Digital Design Using Open-Source tools

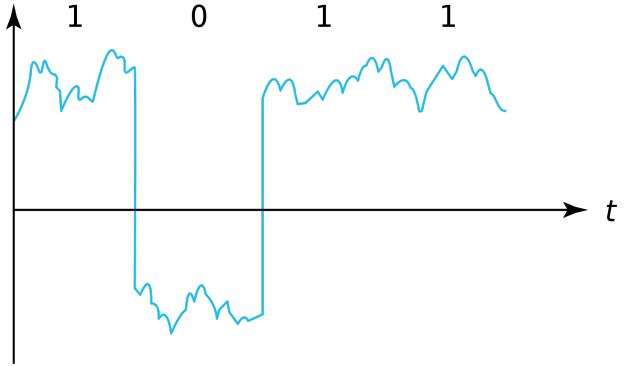
Workshop-CANELOS24

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Sistemas Digitales

Digital Signal: A signal that represents information as a sequence of discrete values.

In a **binary** system, values are either 0 or 1.



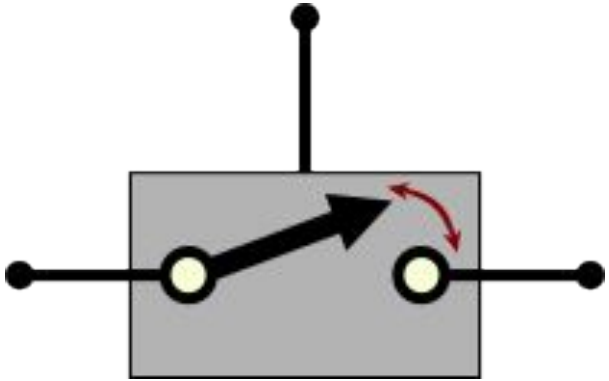
Why Binary?

It comes from the hardware!

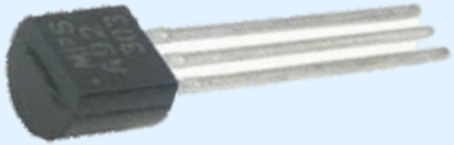
- Switching devices
- Ease of distinguishing 2 states

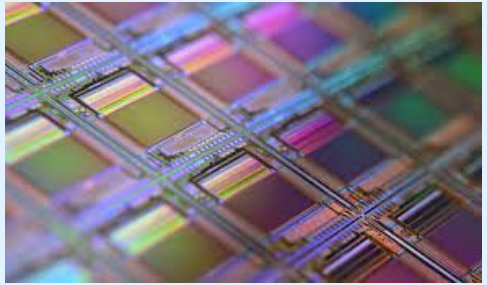
Switching Devices

(switches)



