10) VLSI Design styles.

* full custom (standard cell library + custom design)
  for example: designing a 3input and gate

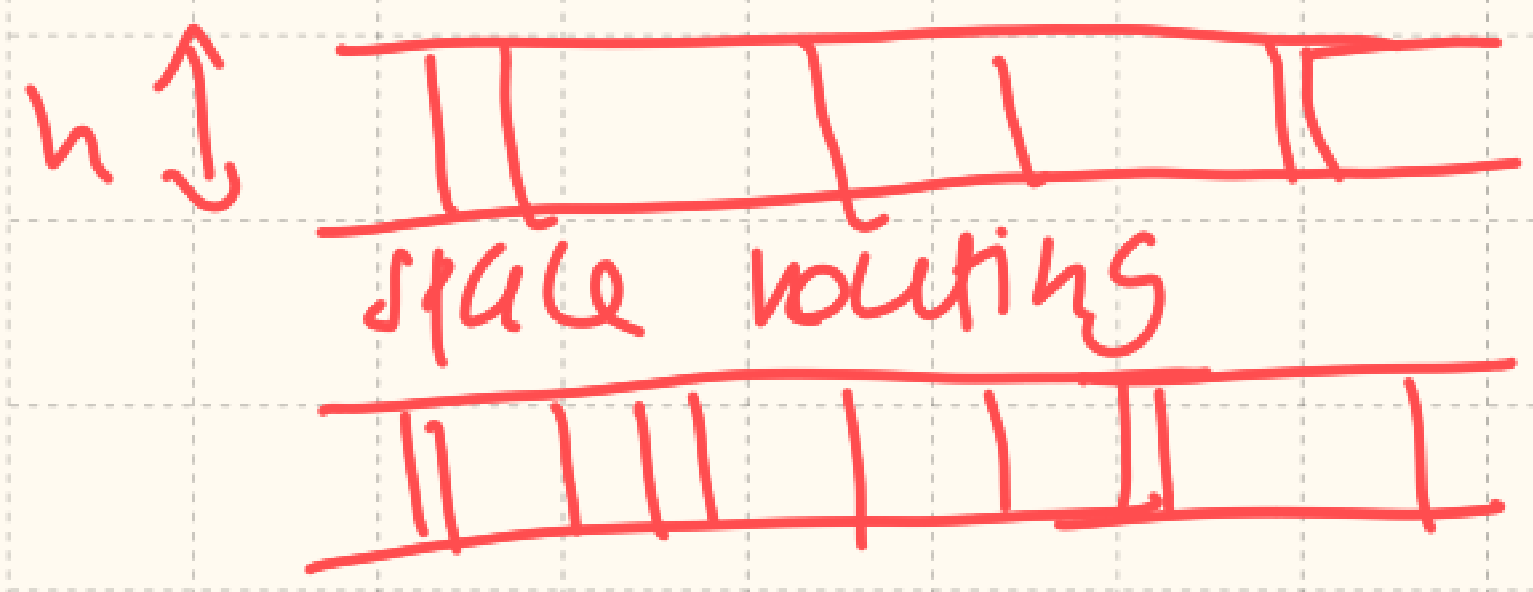
some part of the chip they will use standard cell library and parts which are very critical in performance they use custom design.

Latest technology -> full custom
or
custom design
or
Bulk (10's of millions) ex snapdragon

