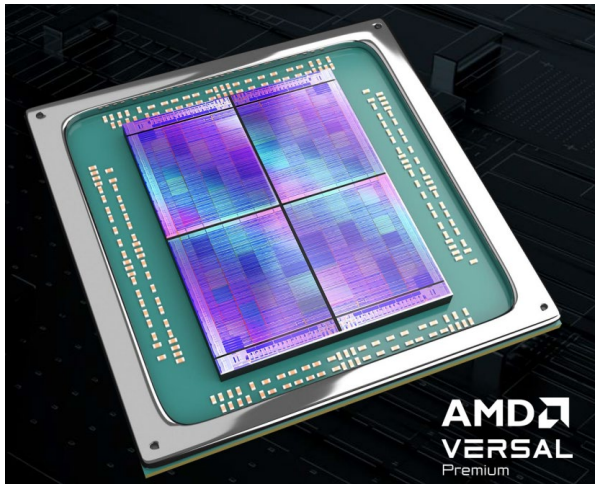
EE1007E : Introduction to Digital Design and ICs

Lecture 4 – HDLs

In July (and at HotChips), AMD announced a new FPGA, built from chiplets

- Novel two-by-two super logic region, SLR, (chiplet) arrangement for enhanced routability and lower latency
- Architectural innovation based on 4th gen SSI (stacked silicon interconnect) technology to maximize performance



D. Gaitonde, HotChips'23

Review

- Design metrics:
  - Robustness, performance, power, cost
- (Brief) History of integrated circuit design

Implementation Alternatives

Implementation Alternative Summary

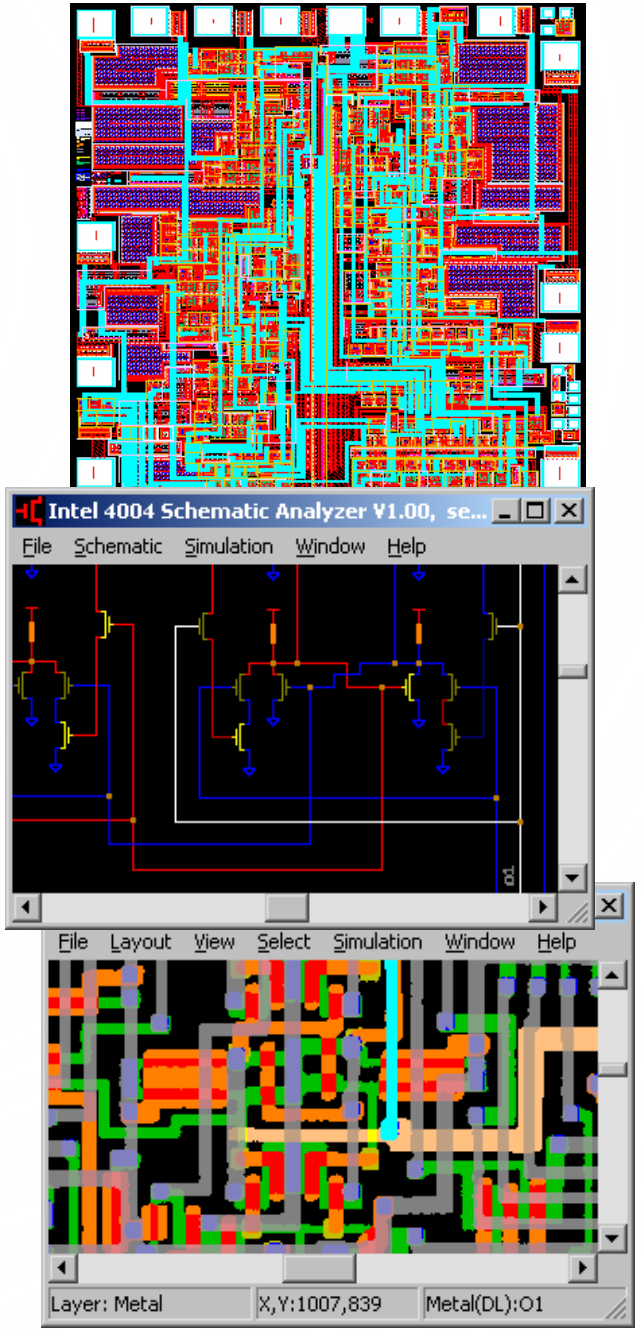
Full-custom:	Every transistors layout hand-drawn and optimized.
Standard-cell:	Logic gates and “macros” automatically placed and routed.
Gate-array (structured ASIC):	Partially prefabricated wafers with arrays of transistors customized with metal layers or vias.
FPGA:	Prefabricated chips that can be customized “in the field”
Microprocessor:	Instruction set interpreter customized through software.
Domain-specific processor:	Special instruction set interpreters (ex: DSP, NPU, GPU).

These days, “ASIC” almost always means standard-cell design.

Full-Custom

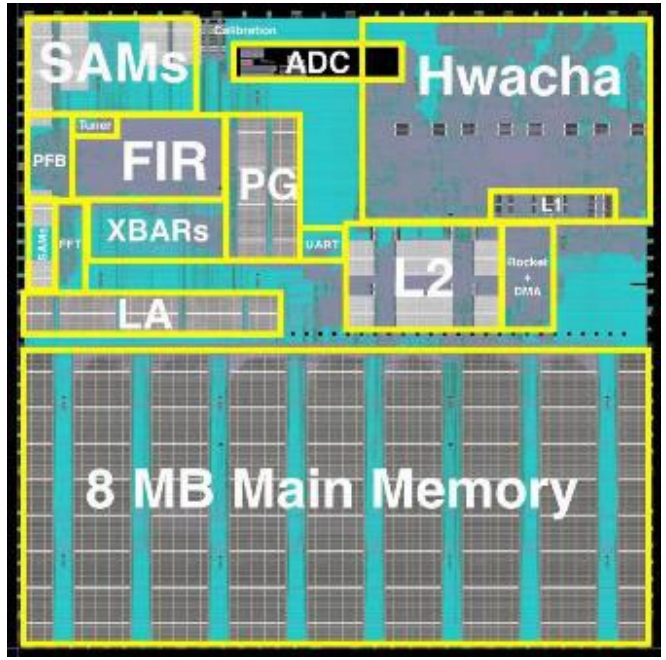
- Circuit styles and transistors are custom sized and drawn to optimize performance and power.
- High NRE (non-recurring engineering) costs
  - Time-consuming layout iterations

- Common today for **analog design.**
- In digital world – for **I/Os, memories** and **arrays.**

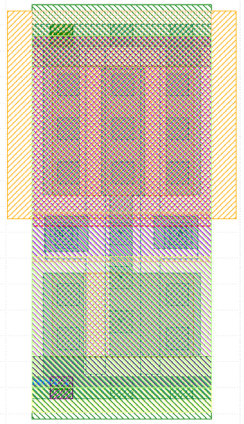


Standard Cells

- Library of logic gates (1000-2000)
  - Combinational: NANDs, NORs, … Sequential: Flip-flops, latches, …
- Each cell comes complete with:
  - layout, schematic, symbol
  - Logic (Verilog), timing, power models.
- Chip layout is automatic, reducing NREs (usually no hand-layout).
- Memories, I/O and analog components complete the ASIC



