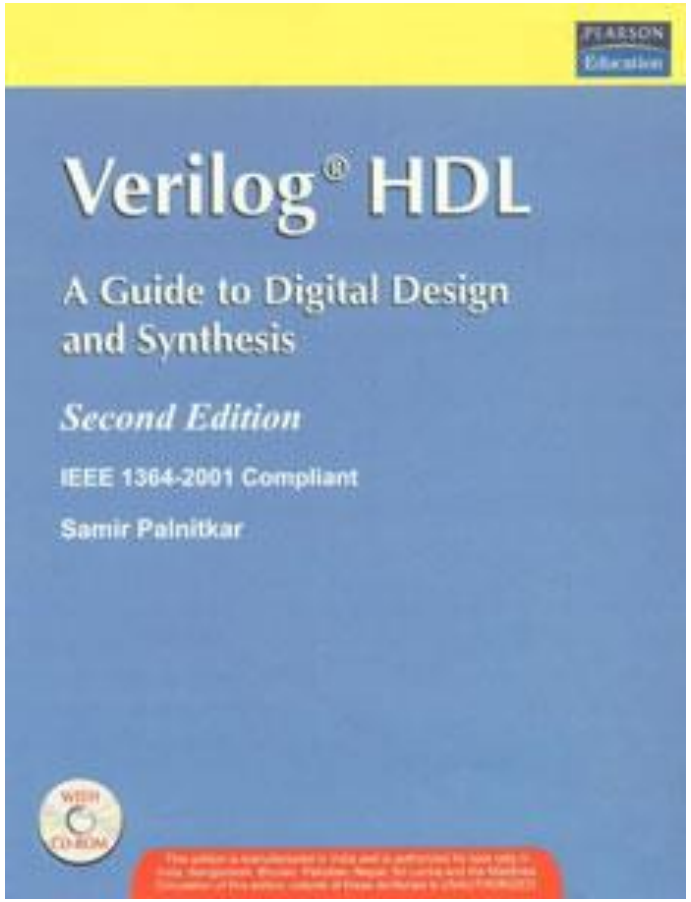
# Verilog HDL

## HIERARCHICAL MODELING CONCEPTS

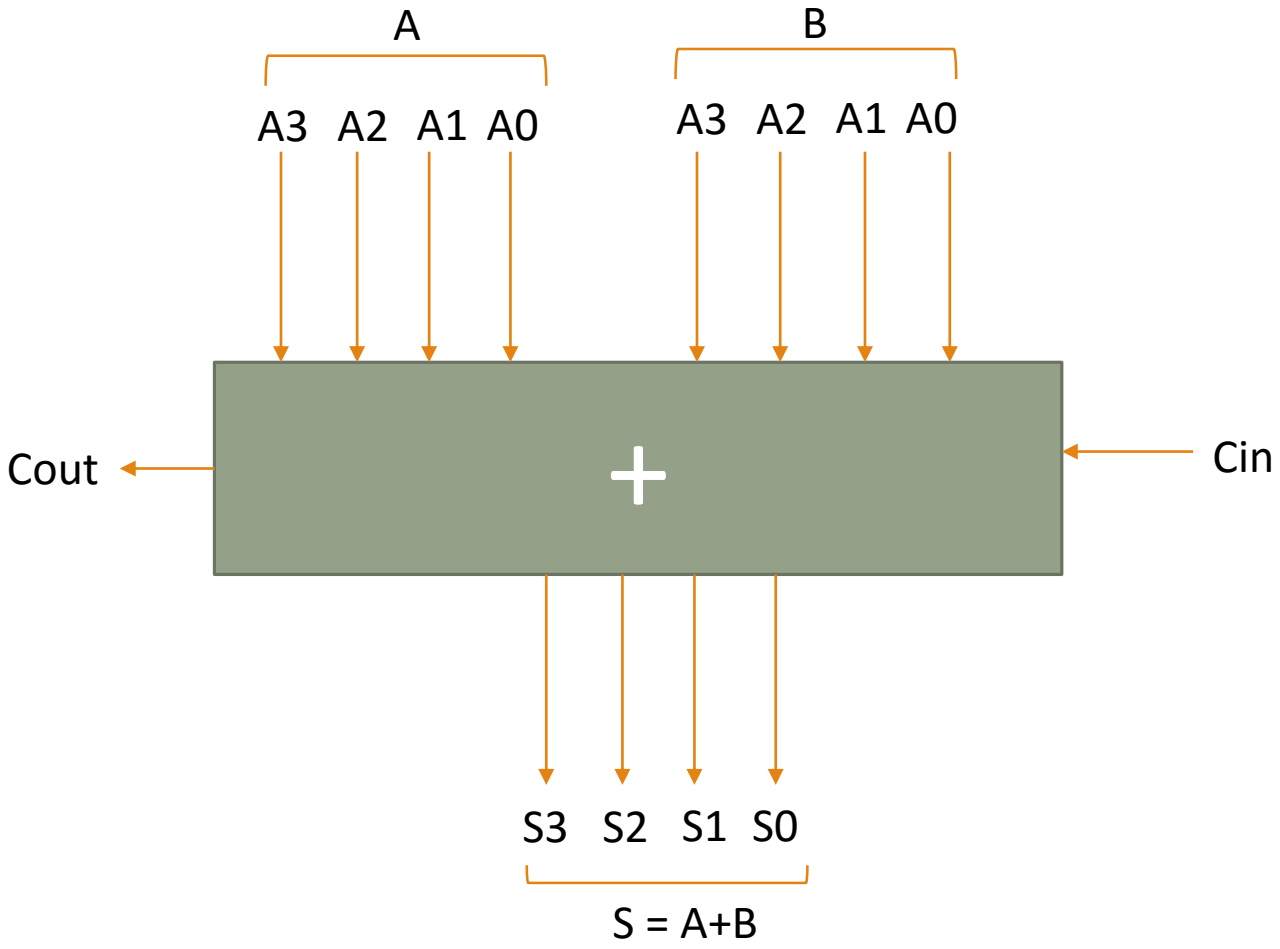
