Session #3

Design Methodologies

- There are two types:
  - Top Down
  - Bottom Up

Top Down:

We define the top level block and identify the sub-blocks necessary to build the top-level block.

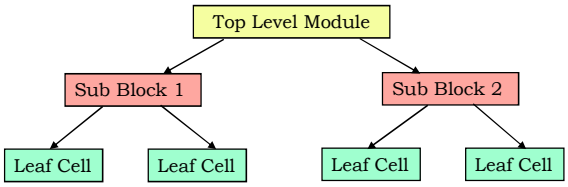
Further, we sub-divide the blocks until we come to leaf cells, which cannot be further divided

Bottom Up:

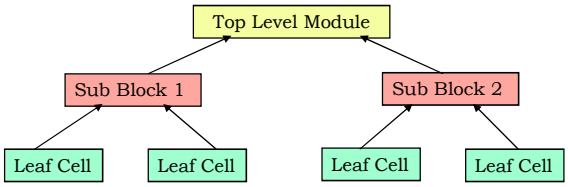
We first identify the building blocks that are available to us. We build bigger cells using these building these blocks.

These cells are used for building higher level blocks until we build the top-level block in the design.

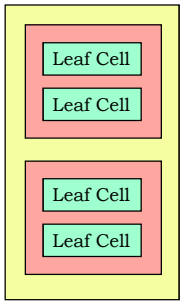
Design Methodologies



Top Down

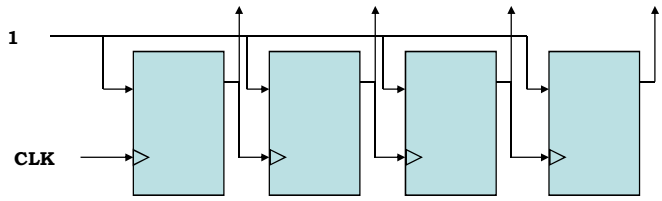


Bottom Up



Final structure

Ripple Counter



- Consider the 4 bit ripple counter
- It is made up of negative edge triggered T-Flip Flops.
- Each of the T-Flip Flops are in turn made up of an inverter and a D-Flip Flop.

What is the best strategy to design and develop the ripple counter in a HDL?

Ripple Counter

Bottom Up Approach

- First Method:
  - Start from the DFF and Inverter. Build a TFF.
  - Using 4 TFF instances build the ripple counter.

Do you have to model or design TFFs four times in this case?

No, one TFF can be designed and four copies of the TFF can be used in the ripple counter design. Each copy of the TFF in ripple counter has the same property has the original single TFF modeled.

- Second Method:
  - Build the ripple counter using 4 TFFs
  - Next design the TFFs using the DFFs and Inverters

Top Down Approach

Do you have to model or design TFFs four times in this case?

Four instances/copies of the TFF can be used in the ripple counter design and later a single TFF can be modeled which will define the properties of the TFF instances in the ripple counter design.

Modules and Module Instances

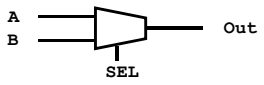
In Verilog:

- ***Module*** defines the entire functionality of the digital circuit in concern.
- ***Module Instance*** is a template to create an object of the original module.
  - Module instance has the same functionality of the original module.
  - It can have the same or different parameters as the original module
- The process of creating objects from a module template is called ***instantiation*** and the objects are called ***instances***.

Creating Verilog Module Instances

2:1 Multiplexer Code:

```
module multiplexer (A, B, Sel, Out) ;
output Out;
input A, B, Sel;
wire S1, S2, Sel_Bar;
   assign S1 = A & Sel_Bar ;
   assign S2 = B & Sel;
   assign Out = S1 | S2;
   assign Sel_Bar = ~Sel;
endmodule
```



Use the module name

Multiplexer Instance:

```
multiplexer mux_instance (
        .A( ),
        .B( ),
        .Sel( ),
        .Out( )
);
```

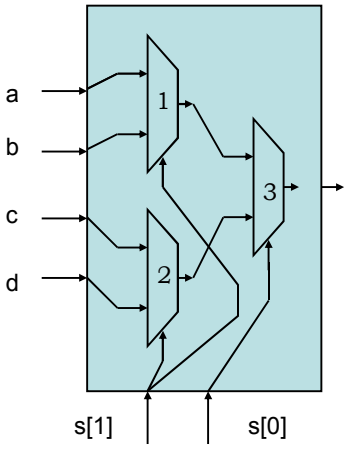
Unique name

Interface to other instances

Any number of instances can be made as per the requirement.

Example:

Constructing a 4:1 Multiplexer with 2:1 Multiplexers




```
module mux_4to1(a, b, c, d, s, out);
        input a,b,c,d;
        input [1:0] s;
        output out;
        wire s1, s2;

multiplexer mux_instance_1 (
        .A( a ), .B( b ),
        .Sel( s[1] ), .Out( s1 )
);
multiplexer mux_instance_2 (
        .A( c ), .B( d ),
        .Sel( s[1] ), .Out( s2 )
);
multiplexer mux_instance_3 (
        .A( s1 ), .B( s2 ),
        .Sel( s[0] ), .Out( out )
);

endmodule
```

Exercise 1

- Design a full adder using two instances of half adders and model it in Verilog.
- Using the full adders model 4-bit full adder in Verilog.
- Verify the 4 bit full adder's functionality in the simulator of your choice.
- What are the corner cases in the functionality of the 4-bit full adder verification?

60 Minutes