Berkeley chip from IEEE Journal of Solid-State Circuits

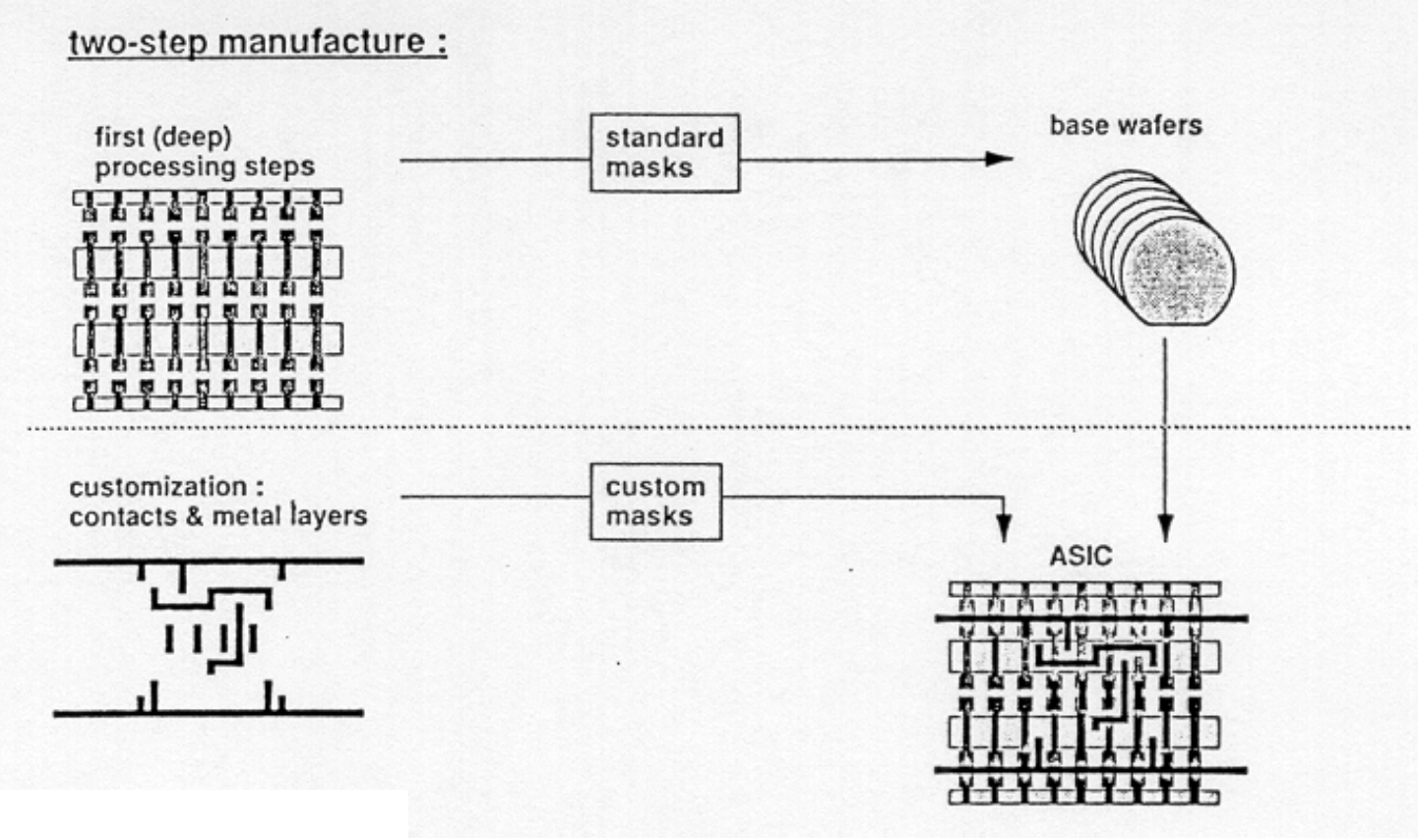


NAND2

SkyWater 130nm process

Gate Array

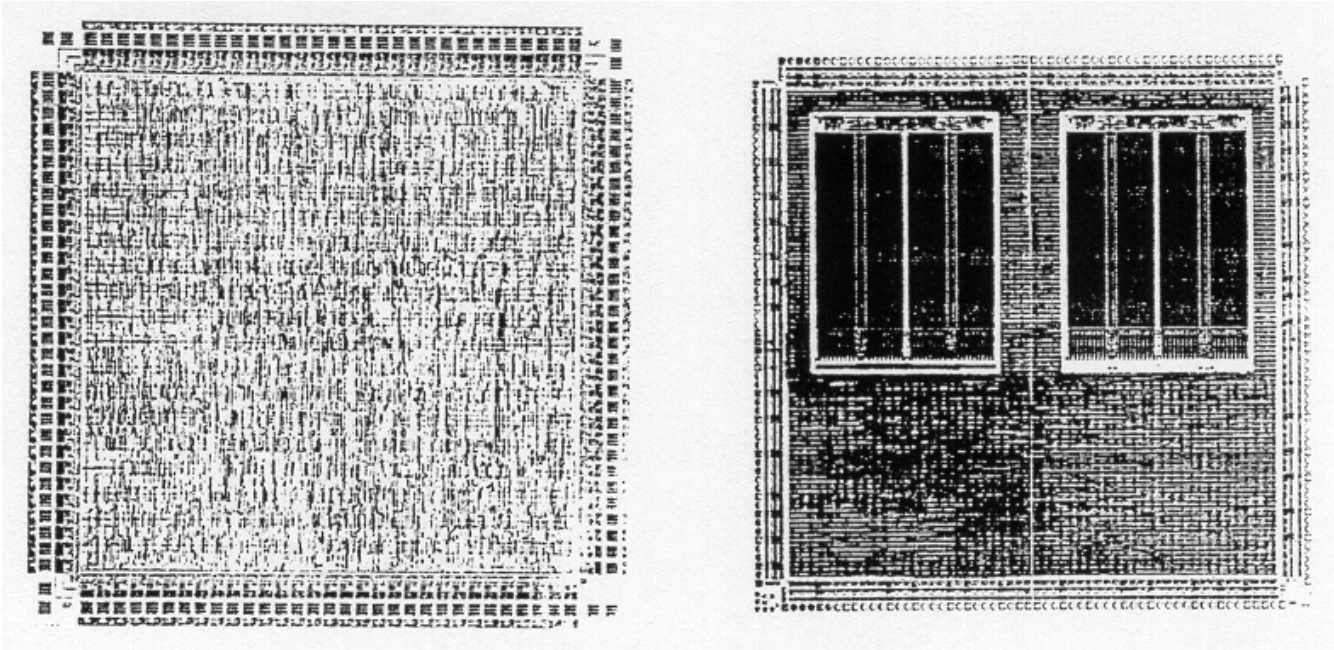
- Prefabricated wafers of rows of transistors. Customize as needed with “back-end” metal processing (contact cuts, metal wires). Could use a different fab.
- CAD software understands how to make gates and registers.