75 years of the MOS transistor

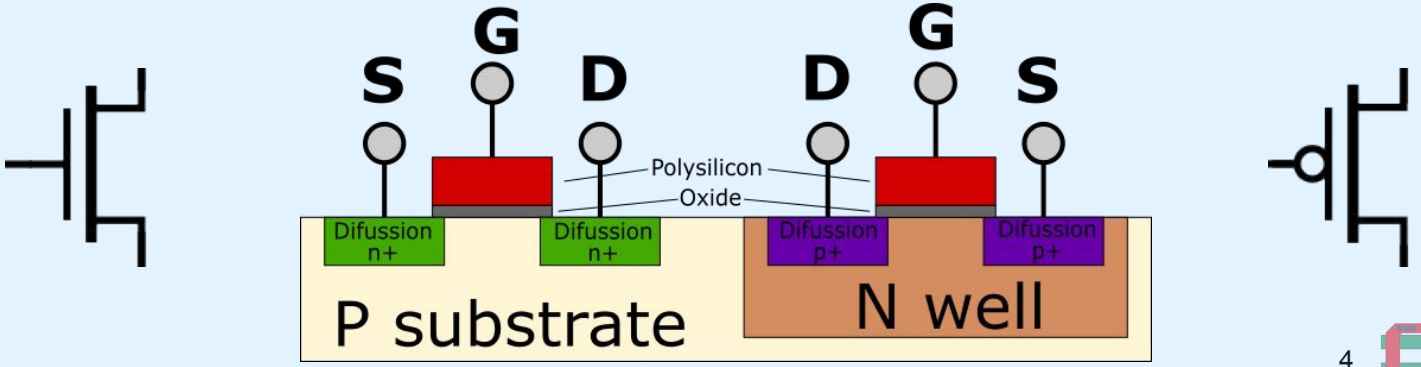




The ultimate switch

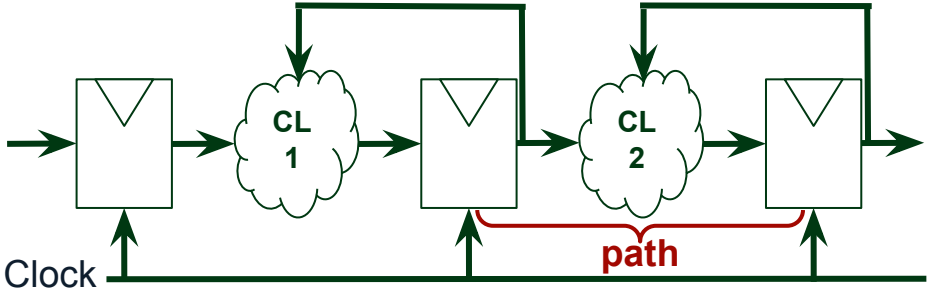
- activated by voltage
- simple geometry
- scalable for mass production

➡ predominant in today's technology

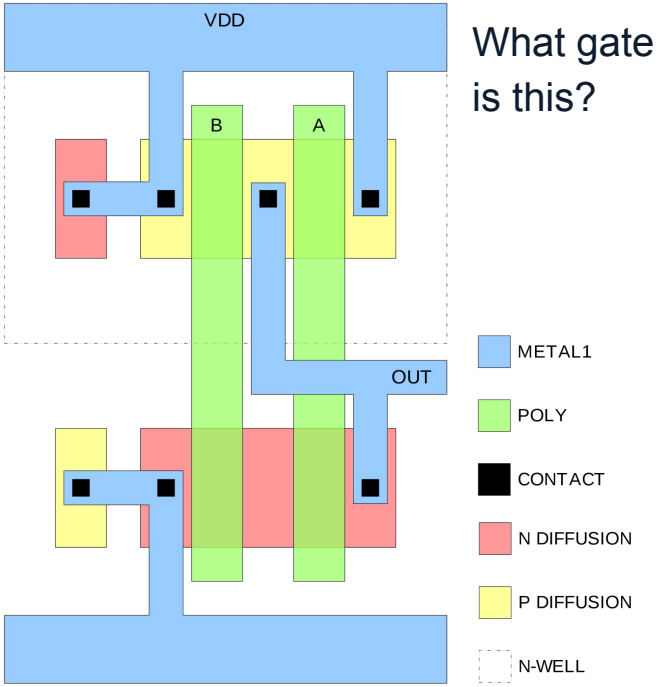


Standard Cells

Given the general structure of synchronous sequential systems, the design can be automated if the delays of the paths are known.

