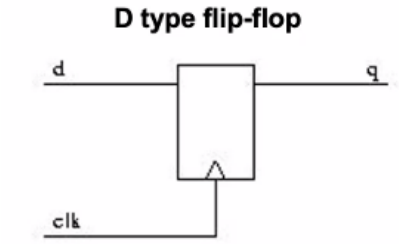
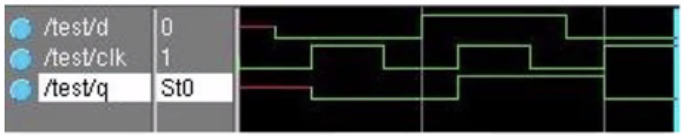
Basic Sequential Elements

D Flip-Flop

A D FF (Flip-Flop) is a sequential logic device used to store 1 bit of data. It has some **input wire D** and an **output wire Q** that reflects the current state of the FF. A clock wire is used to signal when the FF should update the current state Q. On the rising or falling clock edge (a specification of the FF), the value of Q is updated to reflect D at that moment. The diagram below depicts an example timing diagram for a rising edge-triggered FF.



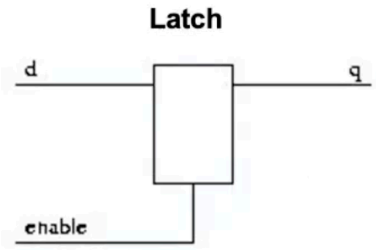


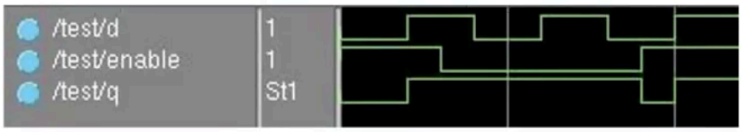
Specifications of a D Flip-Flop

- Like any logic circuit, it takes time for the input signals to propagate through the logic circuit. The **propagation delay** of the FF is the time for Q to update after a clock edge.
- FFs also require that the D signal is held stable for some amount of time before (**setup time**) and time after (**hold time**) the clock edge.

Latch

A Latch is a sequential logic device used to store 1 bit of data. It has some input wire D and output wire Q. Instead of a clock, an enable wire is used to signal when the device should be in the state of updating Q.



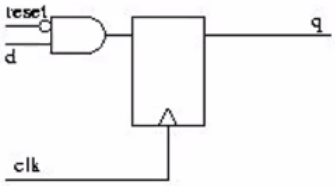


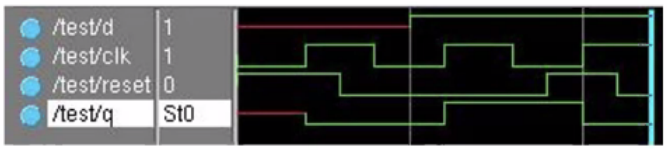
On powerup of a hardware design with Flip Flops, every FF starts with an unknown state. Every output usually drives other logic, so the system starts in an unpredictable state. To remedy this issue, storage devices with reset capabilities are introduced.

D Flip-Flop w/ Synchronous Reset

FFs with synchronous reset respond to the reset signal on the clock edge. See the diagram below. If the reset signal is high, the AND gate is forced to 0, and the next state of the FF becomes 0.

D type flip-flop with synchronous reset



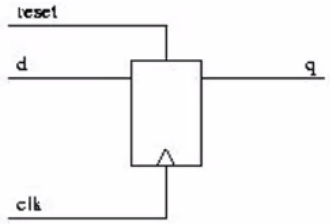


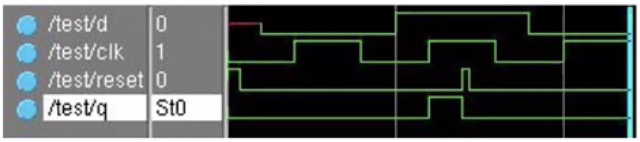
There is a performance/timing disadvantage to synchronous reset FFs. These FFs require that the reset signals are high on the clock edge to work. Reset signals need to be high for at least a clock period to ensure that the FF reads it.

D Flip-Flop w/ Asynchronous Reset

FFs with asynchronous reset respond to the reset signal at any time. The FF responds to it regardless of the clock signal.