# Reference

Verilog HDL: A Guide to Digital Design and Synthesis, 2e, Samir Palnitkar

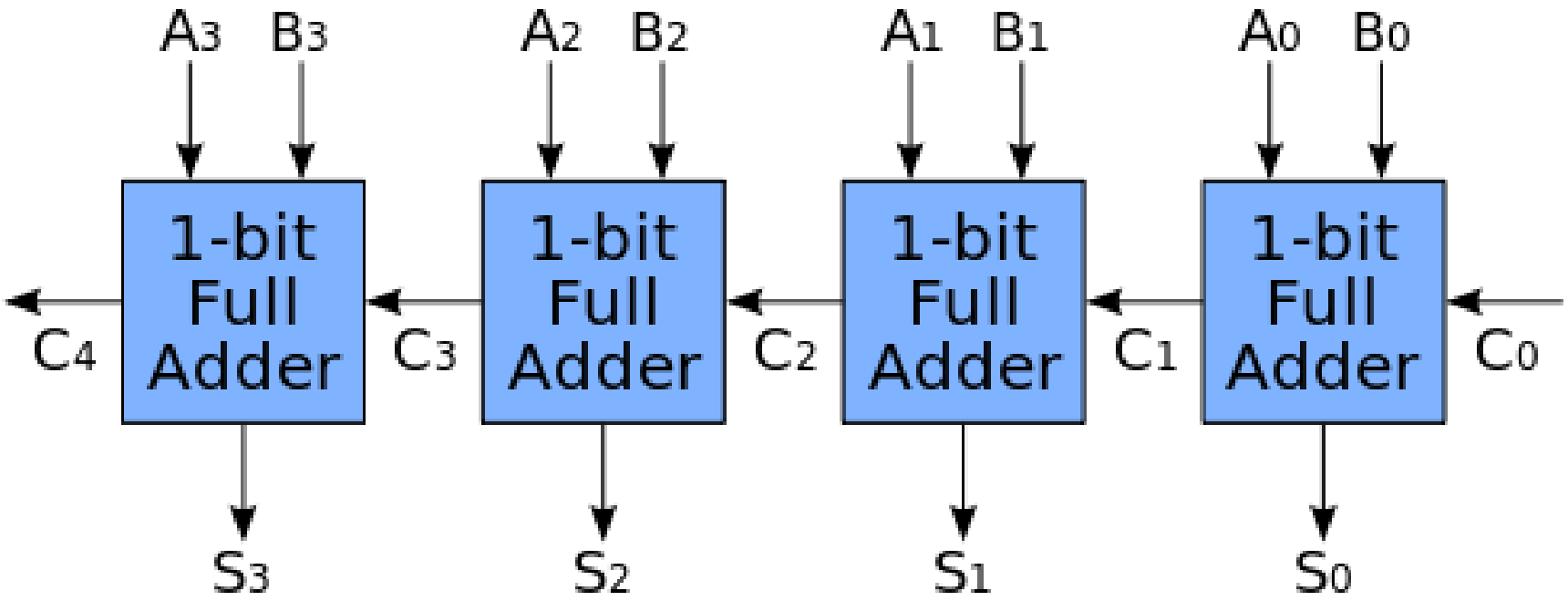
