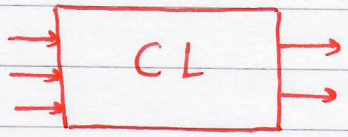
Lecture 4

A peek in the past:

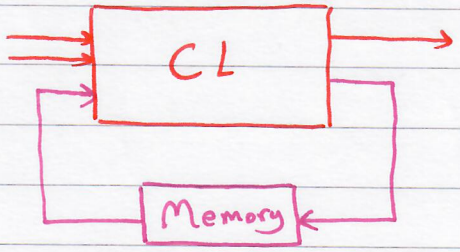
[1] Combinational Logic Circuits:
Circuits where the output is a function of the present input only.

Ex:-Arithmetic/Logical units (adder, subtractor, and)
-Multiplex
-Decoder
-Comparator



[2] Sequential Logic Circuits:
Circuits where the output if a function of the present input and prerious outputs.

Ex: -Flip Flops
- Registers
- Counters



CL with memory state
(SL)

Ali Nawab

Custom Single Purpose Processor

Steps for building a custom SPP:

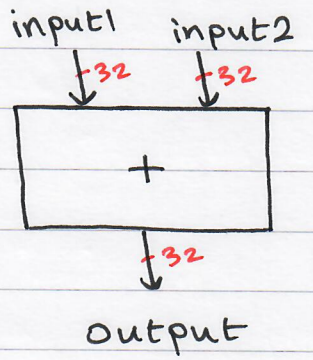
1. Requirements: Carefully read & analyze the requirements which are usually given as code or mathematical formulas (might even be given as plain text!)

2. Design:
   a. Hierarchy: Shows the levels of abstraction for the components of the system.
   b. FSMD (Finite State Machine with Datapath): Describes the flow of code in an FSM with possible parallelism.
   c. DP (datapath): processor components which perform operations and hold data.
   d. CU (Control Unit): The unit which controls the datapath (design as an FSM)
   e. Processor: The whole system where the DP & CU are interconnected.

3. Implementation:
   Converting the design into a synthesizable VHDL code and deploy it on an FPGA

4. Testing:
   Test the correctness of the generated circuit on the FPGA & on ModelSim

Adder (behavioral)

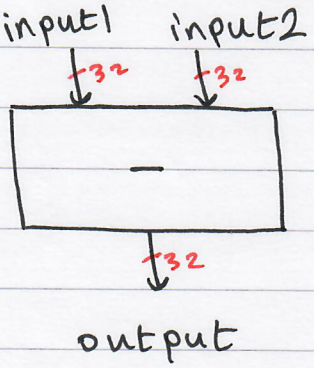



```
library ieee;
use ieee.std_logic_1164.all;

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

architecture behavior of Adder is
begin
   output <= input1 + input2;
end architecture;
```

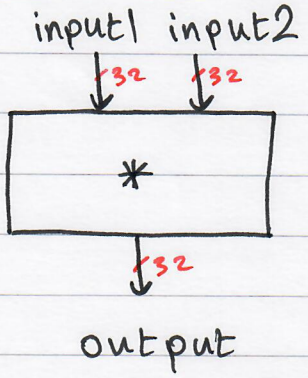
we don't need a library for this file

Note that identefier names in the VHDL code capture the names in the design above

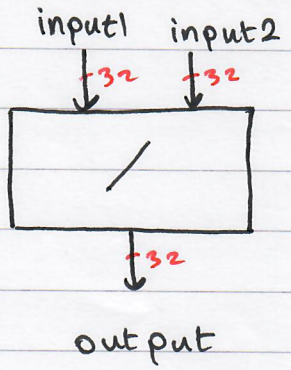
Subtractor (behavioral)



Multiplier (behavioral)

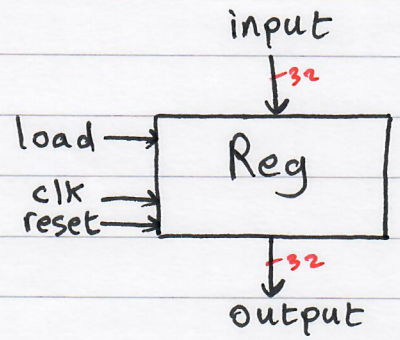


Divider (behavioral)