D type flip-flop with asynchronous reset





For performance reasons these are preferred.

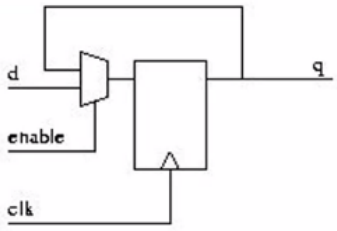
We use FFs and put combinational logic between them. The operating frequency of the FFs depends on the latency of the CL between them. They have to be operated at a slightly lower frequency.

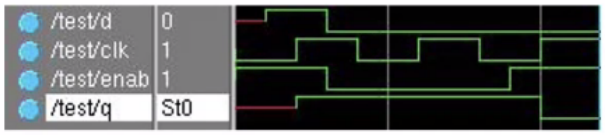
Clock gating: Flip Flops with Enable

Sometimes certain portions of a design are powered down for power efficiency reasons. An **enable** signal is used to turn a FF “on” and “off,” meaning that when enable is low, the output Q of the FF is guaranteed to be held stable.

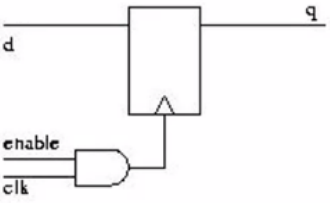
Consider the 2 designs below.

In the design with the multiplexer, the enable signal multiplexes in the input D or the output Q back into the FF. The clock signal still reaches the FF at every cycle. Even though the FF is held stable there is still some power consumption because the clock signal reaches the FF and triggers the input signal to propagate through it.





In the design with the AND gate, a lower enable signal forces the AND gate to 0. The clock signal will only arrive at the FF if enable is on. **This design is better because it consumes less power.**

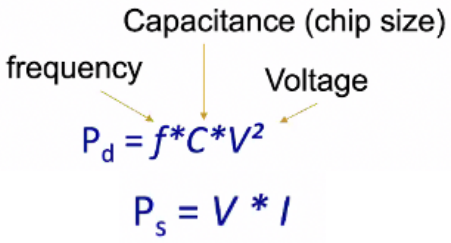


Other Basic Concepts

What determines the Power Consumption of a Chip?

There are 2 types of power usage: dynamic power and static power consumption.
Dynamic power is power used while a design is running. It depends on

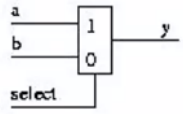
- frequency f: how fast the design is clocked
- capacitance C: how much charge the chip can hold overall
- voltage V: how quickly the electrons are being sent through the design



Static power is independent of the switching frequency f. The frequency is tied into the voltage. Reducing frequency by ½ means that the electrons move ½ as fast through a design. The voltage is reduced as well. Higher voltage means more electrons. **Reducing frequency provides cubic dynamic power reduction.**

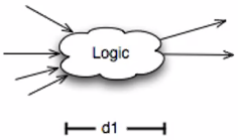
Multiplexers

A multiplexer is a combinational logic device that selects data to output from multiple input signals. The select signal needs to be log2(n) bits wide for an n-input multiplexer.

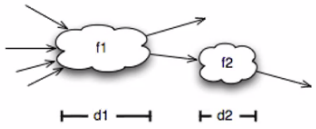


Logic Functions

A logic function is a design that takes input signals and outputs signals using them.
outputs = f(inputs)

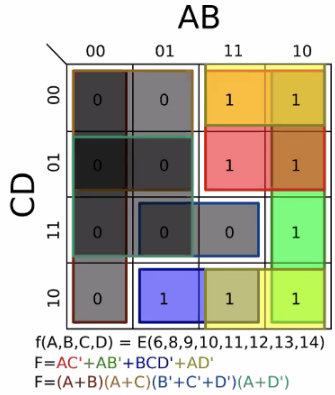


Logic functions can be compounded. A logic cloud sometimes must wait for another.
output = f2(f1(inputs))



Karnaugh Maps

K-Maps are a tool used to simplify boolean expressions. Synthesis tools in Vivado automate logic expression reduction, so it is not necessary to use them by hand. There is motivation to simplify logic expressions because smaller expressions means less gates and less power consumption.



Structural Verilog

The development of synthesis tools (compiles HDL into boolean algebra) made large-scale designs possible.

