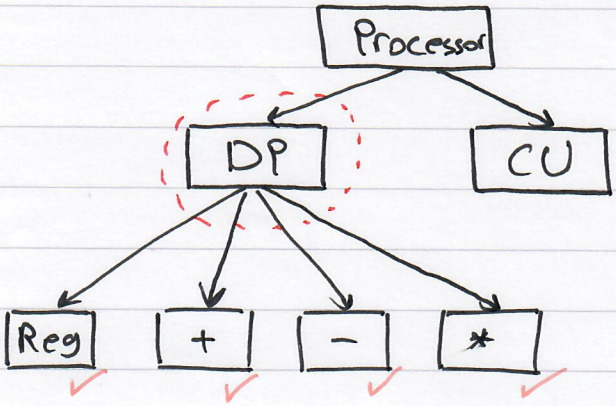
Implement the VHDL code for the SPP designed in Lecture 4.

To implement the SPP fully in VHDL we first look at the hierarchy to know which components need to be described in structural/behavioral VHDL.



We are going for a bottom-up design where we look at the lowest level in the hierarchy.

Since we already have the codes for Reg, Adder, subtractor & multiplier, we will jump one level up and start implementing the DP.

Ali Nawab

```
-- DP in VHDL

library ieee;
use ieee.std_logic_1164.all;

entity DP is
port(
  clk, reset: in std_logic;
  a_ld, b_ld, x_ld, y_ld, z_ld: in std_logic;
  a_i, b_i: in integer;
  z_out: out integer
);
end entity;

architecture structure of DP is

 component Adder is
 port(
    input1, input2: in integer;
    output: out integer
);
end component;

 component Subtractor is
 port(
    input1, input2: in integer;
    output: out integer
);
end component;
```

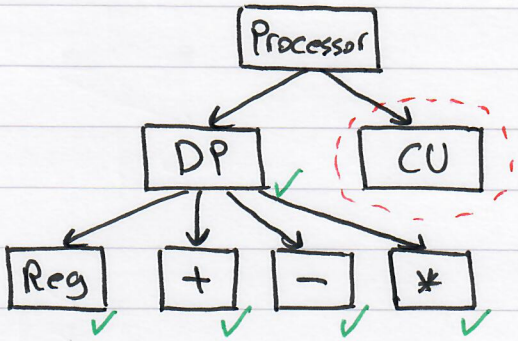
The best way to construct the entity is by looking at the DP as a black box in the processor design [5]

Components declaration is based on the inputs & outputs of each ~~component~~ entity.

```
component Multiplier is
port(
   input1, input2: in integer;
   output: out integer
);
end component;

component Reg is
port(
  clk, reset, load: in std_logic;
  input: in integer;
  output: out integer
);
end component;

Signal a, b, add1, sub1, x, y, mult1: integer;

begin

U0: Reg port map (clk, reset, a_ld, a_i, a);
U1: Reg port map (clk, reset, b_ld, b_i, b);
U2: Adder port map (a, b, add1);
U3: Subtractor port map (a, b, sub1);
U4: Reg port map (clk, reset, x_ld, add1, x);
U5: Reg port map (clk, reset, y_ld, sub1, y);
U6: Multiplier port map (x, y, mult1);
U7: Reg port map (clk, reset, z_ld, mult1, z_out);

end architecture;
```

All wires that connect components together are signals

Port Maps are implemented by looking at the DP [3], each component has its own port map.