**Chapters 2**



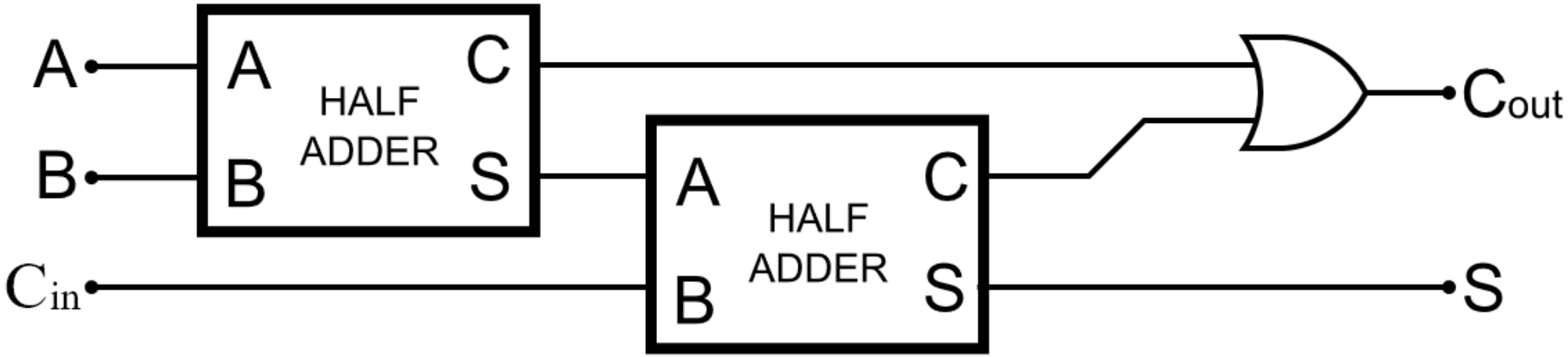
# 4-bit ripple carry adder : top level view