*Semi custom:
uses only standard cell library -> Gates, FF, latches, buffer etc

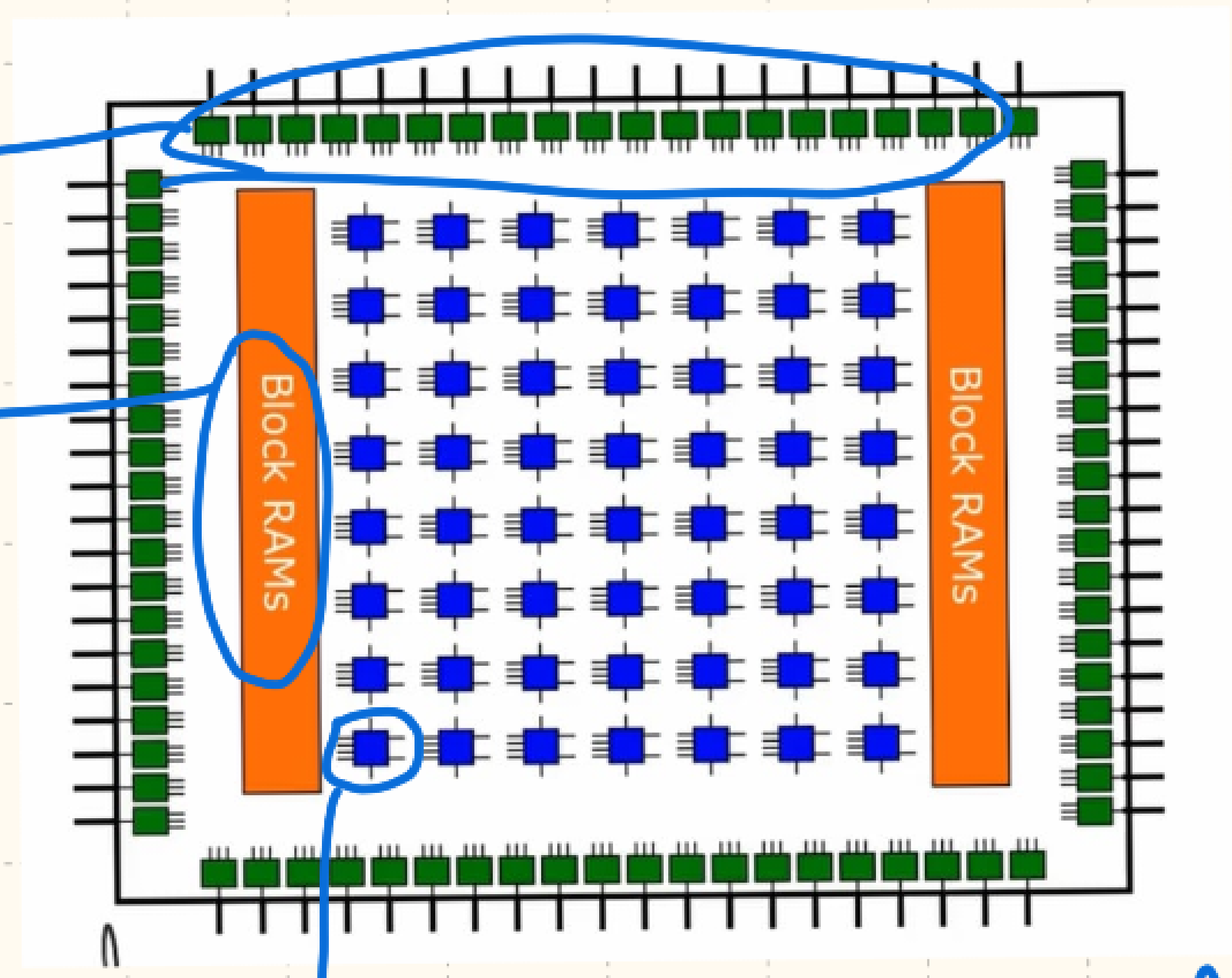
Standard cell have fixed height but varying width
Most common. used by 80% companies

(can directly synthesize the software



Cell library -> library database -> functional info
-> layout info
-> schematic info
-> timing -> performance, power, behavior when noise is present

* FPGA (field programmable gate arrays)
(chip already in hand)
Pre fabricated => much faster



→ configurable logic block (xilinx) (CLB)

CLB → look up tables
muxes → routing and cascading
SRAM → storing
D FFs

FPGA are good for Prototyping or few chips are to be made

* FPGA are also good for parallel processing

* Gate array
small level functions
ex toy car

Time to market
faster
→ FPGA
→ semi
- full
slow

implementation efficiency

1 2 3

Full Custom

Semi Custom (applic. specific IC) ASIC

FPGA (faster to market)

1 2 3

faster speeds

Full custom	Semi custom
* research and development of tech ex 3nm and 4nm	* slower in adapting technologies. ex 14nm or 24 nm
* Exhaustive cells 3 input AND Gates can be designed and implemented	* limited cells Insted of 3 input AND we have to use 4 input or 2 and 2 cascading
* Fully optimized	* Not fully optimized
* slower time to market	* Faster time to market
* only affordable for few	* Affordable for many
* $100 \times 10^6 <$ chips should be produced to breakeven the cost of R and D	$\approx 10 \times 10^6$ chips are produced

ASIC	FPGA
* Slower time to market * due to the long processes	Faster time to market
* Expensive for smaller quantities * even for producing 1000 nos its expensive	Cheaper for small quantities
* More optimized	Almost optimized (lacks in terms of performance)
* low power	Not so much low power
* Many Resources Required	* Only basic expertise Required

