EECS151/251A Introduction to Digital Design and ICs

Lecture 3 – Metrics & Verilog

Sophia Shao

Intel
IDM
advantage
software
silicon & platforms
packaging & process
at-scale manufacturing

LICKING COUNTY, Ohio, Jan. 21, 2022 – Intel today announced plans for an initial investment of more than $20 billion in the construction of two new leading-edge chip factories in Ohio. The investment will help boost production to meet the surging demand for advanced semiconductors, powering a new generation of innovative products from Intel and serving the needs of foundry customers as part of the company's IDM 2.0 strategy. To support the development of the new site, Intel pledged an additional $100 million toward partnerships with educational institutions to build a pipeline of talent and bolster research programs in the region.

https://www.intel.com/content/www/us/en/newsroom/news/intel-announces-next-us-site-landmark-investment-ohio.html

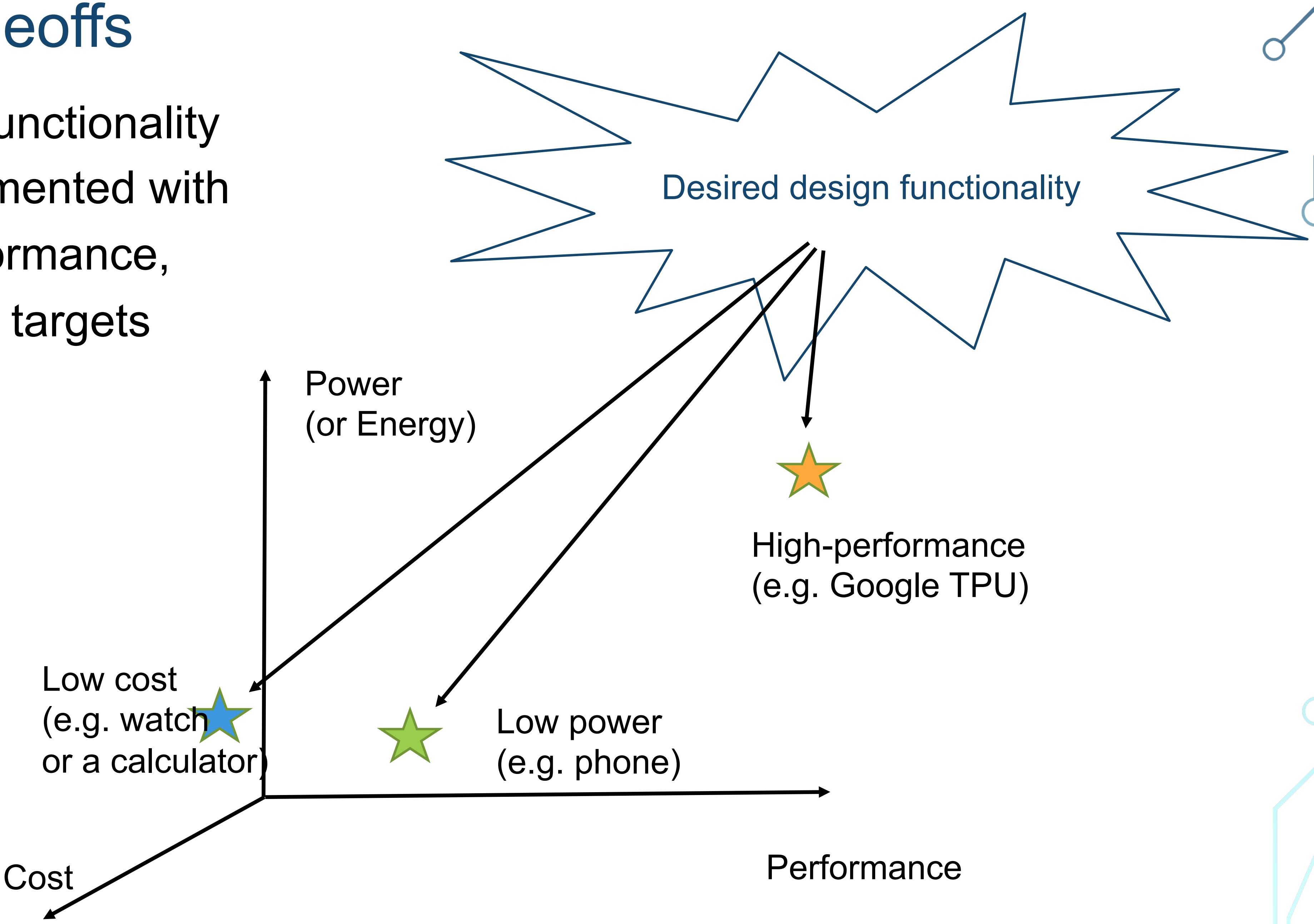
Review

- The design process involves traversing the layers of specification, modeling, architecture, RTL design and physical implementation.
- Digital systems implement a set of Boolean equations

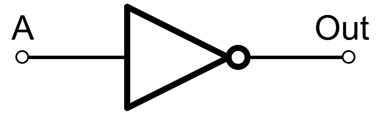
- **Metrics:**
  - **Functionality and Robustness**
  - **Cost**
  - **Performance**
  - **Power and Energy**
- **Verilog!**
  - **Overview**
  - **Combinational Logic**

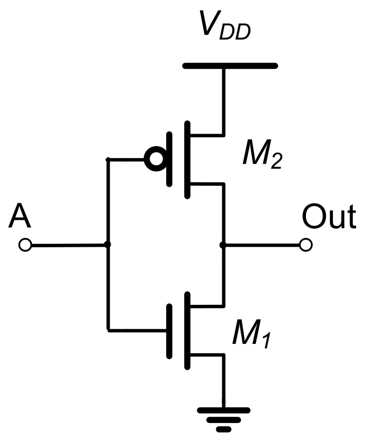
Design Tradeoffs

- The desired functionality can be implemented with different performance, power or cost targets



