Introduction to electronic workflow

Stefan Cristi Zugravel

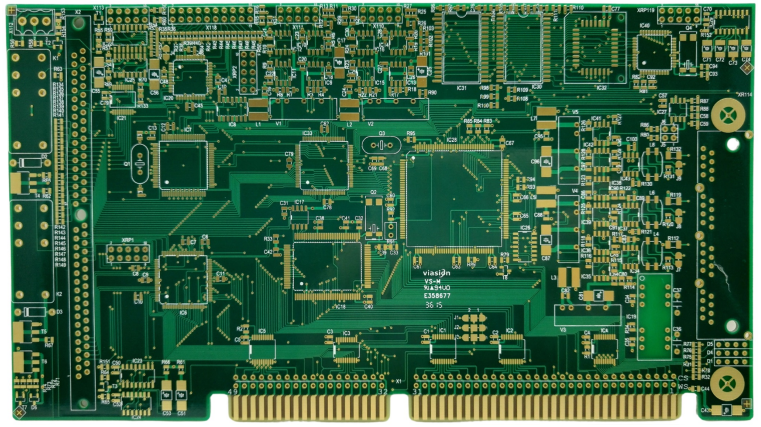
18/02/2026

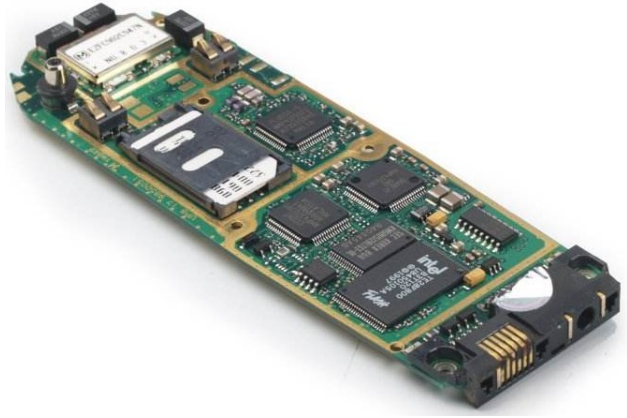
PART 0 – tool introduction

Course objectives

- Manage hardware projects using **Git version control**
- Design **PCB schematics and layouts** using **KiCad**
- Perform **basic SPICE simulations** to validate critical circuits
- Understand **FPGA architecture and HDL design methodology**
- Develop, simulate, and synthesize simple **VHDL modules**
- Use **XILINX VIVADO** and integrate IP cores

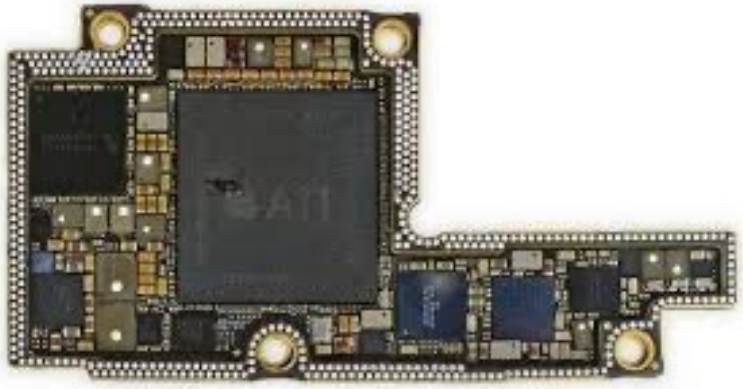
PCB, electronics

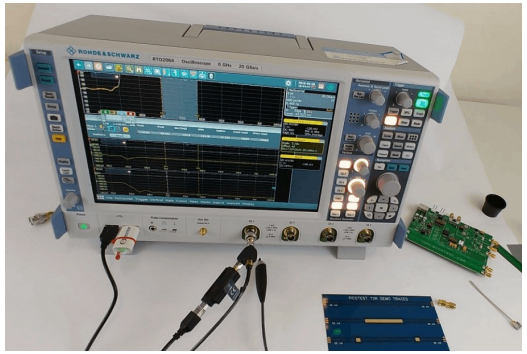


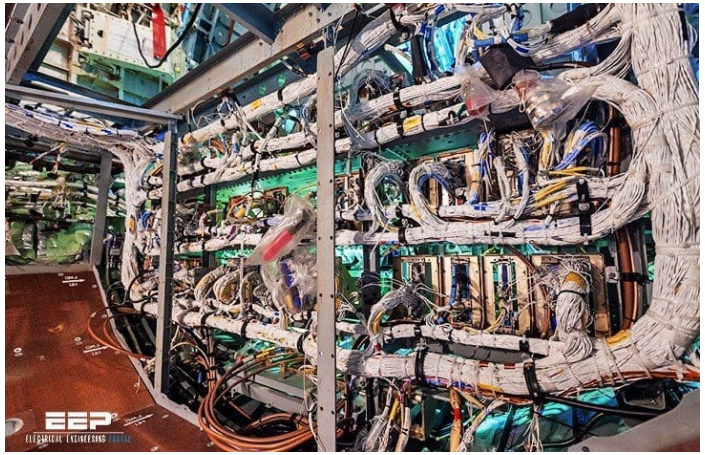


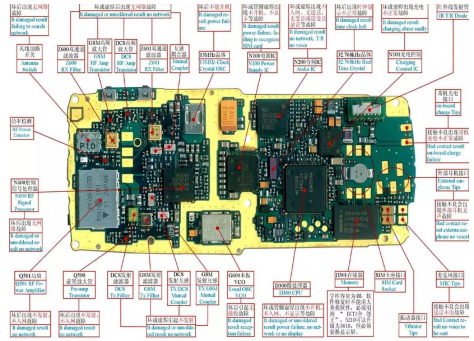
- Consumer Electronics
- Industrial & Automotive
- Telecommunications & Networking
- Medical Electronics
- Aerospace, Defense, & Maritime
- Lighting
- Instrumentation

Ecc ....