Gate Array

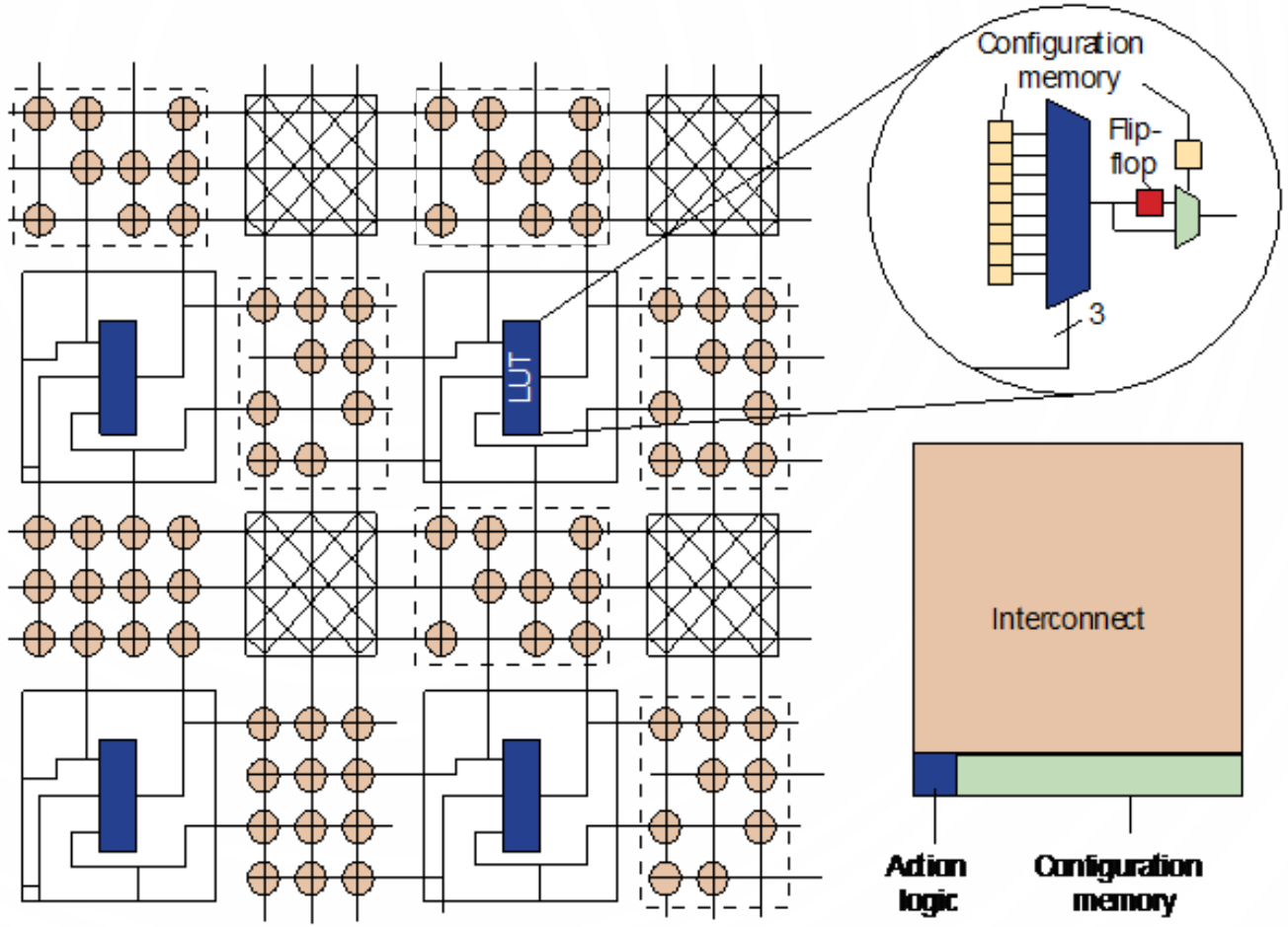
- Shifts large portion of design and mask NRE to vendor.
- Shorter design and processing times, reduced time to market for user.
- Highly structured layout with fixed size transistors leads to large sub-circuits (ex: Flip-flops) and higher per die costs.
- Memory arrays are particularly inefficient, so often prefabricated.
- Displaced by field-programmable gate arrays

Sea-of-gates, structured ASIC, master-slice.



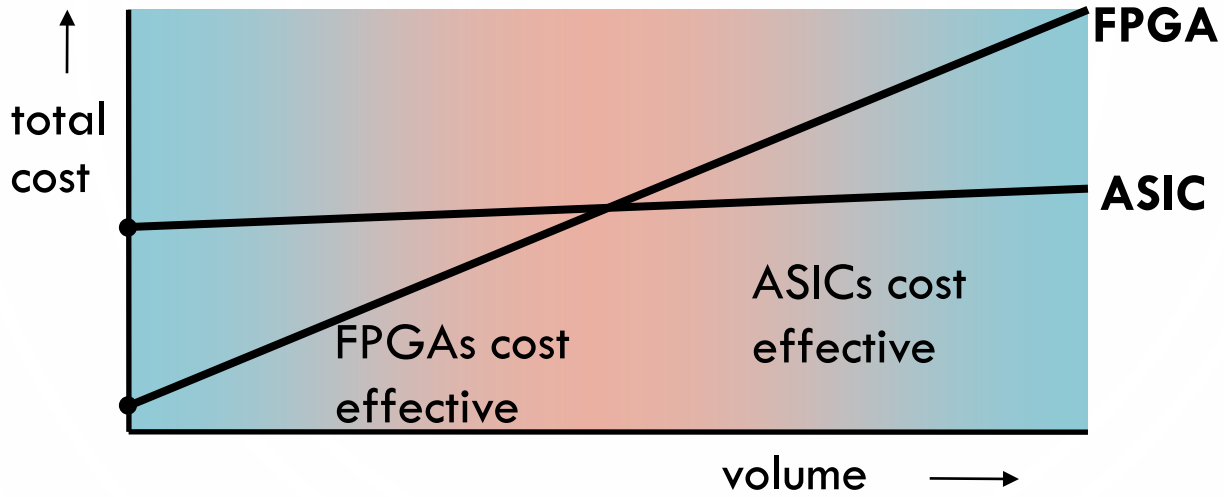
Field Programmable Gate Arrays (FPGA)

- Two-dimensional array of simple logic- and interconnection-blocks.
- Typical architecture: Look-up-tables (LUTs) implement any function of n-inputs (n=3 in this case).
- Optional flip-flop with each LUT.



- Fuses, EPROM, or Static RAM cells are used to store the "configuration".
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Many FPGAs include special circuits to accelerate adder carry-chain and many special cores: RAMs, MAC, Ethernet, USB, PCIe, CPUs, ...

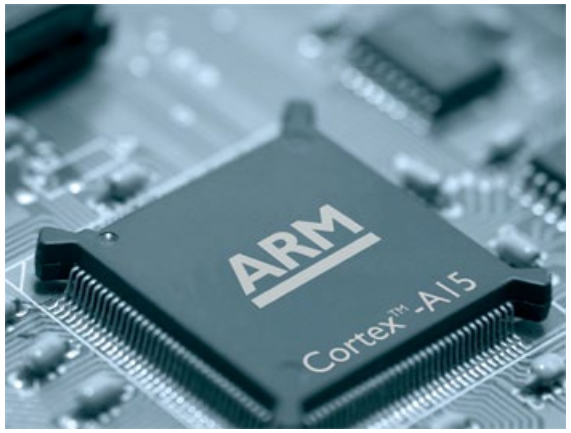
FPGA versus ASIC



- **ASIC:** Higher NRE costs (10's of $M). Relatively Low cost per die (10's of $ or less).
- **FPGAs:** Low NRE costs. Relatively low silicon efficiency ⇒ high cost per part (> 10's of $ to 1000's of $).
- **Cross-over volume** from cost effective FPGA design to ASIC was often in the 100K range.
- **ASICs often have >10x higher performance and 10x lower power for the same application.**