Beneath the Digital Abstraction

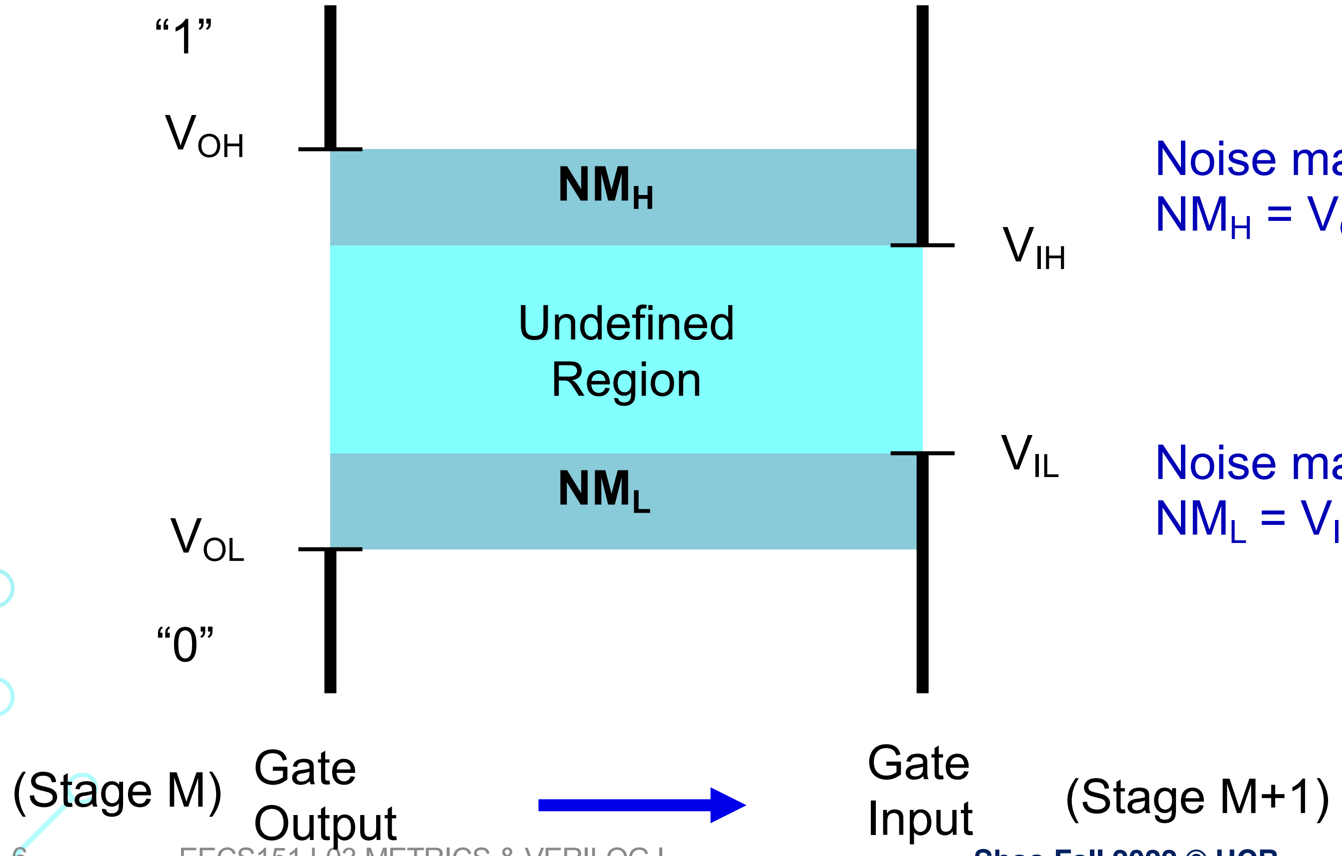




- Logic gate interpret its inputs as 0s(Low) and 1s(High).
- Noise: unwanted variations of voltages and currents in digital circuits.
- Logic levels:
  - Mapping a continuous voltage onto a discrete binary logic variable
  - Low (0): $[0, V_L]$
  - High (1): $[V_H, V_{DD}]$
  - V_L V_H: nominal voltage levels

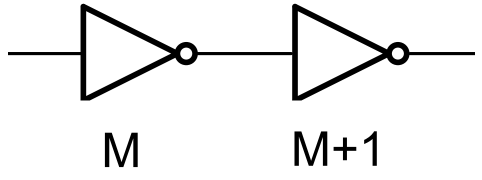
Noise Margins

- A measure of the sensitivity of a gate to noise.
- Represent the levels of noise that can be sustained when gates are cascaded.
- The amount of **noise** that could be added to **a worst-case output** so that the signal can still be interpreted correctly as **a valid input**.



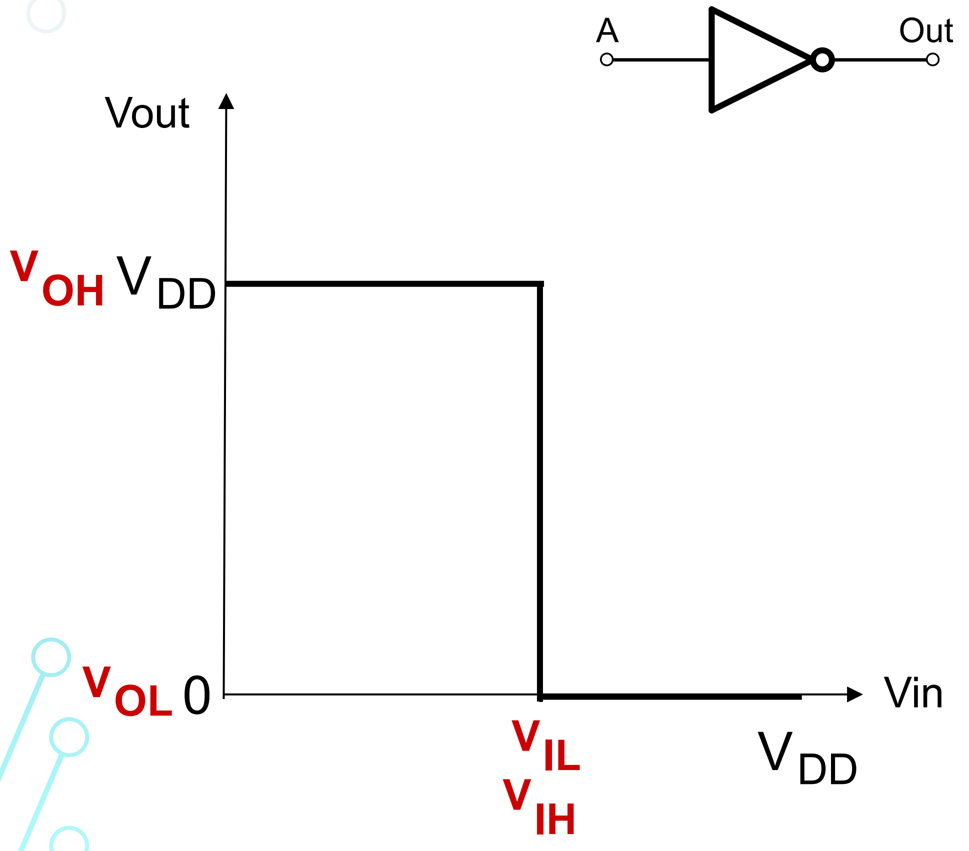
Noise margin high:
$NM_H = V_{OH} - V_{IH}$