-- CU in VHDL

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

entity CU is
port(
   clk, reset: in std_logic;
   a_ld, b_ld, x_ld, y_ld, z_ld: out std_logic
);
end entity;

architecture behavior of CU is

type state is (start, s0, s1, s2, endState);
signal currentS, nextS : state;

begin

clock: process (clk,reset) is
begin
      if(reset ='1') then
          currentS <= start;
      elsif (rising_edge (clk)) then
          currentS <= nextS;
      end if;
end process;
```

Like the DP, the CU entity is constructed from the processor [5]

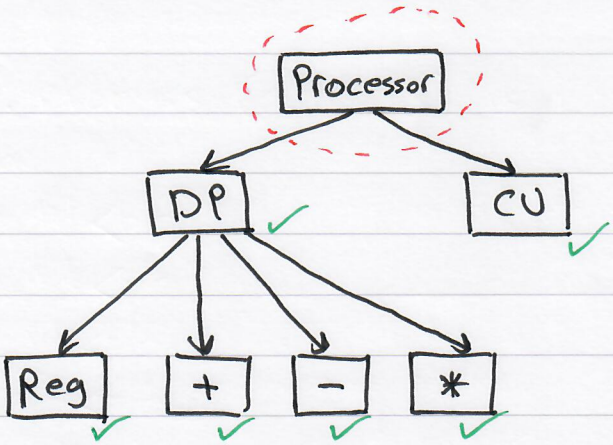
Based on the states from the CU [4]

This process defines the transitions in the FSM by the rising_edge of the clock

```
FSM: process (currentS) is
begin
    a_ld <='0';
    b_ld <='0';
    x_ld <='0';
    y_ld <='0';
    z_ld <='0';

  case currentS is
     when start =>
        nextS <=S0;
     when S0 =>
        a_ld <='1';
        b_ld <='1';
        nextS <=S1;
     when S1 =>
        x_ld <='1';
        y_ld <='1';
        nextS <=S2;
     when S2 =>
        z_ld <='1';
        nextS <=endState;
     when endState =>
        nextS <=endState;
  end case;
 end process;
end architecture;
```

Initialize all loads to '0' so they are closed by default.

The states are obtained from the CU [4]

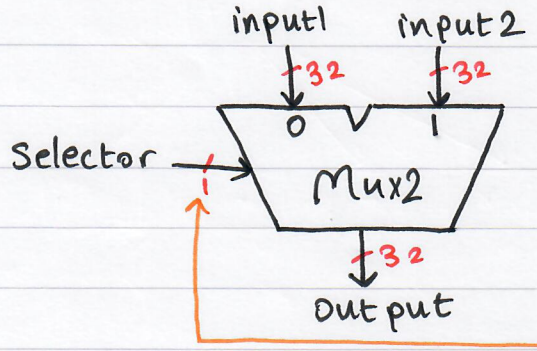


-- Processor in VHDL

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

entity Processor is
port(
   clk, reset: in std_logic;
   a_i, b_i: in integer;
   z_out: out integer
);
end entity;

architecture structure of DP is
   Component DP is .... end Component;
   Component CU is .... end Component;
   Signal a_ld, b_ld, x_ld, y_ld, z_ld: std_logic
begin
  U0: CU port map (clk, reset, a_ld, b_ld, x_ld, y_ld, z_ld);
  U1: DP port map(clk, reset, a_ld, b_ld, x_ld, y_ld, z_ld, a_i, b_i, z_out);
end architecture;
```

The same concepts that apply on DP will apply on the Processor.

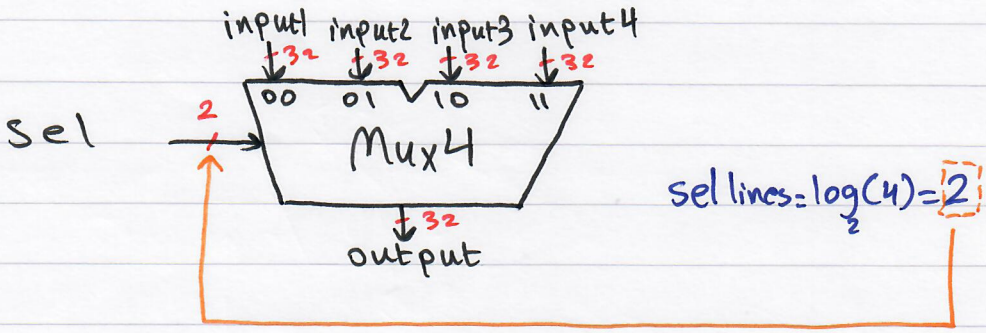
Multiplexer 2x1 (behavioral)