Microprocessors

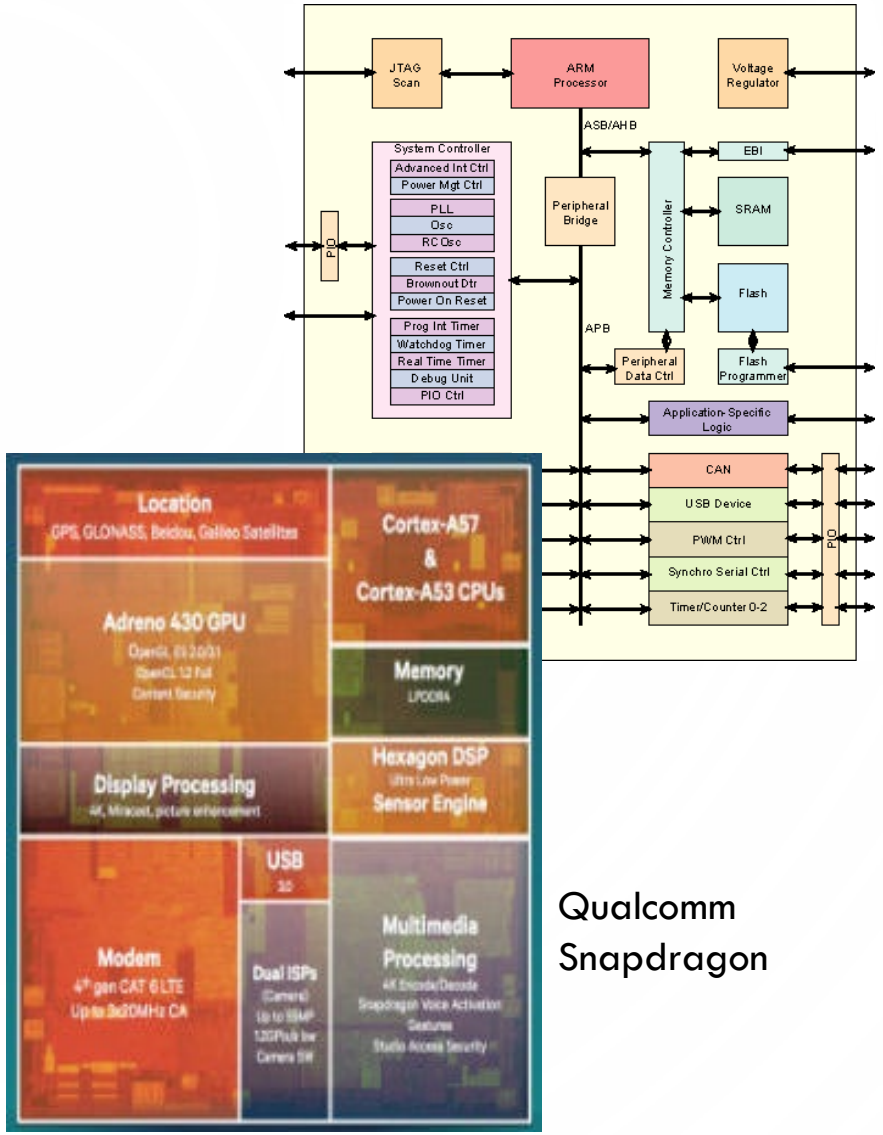
- Where relatively low performance and/or high flexibility is needed, a viable implementation alternative:
  - Software implements desired function
  - "Microcontroller", often with built in nonvolatile program memory and used as single function.
- Two "abstraction" levels:
  - Instruction Set Architecture (ISA)
  - "Synthesizable" RTL model ("soft core", available in HDL)
- Their implementation can be both ASIC or FPGA



§	Assembler
	ADD{cond}{S} Rd, Rn, <Operand2>
	ADC{cond}{S} Rd, Rn, <Operand2>
5E	QADD{cond} Rd, Rm, Rn
5E	QDADD{cond} Rd, Rm, Rn
	SUB{cond}{S} Rd, Rn, <Operand2>
	SBC{cond}{S} Rd, Rn, <Operand2>
	RSB{cond}{S} Rd, Rn, <Operand2>
	RSC{cond}{S} Rd, Rn, <Operand2>
5E	QSUB{cond} Rd, Rm, Rn
5E	QDSUB{cond} Rd, Rm, Rn
2	MUL{cond}{S} Rd, Rm, Rs
2	MLA{cond}{S} Rd, Rm, Rs, Rn
M	UMULL{cond}{S} RdLo, RdHi, Rm, Rs
M	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs
6	UMAAL{cond} RdLo, RdHi, Rm, Rs

System-on-Chip (SOC)

- Brings together: standard cell blocks, custom analog blocks, processor cores, memory blocks, embedded FPGAs, …
- Standardized on-chip buses (or hierarchical interconnect) permit "easy" integration of many blocks.
  - Ex: AXI, …
- "IP Block" business model: Hard- or soft-cores available from third party vendors.
- ARM, inc. is the example. Hard- and "synthesizable" processors.
- ARM and other companies provide, Ethernet, USB controllers, analog functions, memory blocks, …



Qualcomm Snapdragon

- Pre-verified block designs, standard bus interfaces (or adapters) ease integration - lower NREs, shorten TTM.

Administrivia

- Bora's office hour (this week) on Thursday at 1pm
  - Regular office hour on Mondays at 10am
- Lab 1 finished
  - If you couldn't attend the lab, please get checked off in office hours this week
- Lab 2 is more involved
  - Be prepared
  - Verilog primer
- Homework 1 due on Friday
  - Start early

ASIC Design

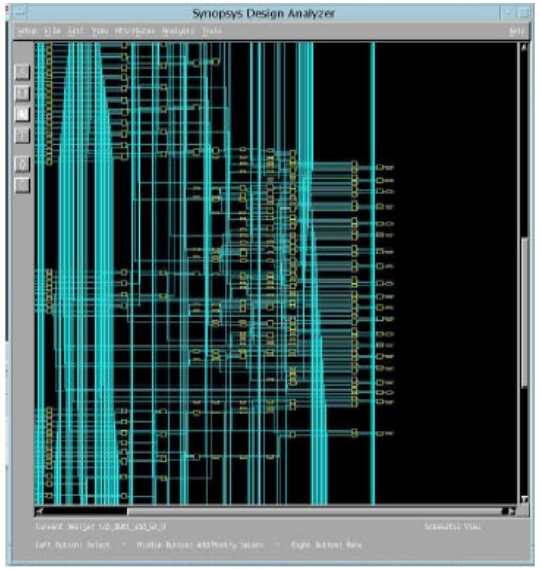
Verilog to ASIC Layout Flow

- "Push-button" approach

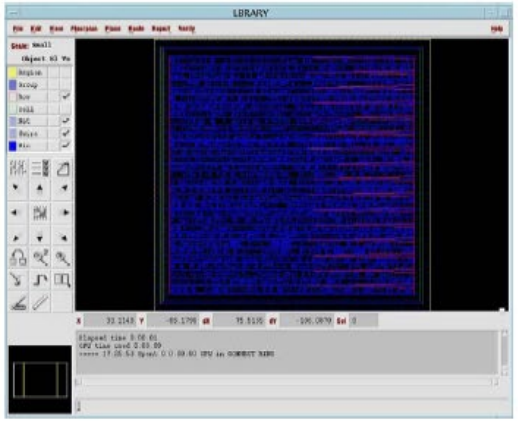
```
module adder64 (a, b, sum);
  input  [63:0] a, b;
  output [63:0] sum;

  assign sum = a + b;
endmodule
```