Noise margin low:
$NM_L = V_{IL} - V_{OL}$

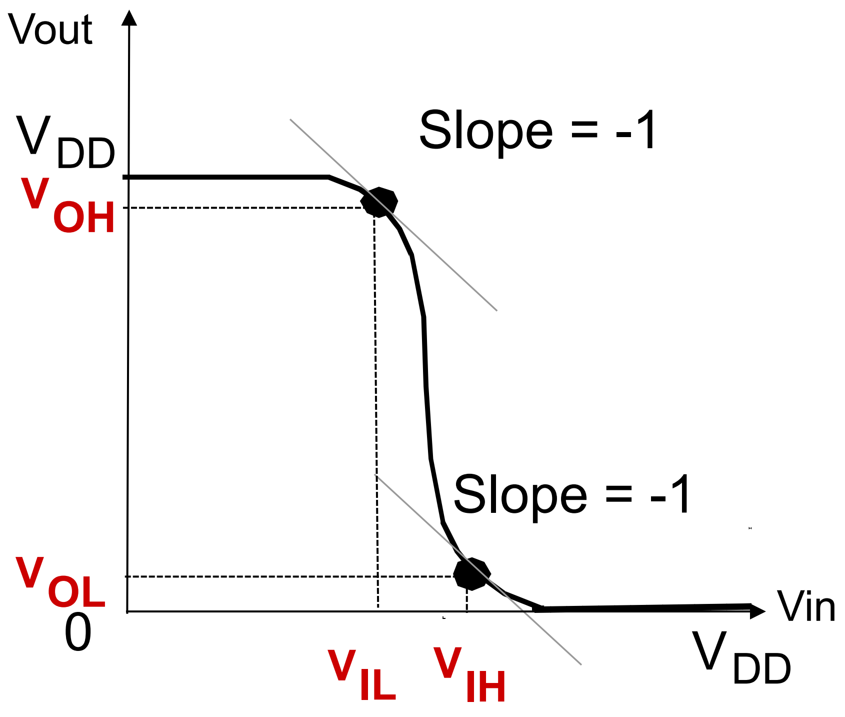


Voltage Transfer Characteristic

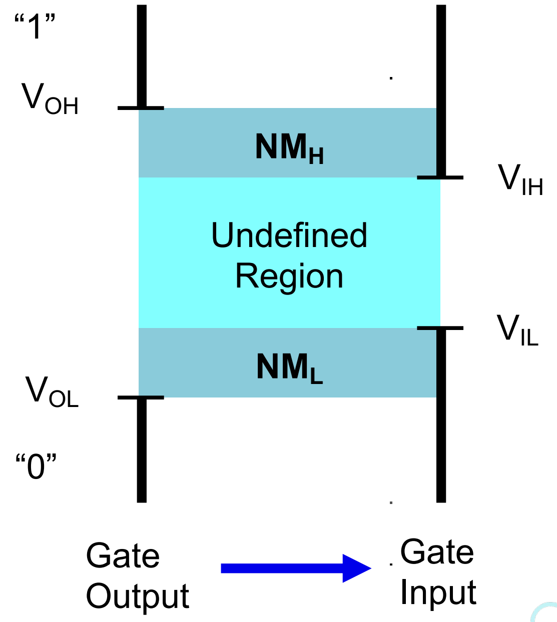
- Describes the output voltage as a function of the input voltage.
- To choose logic levels -> slope = -1 -> maximize noise margin



a) Ideal

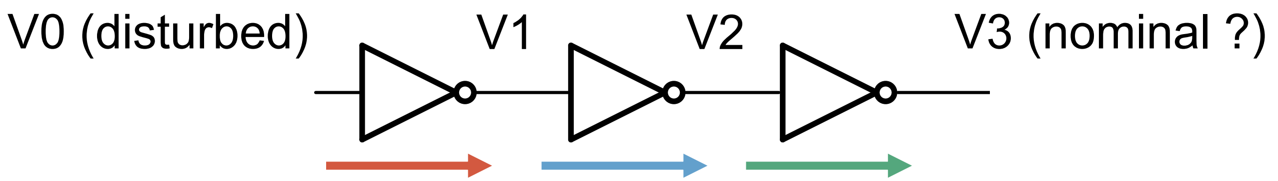


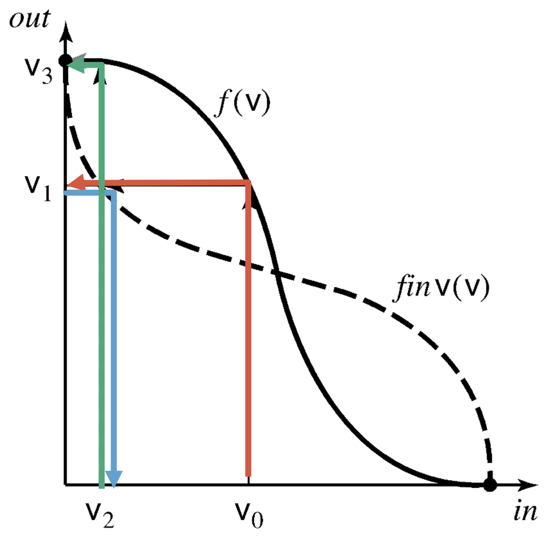
b) Real



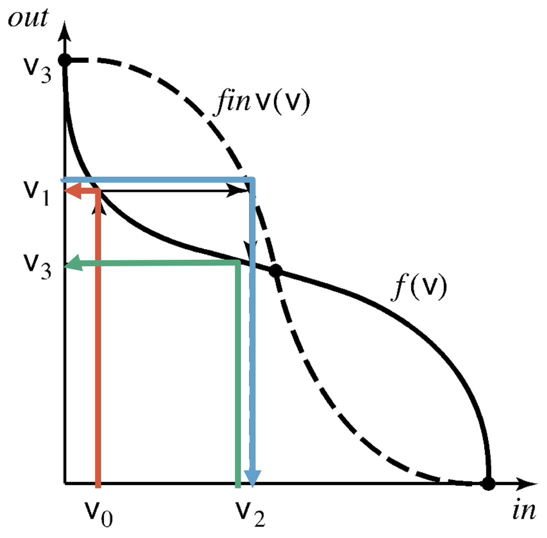
Regenerative Property

- Ensures that a **disturbed** signal gradually **converges back** to one of the **nominal voltage levels** after passing through a number of logical stages.
- Look for a sharp transition in voltage transfer characteristics.