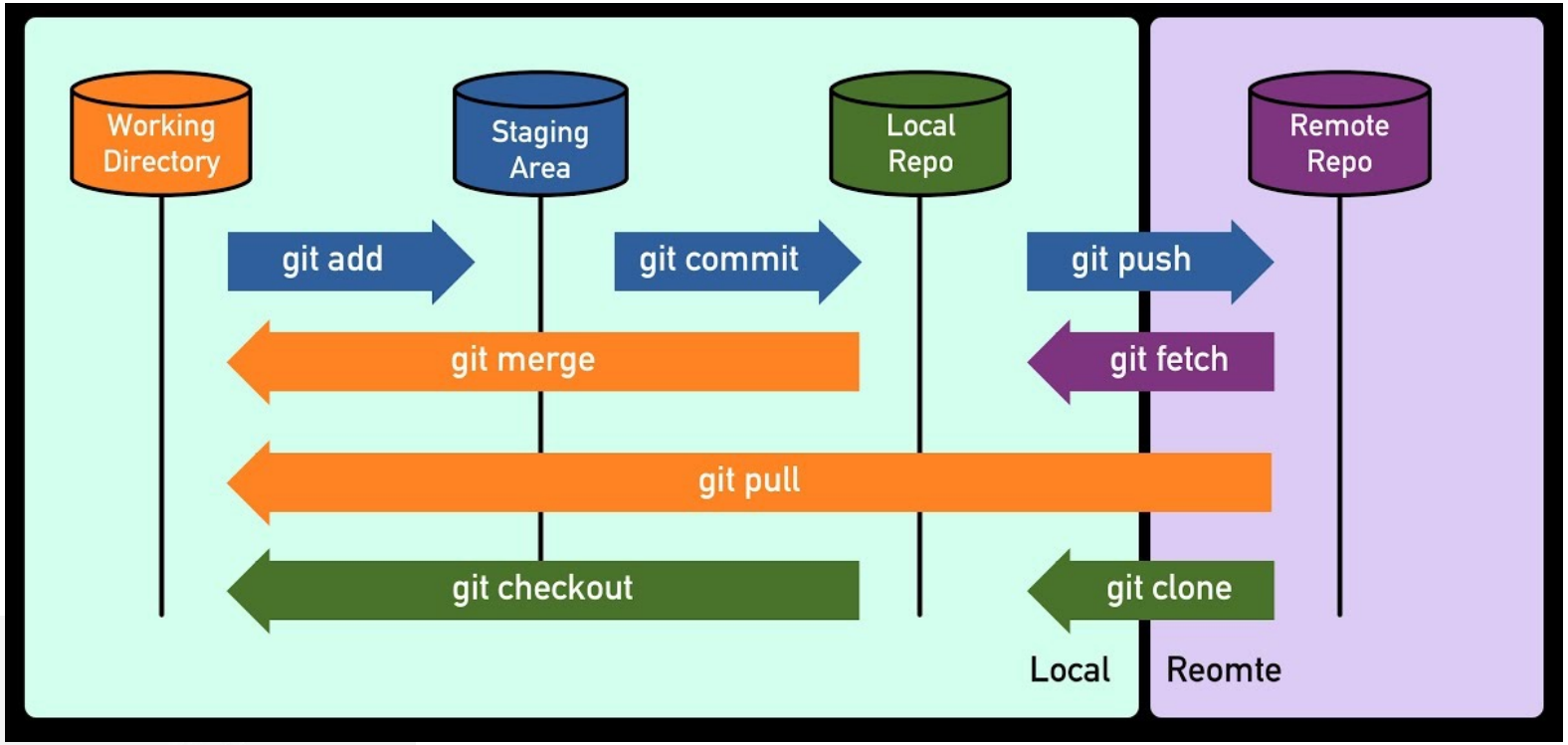


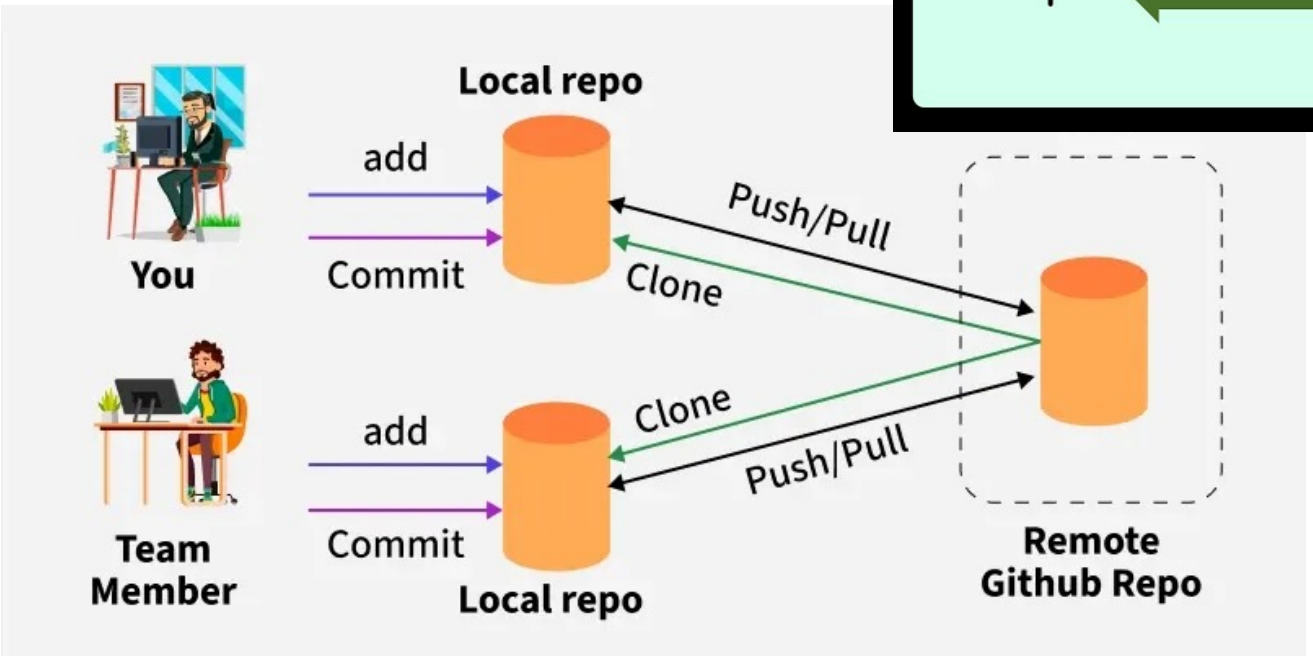


Tools Used

- Git (versioning)
- Draw.io (block diagrams)
- KiCad (Schematic & PCB)
- NGspice & LTspice (simulation)
- XILINX VIVADO (FPGA)

Git - 1


Working Directory
Staging Area
Local Repo
Remote Repo
git add
git commit
git push
git merge
git fetch
git pull
git checkout
git clone
Local
Reomte


Local repo
add
Commit
Push/Pull
Clone
You
Team Member
Clone
add
Push/Pull
Commit
Local repo
Remote Github Repo

Git - 2

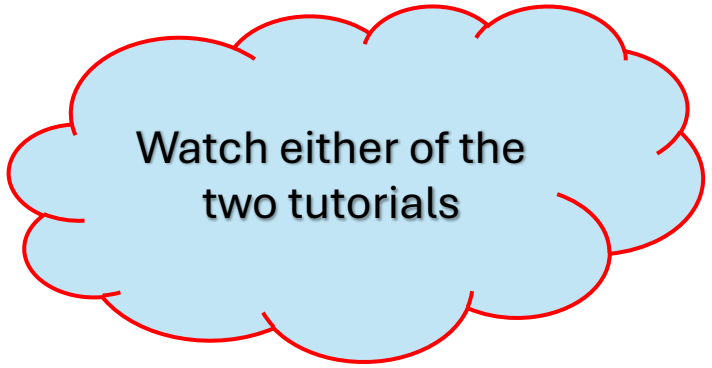
“A distributed system to track changes in source code, enabling multiple people to work together on non-linear development.”

Setup & Basics	Branch & Merge	Investigate & Fix
`git init` - Start repo	`git branch` - List branches	`git log` - View history
`git clone <url>` - Copy repo	`git checkout <branch>` - Switch	`git status` - Check changes
`git add .` - Stage all	`git merge <branch>` - Combine	`git diff` - See differences
`git commit -m "msg"` - Save	`git branch -d <name>` - Delete	`git restore <file>` - Undo changes
`git push` - Upload		`git stash` - Temporarily save
`git pull` - Download		

Lab 0

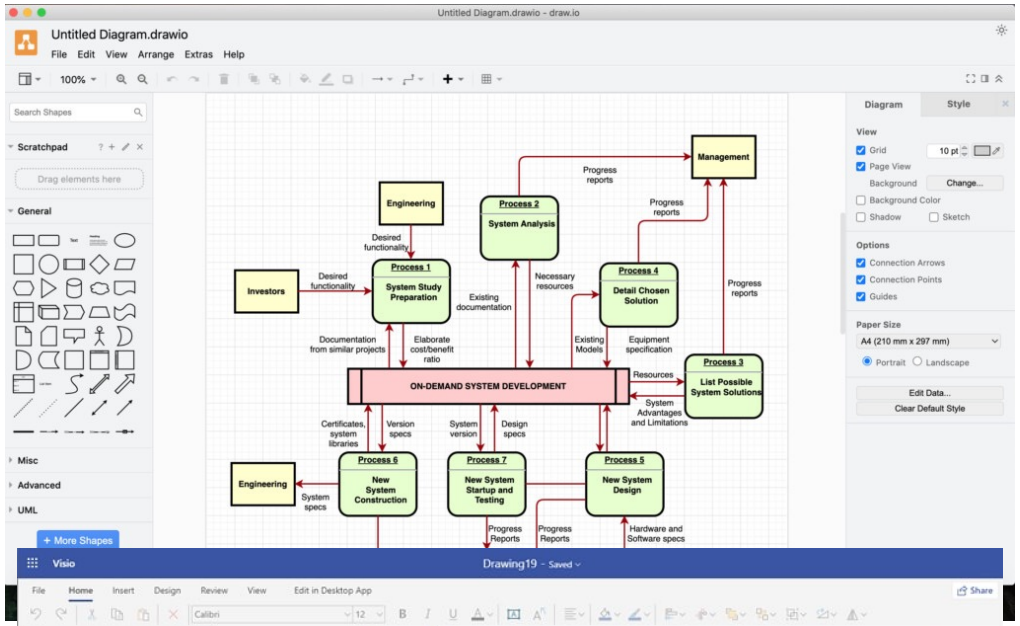
https://www.youtube.com/watch?v=8JJ101D3knE

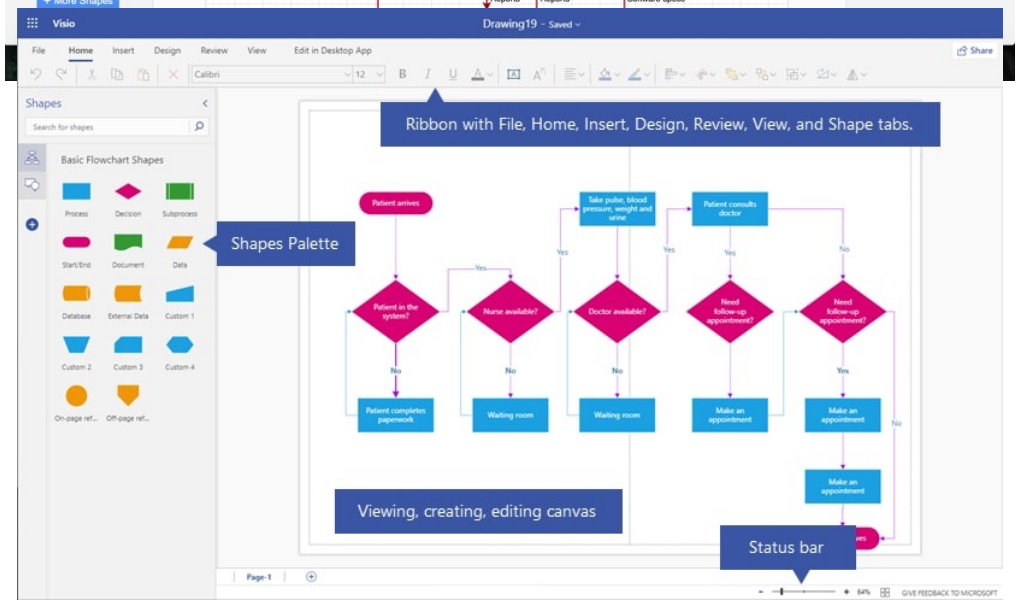
https://www.youtube.com/watch?v=tRZGeaHPoaw