ASIC Design flow:

1) Design specification →Guiding document based on customer. Full details of the product

2) Architecturing →Dividing the problem to smaller blocks and distribute it to your team. →Also know what modules are already available. (IP's, existing designs etc)

3) RTL coding →Design something fresh Verilog, VHDL, C, C++, Python, MATLAB →Register Transfer level (RTL)

4) Verification →verify the hardware, checks the intended specification. (Freshers)

5) Synthesys → Code to Hardware blocks. SCL (standard cell library)

6) DFT Design For testability.

2^3 inputs needed to fully test

. 2^{1000} inputs needed, but not possible : Insert additional hardware to check chip efficiently and use less number of cycles.

7) Timing Analysis →Fix the clock frequency. Check the expectations are met. → commonly merged with DFT

8) Floor planning →extract layout. approx planning of area of chip.

9) Placement

10) Routing interconnects between blocks. Design Rule checks (DRC) or congestion (Many wire overlap) or hotspot (Many components causes heating in particular area)

one team

11) Formal Verification →Do every time a design changes. →Does a mathematical check.

12) Power estimation →estimate the total power consumption →switching causes power consumption

13) Fabrication →Chemical processes

14) Packaging → I/O pins final products

Design specification:

* Customer Requirements → No. of inputs, chip size, speed [Bus speed etc], battery, Power consumption, Area, Logic [function of the chip] outputs, cost, Application [Military, consumer etc]

Architecturing:

Planning for different processes of the chip ex →* 2 Processors - Master slave

* DSP → Video codec or Image or Audio
* Memory
* Private peripherals → (Can only be accessed by one processor usually slave)
* Clocks
* RF
* Memory
* public peripherals (can be accessed by both processors)
* ADC/DAC
* Bus / Glue
* logic

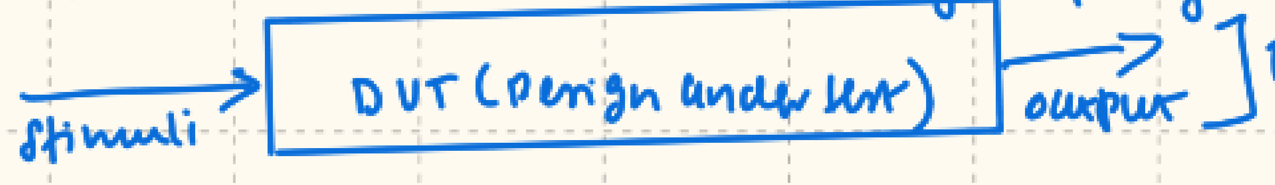
The whole process is divided into sections and assigned into teams

RTL Coding:

Verilog is used because of its simplicity. Other codes include VHDL, C etc

Verification:

* Done using test benches. Test bench is the basic form of verification.



Test bench : Whole another set of code. Check if the DUT is functioning as intended. [system verilog]

* Simulation : usually done only when you have to check for bugs debugging & if the design is very small

Synthesis

* code gets translated to hardware.

* RTL code
  constraints (Area, power, speed etc)
  standard cell library (SCL)
  $\rightarrow$ synthesis tool $\rightarrow$ Netlist (connection of library cells)

based on the constraints given the synthesis tool will select the appropriate cell to build your circuit

* Most used synthesis tool is DC (synopsys) (Design compiler)

Design for testability

* check for process defects → functional defect
  ↳ timing defect
* test plan after manufacturing

* during design phase we have to think how we can insert more circuits so that the design becomes more testable.
  * controllability → control the inputs
  * observability → observe the output