Select lines = $\log_2(\#inputs) = \log_2(2) = 1$

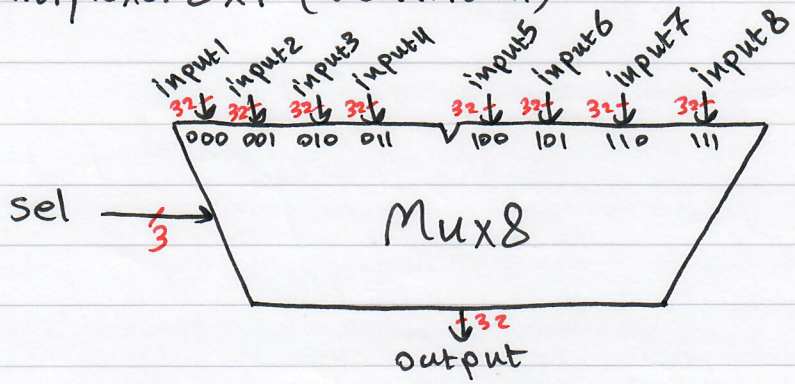
```
library ieee;
use ieee.std_logic_1164.all;

entity Mux2 is
port(
  sel: in std_logic;
  input1, input2: in integer;
  output: out integer
);
end entity;

architecture behavior of Mux2 is
begin
   with sel select
      output <= input1 when '0',
                input2 when others;
end architecture;
```

Multiplexer 4x1 (behavioral)




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

entity Mux4 is
port(
  sel: in std_logic_vector(1 downto 0);
 input1, input2, input3, input4: in integer;
 output: out integer
);
end entity;

architecture behavior of Mux4 is
begin
   with sel select
      output <= input1 when "00",
                input2 when "01",
                input3 when "10",
                input4 when others;
end architecture;
```

Multiplexer 8x1 (behavioral)



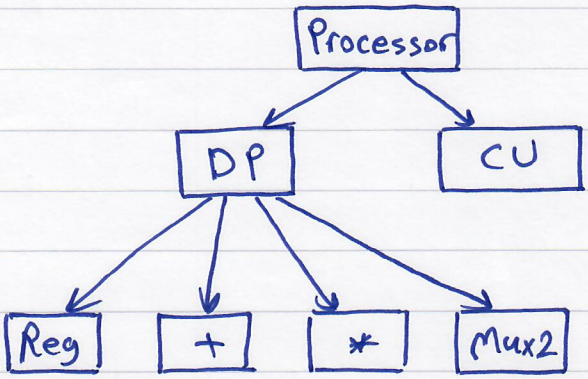
Design a SPP for the following code:

```
int a, b, c, x, y;
x = a + b * c;
y = x + 3;
x = y * 2;
```

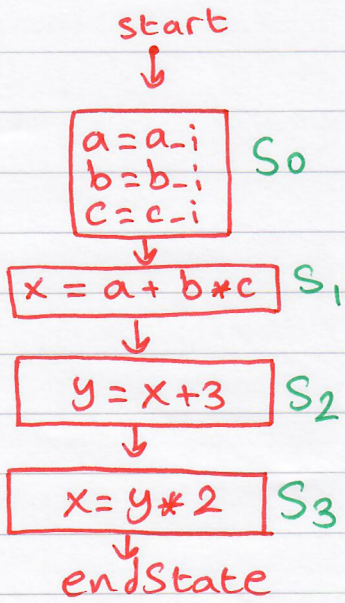
a, b, c are inputs

x, y are outputs

1 Hierarchy



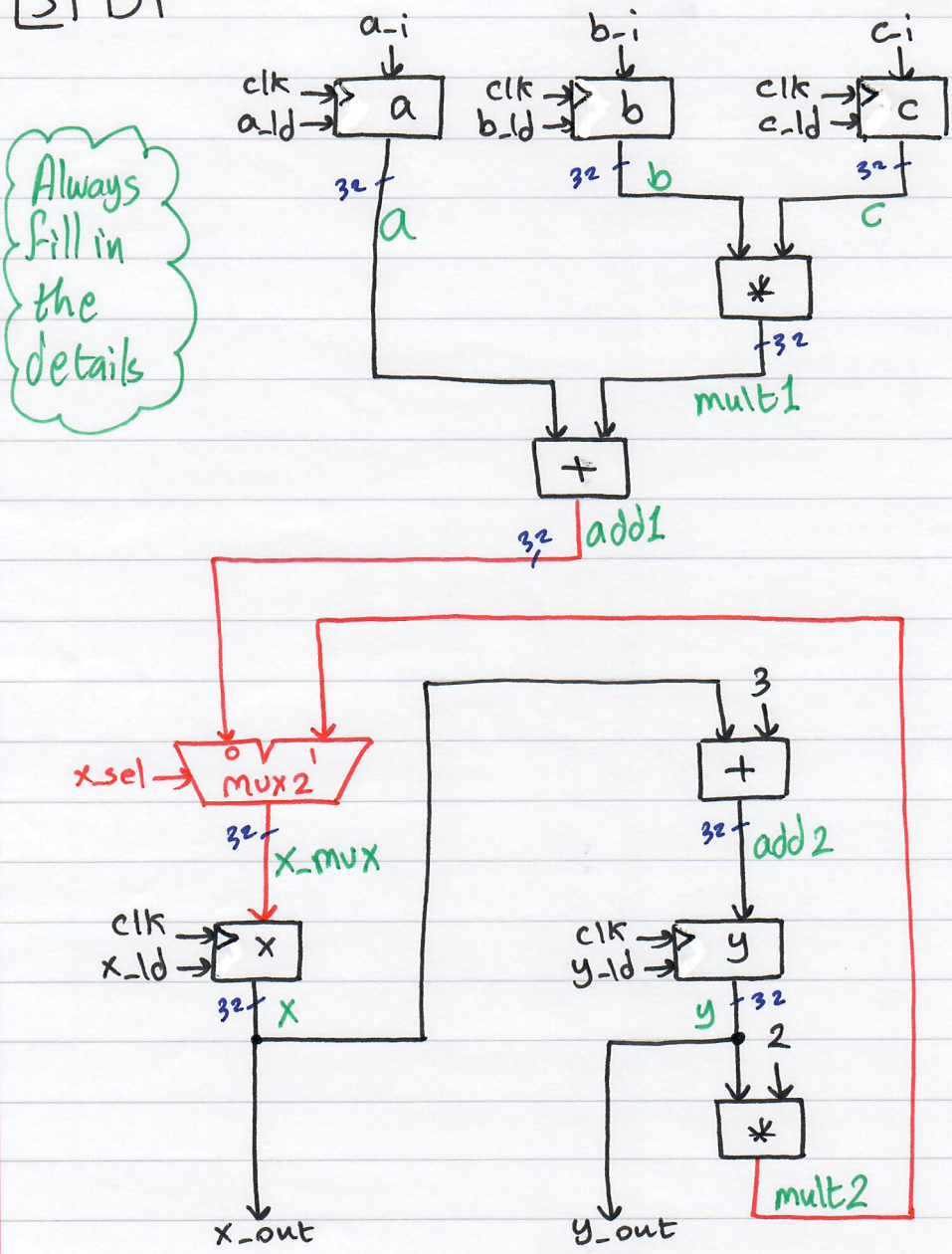
2 FSMD



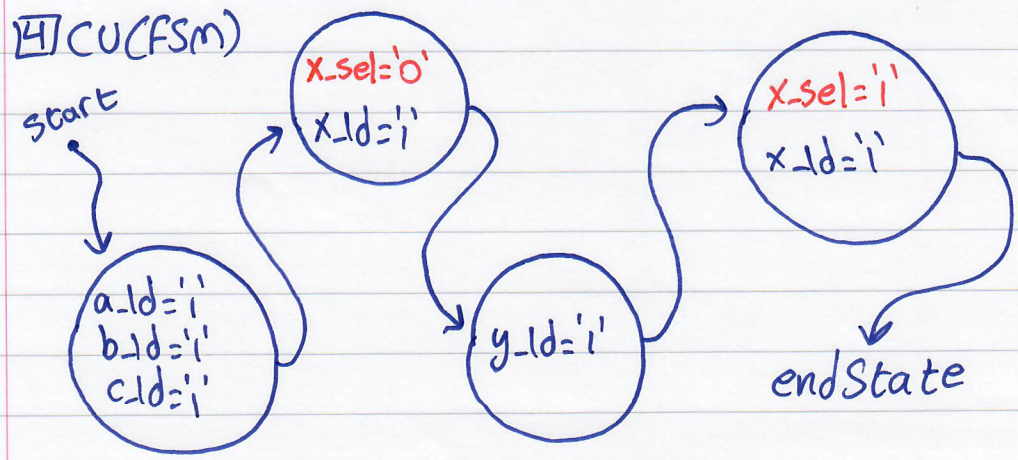
Always fill in the details

3 DP