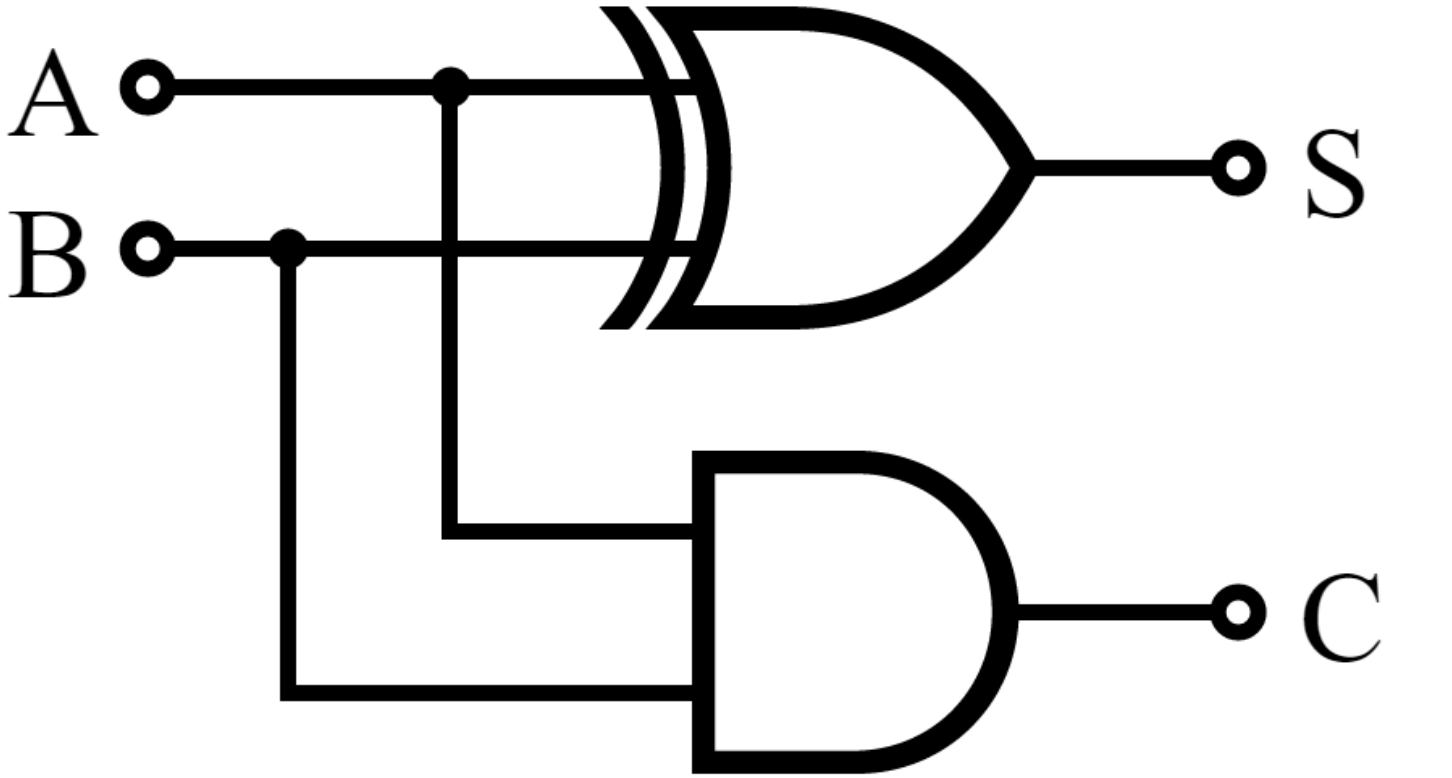
# 4-bit adder implemented using full adders