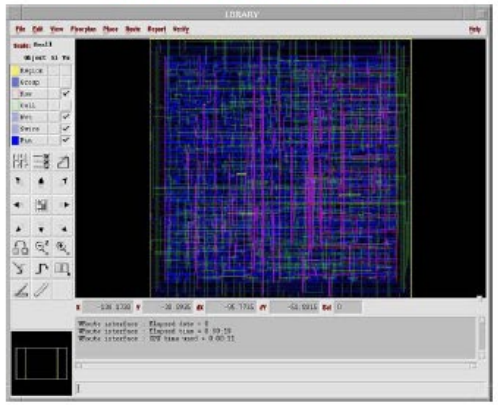
After Synthesis

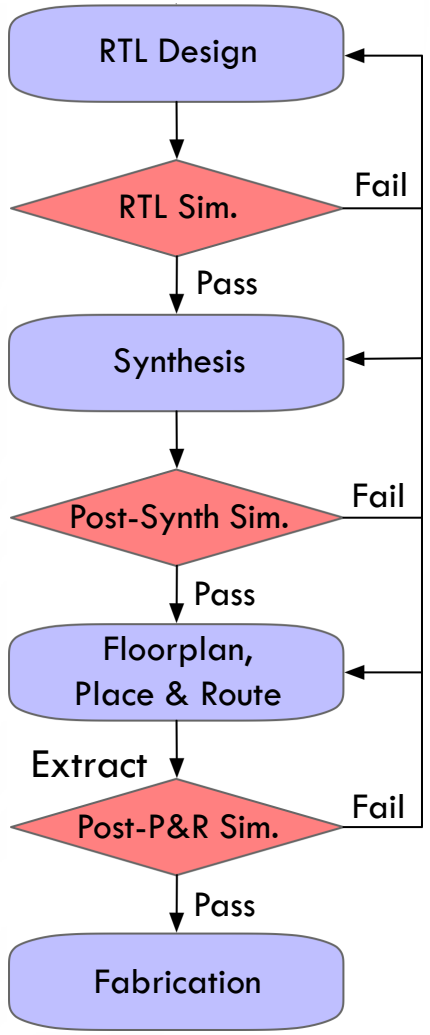


After Placement

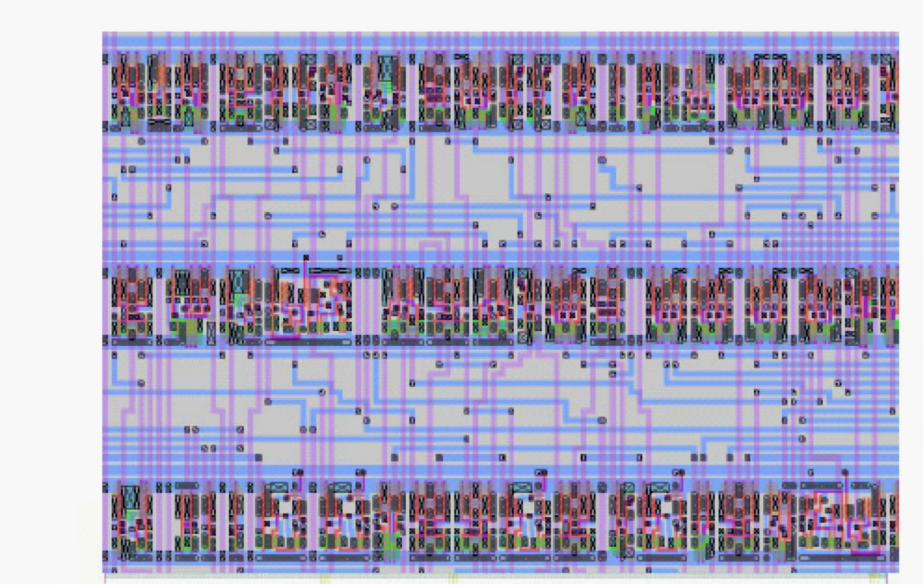


After Routing

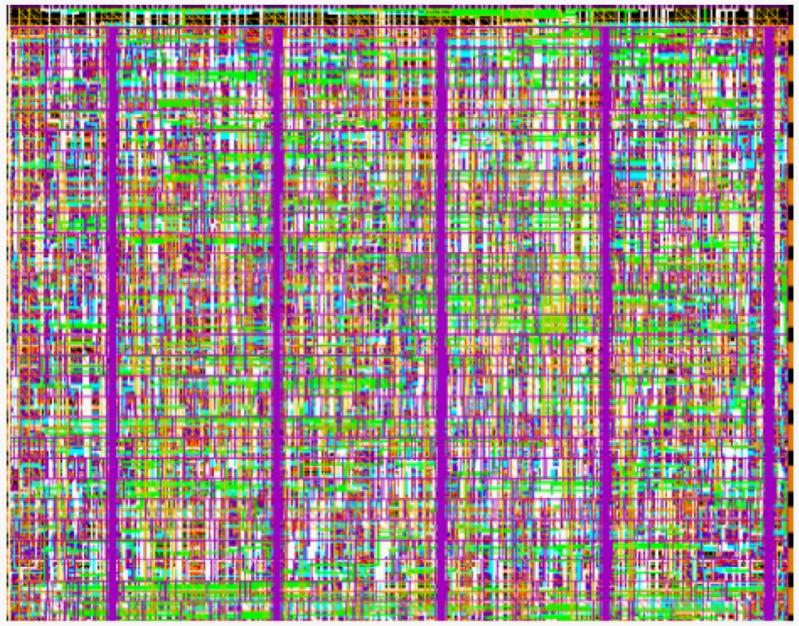




Standard cell layout methodology



Old times: 1μm, 2-metal process



Modern sub-100nm process
"Transistors are free things that fit under wires"

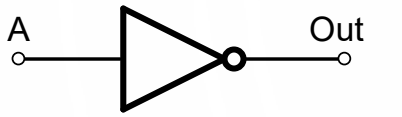
- With limited # metal layers, dedicated routing channels were needed
- Currently area dominated by wires

Standard Cells (ASAP7)

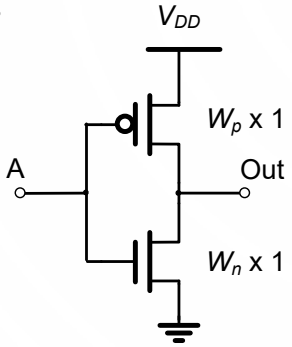
Name

1x Inverter

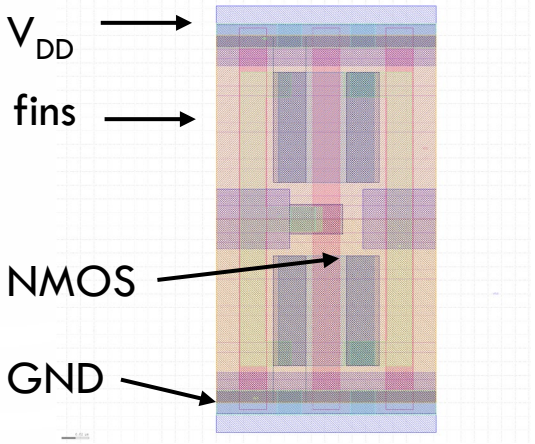
Symbol



Schematic



Layout (gds)



INVX1_ASAP7_75T_R

asap7sc7p5t_24_INVBUF_RVT_TT Cell Library: Process , Voltage 0.70, Temp 25.00

Truth Table

INPUT	OUTPUT
A	Y
0	1
1	0

Footprint

Cell Name	Area
INVx1_ASAP7_75t_R	0.69984

Pin Capacitance Information

Cell Name	Pin Cap(ff)	Max Cap(ff)
	A	Y
INVx1_ASAP7_75t_R	0.43869	46.08000

Leakage Information

Cell Name	Leakage(pW)		
	Min.	Avg	Max.
INVx1_ASAP7_75t_R	0.00000	51.15440	53.32100

Delay Information

Delay(ps) to Y rising :

Cell Name	Timing Arc(Dir)	Delay(ps)		
		Min	Mid	Max
INVx1_ASAP7_75t_R	A->Y (FR)	5.53739	36.47100	284.32000

Verilog for logic sim

```
`celldefine
module INVx1_ASAP7_75t_R (Y, A);
      output Y;
      input A;

      // Function
      not (Y, A);

      // Timing
      specify
            (A => Y) = 0;
      endspecify
endmodule
`endcelldefine
```

Netlist for Spice sim

```
.subckt PM_BUFX10_ASAP7_75T_R%A 2 7 10 13 15 23 27 VSS
c21 27 VSS 0.00191995f $X=0.081 $Y=0.135
c22 23 VSS 0.0244917f $X=0.0565 $Y=0.1325
c23 13 VSS 0.00510508f $X=0.135 $Y=0.135
c24 10 VSS 0.0605852f $X=0.135 $Y=0.0675
c25 2 VSS 0.0649407f $X=0.081 $Y=0.0675
r26 23 27 1.66358 $w=1.8e-08 $l=2.45e-08 $layer=M1
$thickness=3.6e-08 $X=0.0565
...
```

.lib for synthesis

```
cell (INVx11_ASAP7_75t_R) {
   area : 3.03264;
   cell_leakage_power : 562.699;
   pg_pin (VDD) {
     pg_type : primary_power;
     voltage_name : "VDD";
   }
   pg_pin (VSS) {
     pg_type : primary_ground;
     voltage_name : "VSS";
   }
   leakage_power () {
     value : 538.867;
     when : "(A * !Y)";
     related_pg_pin : VDD;
   }
...
```

+ .lef for place and route, and other files for LVS and DRC

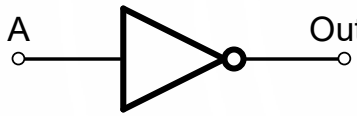
Standard Cells (SkyWater 130nm)