design

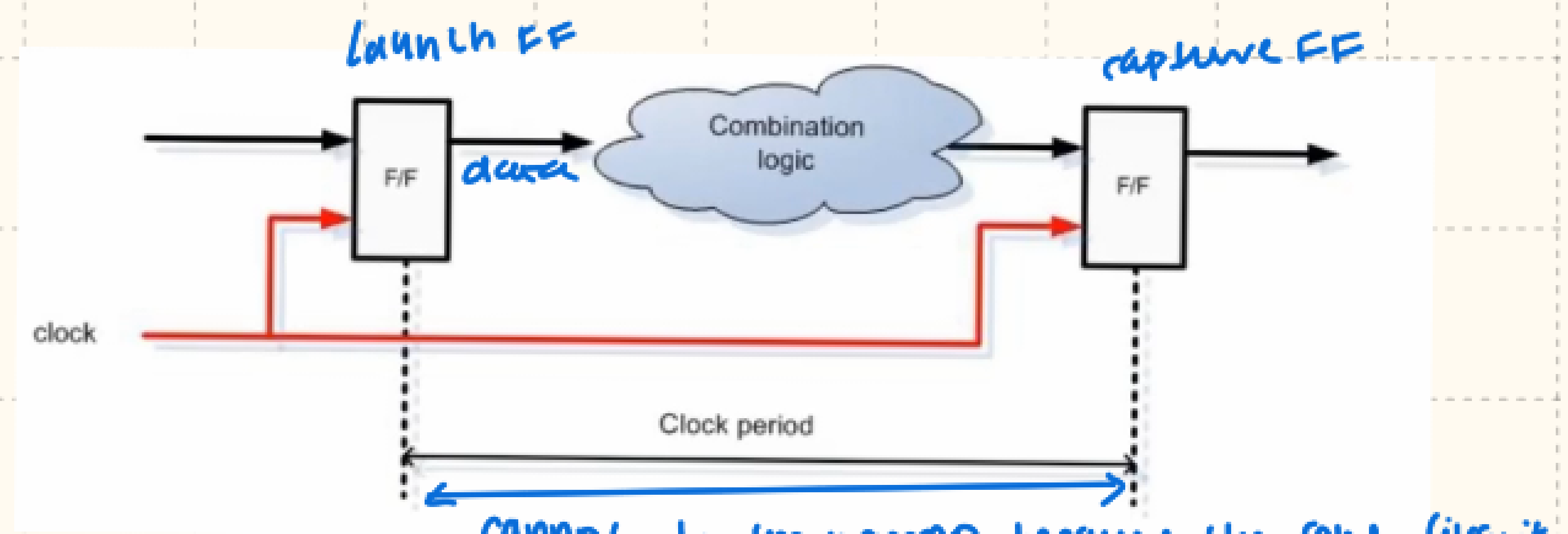
2 phases
* DFT insertion and Test generation
* Test application - Fab

software TESSENT

Test cost > 60% of total cost

Timing Analysis:

* fix the clock frequency
* clock synchronizes the data from FF to FF



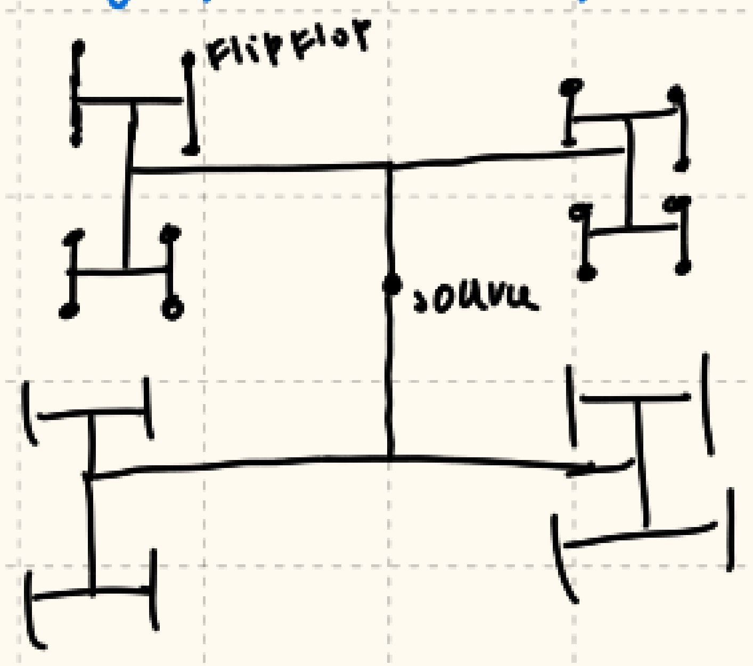
cannot be too narrow because the comb. circuit needs to settle down and also satisfy your setup time.

several data is launched and the path with the maximum delay is called the critical path.

critical path decides your clock period.

clock tree → How the clock is taken from source to different flip flops at the same time.

commonly used clock tree is H tree



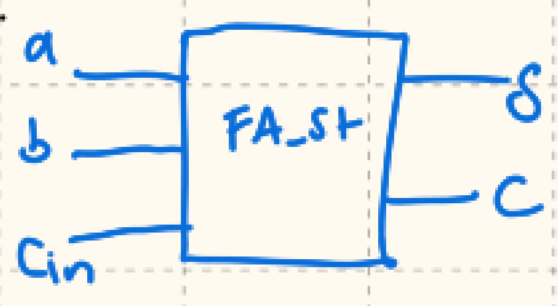
Formal Verification:

* performed throughout the design cycle.