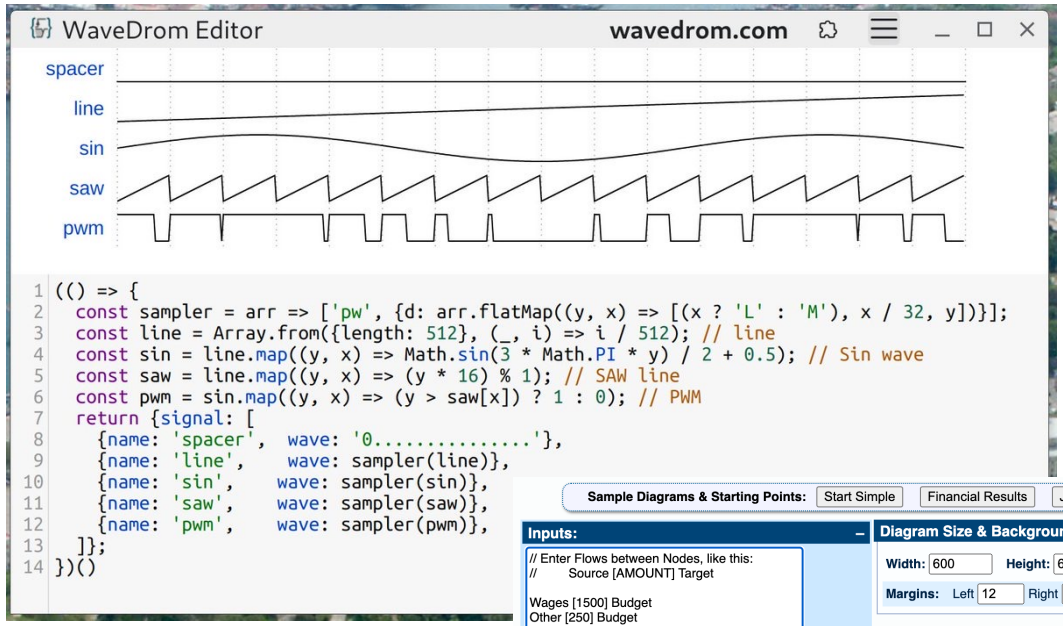


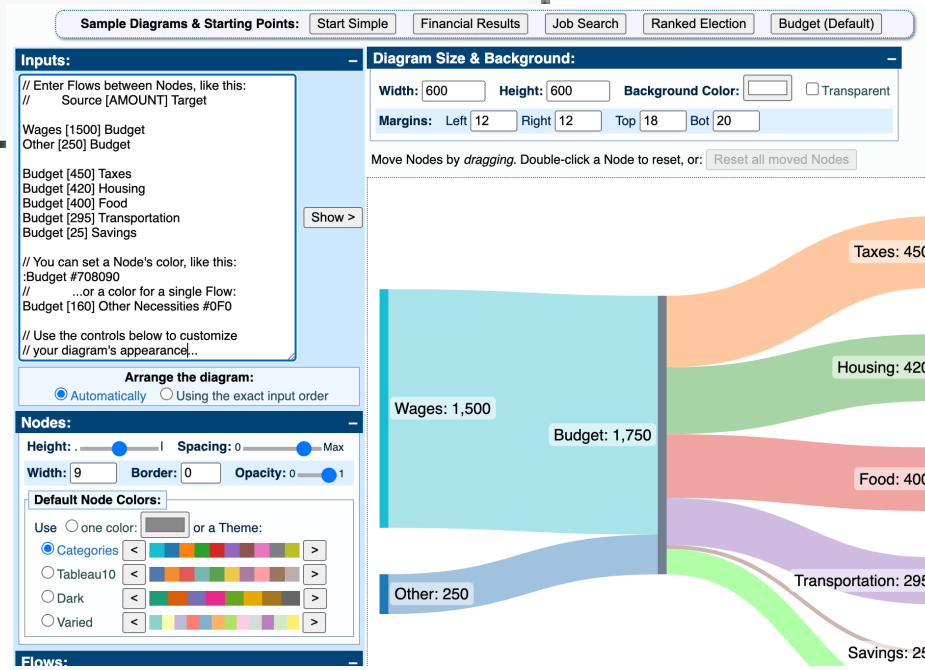
- Create a new Repo with a .gitignore and a license
- Push a commit, create a new branch and do a merge

Draw.io – wavedrom – sankeymatic – MS Visio







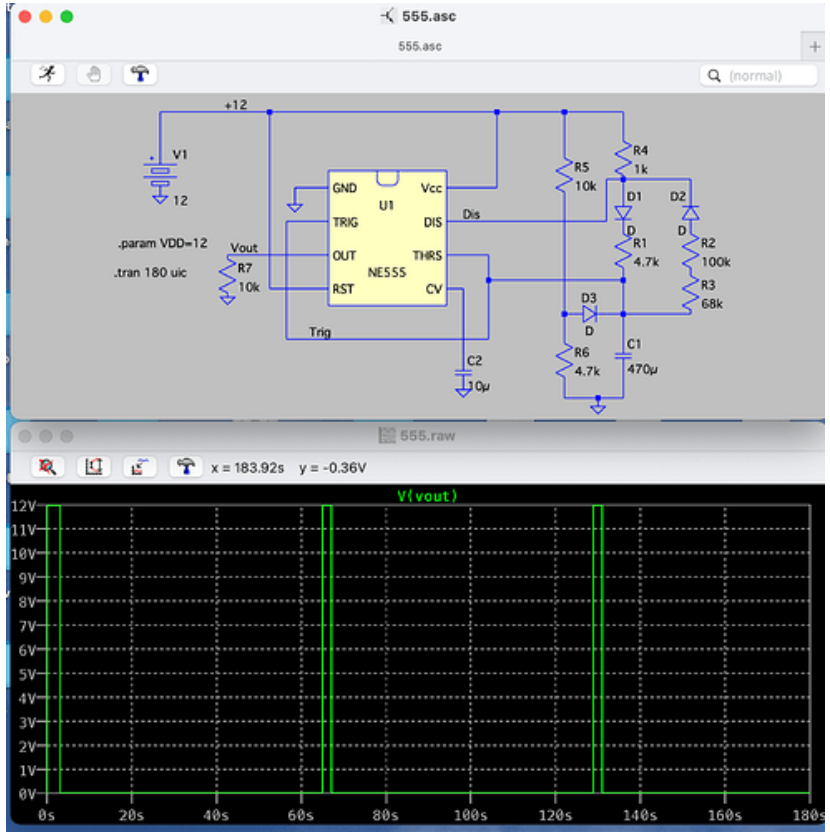


Major Electronic Design Automation (EDA) vendors

SPICE simulation

Simulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis

standard tool for simulating analog electronic circuits



SIMULATION

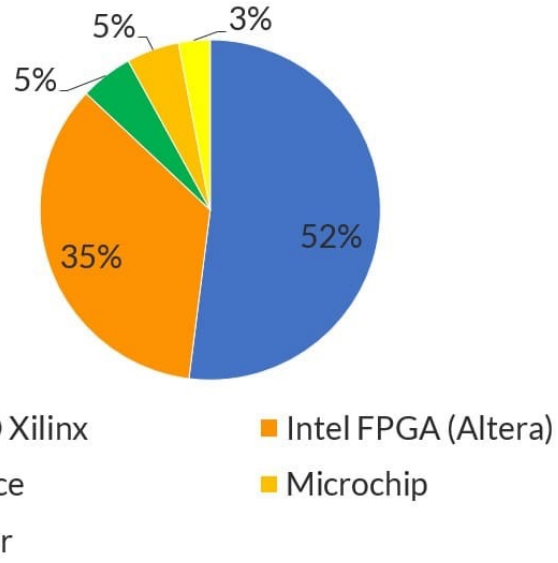
LTspice® LTSPice v/s PSpice® PSpice

Parameter	Linear Technology's SPICE	Personal SPICE
Cost & Licensing	Completely free (even for commercial use)	Paid (Full version); free version is limited
Simulation Speed (Switching Circuits)	Extremely fast, optimized for SMPS & analog switching	Slower in power circuits, can require convergence tweaks
User Interface & Learning Curve	Lightweight UI, scripting-based setup	Advanced GUI with easier access to features
Analog vs Digital Support	Strong analog focus, minimal digital logic	True mixed-signal with full digital gate library
Monte Carlo & Tolerance Analysis	Supported via scripting (`.step, gauss()`) – powerful but manual	Native built-in support with statistical tools
Integration with PCB Design Tools	No native PCB integration	Fully integrated with OrCAD/Allegro (Cadence Suite)
Model Library Support	Best with Analog Devices models; 3rd party models need tweaks	Wide vendor-neutral support (TI, ST, NXP, ON Semi, etc.)
Simulation Features (Advanced)	Supports behavioral modeling, parametric sweeps, some noise	Adds sensitivity, reliability, yield analysis, fault simulation
Best Use Case Fit	Analog design, SMPS, filters, discrete validation	Full system-level sim (MCU + analog + bus-level behavior + PCB)

