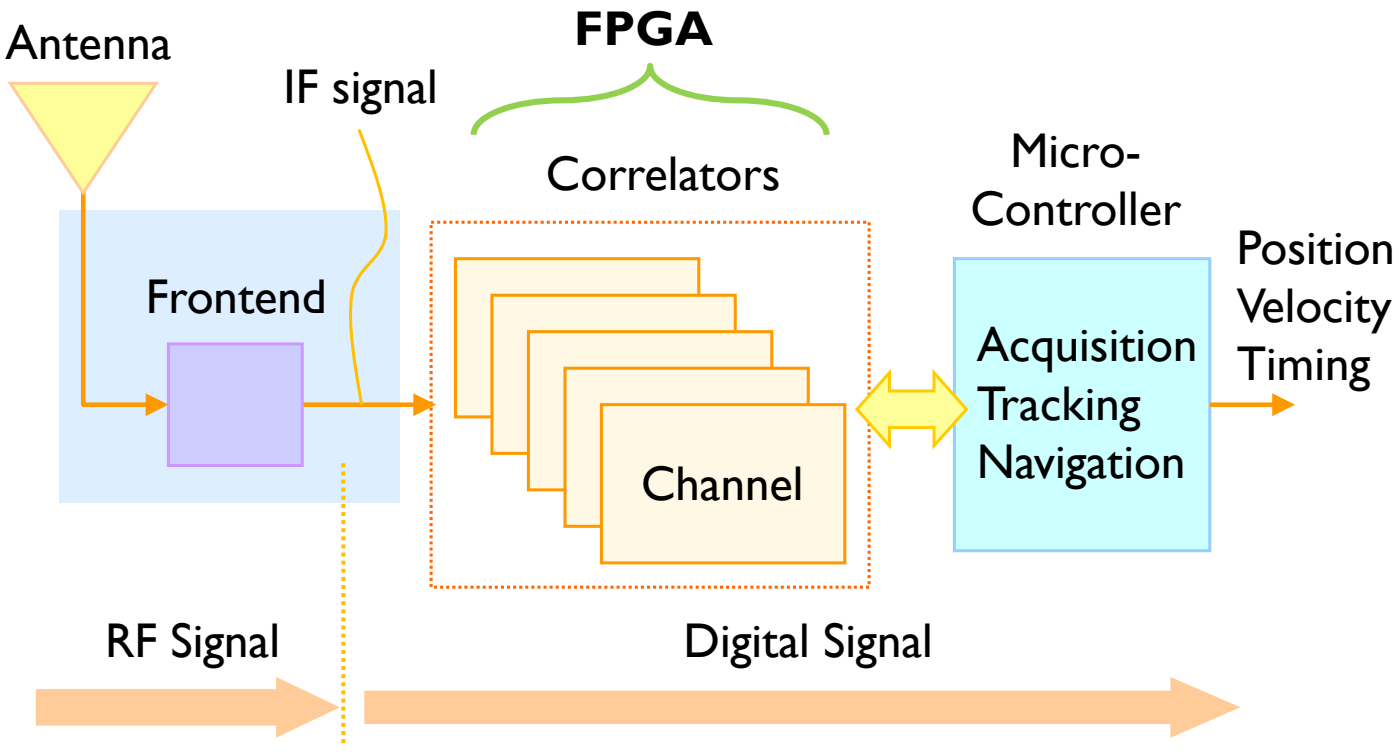
FPGA Basics 3

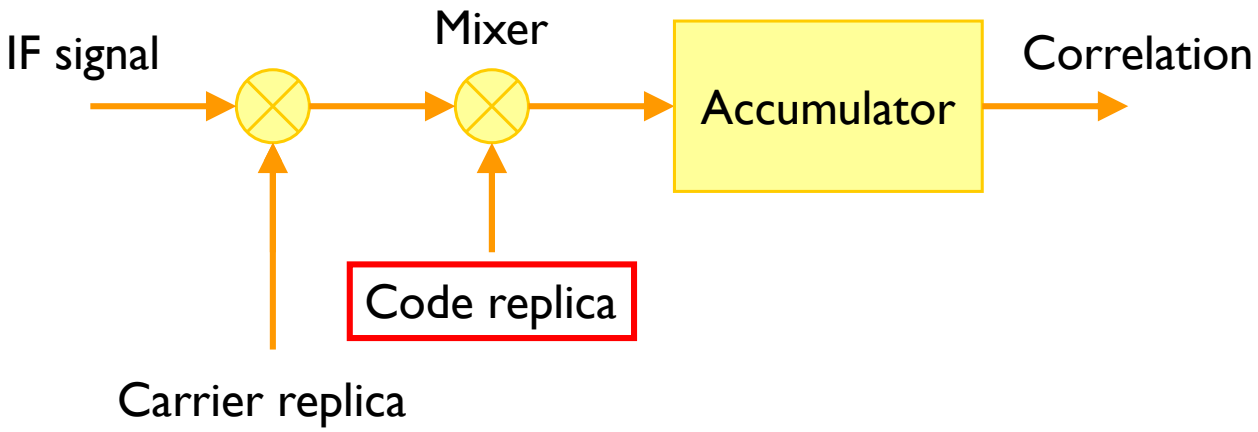
Sequential Logic Design and Simulation

