COMPUTER ARCHITECTURE LAB4

PACKAGE AND COMPONENT

FCIS Ainshams University
Spring2021

AGENDA

- Component Mapping Example (hands-on):
  Micro-operations: register load/ reset
- Package and Components
- Assignment: MIPS Package

COMPONENT MAPPING EXAMPLE

Using “FLOP REGISTER” module, implement the following micro-operations

C: $R1 \leftarrow 0$

L: $R2 \leftarrow R1$

How to solve this problem?

DESIGN STEPS…

To design any H/W module:

1. Define the components that will perform the required functionality
   - Ex: Mux's, Dec's, Reg's…etc.
2. Connect them together
3. Define suitable control signals to perform the required functionality
   - Ex: selections of Mux's, Loads of Reg's…etc.

Converting the draw to VHDL code becomes a **one-to-one mapping**

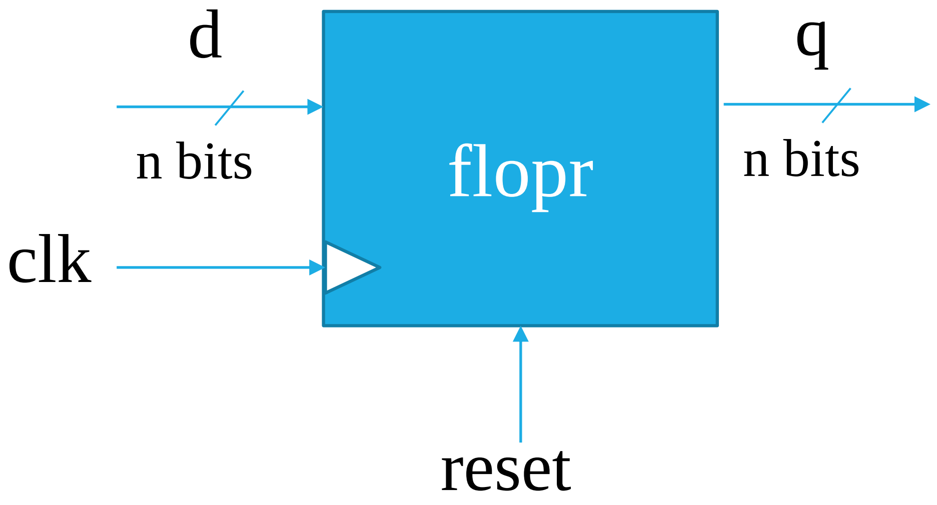
SOLUTION

C: $R1 \Leftarrow 0$

L: $R2 \Leftarrow R1$

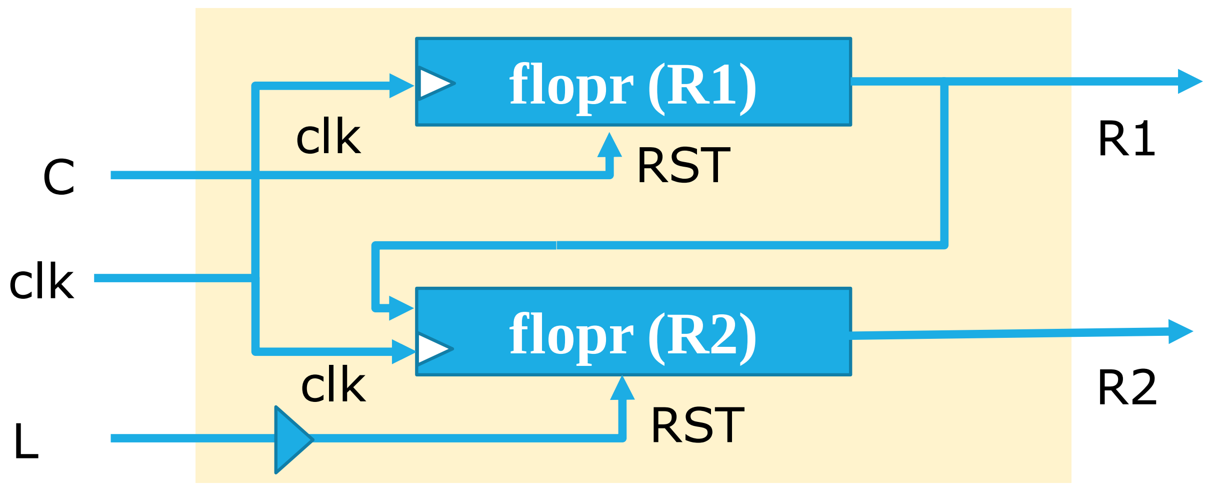
1. Required components:
   - 2 registers
2. Connections:
   - R1 output connected to R2 input ($R2 \Leftarrow R1$)
3. Control signals:
   - When C = 1: RST_{R1} = ‘1’ $\Rightarrow$ $RST_{R1} = C$
   - When L = 1: RST_{R2} = ‘0’ $\Rightarrow$ $RST_{R2} = not(L)$