Regenerative gate



Non-regenerative gate

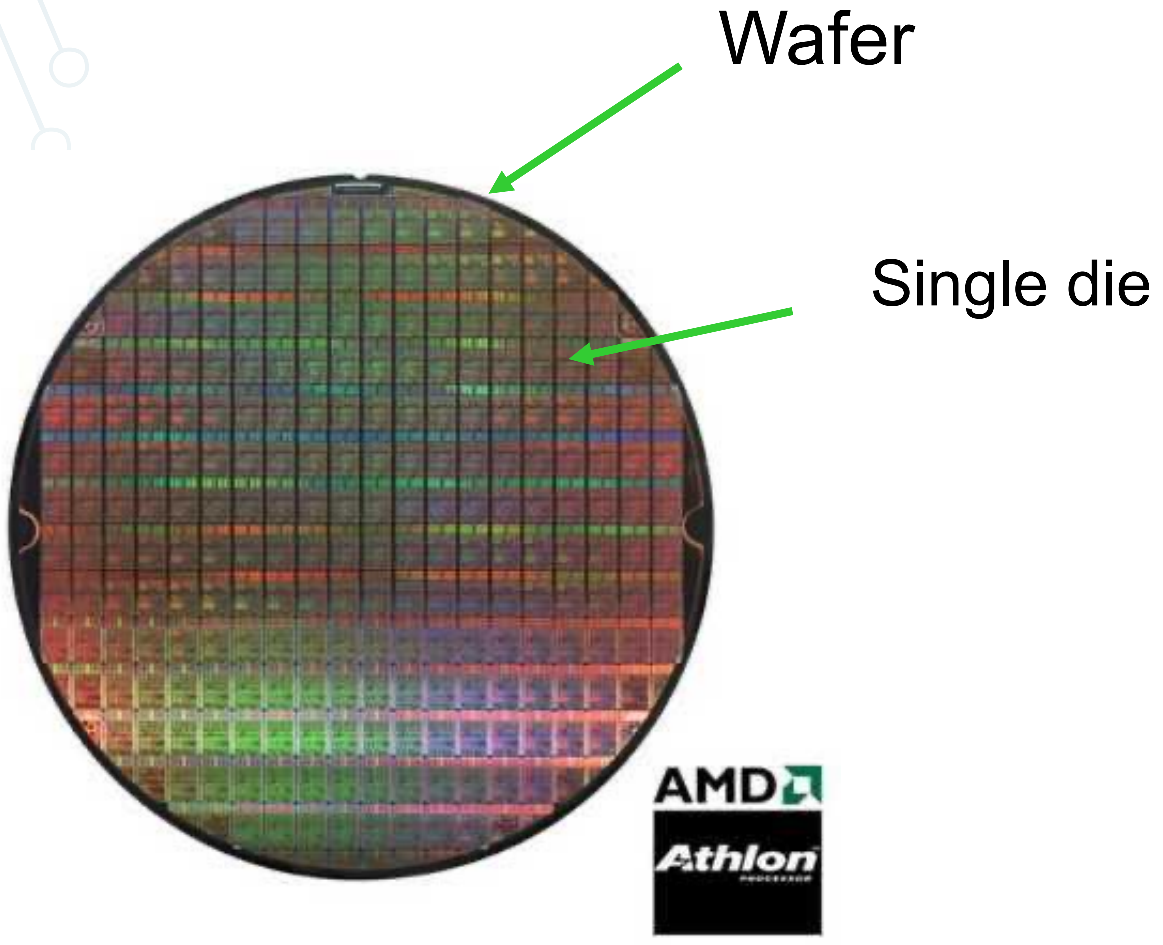
Cost

- Non-recurring engineering (NRE) costs
  - Fixed, one-time cost to research, design, and verify a new piece of HW.
  - Amortized over all units shipped
    - E.g. $20M in development adds $.20 to each of 100M units
- Recurring costs
  - Cost to manufacture, test and package a unit
  - Processed wafer cost is ~10k (around 16nm node) which yields:
    - 1 Cerebras chip
    - 50-100 large FPGAs or GPUs
    - 200 laptop CPUs
    - >1000 cell phone SoCs

$$\boxed{\text{cost per IC} = \text{variable cost per IC} + \frac{\text{fixed cost}}{\text{volume}}}$$

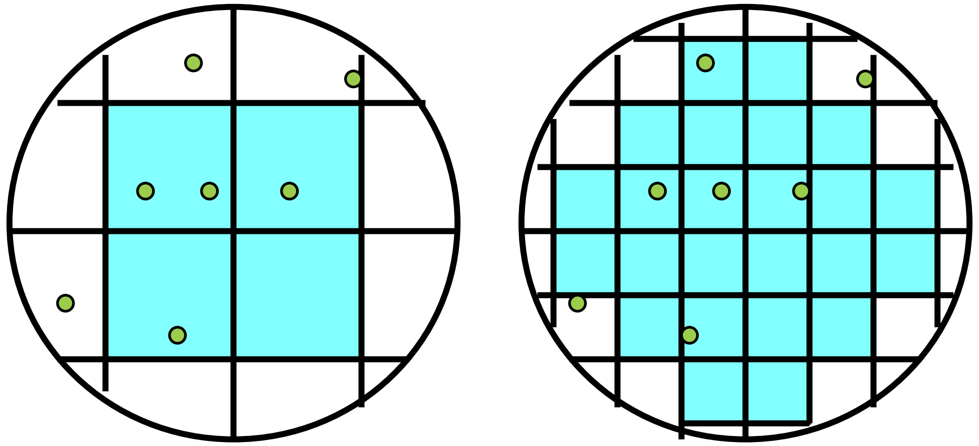
$$\boxed{\text{variable cost} = \frac{\text{cost of die} + \text{cost of die test} + \text{cost of packaging}}{\text{final test yield}}}$$

Die Yield



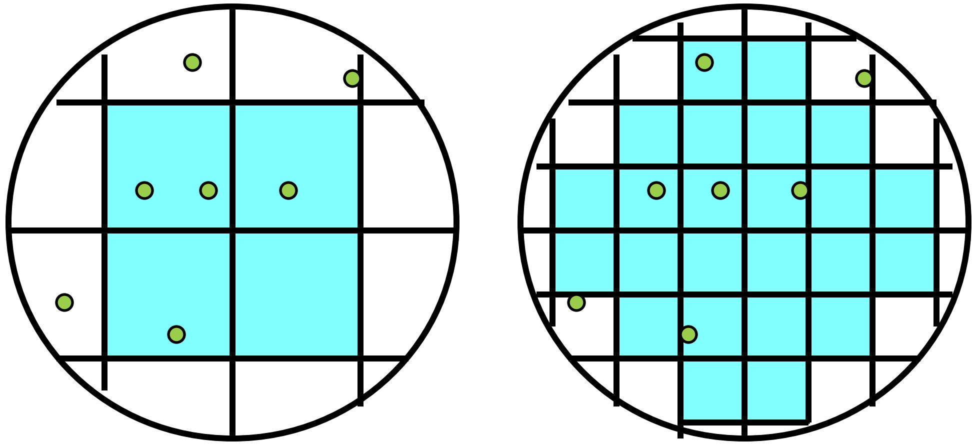
$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$Die\ Yield = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\%$$



Defects

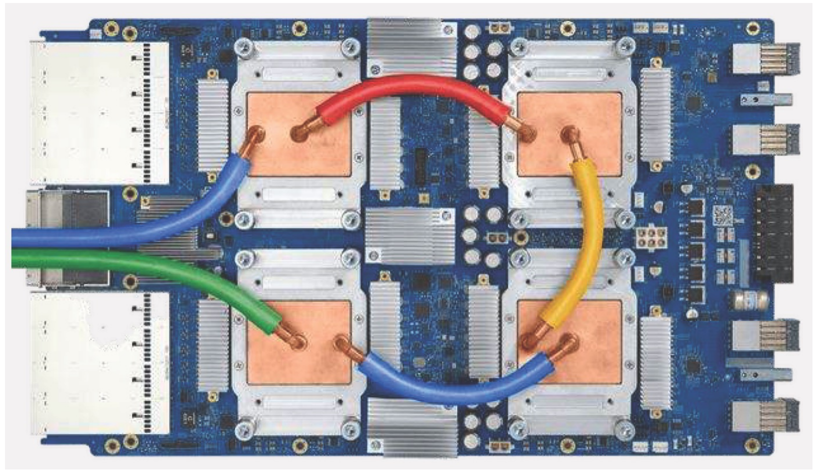
Yield = 0.25



Yield = 0.76

Performance

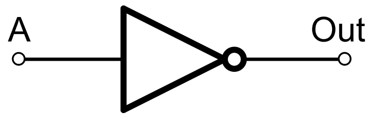
- Throughput
  - Number of tasks performed in a unit of time (operations per second)
    - E.g. Google TPUv3 board performs 420 TFLOPS (10^{12} floating-point operations per second, where a floating-point operation is BFLOAT16)
  - Watch out for 'op' definitions – can be a 1-b ADD or a double-precision FP add (or more complex task)
  - Peak vs. average throughput