GNSS Receiver Architecture

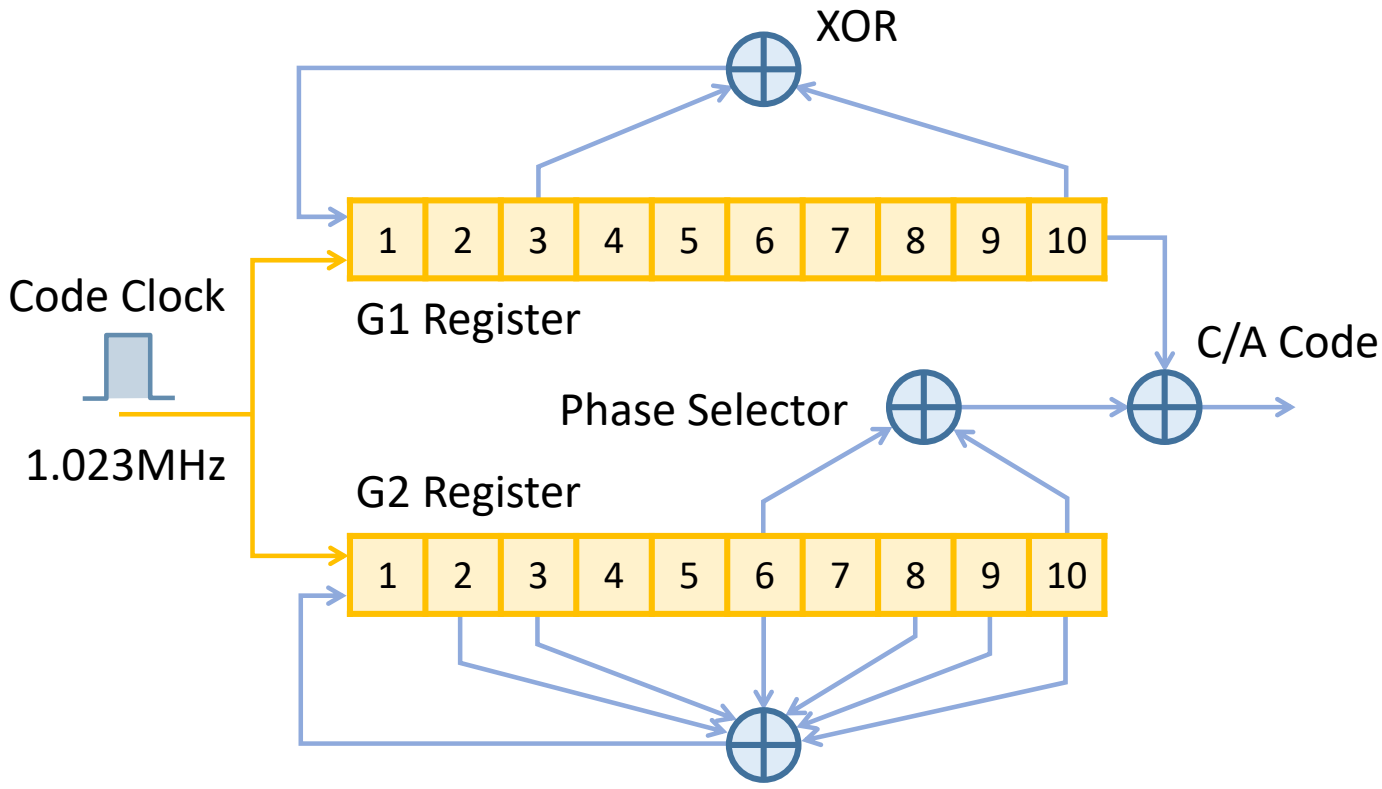


GNSS Correlator Architecture

- 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.