FLOP REGISTER



SOLUTION

Then, the circuit that implements those micro-operations will look like the below



WAIT

- How can we use the previous implementation of the FLOP REGISTER?!
- We need some kind of reusability for our previous codes
- We need some kind of modularity for our previous codes (registers multiplexers decoders …etc)
- Does VHDL support these needs ?

COMPONENT

- Declaring a code as a COMPONENT, provides a way of code partitioning and reusability.

- A COMPONENT is simply a piece of conventional code that contains (LIBRARY declarations, ENTITY and ARCHITECTURE)

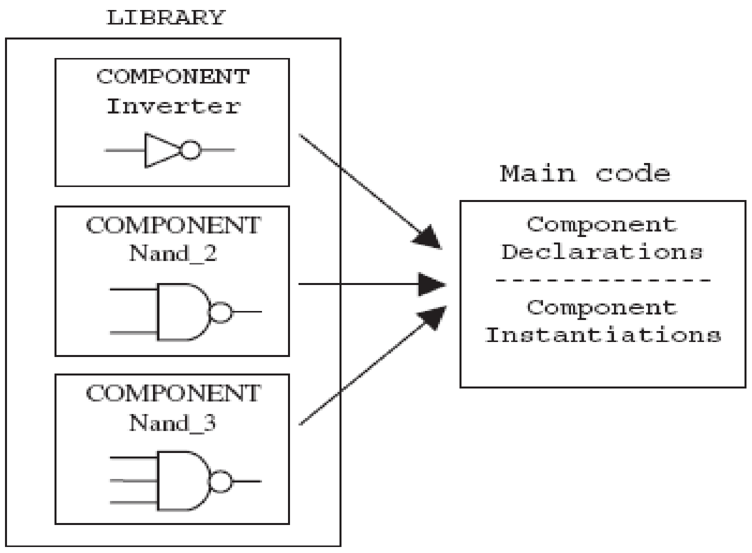
DECLARING COMPONENT

- We can declare the component every time we use it.

- Or, we declare it once in a package and declare (include) this package in our code

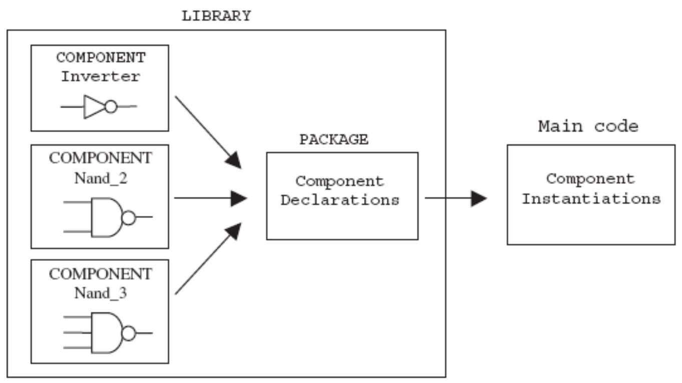
DECLARING COMPONENT

Declare them in the main code itself



DECLARING COMPONENT

- Otherwise, declare them in a PACKAGE
- This avoids repetition of its declaration every time the COMPONENT is instantiated.