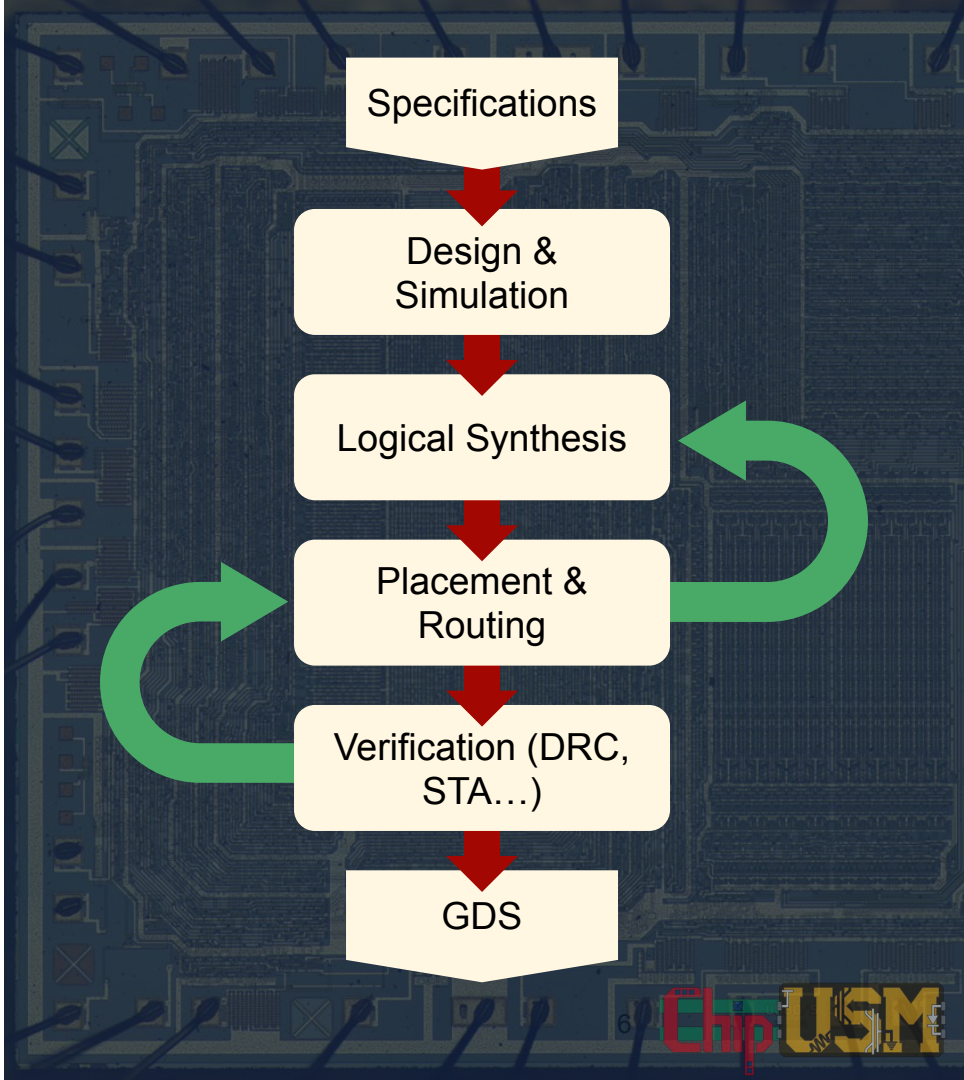


Robust standard cells are designed (analog design), characterizing their delays.



What gate is this?

Design flow for digital integrated circuits

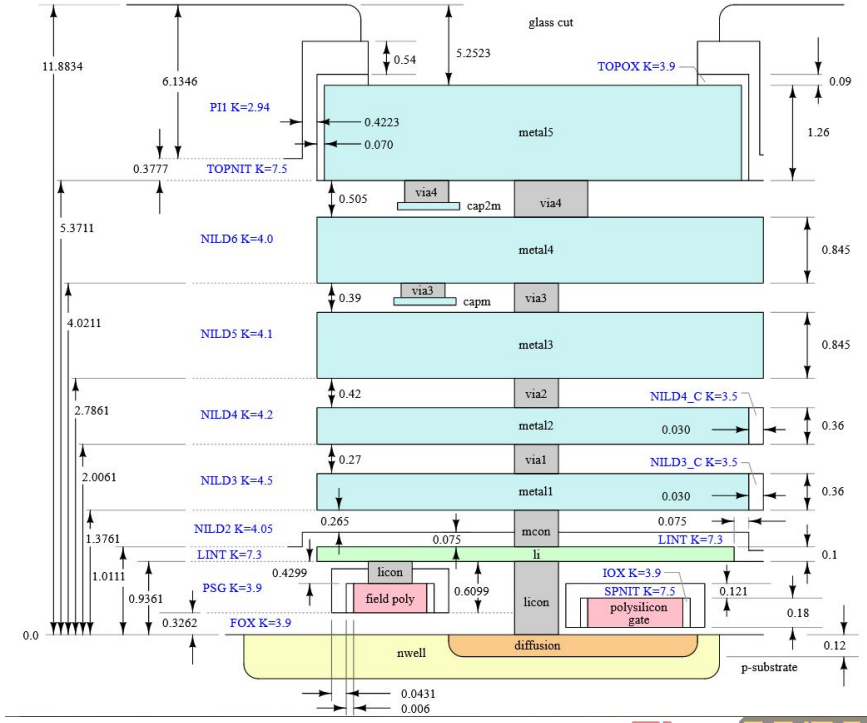


Manufacturing process and PDK (Process Design Kit)

Process Design Kit:

A set of specifications detailing what a manufacturer is capable of producing:

- available material layers
- physical and electrical properties
- \+ a set of rules that ensure manufacturability (Design Rule Check or DRC)