- Both Verilog and VHDL can target simulation and synthesis.
- All Verilog code can be simulated but not all code can be synthesized.
- Simulation
  - frequency/area/performance does not matter
  - used to develop models and evolve them to make it synthesizable
  - non-synthesizable code can be useful for simulation

Verilog Code can be Divided into 2 subsets:	
Structural Verilog (not synthesizable)	Synthesizable Verilog

Design Flow

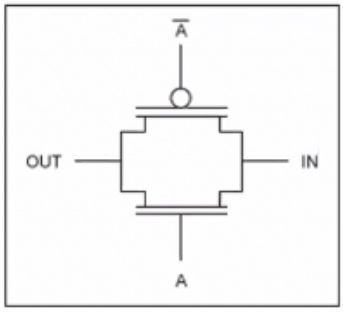
1. guarantee correctness
2. improve performance

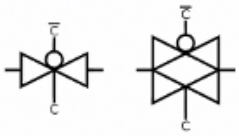
Verilog Syntax	
comments	// this is a comment / *this is also a comment */
string strings are only for testbenches. this command is non-synthesizable	$display(“hellow world!\n”) $display(“Number is %d”, val)

primitives
Primitives are not declared – they can only be instantiated.
 	
[option] specify delay or instance name
	and, or, nor, buff, etc.
	nmos, pmos
	tranif0/1, bufif0/1, rtranif0/1, etc.
	
	and #10 N25 (Z, A, B, C)
	// time delay of #10
	// N25 is the name
	// in () are the inputs and output

Transmission Gate (aka a 3-state gate)

tri-state == there is 0, 1, and an undefined state.





Recall that logic gates usually have **well-defined output**. Logic gates are built using power-efficient CMOS circuits. The the output is either pulled up to Vdd by the PMOS network or pulled down to ground by the NMOS network.

Transmission gates are different. There is no direct path from Vdd to ground. If the gate is open, the IN signal is propagated to the OUT signal. When the gate is closed, IN does not propagate to OUT and the state is unknown. **Z** is used to represent the tri-state.

Verilog supports 4 states:

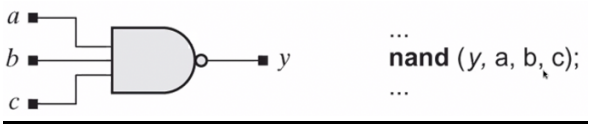
- 0
- 1
- Z (tri-state, signal is neither pulled up or down)
- X (signal is not defined for logic gate) shows up in simulations

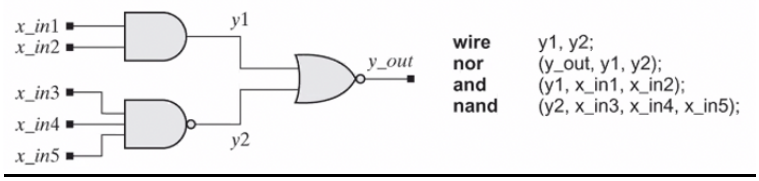
What is 3-State/ High Impedance?

In the table below, there are 2-state primitives in the left column and 3/tri-state in the right.

n-Input	*n*-Output, 3-state
and	buf
nand	not
or	bufif0
nor	bufif1
xor	notif0
xnor	notif1