FPGA design suite

VIVADO®
HLx Editions

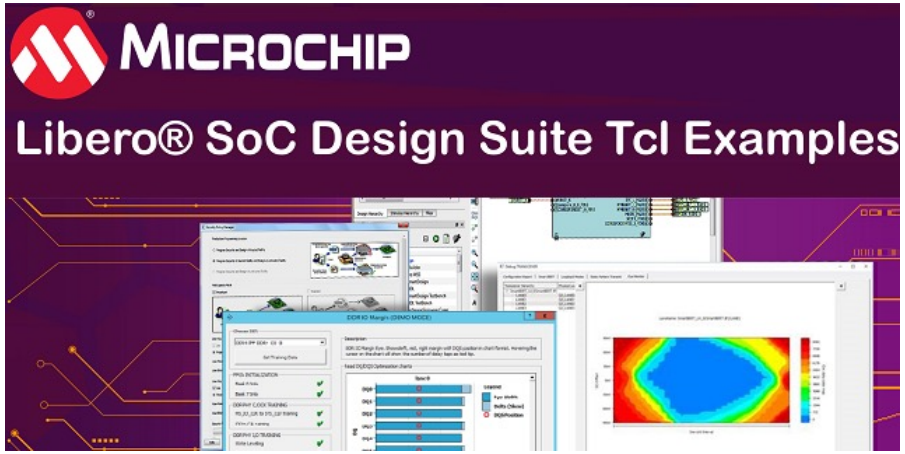
LATTICE RADIANT DESIGN SOFTWARE



Source: The Information Network

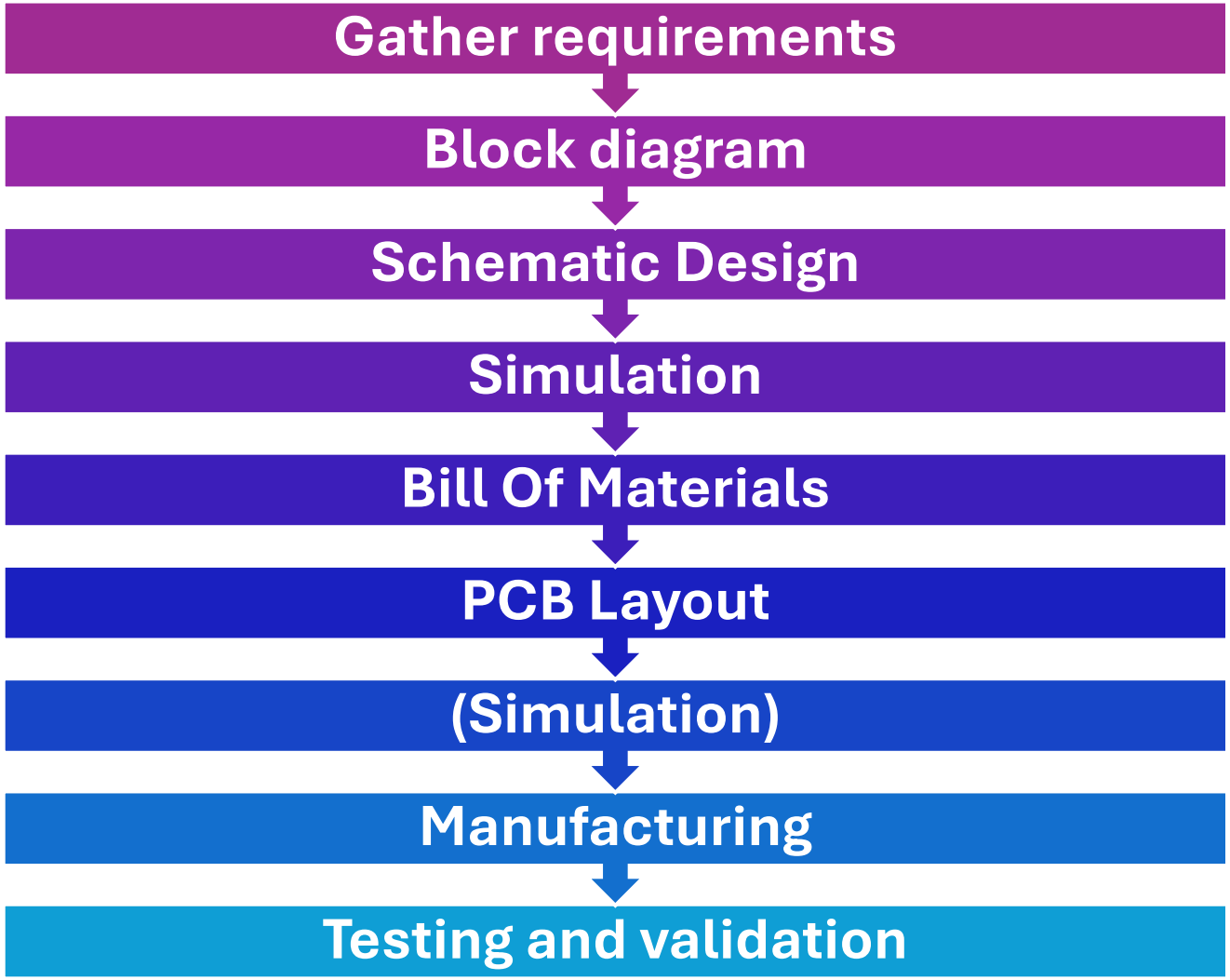
ALTERA®

QUARTUS® II

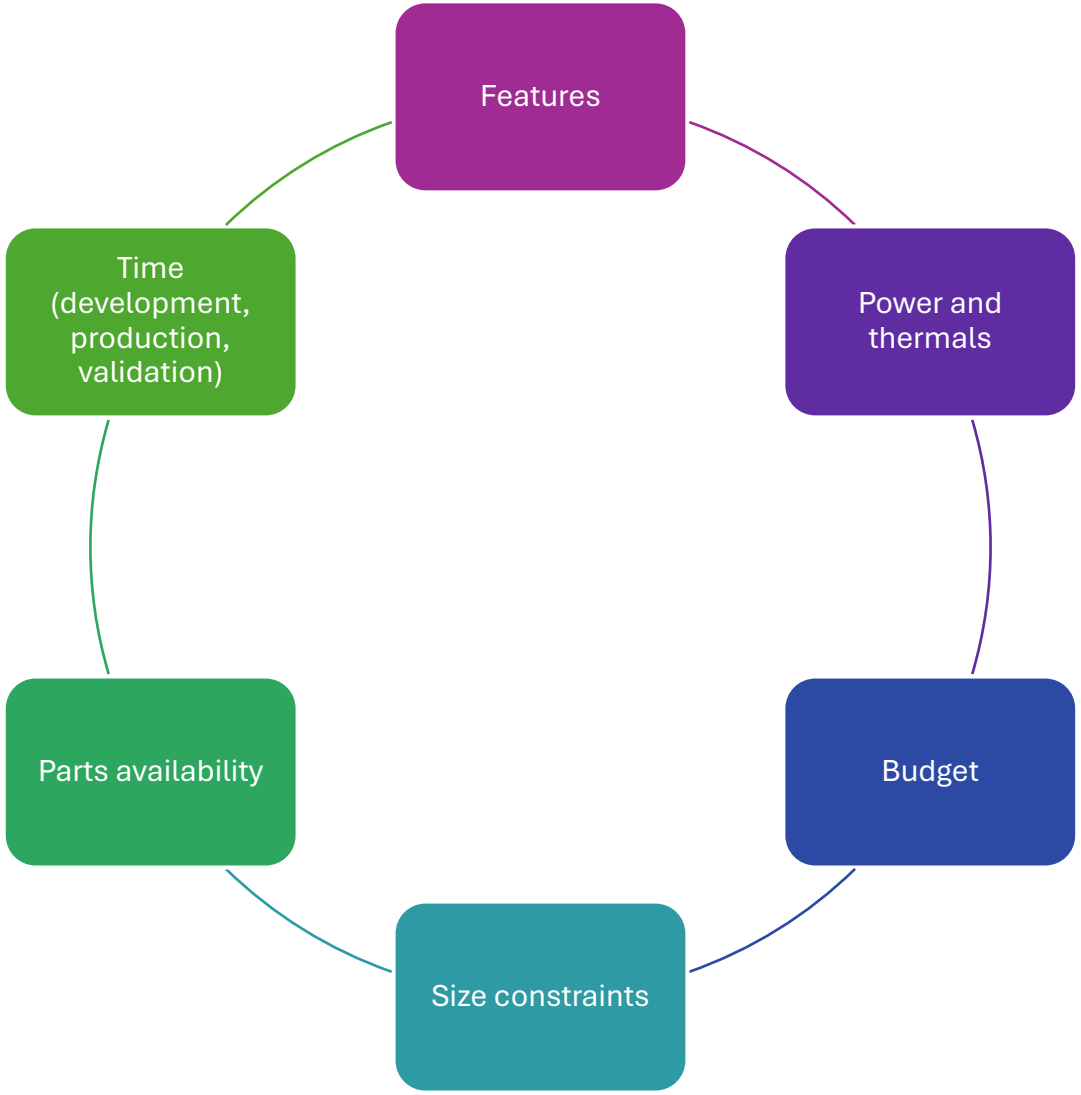


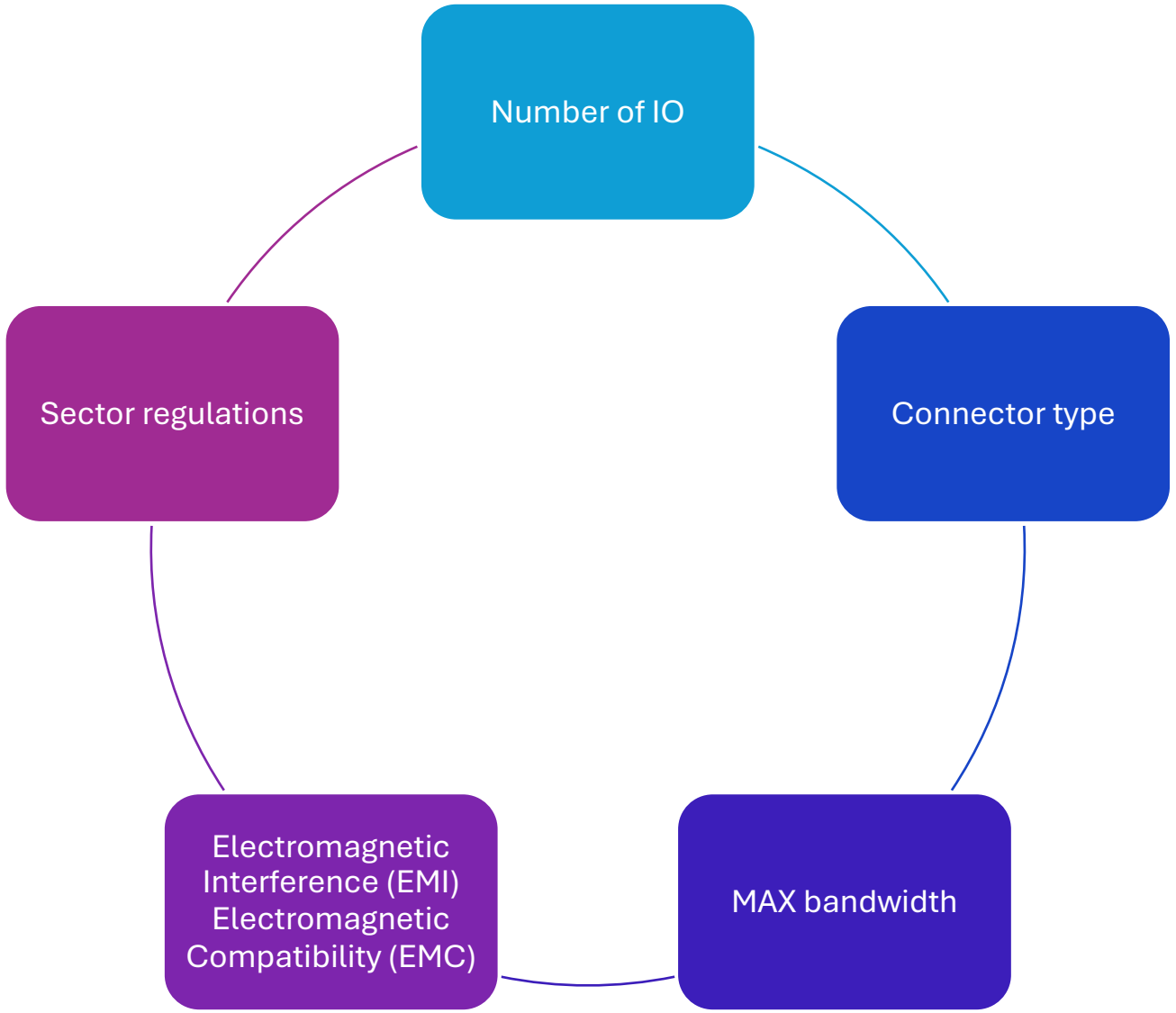
PART 1 – PCB design workflow

PCB design process



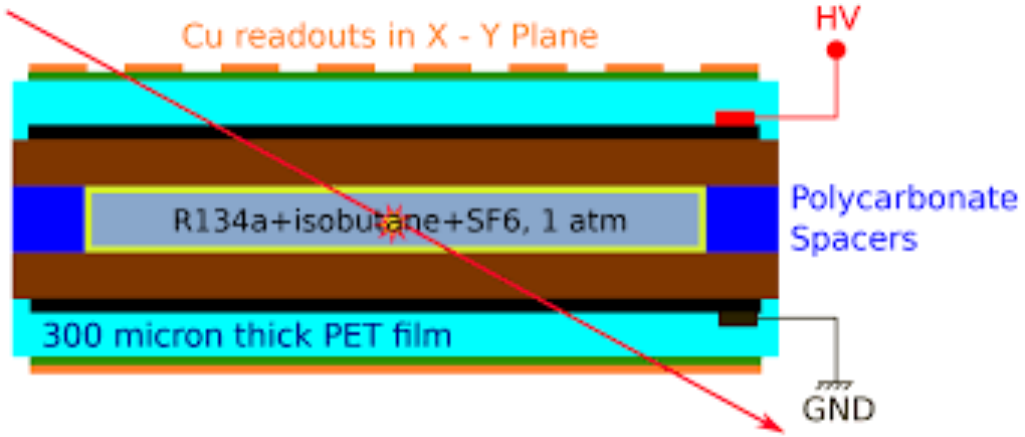
Gather requirements





Course goal (Gather requirements)