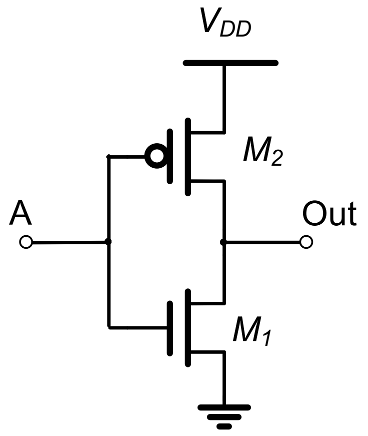


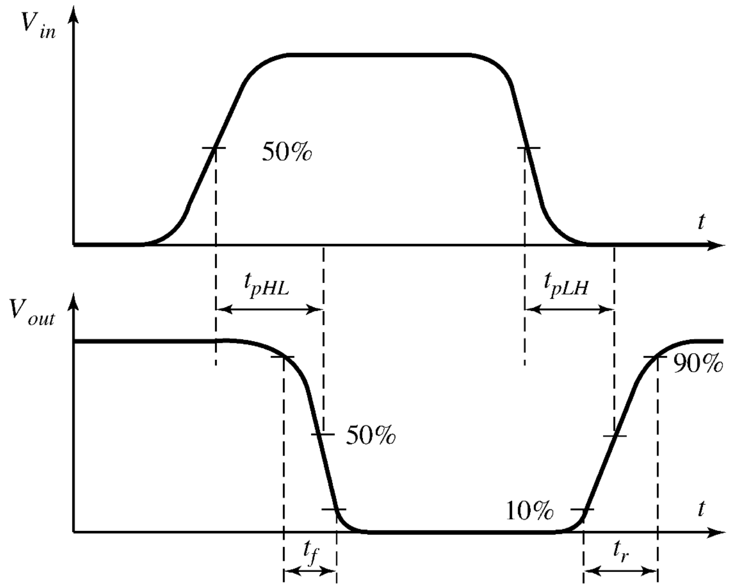
- Latency
  - How long does a task take from start to finish
  - E.g. facial recognition on a phone takes 10's of ms
  - Sometime expressed in terms of clock cycles
  - Average vs. 'tail' latency

Digital Logic Delay

- Changes at the inputs do not instantaneously appear at the outputs
  - There are finite resistances and capacitances in each gate…
- Propagation delay t_p of a logic **gate**:
  - How quickly its output responds to a change at its inputs.
  - Defined as 50% transition points of the input and output waveforms.
  - High-to-low transition: t_{pHL}
  - Low-to-high transition: t_{pLH}
- Rise and fall times t_r t_f
  - For individual **signals**
  - How fast a signal transits
  - 10% - 90%

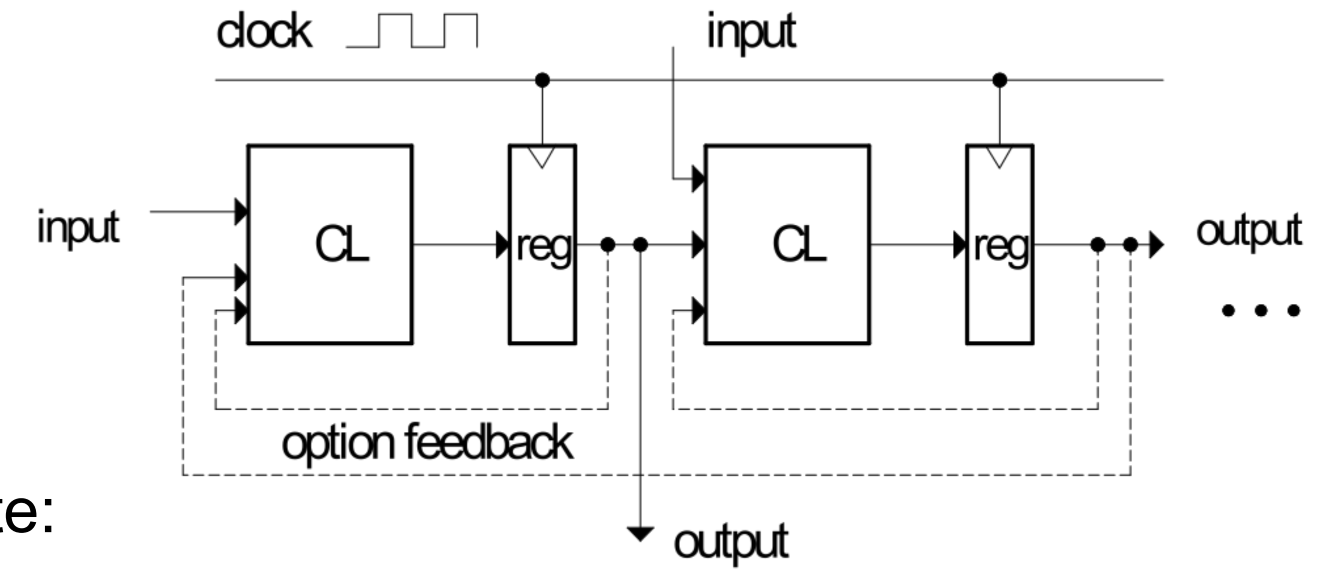






Digital Logic Timing

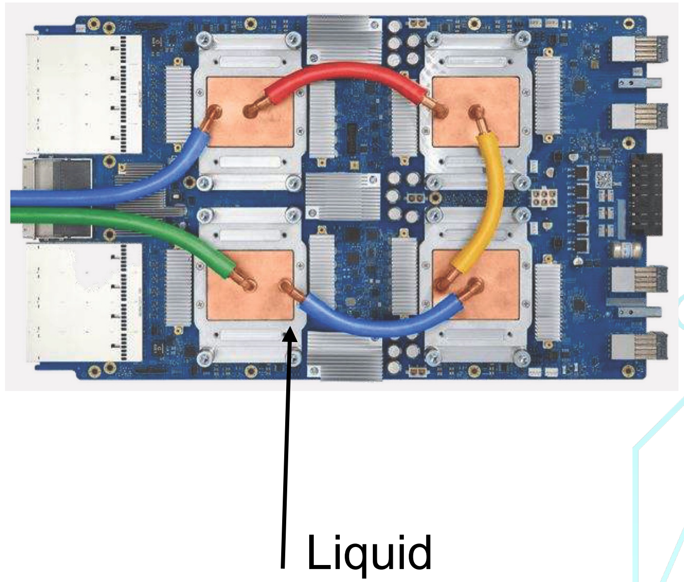
- The longest propagation delay through CL blocks sets the maximum achievable clock frequency.
  - Also need to consider register!



- To increase clock rate:
  - Find the longest path
  - Make it faster

Energy and Power

- Energy (in joules (J))
  - Needed to perform a task
  - Add two numbers or fetch a datum from memory
  - Active and standby
  - Battery stores certain amount of energy (in Ws = J or Wh)
  - That is what utility charges for (in kWh)
- Power (in watts (W))
  - Energy dissipated in time (W = J/s)
  - Peak power vs average powr
  - Sets cooling requirements
    - Heat spreader, size of a heat sink, forced air, liquid, …



Administrivia

- Lab 1 starts this week!
  - Complete Lab 1 on your own and ask for help in office hours.
- Homework 1 this week, due next Friday
  - Start early.
- Discussion 1 starts this week.
  - Know your classmates.