Instantiation of Primitives



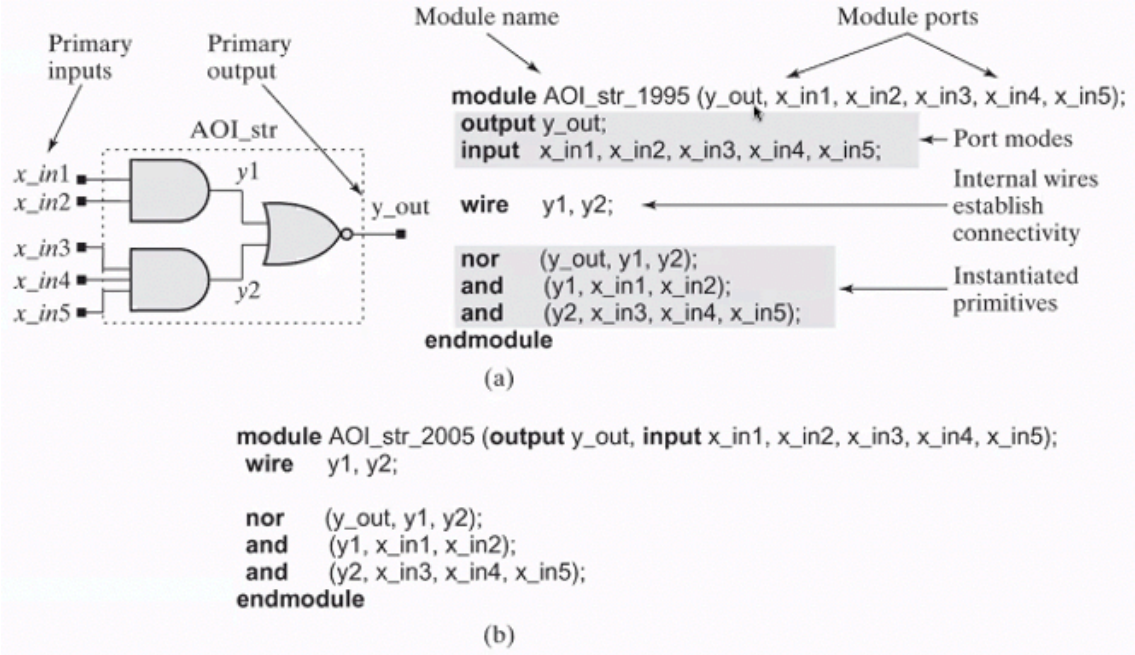


Modules

Hierarchies of modules are used to build complex structures

```
module my_design (module_ports);
... // Declarations of ports go here
... // Functional details go here
endmodule
```

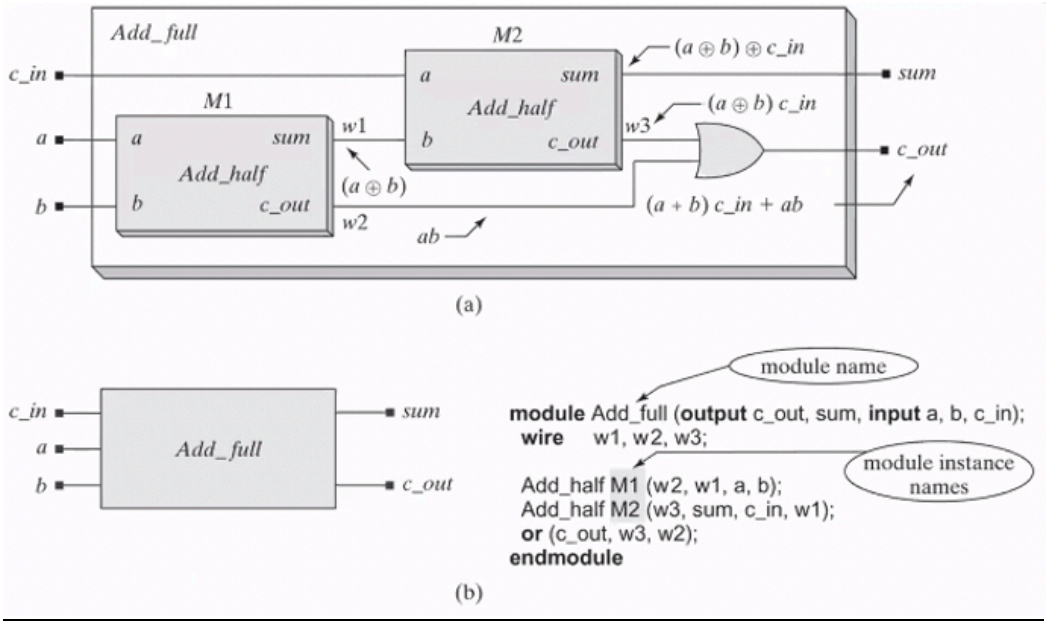
Part a is older Verilog. Part b shows the newer, easier way to declare inputs/outputs.