FLOP REGISTER(AS AN ENTITY)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity flopr is
    generic (n : NATURAL := 32);
    port(clk, reset: in STD_LOGIC;
    d: in STD_LOGIC_VECTOR(n-1 downto 0);
    q: out STD_LOGIC_VECTOR(n-1 downto 0));
end Flopr;
architecture Behavioral of Flopr is
begin
  process(clk, reset)
  begin
    if reset='1' then q <= (others => '0');
    Elsif rising_edge(clk) then
    q <= d;
    end if;
  end process;
end Behavioral;
```

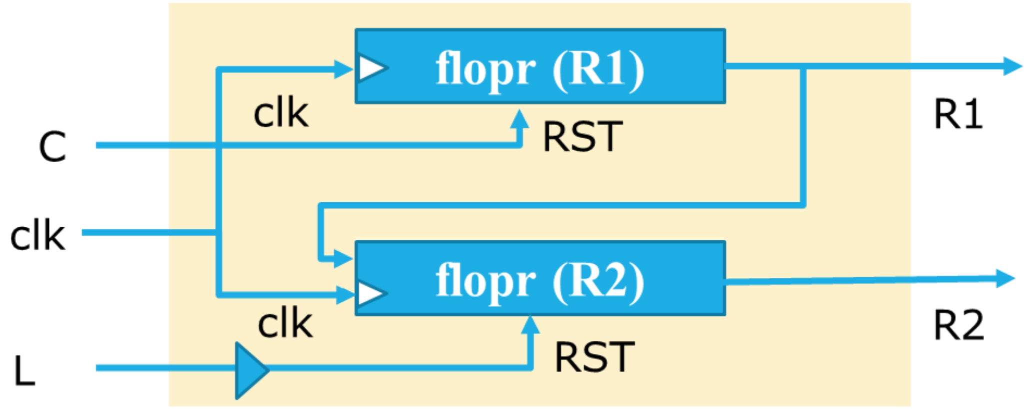
FLOP REGISTER(AS A COMPONENT)

--MyPackage.vhd file

--Define flopr Component (**TYPICAL** as flopr Entity)