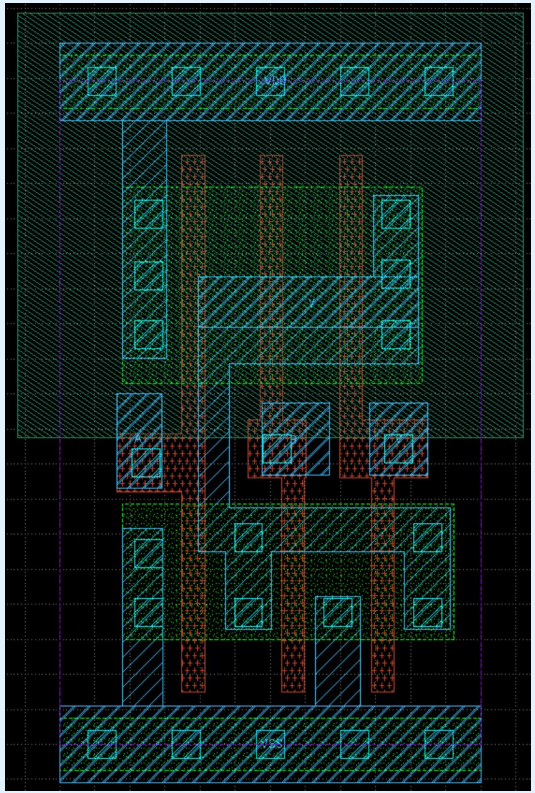


IHP SG13G2

Information in https://github.com/IHP-GmbH/IHP-Open-PDK/tree/main

SG13S is a high performance BiCMOS technology with a 0.13 µm CMOS process. It contains bipolar devices based on SiGe:C npn-HBT's with up to 250 GHz transit frequency (f_T) and 300 GHz maximum oscillation frequency. This process provides 2 gate oxides: A thin gate oxide for the 1.2 V digital logic and a thick oxide for a 3.3 V supply voltage. For both modules NMOS, PMOS and isolated NMOS transistors are offered. Further passive components like poly silicon resistors and MIM capacitors are available. The backend option offers 5 thin Al metal layers, two thick Al metal layers (2 and 3 µm thick)and a MIM layer.

SG13G2 has the same device portfolio as SG13S but much higher bipolar performance with f_T = 300 GHz and 500 GHz maximum oscillation frequency.



More information in

https://github.com/IHP-GmbH/IHP-Open-PDK/blob/main/ihp-sg13g2/libs.doc/doc/SG13G2_os_process_spec.pdf

Open-Source Simulation

Verilog v/s SystemVerilog

En Verilog…

- en lugar de ***logic*** se utilizan dos tipos de datos:
  - ***wire***: para representar cables, conectando módulos o con asignaciones concurrentes.
  - ***reg***: para representar datos guardados, pudiendo inferir FFs, Latches o lógica combinacional.
- no existe la declaración de señales con ***enum***;
- se usa el bloque procedural ***always*** en lugar de ***always_comb*** y ***always_ff***.

	SystemVerilog	Verilog
Combinacional	always_comb	always @(*)
Secuencial	always_ff @(posedge clock)	always @(posedge clock)

La lista de sensibilidad con '*' representa que reacciona a cambios en cualquier señal.

Ejemplo

```
2   //////// Instantiation Template //////////
3   /*
4       counterN #(.N(8)) instance_name (
5           .clk(),
6           .rst(),
7           .en(),
8           .counter()
9       );
10  */
11  ////////////////////////////////////////
12
13  module counterN #(parameter N=8)(
14      input  wire clk,
15      input  wire rst,
16      input  wire en,
17      output reg  [N-1:0] counter
18      );
19
20      reg [N-1:0] counter_next;
21
22      always @(posedge clk) begin
23          if(rst)
24              counter[N-1:0] <= 0;
25          else
26              counter[N-1:0] <= counter_next[N-1:0];
27      end
28
29      always @(*) begin
30          counter_next[N-1:0] = counter[N-1:0];
31          if(en)
32              counter_next[N-1:0] = counter[N-1:0] + 1;
33      end
34
35  endmodule
```

```
1
2
3   module counter_tb ();
4
5       reg clock, reset;
6       wire [3:0] out;
7
8       always #5 clock = ~clock;
9
10      initial begin
11          $dumpfile("signals.vcd");
12          $dumpvars(0, counter_tb);
13          clock = 0;
14          reset = 0;
15          #3
16          reset = 1;
17          #20
18          reset = 0;
19          #200
20          $finish;
21      end
22
23      counterN #(.N(4)) DUT (
24          .clk(clock),
25          .rst(reset),
26          .en(1'b1),
27          .counter(out[3:0])
28      );
29
30  endmodule
31
32
33
34
```

Synthesis using yosys

- First, navigate to the folder Dia_1 > src, then enter the command yosys.