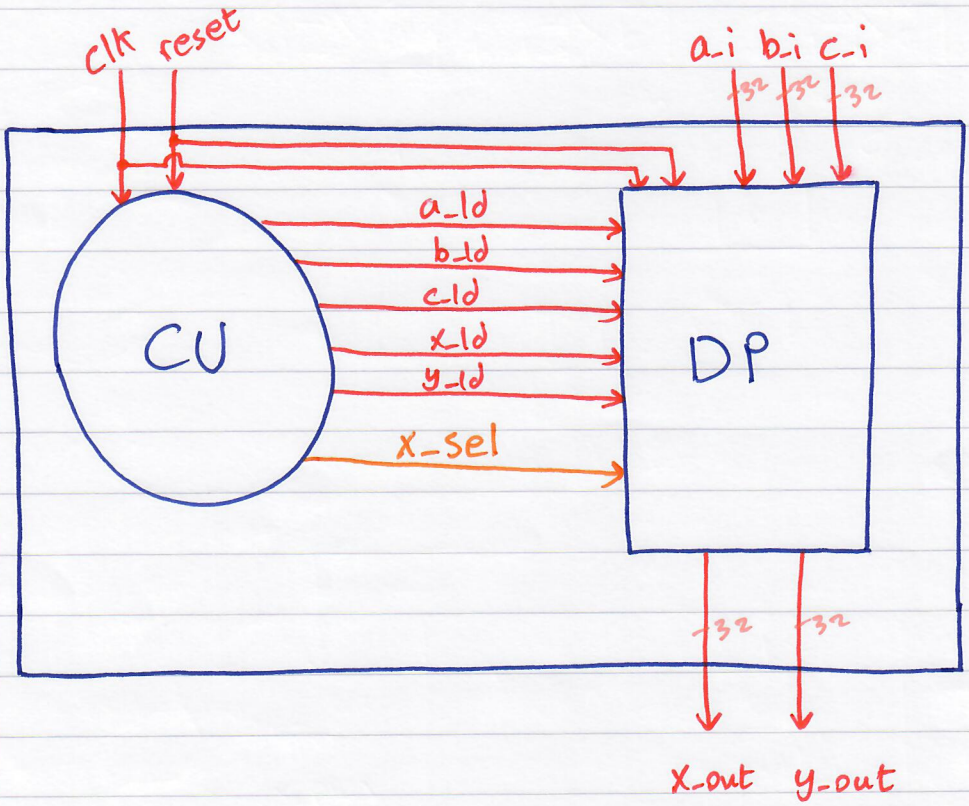
Always fill in the details



4 CU(FSM)



5 Processor



Exercises:

1) Try to implement the previous SPP in VHDL

2) Create an SPP for the following code:

```
X = a*(b-c);
X = X+3;
X = X*X;
X++;
```

a, b, c are inputs
X is the output

Hint: you will need a Mux4 for X.