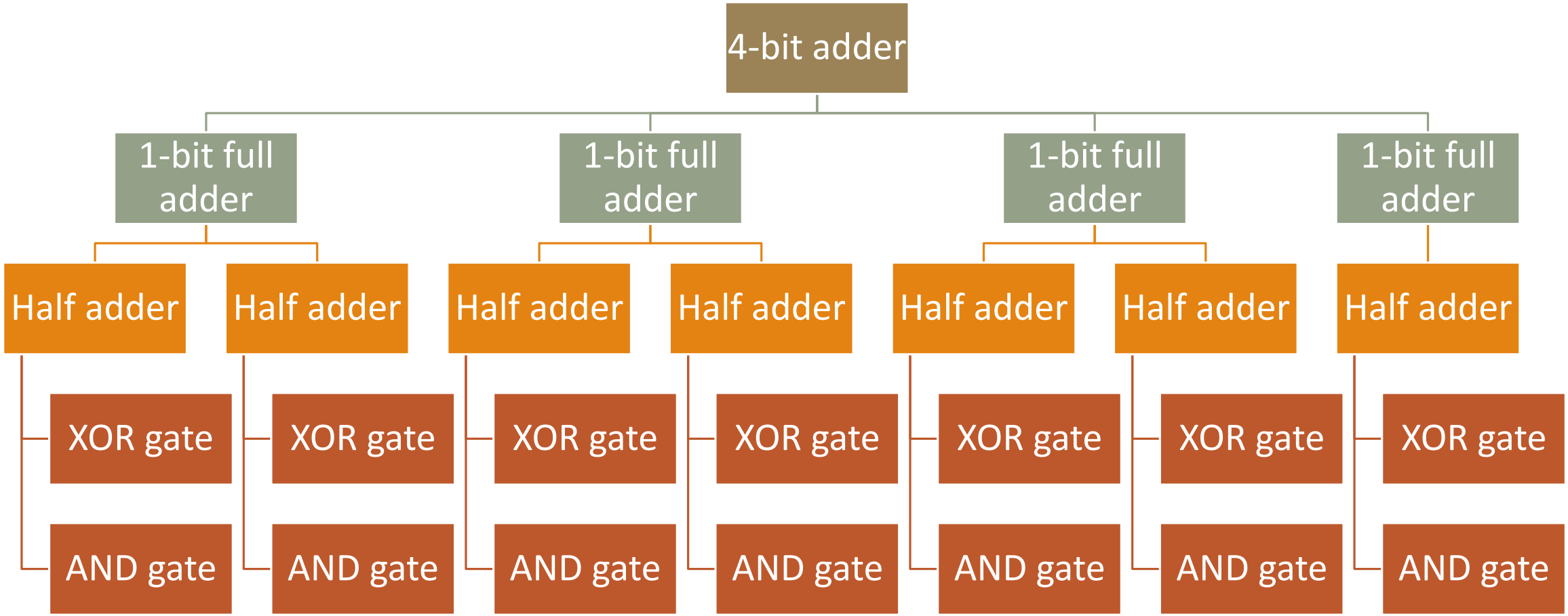
# Full adder built using half adders



# Half adder implemented using logic gates



# Design Hierarchy of the 4-bit adder



# Design Methodologies

There are two basic types of digital design methodologies:

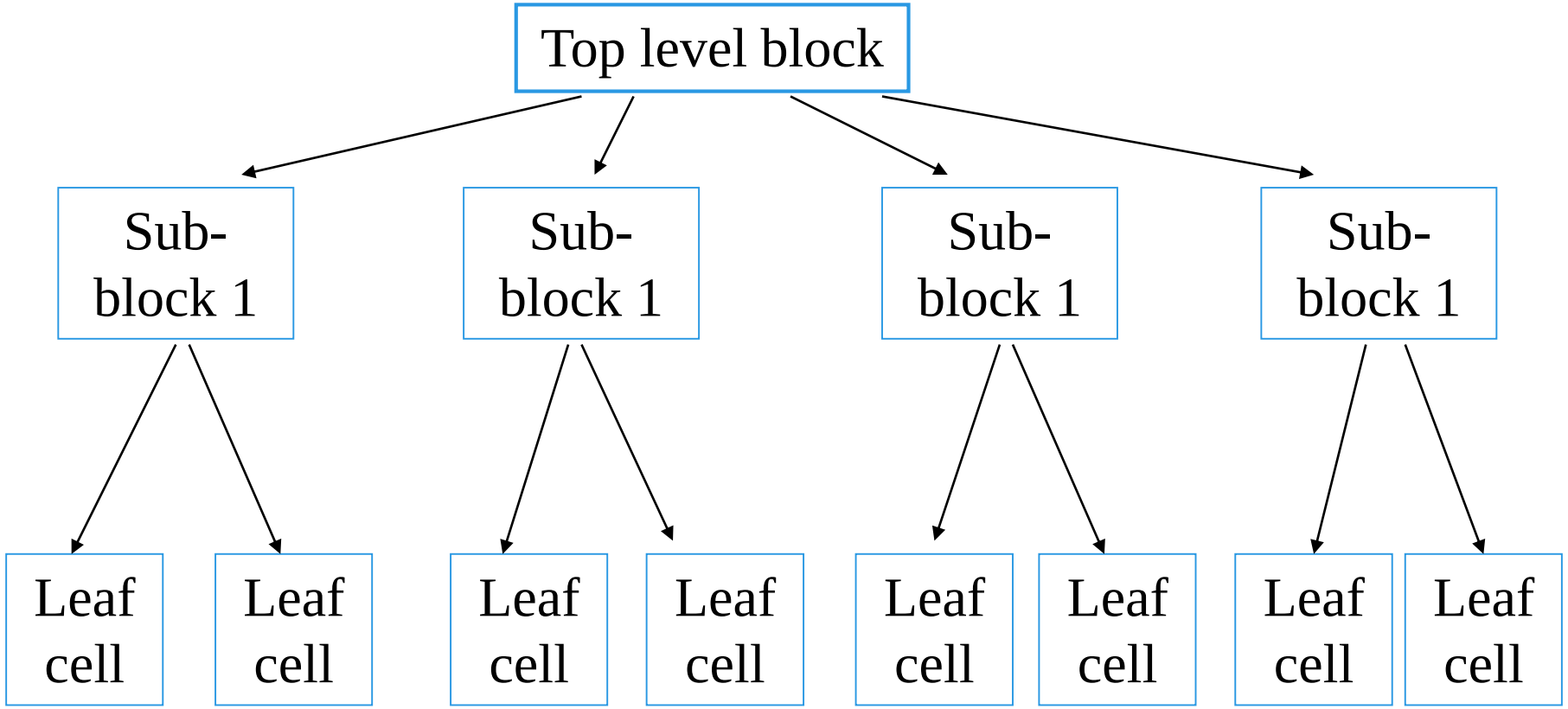
- Top-down design methodology

  we define the top-level block and identify the sub-blocks necessary to build the top-level block. We further subdivide the sub-blocks until we come to leaf cells, which are the cells that cannot further be divided.