```
PACKAGE MyPackage is
Component flopr is
   generic (n : NATURAL := 32);
   port(clk, reset: in STD_LOGIC;
   d: in STD_LOGIC_VECTOR(n-1 downto 0);
   q: out STD_LOGIC_VECTOR(n-1 downto 0));
end Component;
end Behavioral;
```

MAIN MODULE (USES REGISTER)



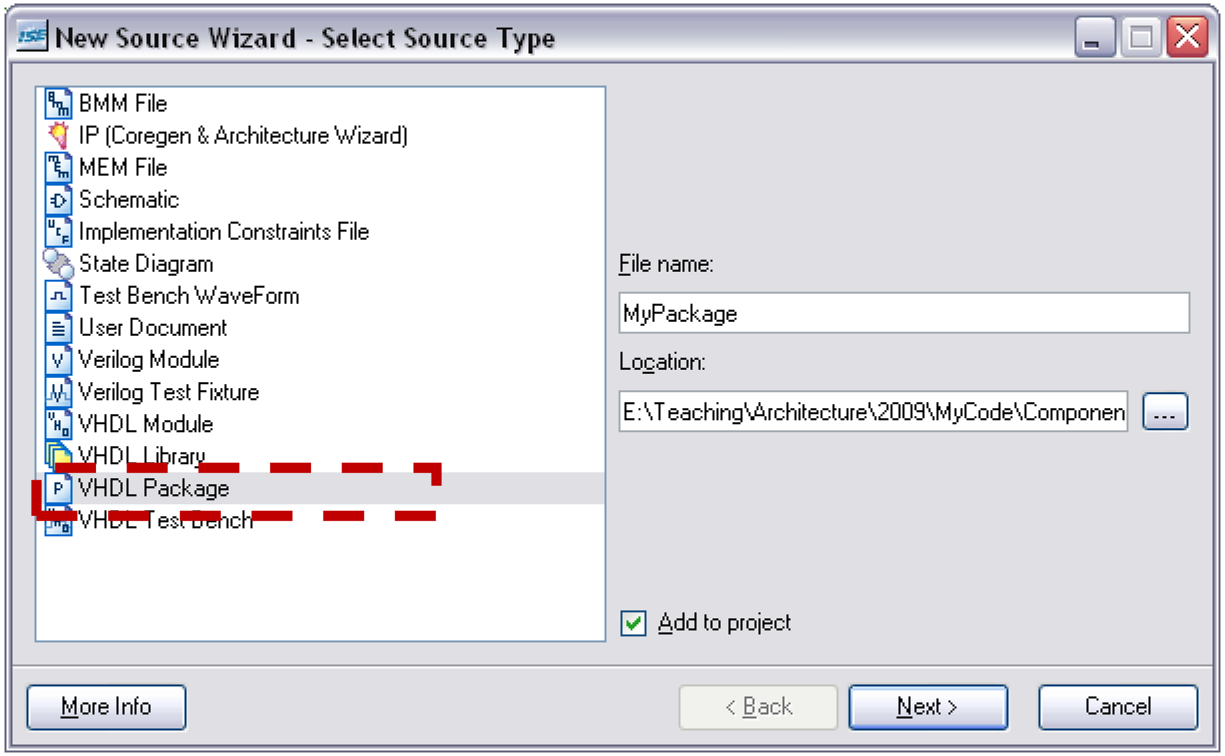
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.MyPackage.all;

entity FloprMainModule is
  generic (n : NATURAL := 32);
     Port ( CLk : in  STD_LOGIC;
           C : in  STD_LOGIC;
           L : in  STD_LOGIC;
           R1 : out  STD_LOGIC_VECTOR (n-1 downto 0);
           R2 : out  STD_LOGIC_VECTOR (n-1 downto 0));
end FloprMainModule;

architecture Behavioral of FloprMainModule is
  SIGNAL Tmp1: STD_LOGIC_VECTOR (n-1 DOWNTO 0);
  SIGNAL Tmp2: STD_LOGIC_VECTOR (n-1 DOWNTO 0);
  Begin
    Tmp1 <= (others => '1');
    R1Map: flopr GENERIC MAP (n) PORT MAP (clk,C,Tmp1,Tmp2);
    R2Map: flopr GENERIC MAP (n) PORT MAP (clk,not(L),Tmp2,R2);
    R1<= Tmp2 ;
end Behavioral;
```

VHDL STEPS

To implement this example in Xilinx, you need to:

- Add the VHDL file of Register (flopr) to the project
- Add a package file to declare a component for it (MyPackage.vhd)
- Write the MainModule that describes the circuit using register component (MainModule.vhd)