- **Metrics:**
  - **Functionality and Robustness**
  - **Cost**
  - **Performance**
  - **Power and Energy**
- **Verilog!**
  - **Overview**
  - **Combinational Logic**

Hardware **Description** Languages

Verilog:
- Simple C-like syntax for structural and behavior hardware constructs
- Mature set of commercial tools for synthesis and simulation
- Used in EECS 151 / 251A

VHDL:
- Semantically very close to Verilog
- More syntactic overhead
- Extensive type system for “synthesis time” checking

System Verilog:
- Enhances Verilog with strong typing along with other additions

BlueSpec:
- Invented by Prof. Arvind at MIT
- Originally built within the Haskell programming language
- Now available commercially: bluespec.edu

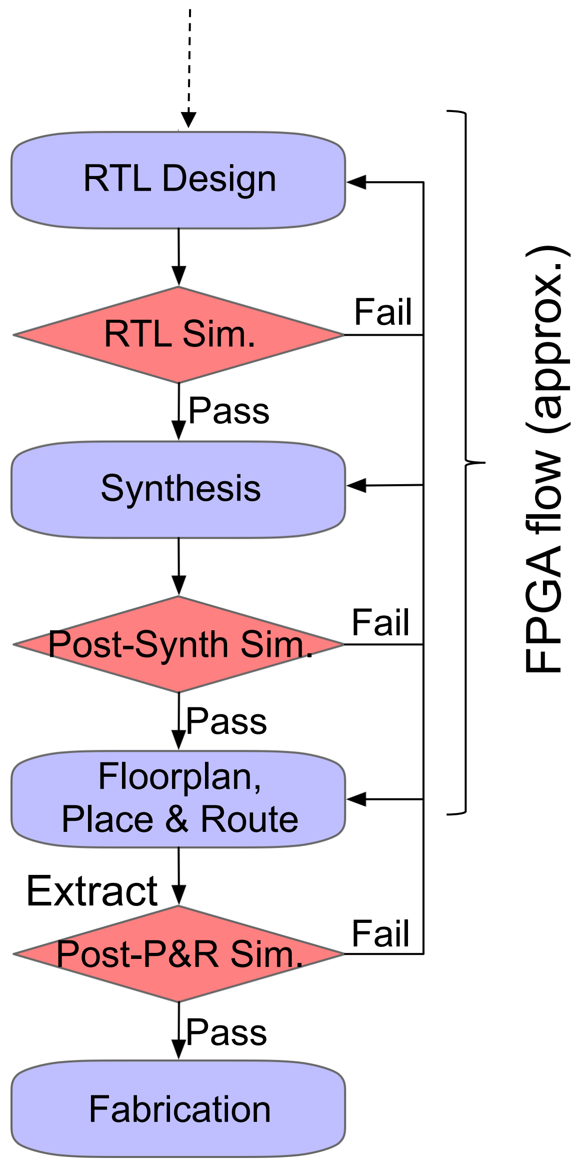
Chisel:
- Developed at UC Berkeley
- Used in CS152, CS250
- Available at: chisel.eecs.berkeley.edu

Verilog: Brief History

- Originated at Automated Integrated Design Systems (renamed Gateway) in 1985. Acquired by Cadence in 1989.
  - Invented as simulation language. Synthesis was an afterthought. Many of the basic techniques for synthesis were developed at **Berkeley** in the 80’s and applied commercially in the 90’s.
  - Around the same time as the origin of Verilog, the US Department of Defense developed VHDL (A double acronym! VSIC (Very High-Speed Integrated Circuit) HDL). Because it was in the public domain it began to grow in popularity.
  - Afraid of losing market share, Cadence opened Verilog to the public in 1990.
- Verilog is the language of choice of Silicon Valley companies, initially because of high-quality tool support and its similarity to C-language syntax.
- VHDL is still popular within the government, in Europe and Japan, and some universities.
- Most major CAD frameworks now support both.

Logic Synthesis

- Verilog and VHDL started out as simulation languages but soon programs were written to automatically convert Verilog code into low-level circuit descriptions (netlists).
- Synthesis converts Verilog (or other HDL) descriptions to an implementation using technology-specific primitives:
  - For FPGA: LUTs, FlipFlops, and BRAMs.
  - For ASICs: standard cells and memory macros.