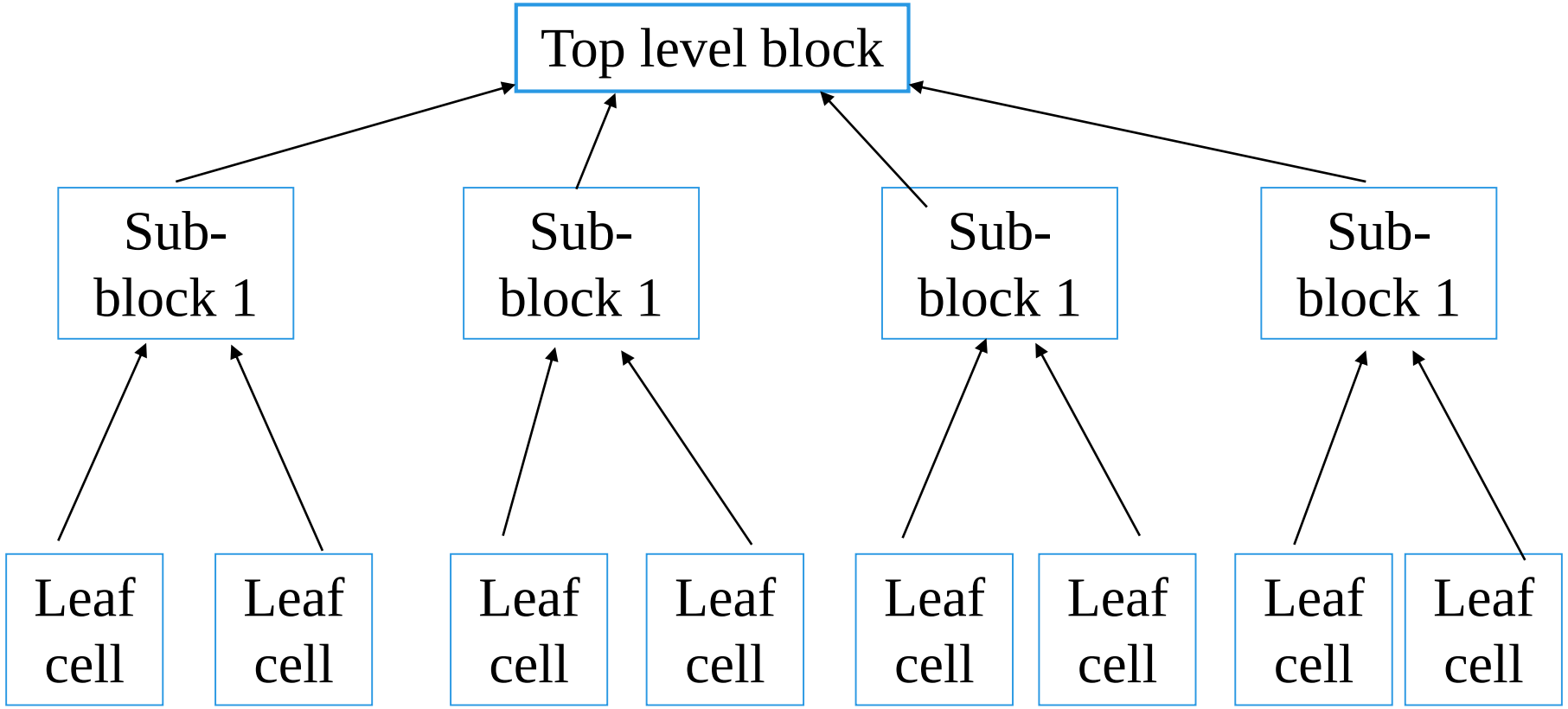
- Bottom-up design methodology

  we first identify the building blocks that are available to us. We build bigger cells, using these building blocks. These cells are then used for higher-level blocks until we build the top-level block in the design

# Top-down design methodology



# Bottom-up design methodology



# Hybrid method

Typically, a combination of top-down and bottom-up flows is used.

- Design architects -> define specifications of the top-level block.
- Logic designers -> break up the functionality of the top level block into blocks and sub-blocks.

At the same time,

- circuit designers -> design optimized circuits for leaf-level cells -> build higher-level cells by using these leaf cells

The flow meets at an intermediate point, where

- switch-level circuit designers have created a library of leaf cells by using switches.
- logic level designers have designed from top-down until all modules are defined in terms of leaf cells.

# Modules

- Hierarchical modeling concepts in Verilog :
  - facilitated by the concept of a module.
- A module is the basic building block in Verilog.
- A module is an element or a collection of lower-level design blocks.

# Modules

- Typically, elements are grouped into modules to provide common functionality that is used at many places in the design.

- A module provides the necessary functionality to the higher-level block through its port interface (inputs and outputs), but hides the internal implementation.
  - allows the designer to modify module internals without affecting the rest of the design.

# Defining a module in Verilog

```
module <module_name> (<module_terminal_list>);
...
<module internals>
...
...
endmodule
```

module_terminal_list describes the input and output terminals of the module.

# Example : Verilog module for adder

```
module RippleCarryAdder4(a,b,cin,s,cout);
  .
  .
  <functionality of the adder>
  .
  .
endmodule
```

# Instances

- A module provides a template from which you can create actual objects.
- When a module is invoked, Verilog creates a unique object from the template.
- Each object has its own name, variables, parameters, and I/O interface.
- The process of creating objects from a module template is called instantiation
- The objects are called instances.

# Example : 4-bit adder

```
// Define the top-level module called RippleCarryAdder4.
//It instantiates 4 FullAdders.
module RippleCarryAdder4(a,b,cin,s,cout);

    input [3:0] a, b;    input cin; // I/O signals will be explained later
    output [3:0] s;      output cout;

    wire cout1, cout2, cout3;  //wires for connecting instances

    //Four instances of the module FullAdder are created.
    //Each has a unique name.
    //Each instance is passed a set of signals.
    //Notice, that each instance is a copy of
    // the module FullAdder.
    FullAdder fa0(a[0],b[0],cin,s[0],cout1);
    FullAdder fa1(a[1],b[1],cout1,s[1],cout2);
    FullAdder fa2(a[2],b[2],cout2,s[2],cout3);
    FullAdder fa3(a[3],b[3],cout3,s[3],cout);

endmodule
```