ADDING A PACKAGE


New Source Wizard - Select Source Type

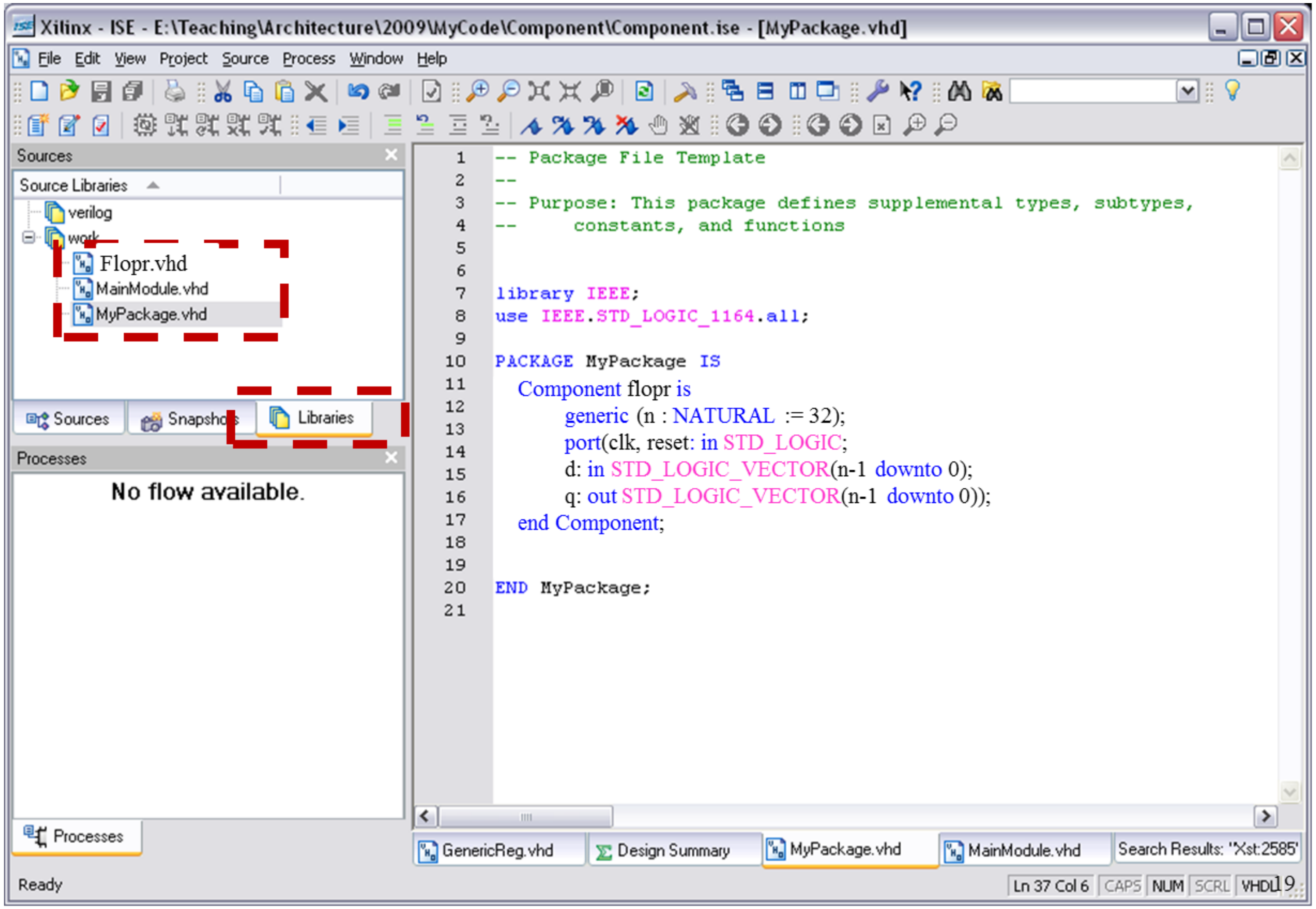
- BMM File
- IP (Coregen & Architecture Wizard)
- MEM File
- Schematic
- Implementation Constraints File
- State Diagram
- Test Bench WaveForm
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench

File name: MyPackage

Location: E:\Teaching\Architecture\2009\MyCode\Componen

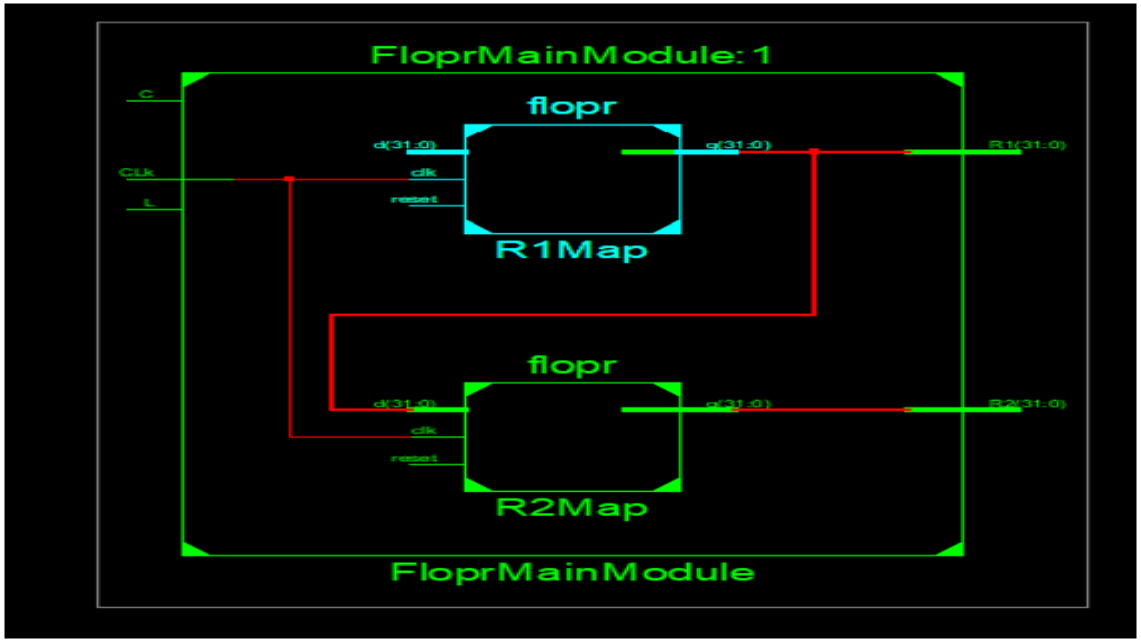
[✓] Add to project

FINAL PROJECT CONTENT



```
-- Package File Template
--
-- Purpose: This package defines supplemental types, subtypes,
--      constants, and functions


library IEEE;
use IEEE.STD_LOGIC_1164.all;

PACKAGE MyPackage IS
  Component flopr is
        generic (n : NATURAL  := 32);
        port(clk, reset: in STD_LOGIC;
        d: in STD_LOGIC_VECTOR(n-1  downto 0);
        q: out STD_LOGIC_VECTOR(n-1  downto 0));
  end Component;


END MyPackage;
```

RTL DESIGN OF MAIN MODULE



TEST BENCH

```
wait for clk_period/2 - 100ns;
l <= ‘1’;
wait for clk_period;
c <= ‘1’;

wait for clk_period*3;
c <= ‘0’;
l <= ‘1’;
wait;
```

ASSIGNMENT: MIPS PACKAGE

- Update “MyPackage” to be used while building the MIPS processor:

Create a component for each module implemented so far:

- flopr (already added)
- mux2
- Sl2
- Signext
- Alu
- Adder

ASSIGNMENT: MIPS PACKAGE

```
component flopr generic(n: integer);
                port(clk, reset: in STD_LOGIC;
                d: in STD_LOGIC_VECTOR(n-1 downto 0);
                q: out STD_LOGIC_VECTOR(n-1 downto 0));
end component;
component mux2 generic(n: integer);
                port(d0, d1: in STD_LOGIC_VECTOR(n-1 downto 0);
                s: in STD_LOGIC;
                y: out STD_LOGIC_VECTOR(n-1 downto 0));
end component;
component sl2
  port(a: in STD_LOGIC_VECTOR(31 downto 0);
      y: out STD_LOGIC_VECTOR(31 downto 0));
end component;
```

ASSIGNMENT: MIPS PACKAGE

```
component signext
   port(a: in STD_LOGIC_VECTOR(15 downto 0);
                   y: out STD_LOGIC_VECTOR(31 downto 0));
end component;
component adder
   port(a, b: in STD_LOGIC_VECTOR(31 downto 0);
           y: out STD_LOGIC_VECTOR(31 downto 0));
end component;
  component alu
   port(a, b: in STD_LOGIC_VECTOR(31 downto 0);
          alucontrol: in STD_LOGIC_VECTOR(2 downto 0);
          result: buffer STD_LOGIC_VECTOR(31 downto 0);
          zero: out STD_LOGIC);
  end component;
```

UPCOMING LABS

- You must bring your package file in addition to the all .vhd files that have been implemented so far:
  - Adder
  - Sl2
  - Signext
  - flopr
  - mux2
  - Alu

Thanks