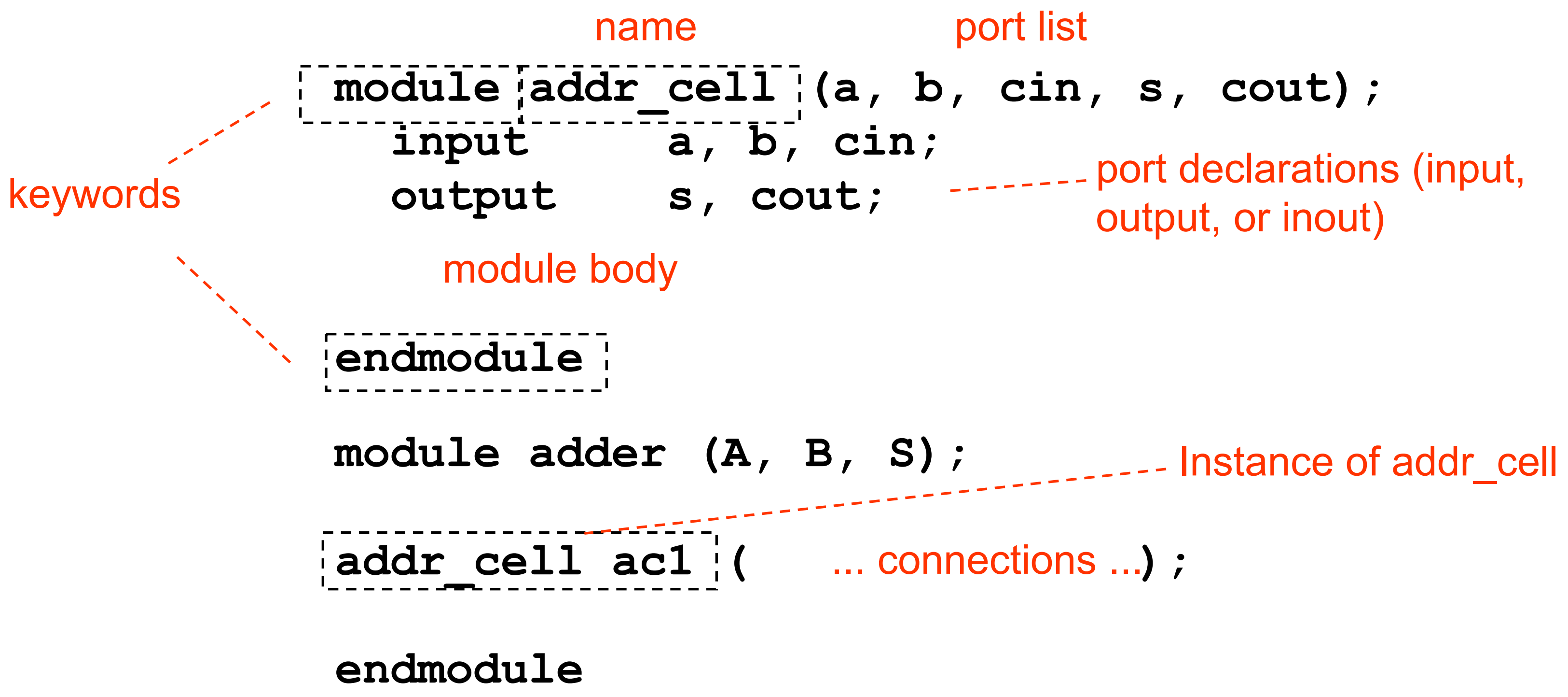


Verilog Introduction

- A `module` definition describes a component in a circuit
- Two ways to describe module contents:
  - **Structural Verilog**
    - List of sub-components and how they are connected
    - Just like schematics, but using text
    - tedious to write, hard to decode
    - You get precise control over circuit details
    - May be necessary to map to special resources of the FPGA/ASIC
  - **Behavioral Verilog**
    - Describe what a component does, not how it does it
    - Synthesized into a circuit that has this behavior
    - Result is only as good as the tools
- Build up a hierarchy of modules. Top-level module is your entire design (or the environment to test your design).

Verilog Modules and Instantiation

- Modules define circuit components.
- Instantiation defines hierarchy of the design.



```
module addr_cell (a, b, cin, s, cout);
   input     a, b, cin;
   output    s, cout;

endmodule

module adder (A, B, S);

addr_cell ac1 (    ... connections ...);

endmodule
```

Note: A module is not a function in the C sense. There is no call and return mechanism. Think of it more like a hierarchical data structure.

Structural Model - XOR example

module name

port list

port declarations

internal signal declarations

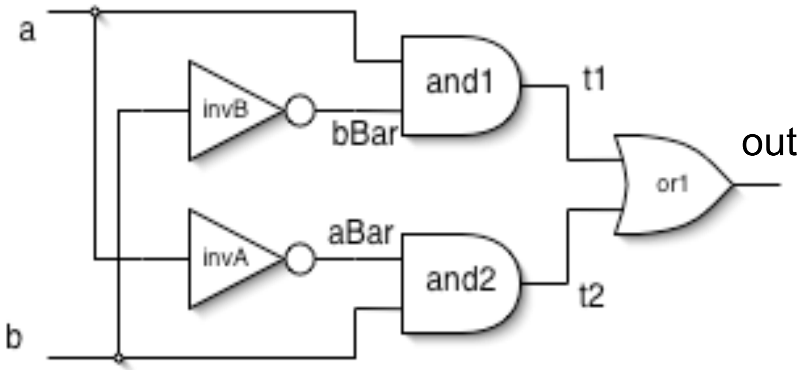
Built-in gates

instances

```
module xor_gate ( out, a, b );
   input      a, b;
   output     out;
   wire       aBar, bBar, t1, t2;

   not invA (aBar, a);
   not invB (bBar, b);
   and and1 (t1, a, bBar);
   and and2 (t2, b, aBar);
   or  or1 (out, t1, t2);

endmodule
```

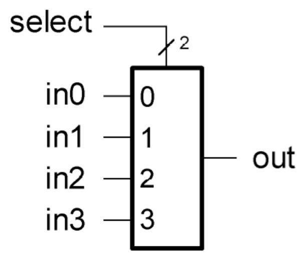


Interconnections (note output is first)

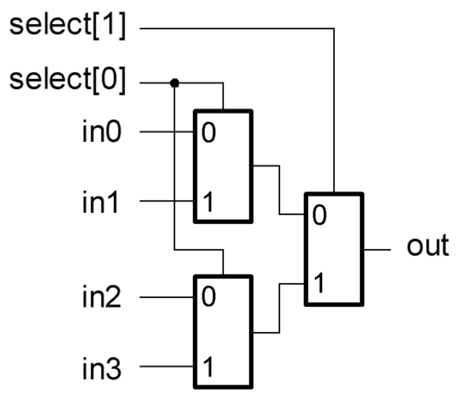
Instance name

- Notes:
  - The instantiated gates are not "executed". They are active always.
  - xor gate already exists as a built-in (so really no need to define it).
  - Undeclared variables assumed to be wires. Don't let this happen to you!

Instantiation, Signal Array, Named ports



a) 4-input mux symbol



b) 4-input mux implemented with 2-input muxes

```
/* 2-input multiplexor in gates */
module mux2 (in0, in1, select, out);
   input in0,in1,select;
   output out;
   wire s0,w0,w1;
   not (s0, select);
   and (w0, s0, in0),
       (w1, select, in1);
   or  (out, w0, w1);
endmodule // mux2
```

```
module mux4 (in0, in1, in2, in3, select, out);
input in0,in1,in2,in3;
input [1:0] select;
output out;
wire w0,w1;
  mux2
    m0 (.select(select[0]), .in0(in0), .in1(in1), .out(w0)),
    m1 (.select(select[0]), .in0(in2), .in1(in3), .out(w1)),
    m3 (.select(select[1]), .in0(w0), .in1(w1), .out(out));
endmodule // mux4
```

Signal array. Declares select[1], select[0]

Named ports. Highly recommended.

Simple Behavioral Model

```
module foo (out, in1, in2);
  input         in1, in2;
  output        out;

    assign out = in1 & in2;

endmodule
```

"continuous assignment"

Connects out to be the logical "and" of in1 and in2.

Short-hand for explicit instantiation of bit-wise"and" gate (in this case).

The assignment continuously happens, therefore any change on the rhs is reflected in **`out`** immediately (except for the small delay associated with the implementation of the practical &).

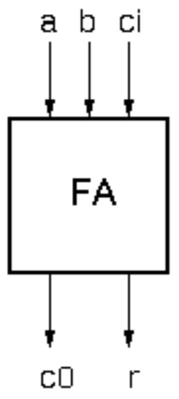
Not like an assignment in C that takes place when the program counter gets to that place in the program.

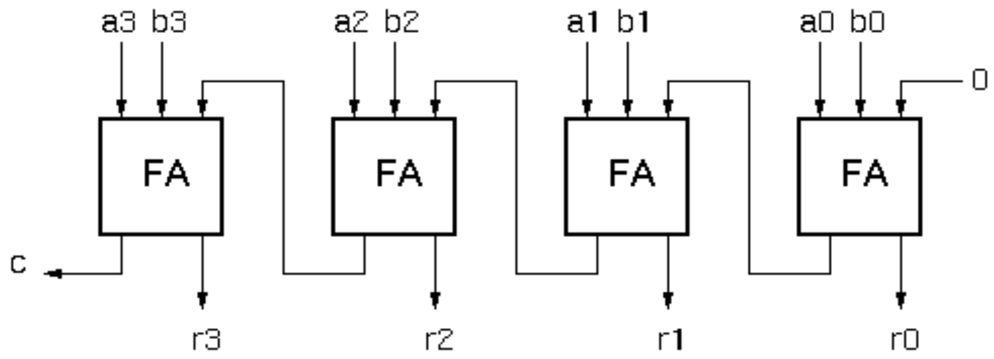
Example - Ripple Adder

```
module FullAdder(a, b, ci, r, co);
   input a, b, ci;
   output r, co;

   assign r = a ^ b ^ ci;
  assign co = a&ci | a&b | b&cin;

endmodule
```