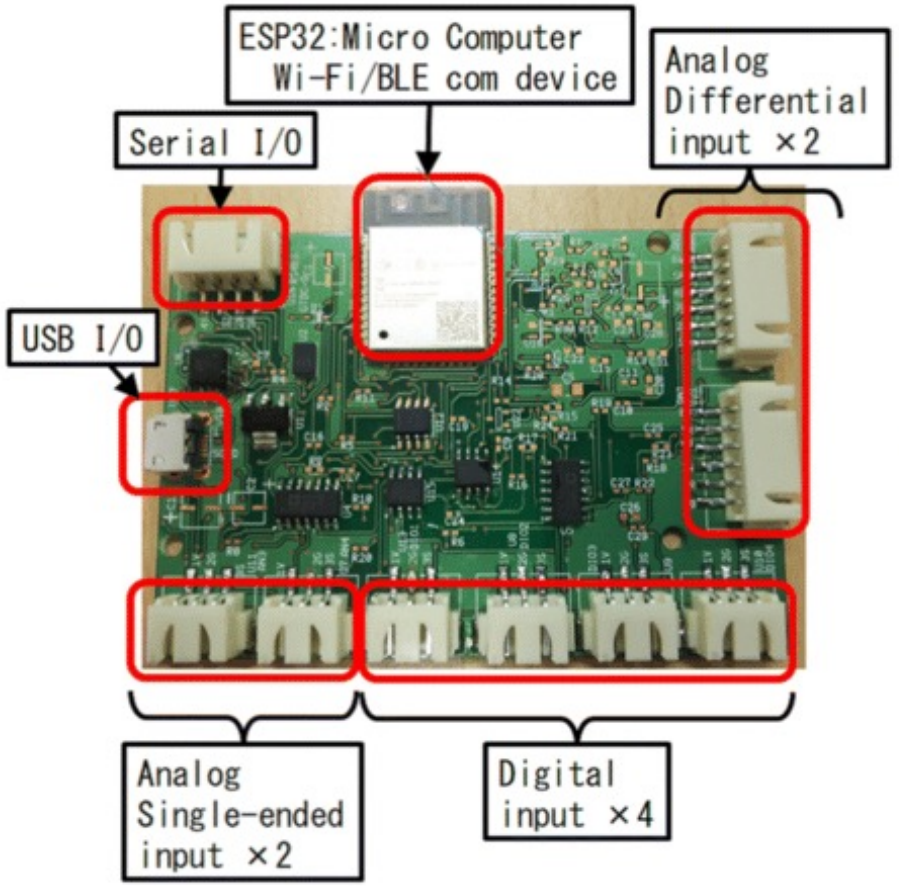
Resistive Plate Chamber (RPC) diagram



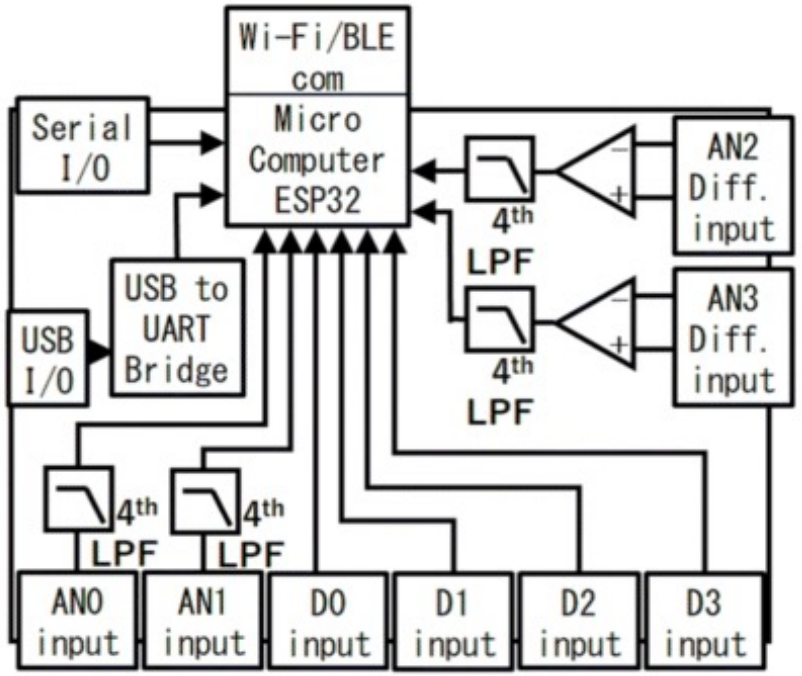
Develop a custom module used to detect currents from 5 uA to 700 uA.

- **8 inputs connected to a 16 bit ADC (8 to 860 SPS)**
- **Shunt resistor with Current-Sense Amplifier**
- **20 KHz -3 dB FEE bandwidth low pass filtering**
- **Microcontroller data taking with of the shelf module.**
- **Low noise**
- **FEE insulated from digital section**

