port map(cin=>c(0),x=>x(0),y=>yt(0),s=>s(0),cout=>c(1));
  f1: fulladd port map(cin=>c(1),x=>x(1),y=>yt(1),s=>s(1),cout=>c(2));
  f2: fulladd port map(cin=>c(2),x=>x(2),y=>yt(2),s=>s(2),cout=>c(3));
  f3: fulladd port map(cin=>c(3),x=>x(3),y=>yt(3),s=>s(3),cout=>c(4));

end struct;
```

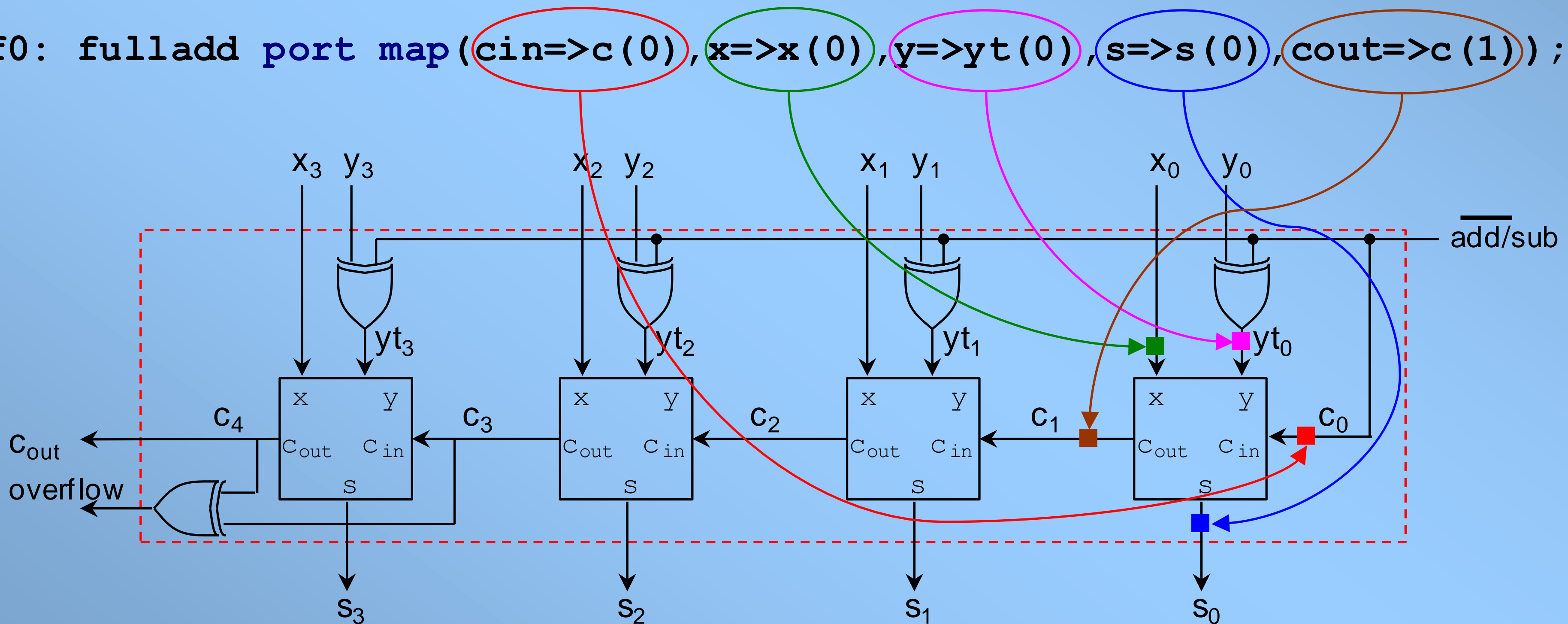
✓4-bit 2's complement Adder

- Use of '***port map***' statement:

```
port map (signal in full adder => signal in top file, ...)
```

- Instantiating and connecting the first full adder:

```
f0: fulladd port map(cin=>c(0),x=>x(0),y=>yt(0),s=>s(0),cout=>c(1));
```



Instructor: Daniel Llamocca

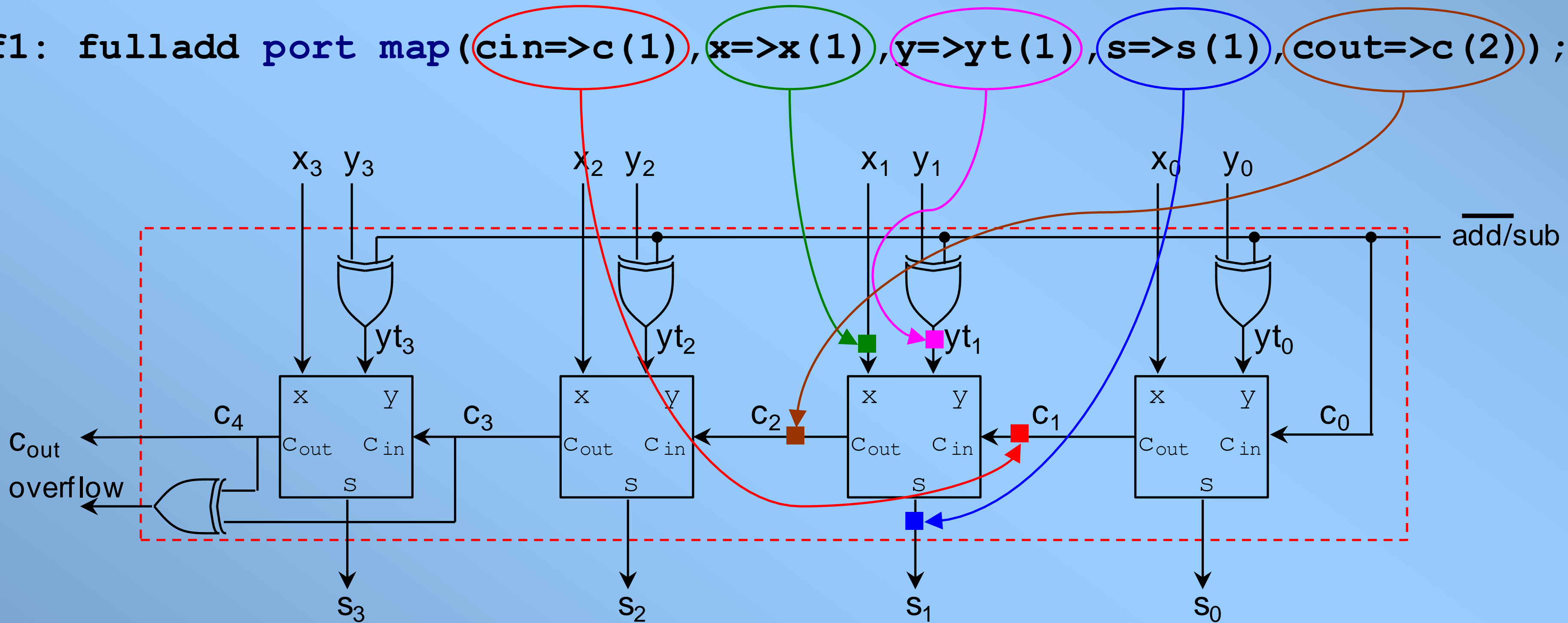
✓4-bit 2's complement Adder

- Use of '***port map***' statement:

```
port map (signal in full adder => signal in top file, ...)
```

- Instantiating and connecting the second full adder:

```
f1: fulladd port map(cin=>c(1),x=>x(1),y=>yt(1),s=>s(1),cout=>c(2));
```

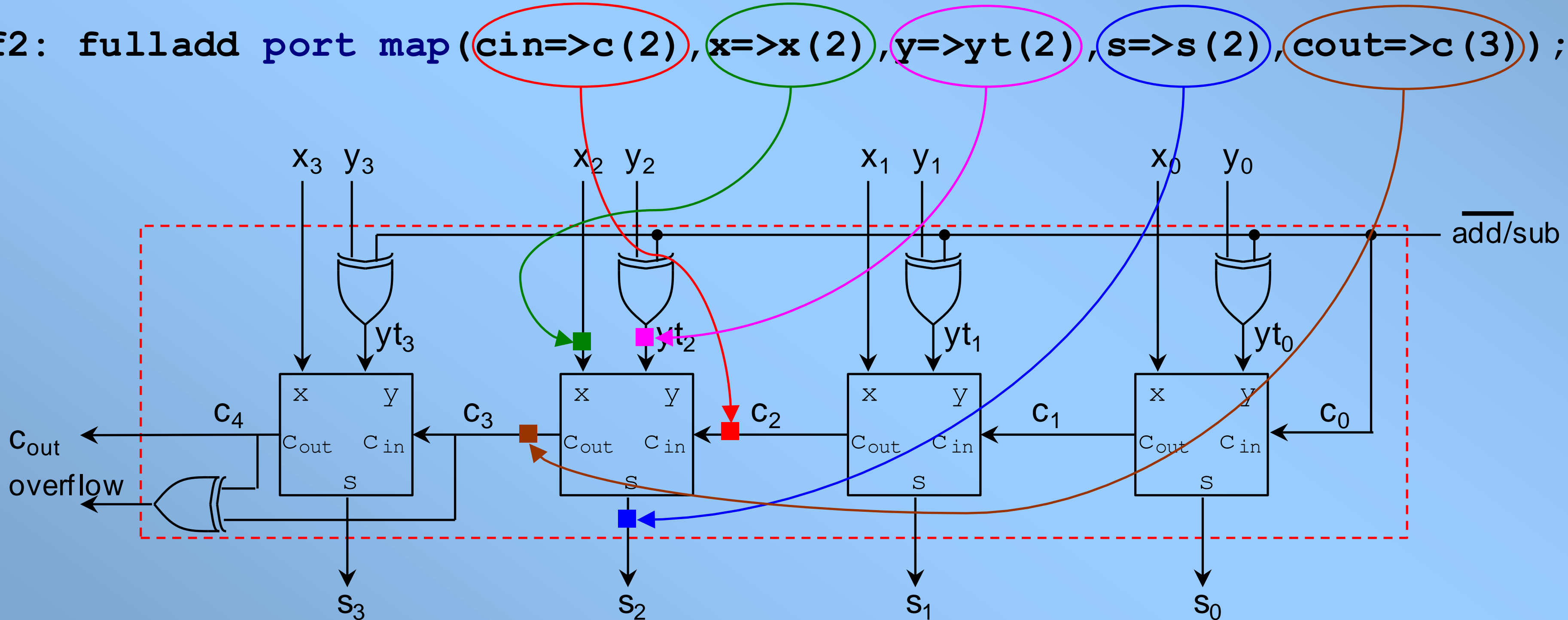


✓4-bit 2's complement Adder

- Use of '***port map***' statement:

  port map (***signal in full adder => signal in top file, ...***)

- Instantiating and connecting the third full adder:



Instructor: Daniel Llamocca

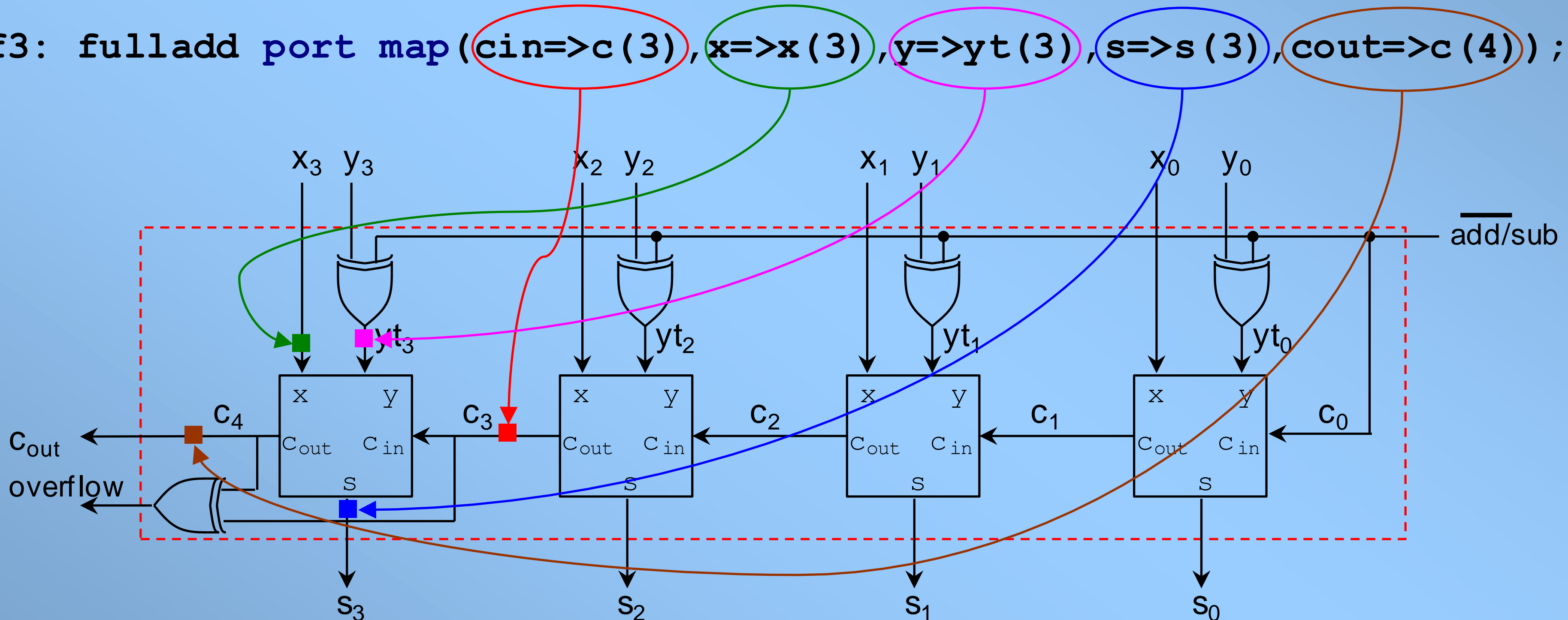
✓4-bit 2's complement Adder

- Use of '***port map***' statement:

```
port map (signal in full adder => signal in top file, ...)
```

- Instantiating and connecting the fourth full adder:

```
f3: fulladd port map(cin=>c(3),x=>x(3),y=>yt(3),s=>s(3),cout=>c(4));
```



✓STRUCTURAL DESCRIPTION

- In the 4-bit adder example, if we wanted to use say 8 bits, we would need to instantiate 8 full adders and write 8 port map statements.
- **For-generate:** Instantiating components can be a repetitive task, thus the ***for-generate*** statement is of great help here:

```
yt(0) <= y(0) xor addsub; yt(1) <= y(1) xor addsub;
yt(2) <= y(2) xor addsub; yt(3) <= y(3) xor addsub;

f0: fulladd port map(cin=>c(0),x=>x(0),y=>yt(0),s=>s(0),cout=>c(1));
f1: fulladd port map(cin=>c(1),x=>x(1),y=>yt(1),s=>s(1),cout=>c(2));
f2: fulladd port map(cin=>c(2),x=>x(2),y=>yt(2),s=>s(2),cout=>c(3));
f3: fulladd port map(cin=>c(3),x=>x(3),y=>yt(3),s=>s(3),cout=>c(4));
```

```
-- continuation from previous page
c(0) <= addsub; cout <= c(4);
overflow <= c(4) xor c(3);

gi: for i in 0 to 3 generate
      yt(i) <= y(i) xor addsub;
      fi: fulladd port map(cin=>c(i),x=>x(i),y=>yt(i),s=>s(i),cout=>c(i+1));
   end generate;
end struct;
```