# FPGA Basics 1

Combinational Logic Design in Verilog HDL

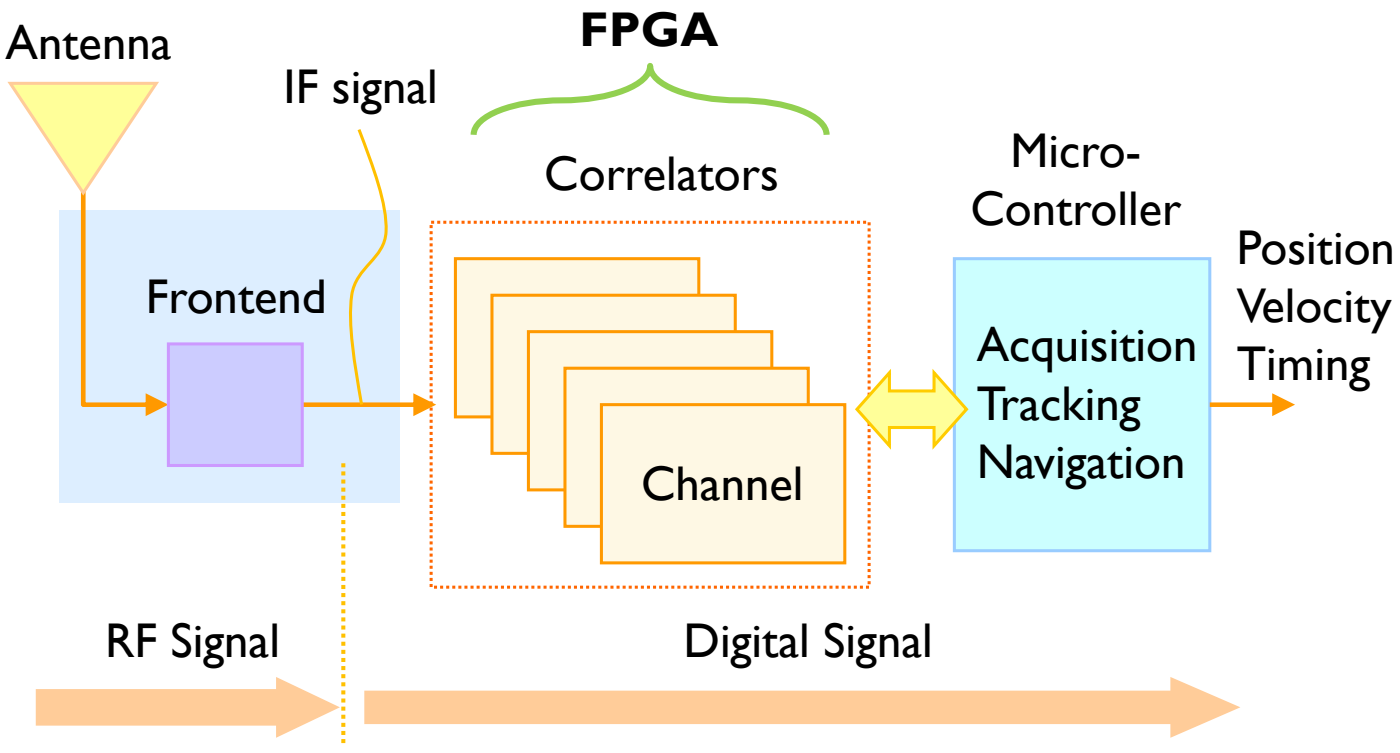
## What is FPGA?

- A field-programmable gate array (**FPGA**) is an integrated circuits designed to be configured by a user.
- The FPGA configuration is generally specified using hardware description language (**HDL**)
- FPGAs contain an array of programmable logic block and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together.

# What can FPGAs be used for?

- FPGAs can be used to solve any problems which is computable.
- Hardware acceleration is a main FPGA use case.
- The advantage lies in that FPGAs are sometimes significantly faster for some application because of their parallel nature.
- Specific applications of FPGAs include digital signal processing, computer vision, speech recognition, cryptography and a growing range of other areas.