Block diagram – 1

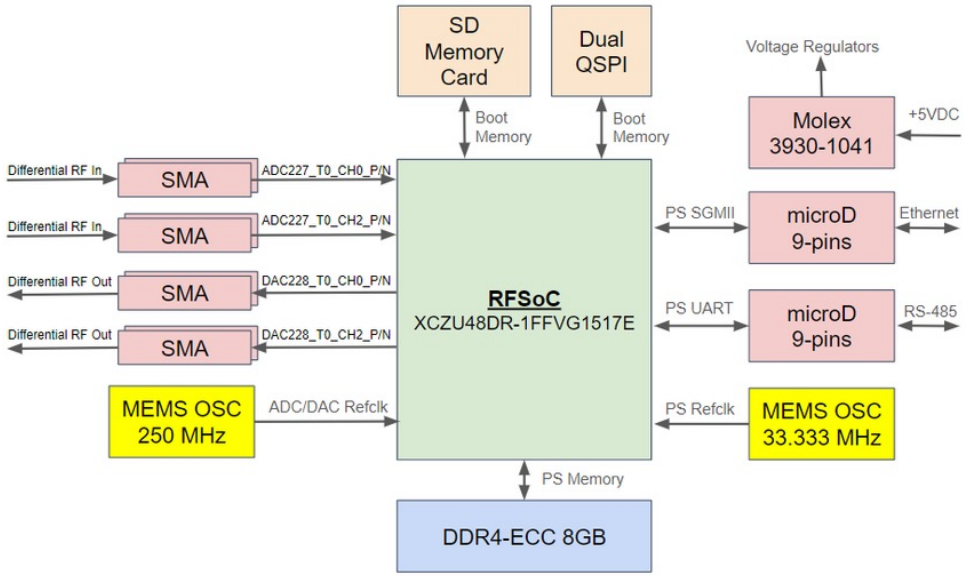


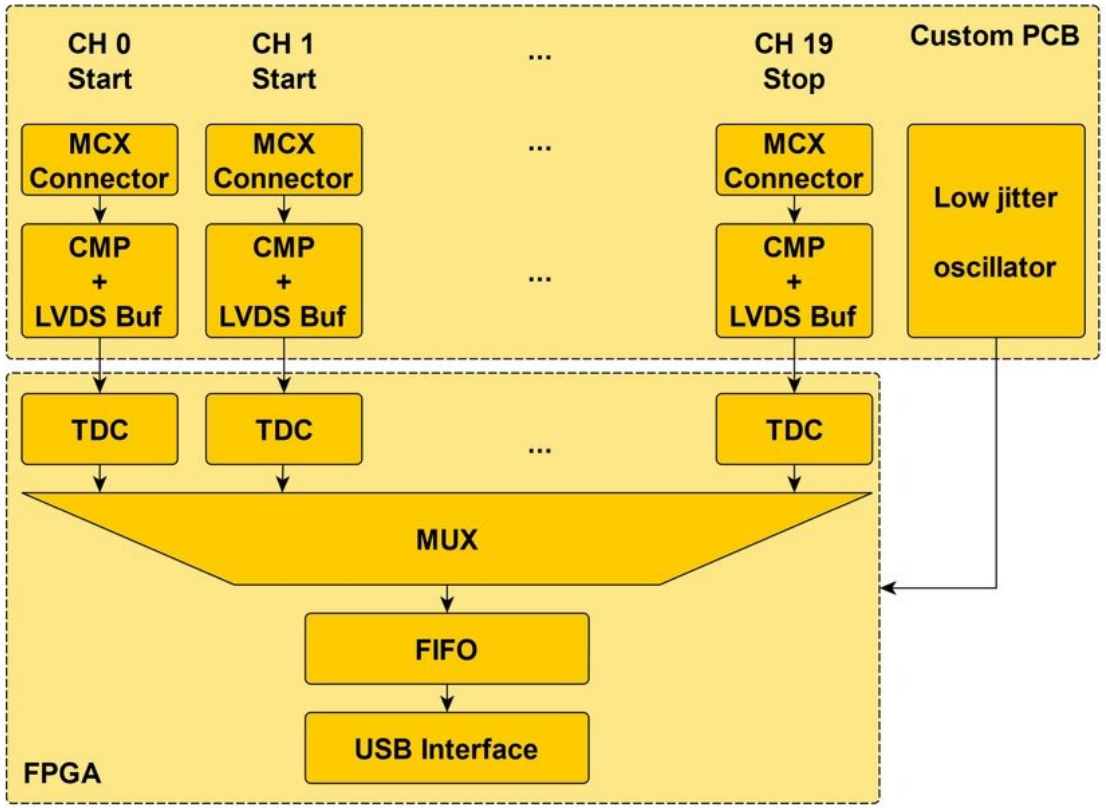
(a)PCB appearance



(b)Block diagram

Block diagram – 2

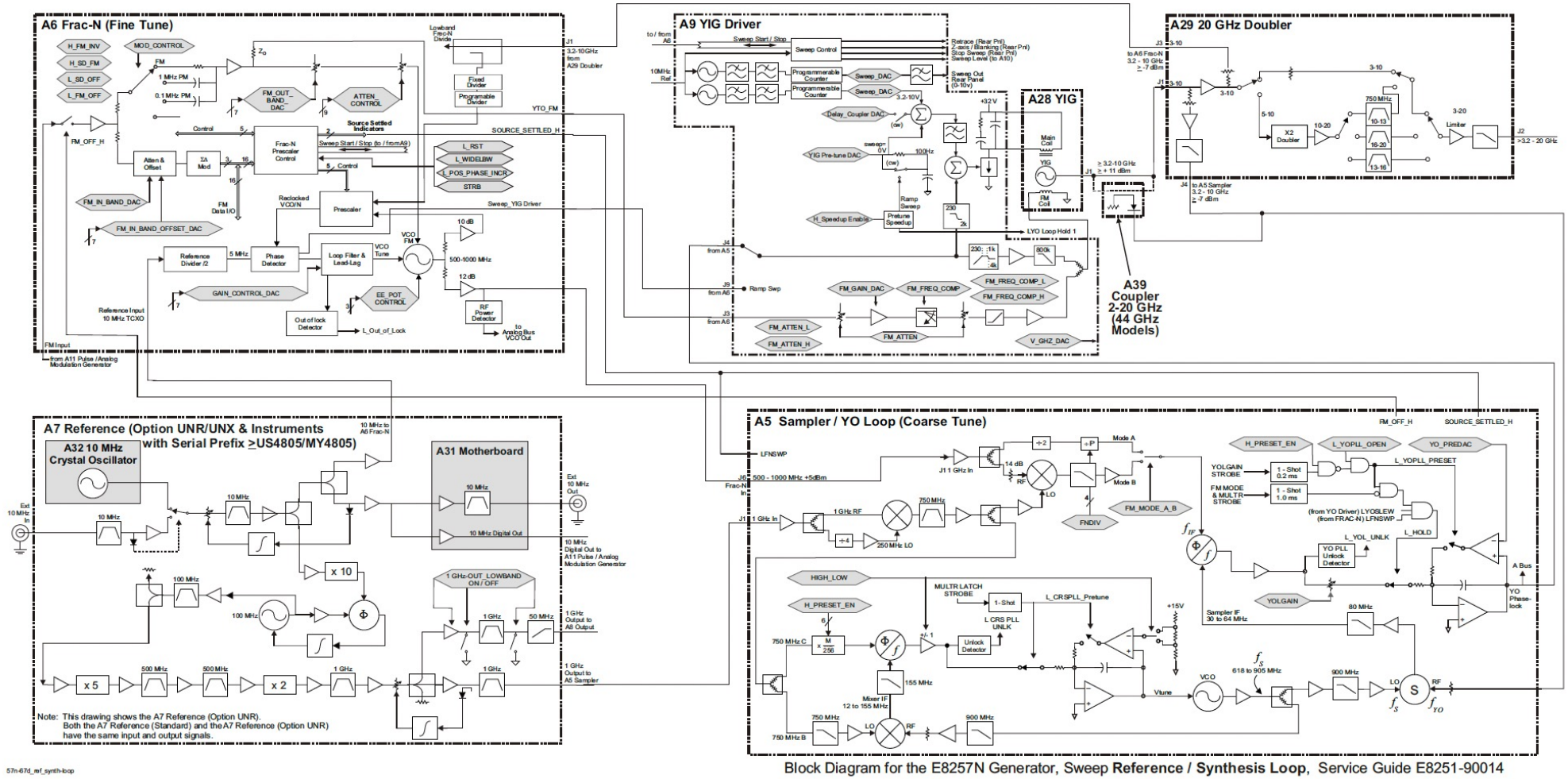




Agilent E8257D block diagram example

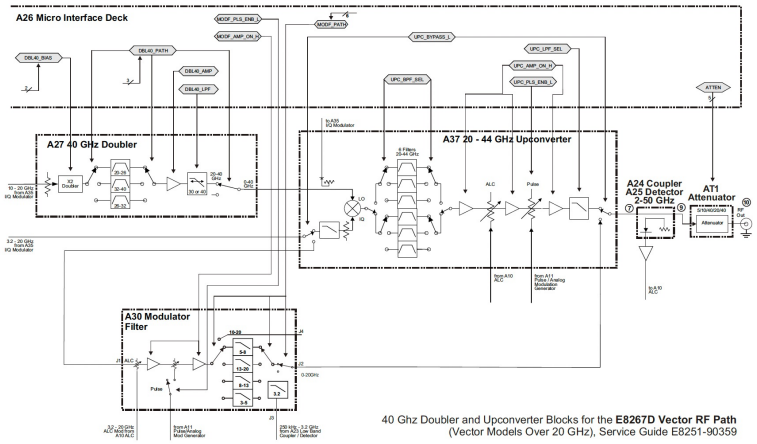
Agilent Technologies
E8257D/67D, E8663D
PSG Signal Generators

Service Guide