C/A Code Generator



Maximum Length Sequence

- A maximum length sequence, also sometimes called an m-sequence, is a type of pseudorandom binary sequence.
- They are bit sequences generated using maximal linear-feedback shift registers.
- They are periodic and reproduce every binary sequence, except the zero vector.
  - For length-m registers, they produce a sequence of length $2^m - 1$.

Sequential Logic in Verilog

- Sequential logic defines modules that have memory.
  - Flip-Flops, Latches, Registers, Finite State Machines, …
- Sequential logic is triggered by a "clock" event.
  - Flip-Flops are sensitive to the transitioning (edge) of clock.
  - Latches are sensitive to level of the signal
- Combinational constructors are not sufficient.
  - We need new constructor: `always`
  - Whenever the event in the sensitivity list occurs, the statement is executed.

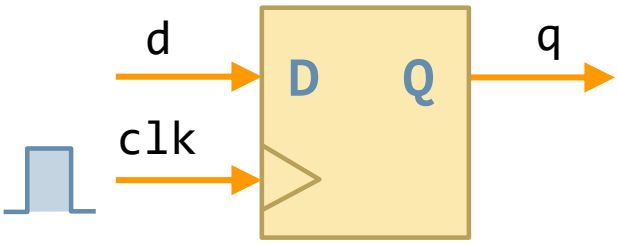
```
always @ (sensitivity list)
    statement;
```

D Flip-Flop

```
module flop(input clk,
            input d,
            output reg q);

  always @ (posedge clk)
    q <= d;

endmodule
```

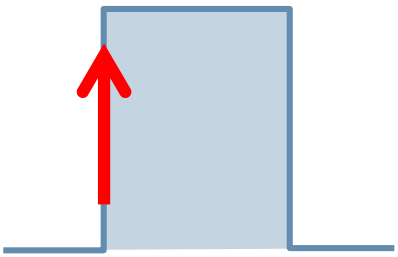


D Flip-Flop

```
module flop(input clk,
            input d,
            output reg q);

 always @ (posedge clk)
    q <= d;

endmodule
```



- The `posedge` defines a rising edge.
- The process will trigger only if the `clk` signal rises.

D Flip-Flop

```
module flop(input clk,
            input d,
            output reg q);

  always @ (posedge clk)
    q <= d;

endmodule
```

- Once the `clk` signal rises, the value of `d` will be copied to `q`.
- "assign" statement is not used within always block.
- The "`<=`" describes a "non-blocking" assignment.

D Flip-Flop

```
module flop(input clk,
            input d,
            output reg q);

  always @ (posedge clk)
    q <= d;

endmodule
```

- Assigned variables need to be declared as `reg`.
- The name `reg` does not necessarily mean that the value is a register.

D Flip-Flop with Synchronous Reset

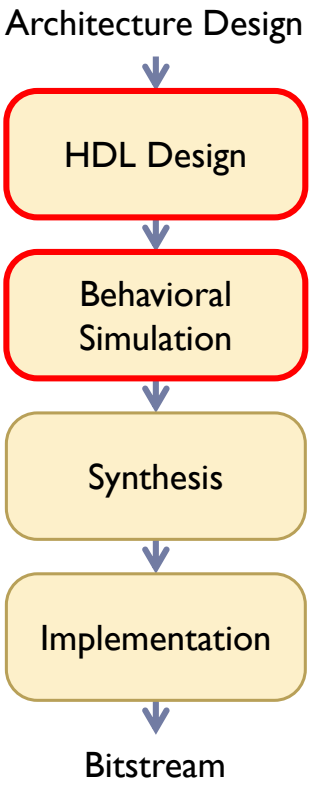
```
module flop(input clk,
            input rstn,
            input d,
            output reg q);

  always @ (posedge clk) begin
    if (!rstn) q <= 0;
    else       q <= d;
  end

endmodule
```

- Reset only happens when the clock rises.

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).

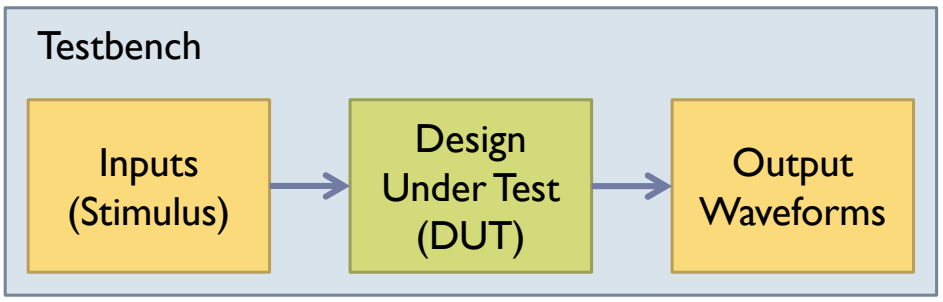
This step ensure that the design works as intended using a specified **testbench** module.