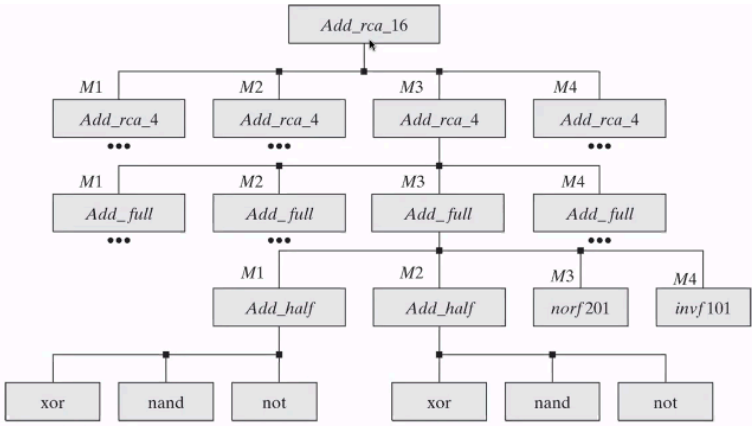
Design Hierarchy of a Full Adder.

A half adder takes 2 input bits and adds them to get a sum and carryout.
A full adder takes 2 input bits and a carry-in to get a sum and carryout.

Using 2 half adders to build a full adder:
A half adder is used to add a and b. Their sum is sent to another half adder with c_in to get the real sum. If a carryout happens for either hald-adder, there is a carryout for the full adder.



Design hierarchy of a 16-bit adder



- scope is access using the period (.) operation
  - M3.M3.M2 referes to an instance of Add_half
  - M3.M3.w3 refers to a wire in the instance

Port Association

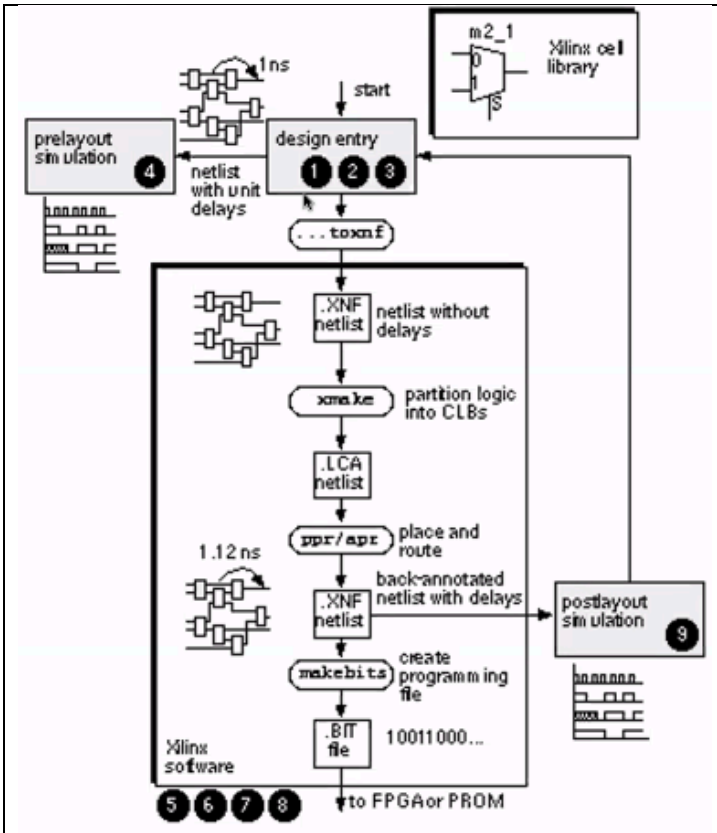
.portName(wire) so that you don't have to remember order. Port association!

Half Adder and Full Adder Implementation

```
module half_add (X, Y, S, C);

input X, Y ;
output S, C ;

xor SUM (S, X, Y);
and CARRY (C, X, Y);

endmodule
```

```
module full_add (A, B, CI, S, CO) ;

input A, B, CI ;
output S, CO ;

wire S1, C1, C2;

// build full adder from 2 half-adders
half_add PARTSUM (.X(A), .Y(B), .S(S1), .C(C1));
half_add SUM (.X(S1), .Y(CI), .S(S), .C(C2));

// ... and an OR gate for the carry
or CARRY (CO, C2, C1);

endmodule
```

FPGA Design Flow



1] **design entry**: describe Verilog code using structural or behavioral Verilog

2] try to **synthesize** it
The synthesis tools produce a netlist that has combination logic and FFs

[FPGAs/ASICS provide these 4: and , or, not, FFs]