# GNSS Receiver Architecture



# Types of Logic Circuits

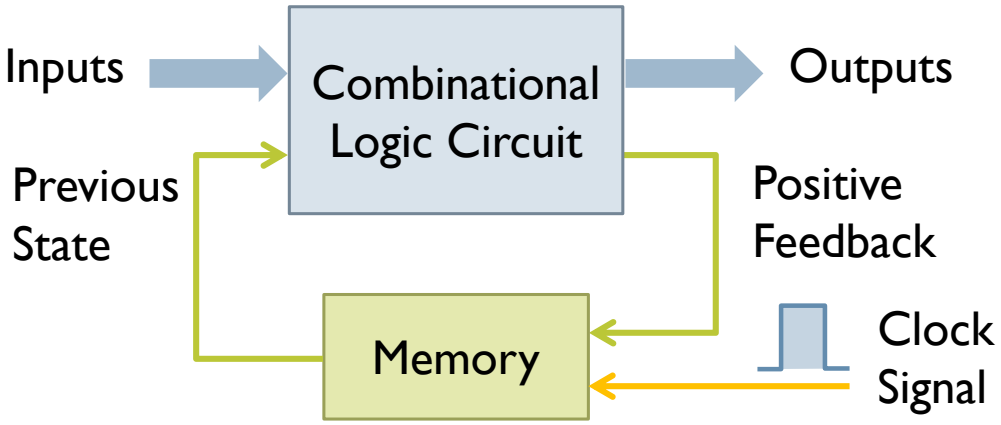
- There are two types of logic circuits depends on their output and memory used.
  - Combinational logic circuits
  - Sequential logic circuits
- The outputs of a combinational logic circuit are determined from only the present combination of inputs.
- The outputs of a sequential logic circuit are determined from both the present combination of inputs and previous outputs. That means sequential logic circuits use memory elements to store the value of previous outputs.

# Combinational Logic Circuits

- Combinational logic circuits are made up from basic logic NAND, NOR or NOT gates that are "combined" together to produce more complicated logic circuits.

# Sequential Logic Circuits

- In sequential logic circuits, the things happen in a "sequence", one after another, and the clock signal determines when things will happen next.




# What is an HDL?

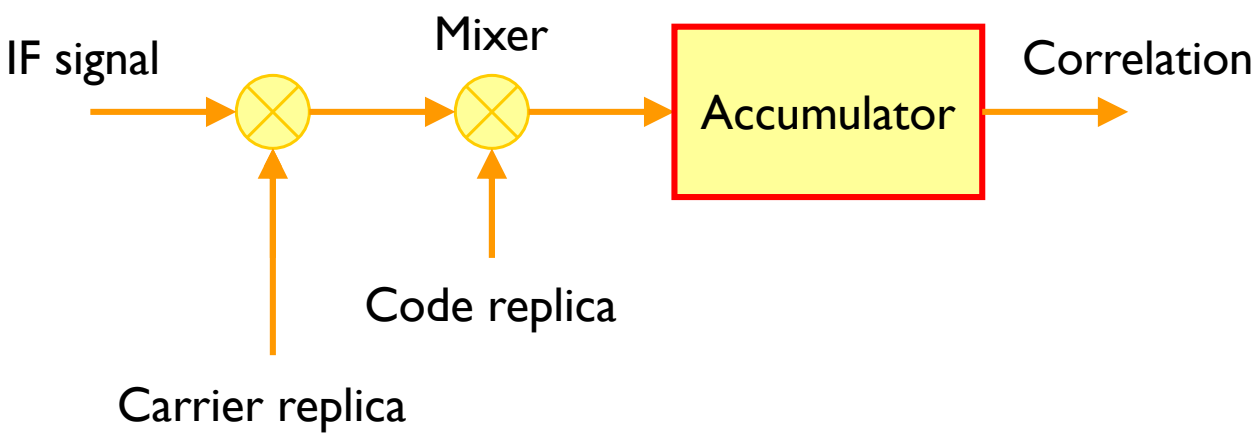
- Hardware description languages allow you to describe a logic circuit using words and symbols, and then development software can convert that textual description into configuration data that is loaded into the FPGA.
- The most popular hardware description languages are Verilog and VHDL.

## Programing Languages vs. HDLs

- HDLs resemble high-level programming languages such as C and Python, but it is important to understand that there is a fundamental difference: statements in HDL code involve parallel operation.
- When we write a computer program, we understand that the processor will execute lines of code one at a time, following the top-to-bottom organization.
- In HDL code, we are describing digital hardware, and separate portion of this hardware can operate simultaneously, even though the corresponding lines of code are written using a top-to-bottom organization.

## GNSS Correlator Architecture

- Correlators are the key operation for GNSS receivers to synchronize with the incoming signal.
- The maximum correlation peak is acquired when the both code and carrier replicas match the incoming signal.



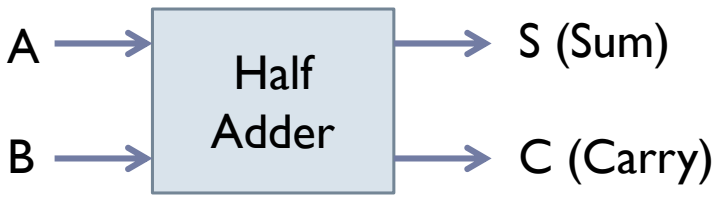

# Binary Addition of Two Bits

- When the two single bits are added together, the addition of "0+0", "0+1" and "1+0" results in either a "0" or a "1".
- The sum of "1+1" is equal to "2", but it does not exist in binary.
- The decimal "2" is equal to the binary "10", in other words a zero for the sum plus an extra carry bit.