Name

Symbol

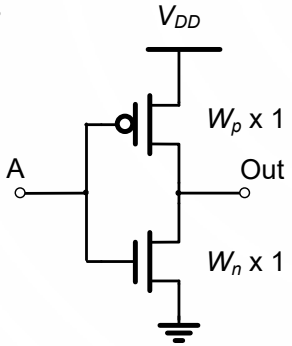
INVx1

1x Inverter

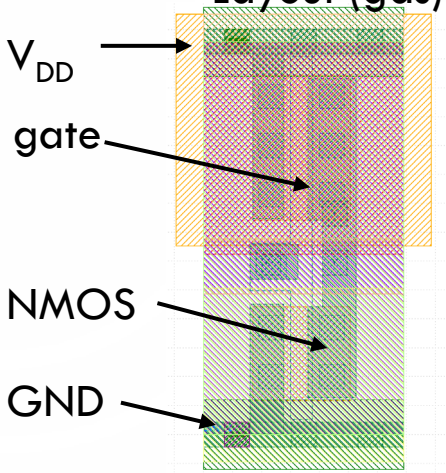


sky130_fd_sc_hd__inv_1

Schematic



Layout (gds)



.lib for synthesis

```
{  "area": 3.7536,
"cell_footprint":
"sky130_fd_sc_hd__inv",
"cell_leakage_power": 7.628985,
"driver_waveform_fall": "ramp",
"driver_waveform_rise": "ramp",
"leakage_power": [    {
"value": 14.5579575,      "when":
"A"    },    {      "value":
0.7000129,      "when": "!A"    }
],  "pg_pin,VGND": {
"pg_type": "primary_ground",
"related_bias_pin": "VPB",
"voltage_name": "VGND"  },
"pg_pin,VNB": {    "pg_type": …
```

Verilog for logic sim

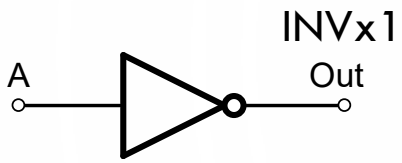
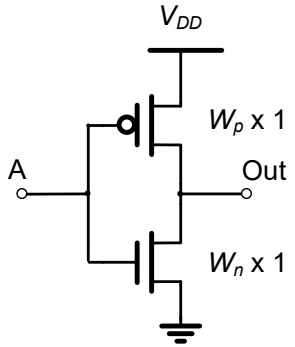
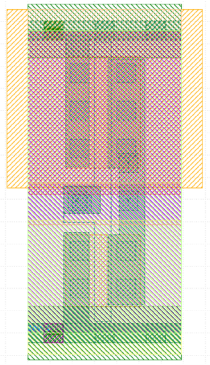
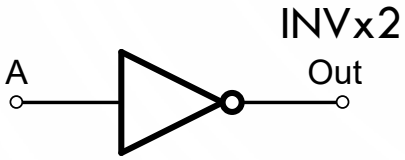
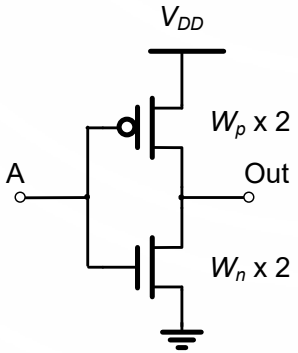
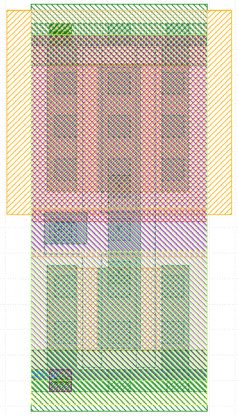
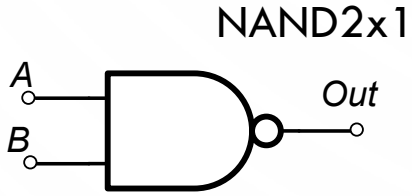
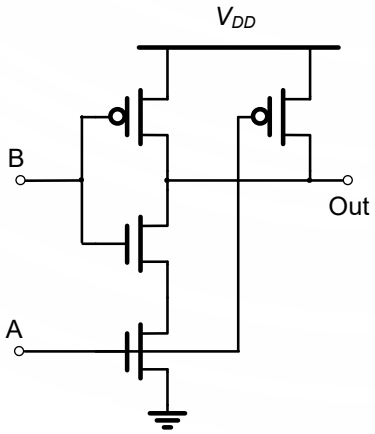
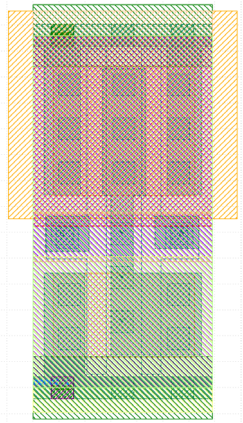
```
`celldefinemodule
sky130_fd_sc_hd__inv_1 (
    Y   ,
    A   ,
    VPWR,
    VGND,
    VPB ,
    VNB);
    output Y   ;
    input  A   ;
    input  VPWR;
    input  VGND;
    input  VPB ;
    input  VNB ;
sky130_fd_sc_hd__inv base (
.Y(Y),         .A(A),
.VPWR(VPWR),         .VGND(VGND),
.VPB(VPB),         .VNB(VNB)
);endmodule`endcelldefine
```

Netlist for Spice sim

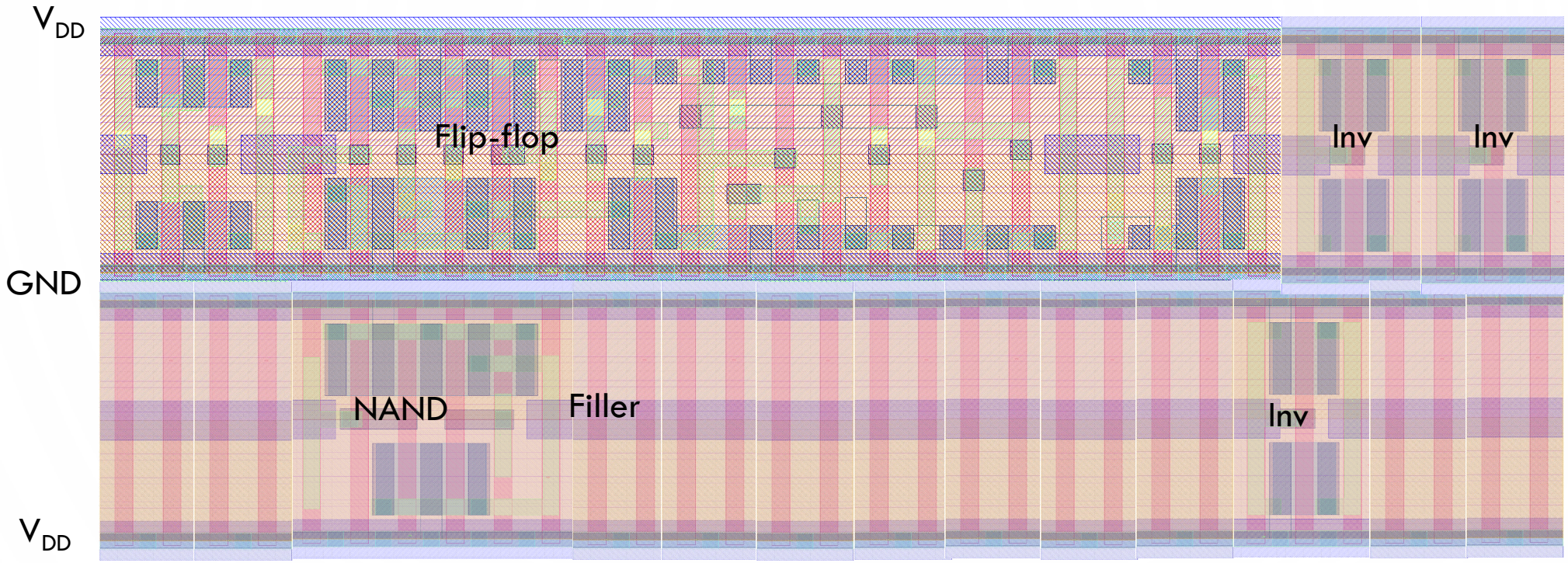
```
.subckt sky130_fd_sc_hd__inv_1 A VGND VNB VPB VPWR YX0 VGND A Y
VNB sky130_fd_pr__nfet_01v8 w=650000u l=150000uX1 VPWR A Y VPB
sky130_fd_pr__pfet_01v8_hvt w=1e+06u l=150000u.ends
```