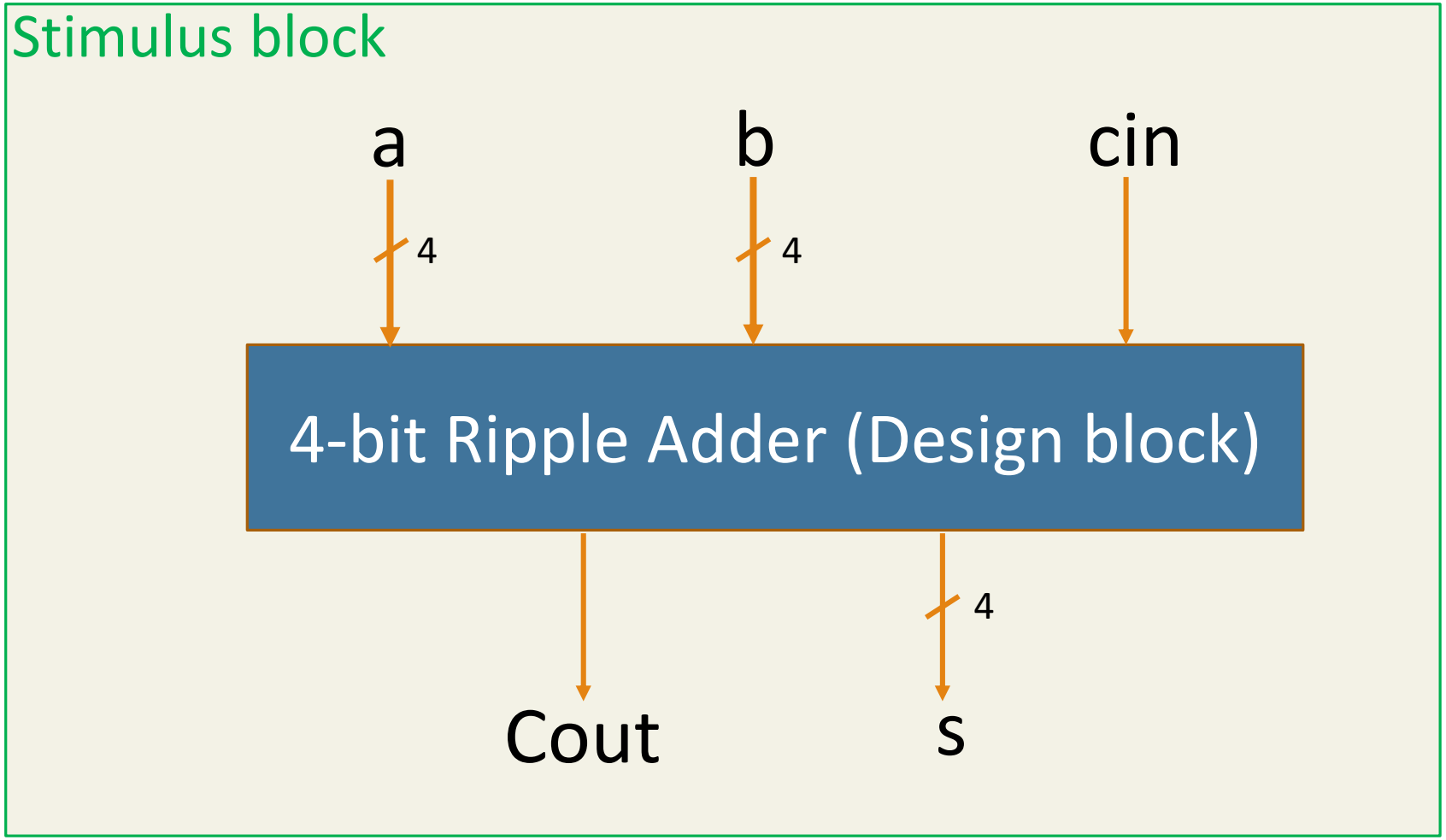
Assume that module **FullAdder** is defined elsewhere in the design as :

```
module FullAdder(a,b,cin,s,cout);
    input a,b,cin; output s,cout;
    assign {cout,s} = a + b + cin;
endmodule
```

# Components of a Simulation

- Once a design block is completed, it must be tested.
- The functionality of **the design block** can be tested by applying stimulus and checking results.
- We call such a block the **stimulus block**.
- The stimulus block can be written in Verilog.
- The stimulus block is also commonly called a **test bench**.

# Stimulus Block Instantiates Design Block



# Example testbed (Stimulus block)

```
module stimulus;
    reg [3:0] a, b;
    wire [3:0] s;
    wire cout;

    // instantiate the design block
    RippleCarryAdder4 rca(a,b,1'b0,s,cout);
    initial
    begin
        a <= 4'b1100; //assign value for a
        b <= 4'b0010; //assign value for b
        #1 //wait for 1 simulation time unit
        $display("%d + %d = %d, %d", a,b,s,cout); //print the result
    end
endmodule
```