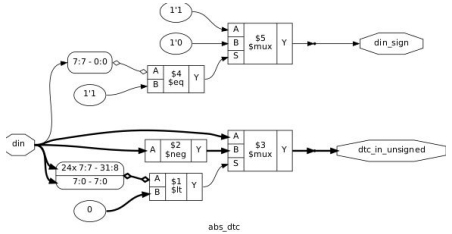
```
/Dia_1/src$ yosys
```

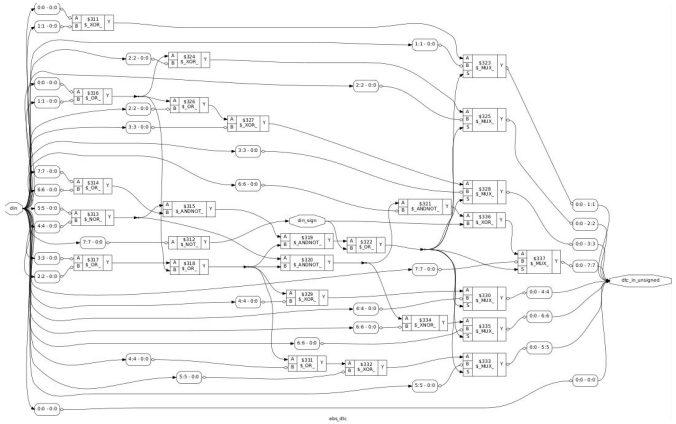
- Next, use the following command to load the Verilog file. At this step, the linter will let us know if there are any errors in our code.

```
yosys> read_verilog abs_dtc.v
yosys> show
```

- Finally, we can run the following command to run the synthesis process.

```
yosys> synth
yosys> show
```





Simulating using Icarus Verilog

Icarus Verilog allows for generating netlists and simulating Verilog code:

- Source code and testbench .v
- Add to the testbench:

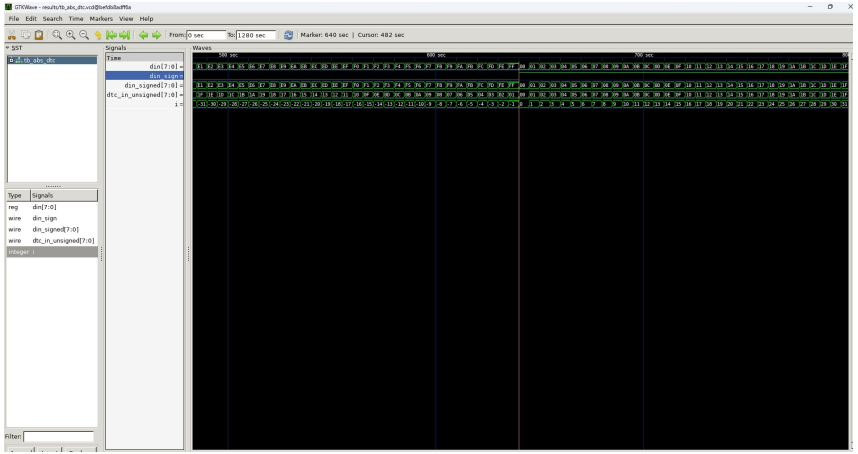
```
$dumpfile("signals.vcd")
$dumpvars(0, <testbench_name>)
```

- Simulate with:

```
Dia_1$ iverilog -o results/abs_dtc src/abs_dtc.v testbenches/tb_abs_dtc.v
Dia_1$ vvp results/abs_dtc > results/abs_dtc.log
```

- Load .vcd file in GTKWave

```
Dia_1$ gtkwave results/tb_abs_dtc.vcd
```



Practical Exercise

In this exercise, you will implement a simple **Digital Timer Circuit (DTC)** using Verilog. The module provided has the following key components:

Module Description:

- **Inputs:**
  - dtc_in [7:0]: An 8-bit input that acts as the timer threshold.
  - trig: A trigger signal to start the timer.
  - clk: A clock signal for synchronization.
  - rst: A reset signal to initialize or reset the timer.
- **Output:**
  - dtc_out: A signal that is high (1) when the timer is active, and low (0) when it is not.

Behavior:

- When the trig signal goes high, the timer starts counting from zero and the output dtc_out will toggle from high (0) to low (1).
- The timer will continue counting until it reaches the value in dtc_in, at which point the output dtc_out will toggle from high (1) to low (0).
- The module can be reset at any time using the rst signal, which resets both the counter and output.

Practical Exercise

With the design now complete, the next step is to simulate it and confirm that it functions as expected.

Simulation Instructions:

1. **Testbench**:
   - You will be provided with a pre-written testbench that applies various input values (e.g., `dtc_in`, `trig`, `clk`, `rst`) to your design.
   - The testbench will automatically generate the necessary stimuli for testing your `dtc` module.
2. **Run the Simulation**:
   - Use the provided testbench to simulate your design.
3. **Compare with Golden Results**:
   - After running the simulation, you need to compare your simulation results with the **golden results** available in the folder `Dia_1/results/golden_results`.
   - Ensure your output matches the golden reference to verify that your design works as expected.

Any Questions?