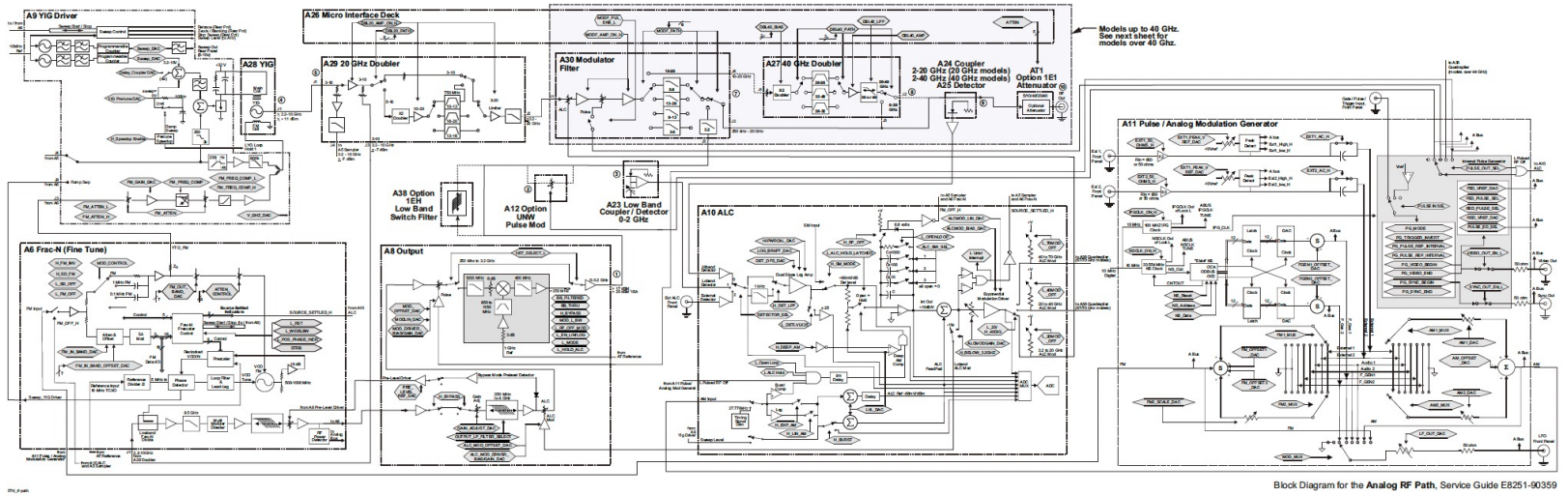


Block Diagram for the E8257N Generator, Sweep **Reference / Synthesis Loop**, Service Guide E8251-90014

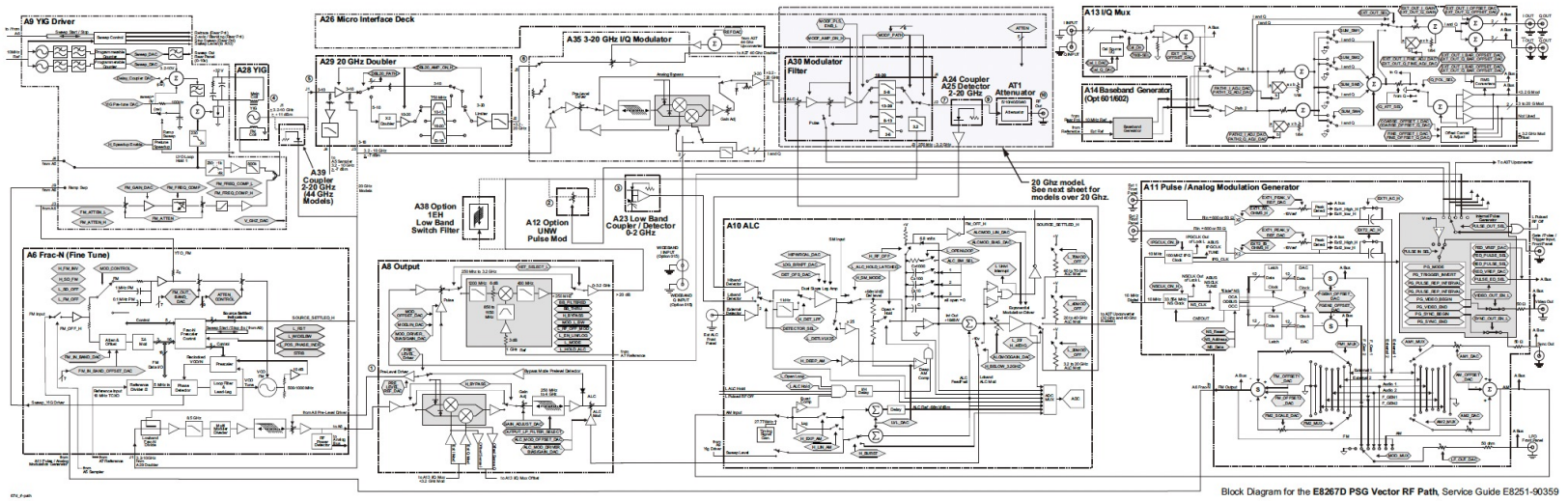
Agilent E8257D block diagram example



40 Ghz Doubler and Upconverter Blocks for the **E8267D Vector RF Path** (Vector Models Over 20 GHz), Service Guide E8251-90359