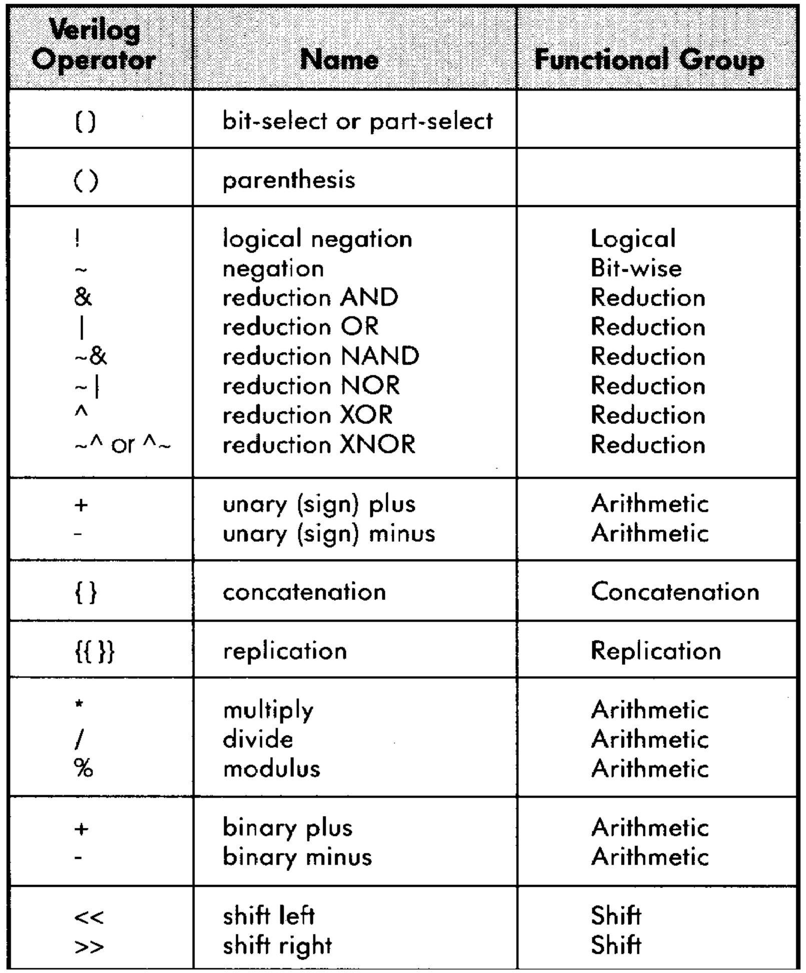




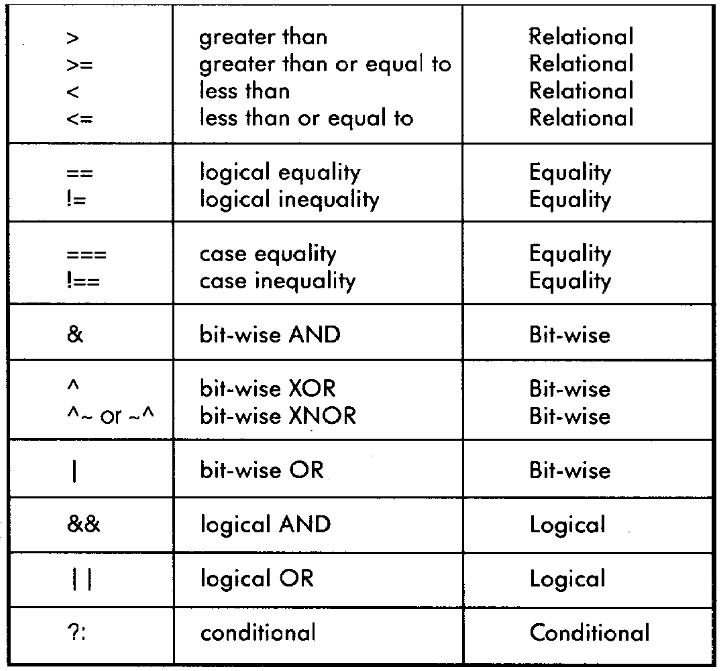
```
module Adder(A, B, R);
   input [3:0] A;
   input [3:0] B;
   output [4:0] R;

   wire c1, c2, c3;
   FullAdder
   add0(.a(A[0]), .b(B[0]), .ci(1'b0), .co(c1),   .r(R[0]) ),
   add1(.a(A[1]), .b(B[1]), .ci(c1),   .co(c2),   .r(R[1]) ),
   add2(.a(A[2]), .b(B[2]), .ci(c2),   .co(c3),   .r(R[2]) ),
   add3(.a(A[3]), .b(B[3]), .ci(c3),   .co(R[4]), .r(R[3]) );
endmodule
```

Verilog Operators & Logic Values



Verilog Operator	Name	Functional Group
[ ]	bit-select or part-select	
( )	parenthesis	
!	logical negation	Logical
~	negation	Bit-wise
&	reduction AND	Reduction
|	reduction OR	Reduction
~&	reduction NAND	Reduction
~|	reduction NOR	Reduction
^	reduction XOR	Reduction
~^ or ^~	reduction XNOR	Reduction
+	unary (sign) plus	Arithmetic
-	unary (sign) minus	Arithmetic
{ }	concatenation	Concatenation
{{ }}	replication	Replication
*	multiply	Arithmetic
/	divide	Arithmetic
%	modulus	Arithmetic
+	binary plus	Arithmetic
-	binary minus	Arithmetic
<<	shift left	Shift
>>	shift right	Shift



>	greater than	Relational
>=	greater than or equal to	Relational
<	less than	Relational
<=	less than or equal to	Relational
==	logical equality	Equality
!=	logical inequality	Equality
===	case equality	Equality
!==	case inequality	Equality
&	bit-wise AND	Bit-wise
^	bit-wise XOR	Bit-wise
^~ or ~^	bit-wise XNOR	Bit-wise
|	bit-wise OR	Bit-wise
&&	logical AND	Logical
||	logical OR	Logical
?:	conditional	Conditional

Continuous Assignment Examples

```
wire [3:0] A, X,Y,R,Z;
wire [7:0] P;
wire r, a, cout, cin;
```

```
assign R = X | (Y & ~Z);                 use of bit-wise Boolean operators

assign r = &X;                           example reduction operator

assign R = (a == 1'b0) ? X : Y;          conditional operator

assign P = 8'hff;                        example constants

assign P = X * Y;                        arithmetic operators (use with care!)

assign P[7:0] = {4{X[3]}, X[3:0]};       (ex: sign-extension)

assign {cout, R} = X + Y + cin;          bit field concatenation

assign Y = A << 2;                       bit shift operator

assign Y = {A[1], A[0], 1'b0, 1'b0};     equivalent bit shift
```

Review

- Design metrics:
  - Functionality and robustness:
  - Cost
  - Performance
  - Power and Energy
- Verilog
  - Hardware description language: describe hardware!
  - Logic synthesis: Verilog -> Gate-level netlists
  - Used in both ASIC and Verilog
  - Assign statement