+ .lef for place and route, and other files

Standard Cells

Name	Symbol	Schematic	Layout
1x Inverter			
2x Inverter			
1x NAND2			

Standard Cell Placement

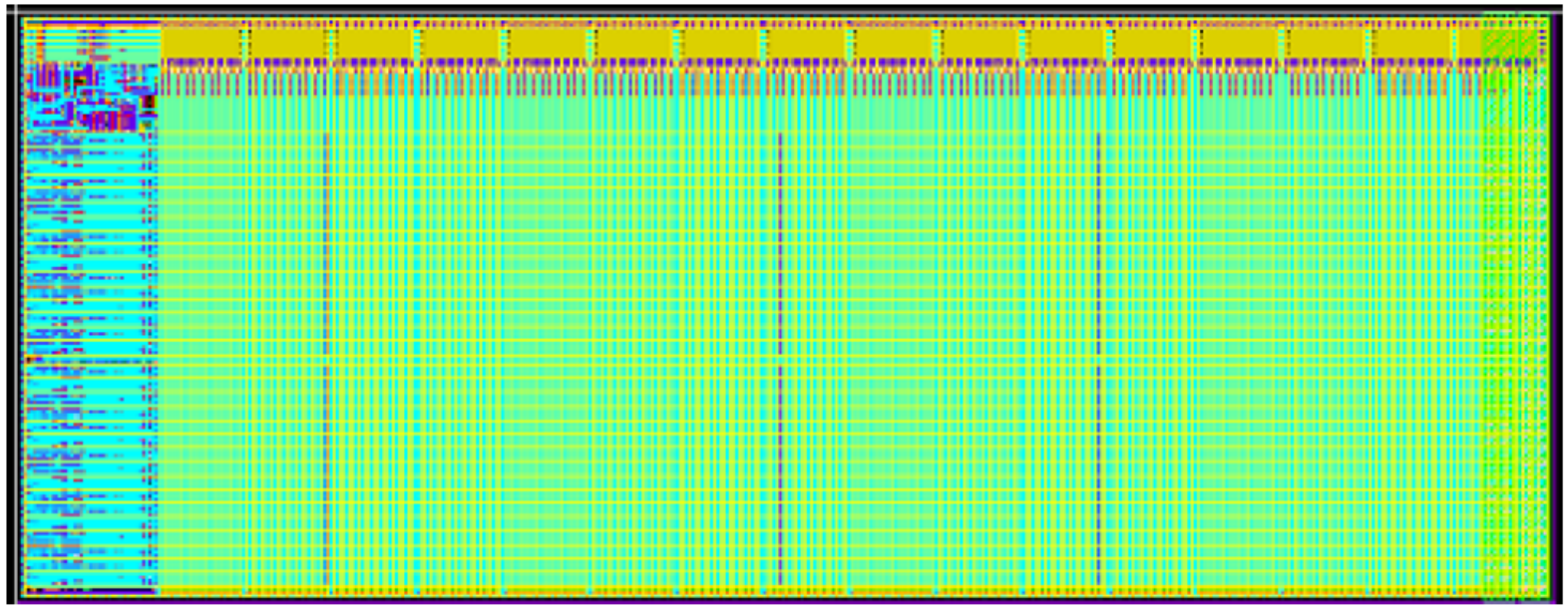


Standard cells abut and flip

Router connects inputs and outputs

Macro modules

256×32 (or 8192 bit) SRAM Generated by hard-macro module generator



- Generate highly regular structures (entire memories, multipliers, etc.) with a few lines of code
- Verilog models for memories automatically generated based on size

Quiz

True or false?

a) FPGAs are usually faster than ASICs

b) Verilog syntax is different for FPGAs and ASICs

c) One cannot change the FPGA function in a deployed product

www.yellkey.com/letter

	abc
1.	FFF
2.	FFT
3.	FTF
4.	FTT
5.	TFF
6.	TFT
7.	TTF
8.	TTT

Hardware Description Languages

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 to support verification

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

Verilog: Brief History

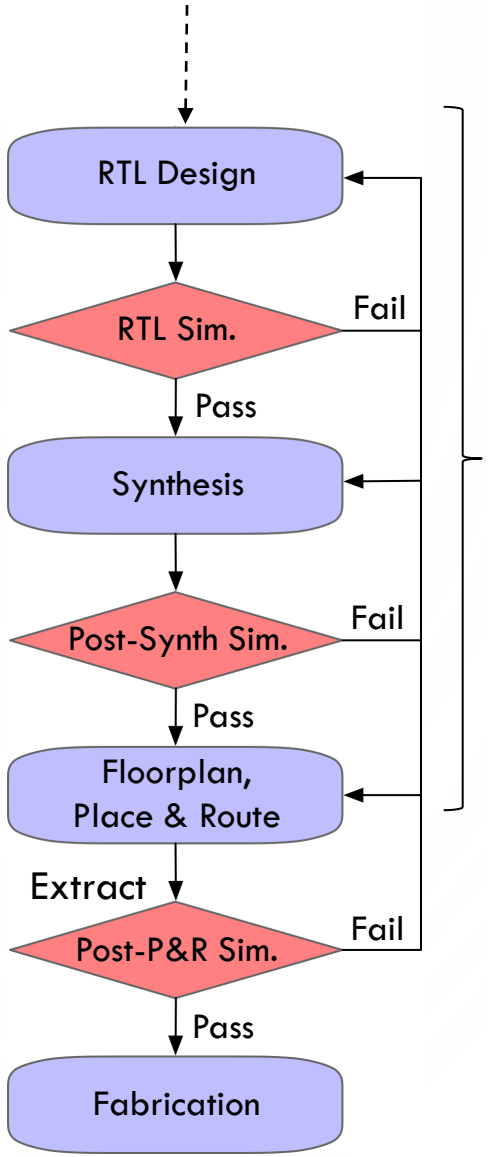
- Originated at Automated Integrated Design Systems (renamed Gateway) in 1985. Acquired by Cadence in 1989.
- Invented as simulation language. Synthesis was an afterthought. Many of the basic techniques for synthesis were developed at Berkeley in the 80's and applied commercially in the 90's.
- Around 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.
- An IEEE working group was established in 1993, and ratified IEEE Standard 1394 (Verilog) in 1995. We use IEEE Std 1364-2005.
- 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.

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).