Register (behavioral)

- parallel load
- edge-triggered




```
library ieee;
use ieee.std_logic_1164.all;
entity Reg is
port(
   clk, reset, load : in std_logic;
   input: in integer;
   output: out integer
);
end entity;
architecture behavior of Reg is
begin
   process (clk, reset) is
   begin
    if (reset = '1') then
        output <= 0;
    elsif(rising_edge(clk) and load = '1') then
        output <= input;
    end if;
   end process;
end architecture;
```

Design & Implement a custom SPP that functions according to the following code:

```
x = a + b;
y = a - b;
z = x * y;
```

* all variables are integers
* z is the output

First let's figure out the inputs & outputs:

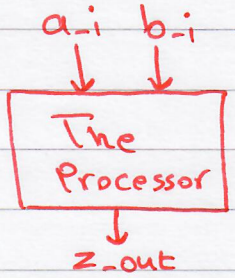
`x = a + b;` → we need the values of a & b but we don't have actual values for them, nor does the code flow indicate any values for them.

∴ a, b are inputs.

`z = x * y` → x, y are already assigned values in the first two lines of code, so they are not inputs.

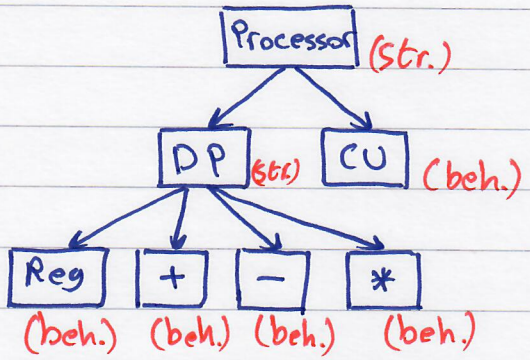
z is given as an output

∴ The system will look like this:

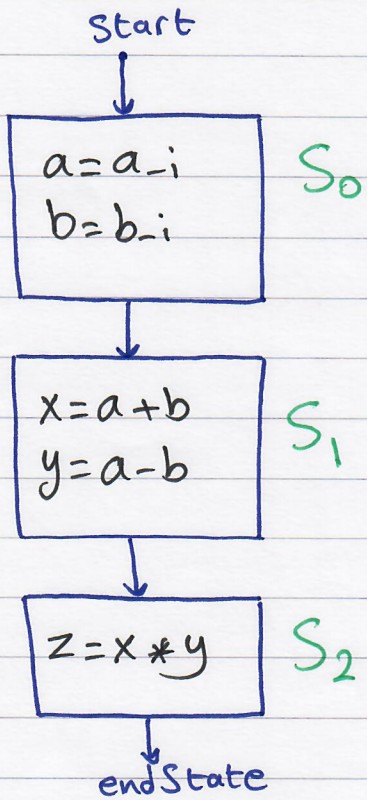


Design:

[1] Hierarchy

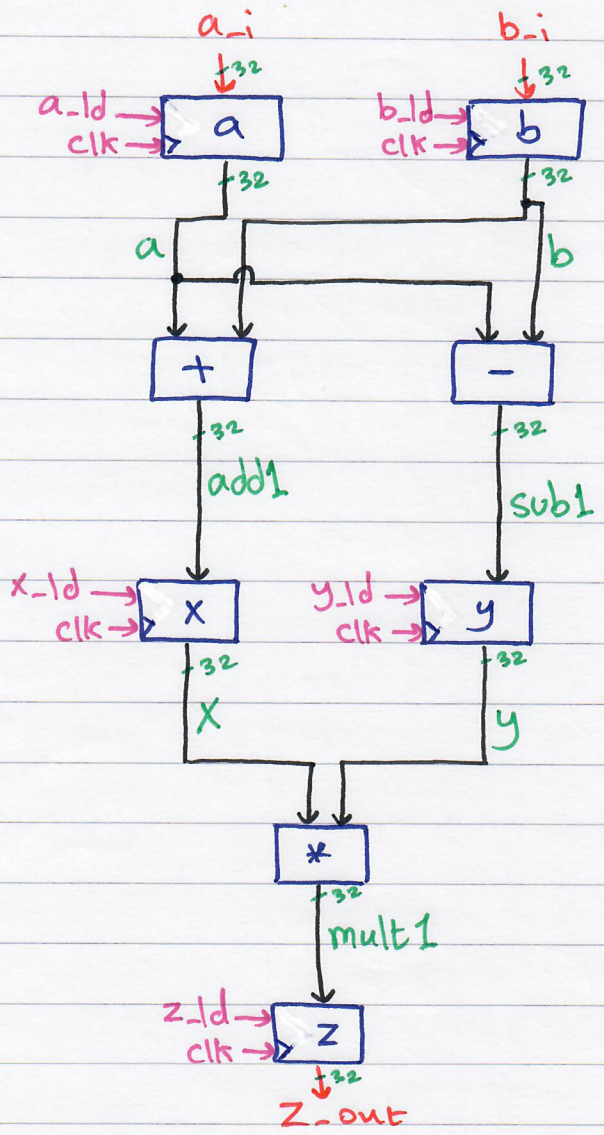


[2] FSMD



Always Fill In The Details !!!

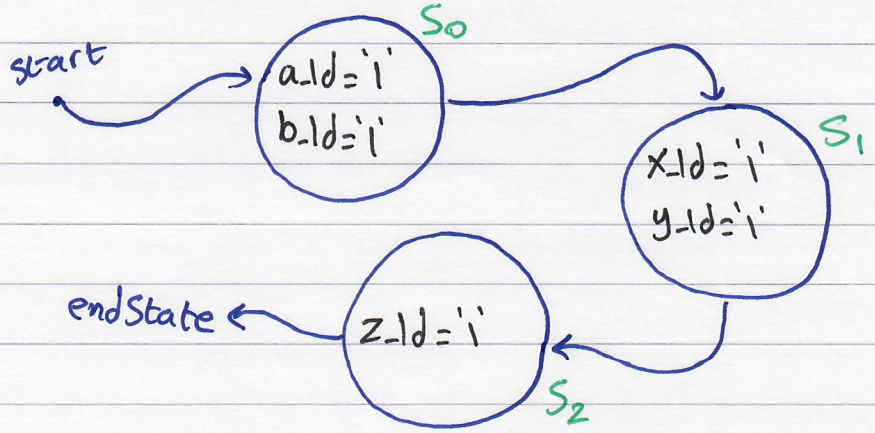
[3] DP



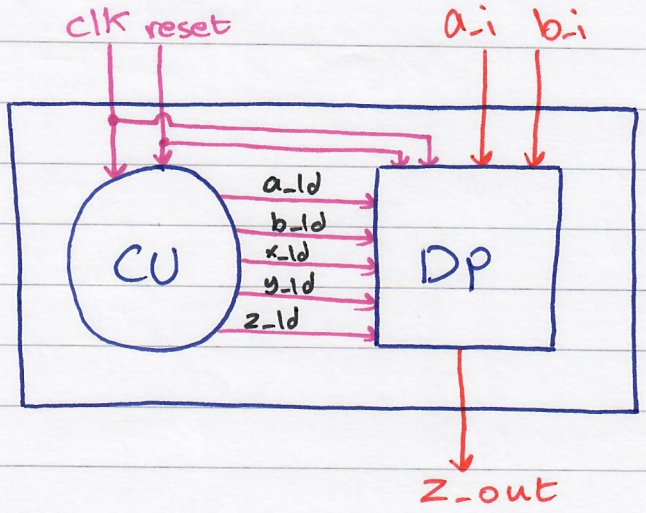
Always Fill In The Details!!!

Filling in more details like signal names make it easier to write VHDL code later

[4] CU (FSM)



[5] Processor



For the implementation part, we will need to write VHDL code for all the components in the hierarchy with details taken from all previous designs, and remember that the implementation must follow the details of the design.

We will stop at the design phase for this example, the next example will include [redacted] implementation.

Exercises:

Create a custom SPP for the following codes

```
x = a + b;
y = x + b;
z = y - a;
```

```
z = a + b;
x = a - b;
z = x + 3;
```