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Carry Sum

# Half Adder




$$\begin{array}{r} A \\ +\ B \\ \hline \{\ C\ ,\ S\ \} \end{array}$$

Truth Table

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

S = A XOR B = A ^ B

C = A AND B = A & B

# Half Adder Verilog Example

```
module half_adder(A, B, S, C);

  input A, B;
  output S, C;

  assign S = A ^ B;
  assign C = A & B;

endmodule
```

# Modules

- A "module" is a basic unit in Verilog code that implement a certain functionality.
- A module should be enclosed within "module" and "endmodule" keywords.
- Name of the module should be given right after the "module" keyword, and a list of "ports" should be declared as well.

```
module half_adder(A, B, S, C);
          Name          Ports
  ...
endmodule
```

# Types of Ports

- **input**
  - Simple input
  - The design module can only receive values from outside.
- **output**
  - Simple output
  - The design module can only send values to outside.
- **inout**
  - Tri-state bi-directional port
  - The design module can either send or receive values.

# Wires and Assignments

- To make a continuous assignment to an internal signal in the module, the signal must first be declared as a "**wire**".
  - Ports are by default considered as signals of type "wire".
- Any "wire" can be driven continuously with a value by the "**assign**" statement.

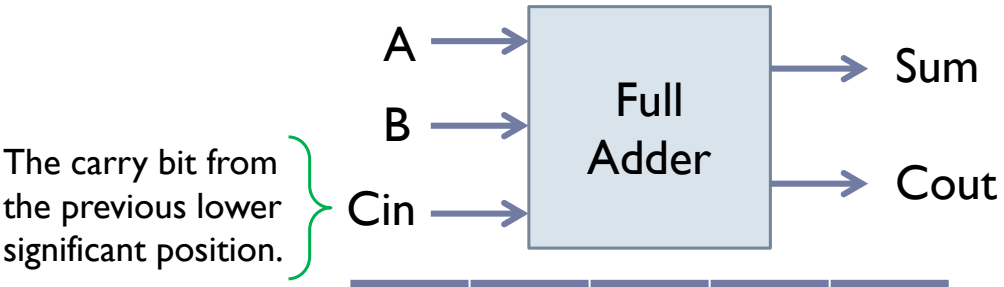
```
input A, B;
output S, C;

assign S = A ^ B;
assign C = A & B;
```

# Bitwise Operators

- Each bit is operated.
- Result is the size of the largest operand, and the smaller operand if left extended with zeros to the size of the bigger operand.

| Character | Operation |
|---|---|
| ~ | NOT (invert) each bit |
| & | AND each bit |
| \| | OR each bit |
| ^ | XOR each bit |

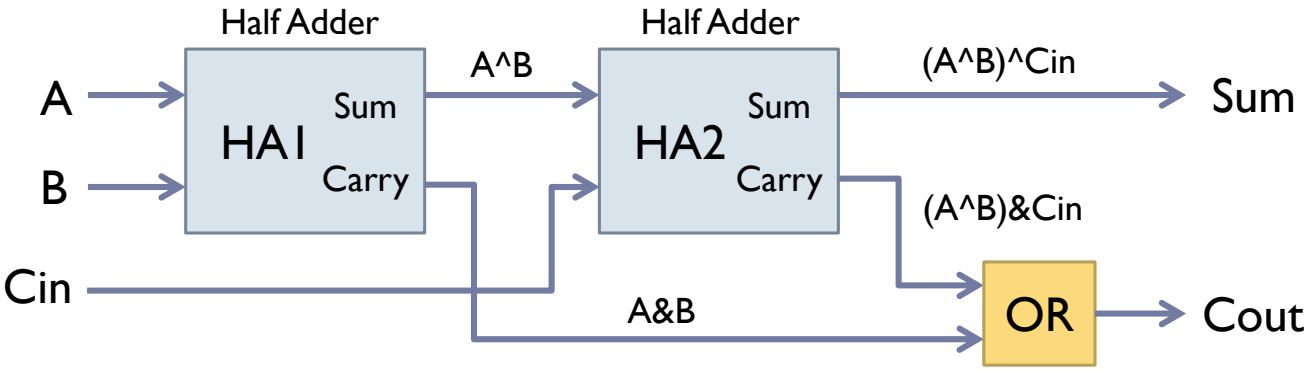
# Full Adder




| A | B | Cin | Sum | Cout |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

# Hierarchical Design in Verilog

- A hierarchical methodology is used to design simple components to construct more complex one.
- For example, the full adder is basically two half adders connected together.