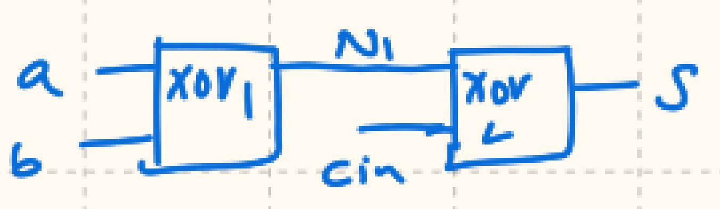
* If any component is added or removed from the design at any stage, you verify the correctness using different methods.

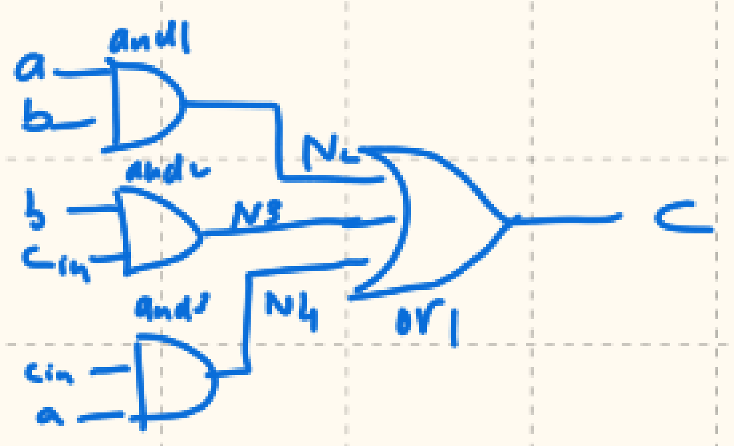
* different methods
  * Equivalence checking
  * property checking
  * boolean algebra
  * computational tree logic
  * Assertions

code for full adder (structural-1)



$S = a \oplus b \oplus C_{in}$ $C = a \cdot b + b \cdot c_{in} + C_{in} a$





```
module FA_st (S, C, a, b, Cin);
input a, b, Cin;
output S, C;
wire N1, N2, N3, N4;
Xor Xor1 (N1, a, b);
Xor Xor2 (S, N1, Cin);
and and1 (N2, a, b);
and and2 (N3, b, Cin);
and and3 (N4, Cin, a);
Or or1 (C, N2, N3, N4);
endmodule
```

A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Power estimation: check how good the routing is

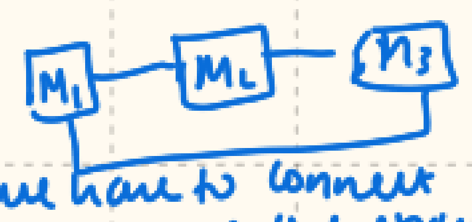
* Power dissipation mainly occurs due to switching
* The areas where more power is dissipated is called hotspots.
* ~65% interconnects
  ~21% clocks (higher frequency = more power consumed.
  ~rest to I/O and logic

Verilog basics

circuit should be known ↳ Data flow → used when Boolean eq. or combinational logic

latches, FF ↳ Behavioral → circuit is not known all seq. elements are designed

↳ Structural →

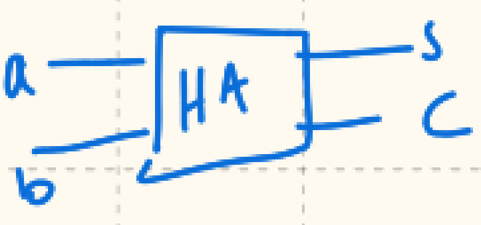


when we have to connect different modules we use structural design style

The internals will be either Dataflow or behavioral.

The integration of blocks will be structural

Data Flow [Half adder]



$s = a \oplus b$

$c = a \cdot b$

```
module HA (s, c, a, b);
input  a, b ;
output s, c ;
assign s = a ^ b ;
assign c = a & b;

endmodule
```

will remain same for all design styles

xor symbol

and symbol

Behavioral design (Half adder)