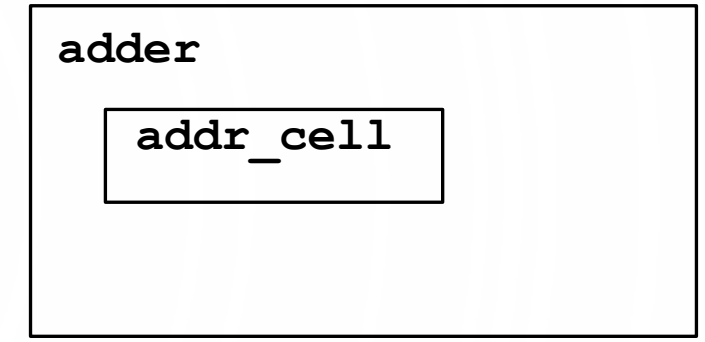
Logic Synthesis

- 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 a logic mapping by using technology-specific primitives:
  - For FPGA: LUTs, flip-flops, and BRAMs
  - For ASICs: standard cells and memory macros
- In addition, synthesis algorithms optimize the implementation for delay and power



Verilog Modules and Instantiation

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



name

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

keywords

port declarations (input, output, or inout)

module body

```
endmodule

module adder (
    input A …
);
addr_cell ac1 (                    );

endmodule
```

Instance of addr_cell

… connections …

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. Testbench is typically the top-level module

Structural Verilog

Structural Model - XOR example

```
module xor_gate (
    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
```

module name

port declarations

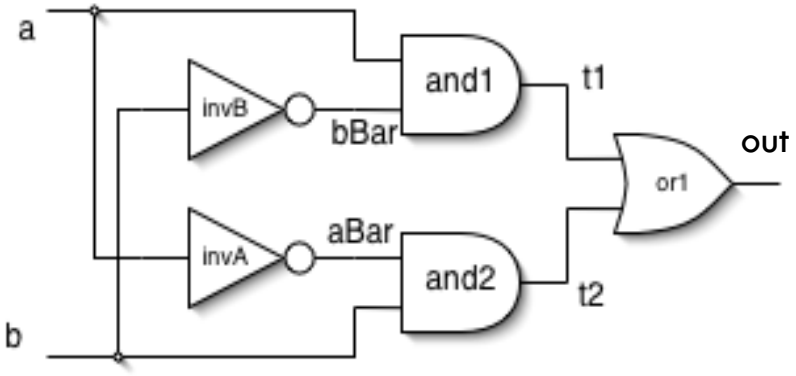
internal signal declarations

Built-in gates

instances

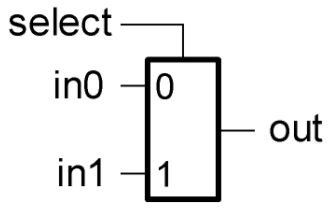
Instance name

Interconnections (note output is first)

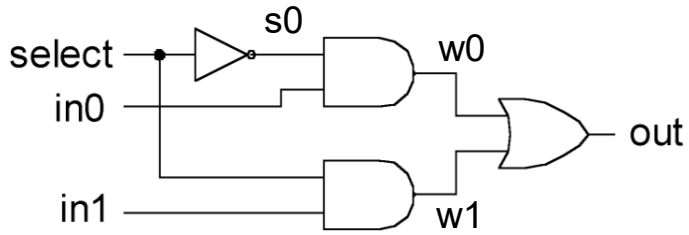


- Notes:
  - The instantiated gates are not “executed”. They are active always.
  - xor gate already exists as a built-in (so really no need to define it).
  - Undeclared variables assumed to be wires. Don’t let this happen to you!

Structural Example: 2-to1 mux



a) 2-input mux symbol



b) 2-input mux gate-level circuit diagram

```
/* 2-input multiplexer in gates */
module mux2 (
    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
```

C++ style comments

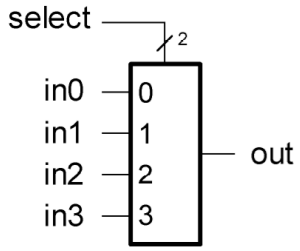
Built-ins don't need instance names

Multiple instances can share the same "main" name.

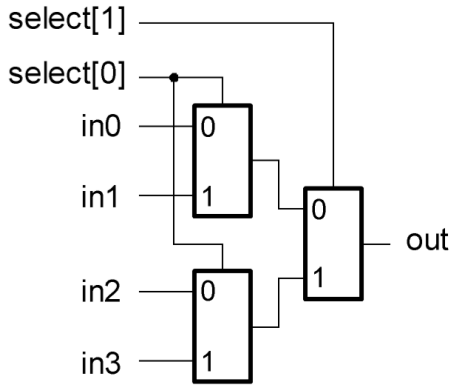
Built-ins gates can have > 2 inputs. Ex:

```
and (w0, a, b, c, d);
```

Instantiation, Signal Array, Named ports



a) 4-input mux symbol



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

```
/* 2-input multiplexer in gates */
module mux2 (
   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 (
    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.

Aside: Netlists

- **Netlist** is a description of the connectivity of an **electronic circuit**
  - Netlist consists of a list of components in a circuit and a list of the nodes they are connected to. A wire (net) connects two or more components
- Structural Verilog is one form of describing a netlist
- Most common netlist format is EDIF (Electronic Design Exchange Format)
  - Established in 1985, still in use

Behavioral Verilog

Simple Behavioral Model

```
module foo (
    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 &).

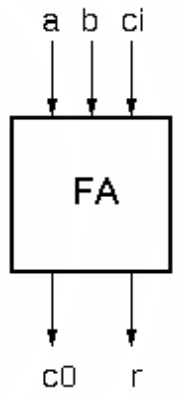
Not like an assignment in C that takes place when the program counter gets to that place in the program.

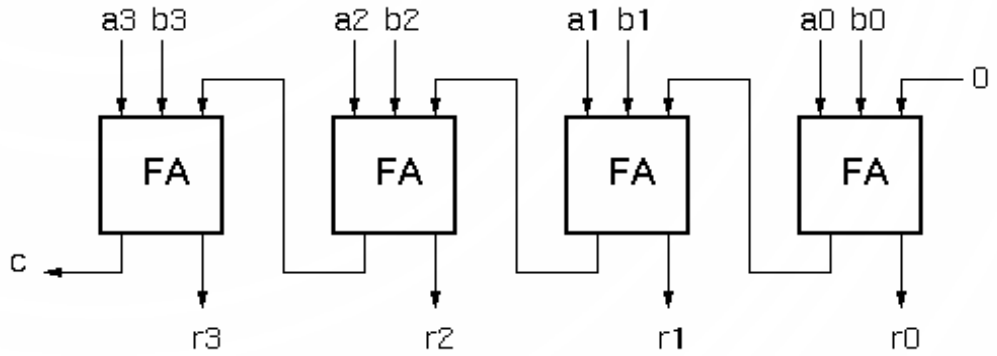
Example - Ripple Adder

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

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

endmodule
```






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

    wire c1, c2, c3;
    full_adder
    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
```

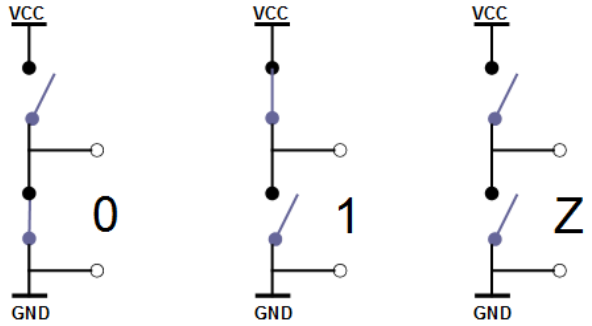
Verilog Operators

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

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

Values

Logic	Description
0	Logic '0' or false
1	Logic '1' or true
x	Don't care or unknown value
z	High impedance state



Verilog Numbers

Constants:

`14` ordinary decimal number

`-14` 2's complement representation

`12'b0000_0100_0110` binary number ("_" is ignored)

`12'h046` hexadecimal number with 12 bits

Signal Values:

By default, values are unsigned

e.g., `C[4:0] = A[3:0] + B[3:0];`

if A = 0110 (6) and B = 1010(-6)
C = 10000 (not 00000)

i.e., B is zero-padded, not sign-extended

`wire signed [31:0] x;`

Declares a signed (2's complement) signal array.

Summary

- Digital systems can be implemented by using general-purpose and specialized processors, FPGAs or ASICs
  - ASICs are the most efficient, but costly
- Hardware description languages are used for simulation, synthesis and verification
  - Verilog, VHDL, Chisel, Bluespec
- Verilog is the most-commonly used HDL
- We have introduced examples of structural and behavioral Verilog