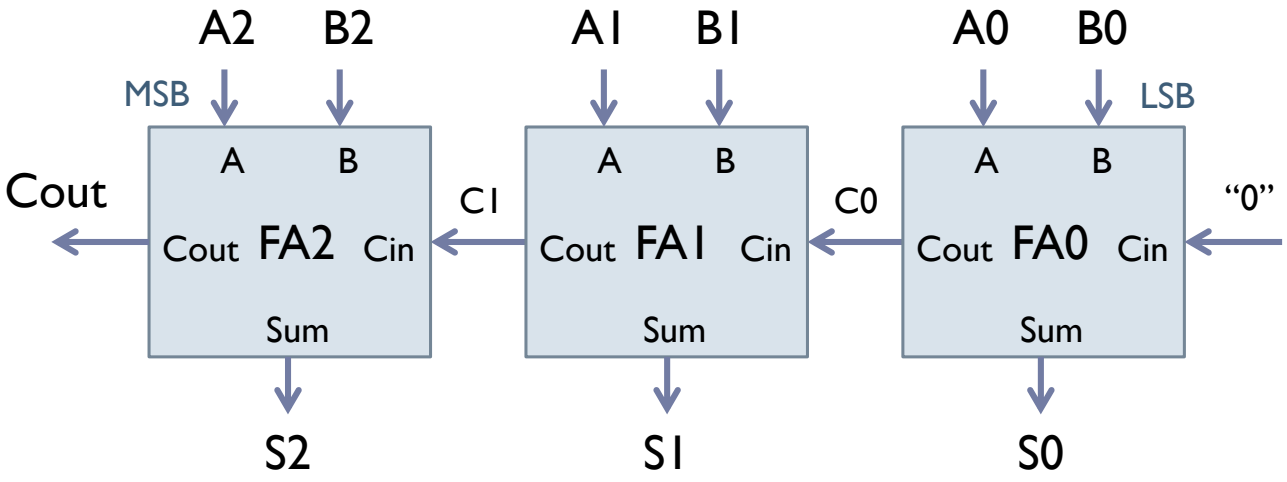



# Full Adder Verilog Example

```
module full_adder(A, B, Cin, Sum, Cout);

  input A, B, Cin;
  output Sum, Cout;
  wire S1, C1, C2;

  half_adder HA1(A, B, S1, C1);
  half_adder HA2(S1, Cin, Sum, C2);

  assign Cout = C1 | C2;

endmodule
```

# 3-Bit Adder

- If you want to N-bit adder, N number of full adders need to be connected or "cascaded" together to produce what is known as a "ripple carry adder".




# 3-Bit Adder Verilog Example

```
module adder(A, B, Sum, Cout);

  input [2:0] A, B;
  output [2:0] Sum;
  output Cout;
  wire C0, C1;

  full_adder FA0(A[0], B[0], 0, Sum[0], C0);
  full_adder FA1(A[1], B[1], C0, Sum[1], C1);
  full_adder FA2(A[2], B[2], C1, Sum[2], Cout);

endmodule
```

# Another 3-Bit Adder Verilog Example

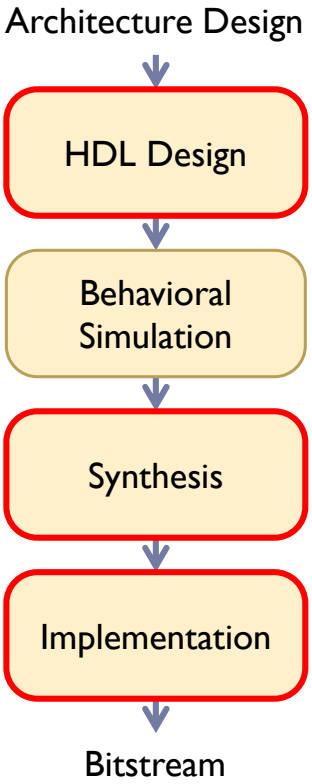
```
module adder3(A, B, Sum, Cout);

  input [2:0] A, B;
  output [2:0] Sum;
  output Cout;

  assign {Cout, Sum} = A + B;

endmodule
```

# Concatenation Operators

- Multi-bit wires and variables can be clubbed together using concatenation operators { and } separated by commas.

```
wire [7:0] high1, low1;
wire [7:0] high2, low2;
wire [15:0] data;

assign data = {high1, low1};
assign {high2, low2} = data;
```

# Design Flows




This is the process of creating the hardware logic itself, typically by writing register-transfer level (**RTL**) using a hardware description language (HDL).

This step ensure that the design works as intended using a specified testbench module.

This process transforms the HDL design into a gate level representation (**netlist**).

This step provides all the features necessary to optimize, place and route the netlist onto the available device resources of the target part.