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.

What is a Testbench?

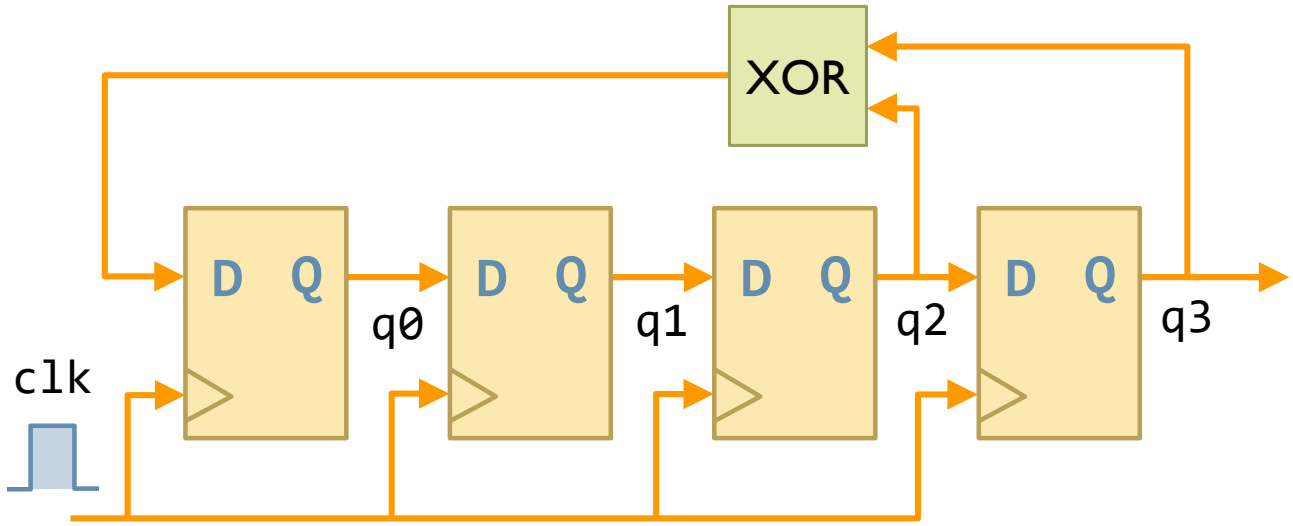
- Testbenches consist of non-synthesizable Verilog code which generates inputs to the design and checks that the outputs are correct.
- The diagram below shows the typical architecture of a simple testbench.



Simulation Tools

- The stimulus block generates the inputs to the FPGA design under test, and the output block shows the output waveforms to ensure they have the correct values.
- Many freely available software packages offer behavioral simulation capability.
  - Commercial: **Vivado** (AMD/Xilinx) and **Quartus** (Intel)
  - Open Source: **Icarus Verilog** and **GTKWave**
  - Online: **EDA Playground** (https://www.edaplayground.com/)

Linear Feedback Shift Register (LFSR)



Linear Feedback Shift Register (LFSR)

q3	q2	q1	q0	Decimal	Hex
0	0	0	1	1	1
0	0	1	0	2	2
0	1	0	0	4	4
1	0	0	1	9	9
0	0	1	1	3	3
0	1	1	0	6	6
1	1	0	1	13	d
1	0	1	0	10	a
0	1	0	1	5	5
1	0	1	1	11	b
0	1	1	1	7	7
1	1	1	1	15	f
1	1	1	0	14	e
1	1	0	0	12	c
1	0	0	0	8	8
0	0	0	1	1	1

LFSR Verilog Example

```
module lfsr(input clk,
            input rstn,
            output reg [3:0] q);

  always @ (posedge clk) begin
    if (!rstn) q <= 4'b1;
    else       q <= {q[2:0], q[2] ^ q[3]};
  end

endmodule
```

Testbench

lfsr_tb.v

```
module lfsr_tb;

  reg clk;
  reg rstn;
  wire [3:0] q;

  lfsr dut(clk, rstn, q);

  always #5 clk <= ~clk;
```

Continued…

Testbench

```
  initial begin
    clk  <= 0;
    rstn <= 0;

    #20 rstn <= 1;
    #80 rstn <= 0;
    #50 rstn <= 1;

    #200 $finish;
  end

endmodule
```