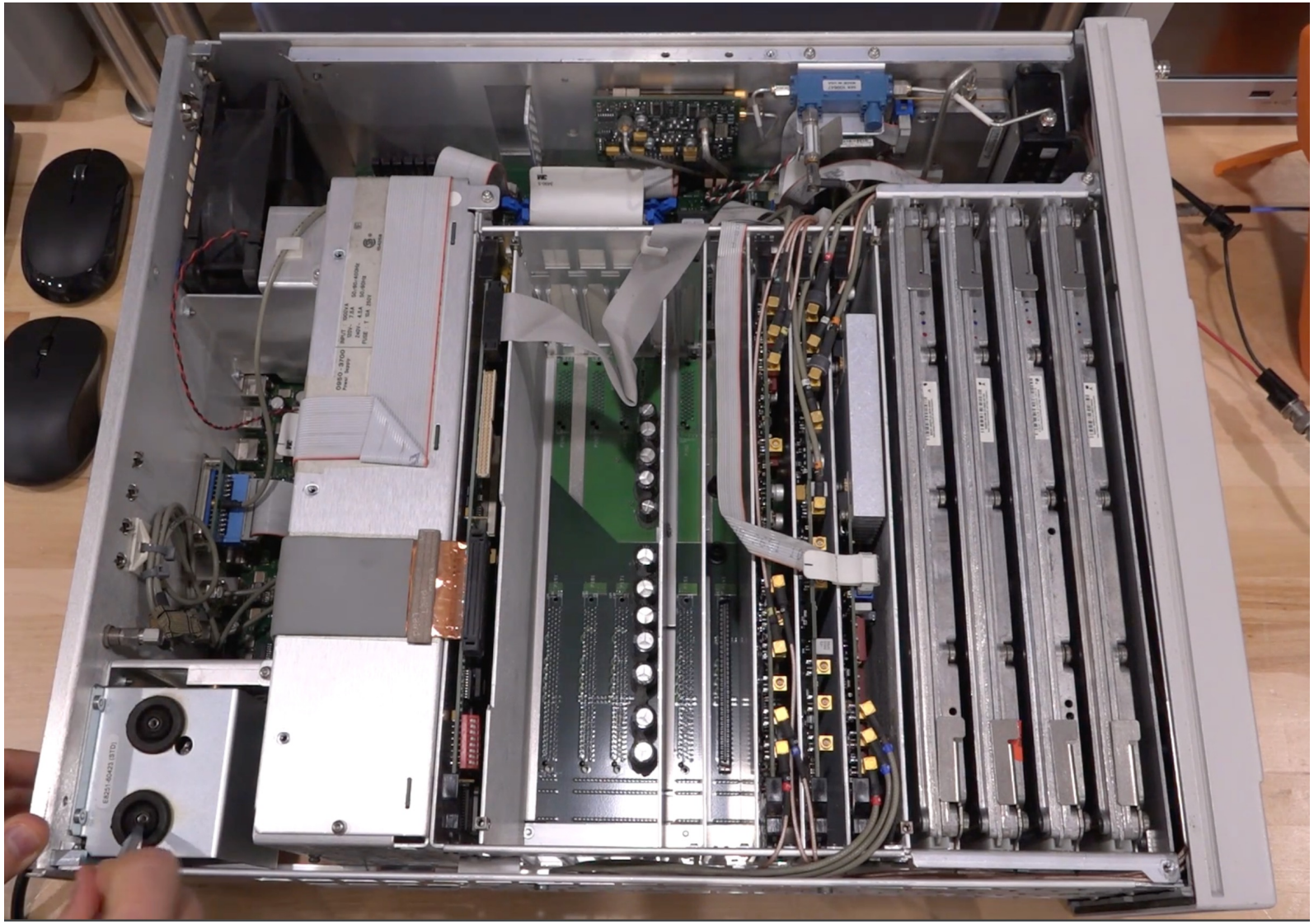
Block Diagram for the **Analog RF Path**, Service Guide E8251-90359



Block Diagram for the **E8267D PSG Vector RF Path**, Service Guide E8251-90359

Agilent E8257D

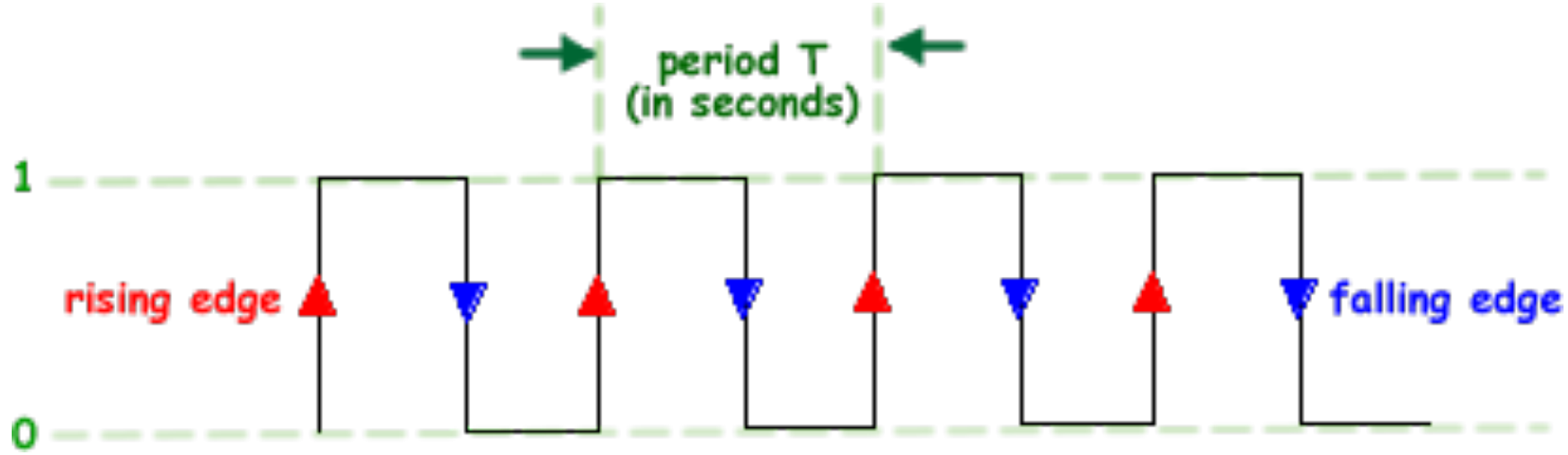
https://www.youtube.com/watch?v=GaaA46r7DUk



A bit of nomenclature

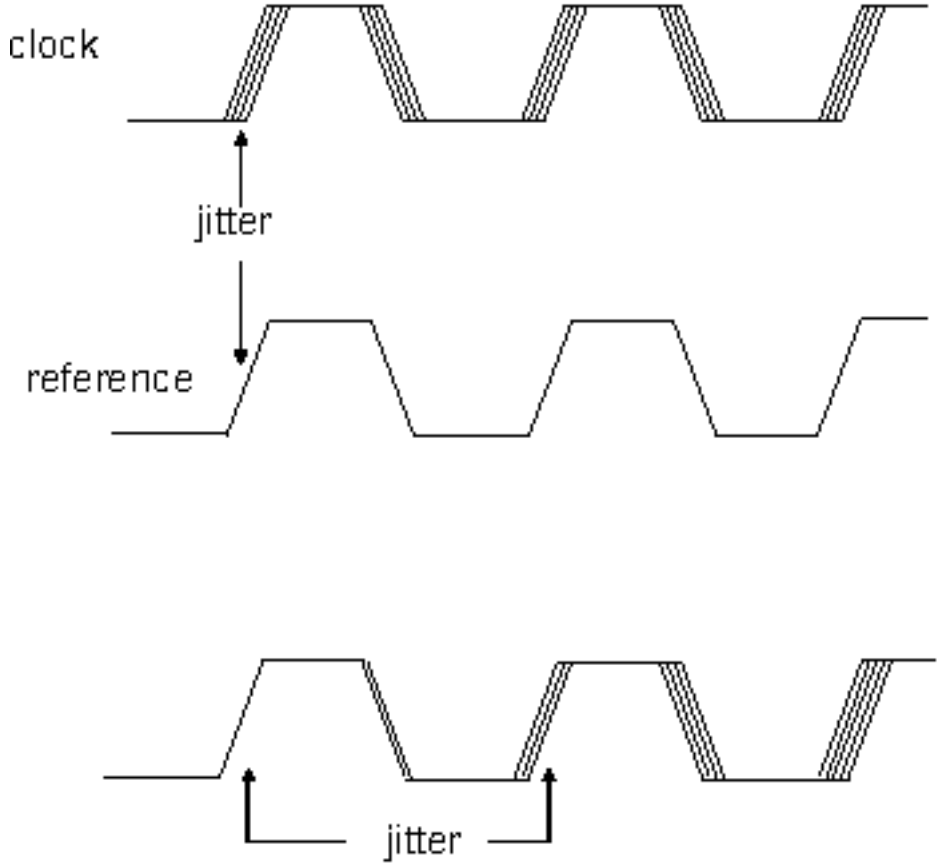
Clock

A **clock** is a periodic timing signal used to synchronize operations in an electronic system, defining when data is sampled, transferred, or processed.



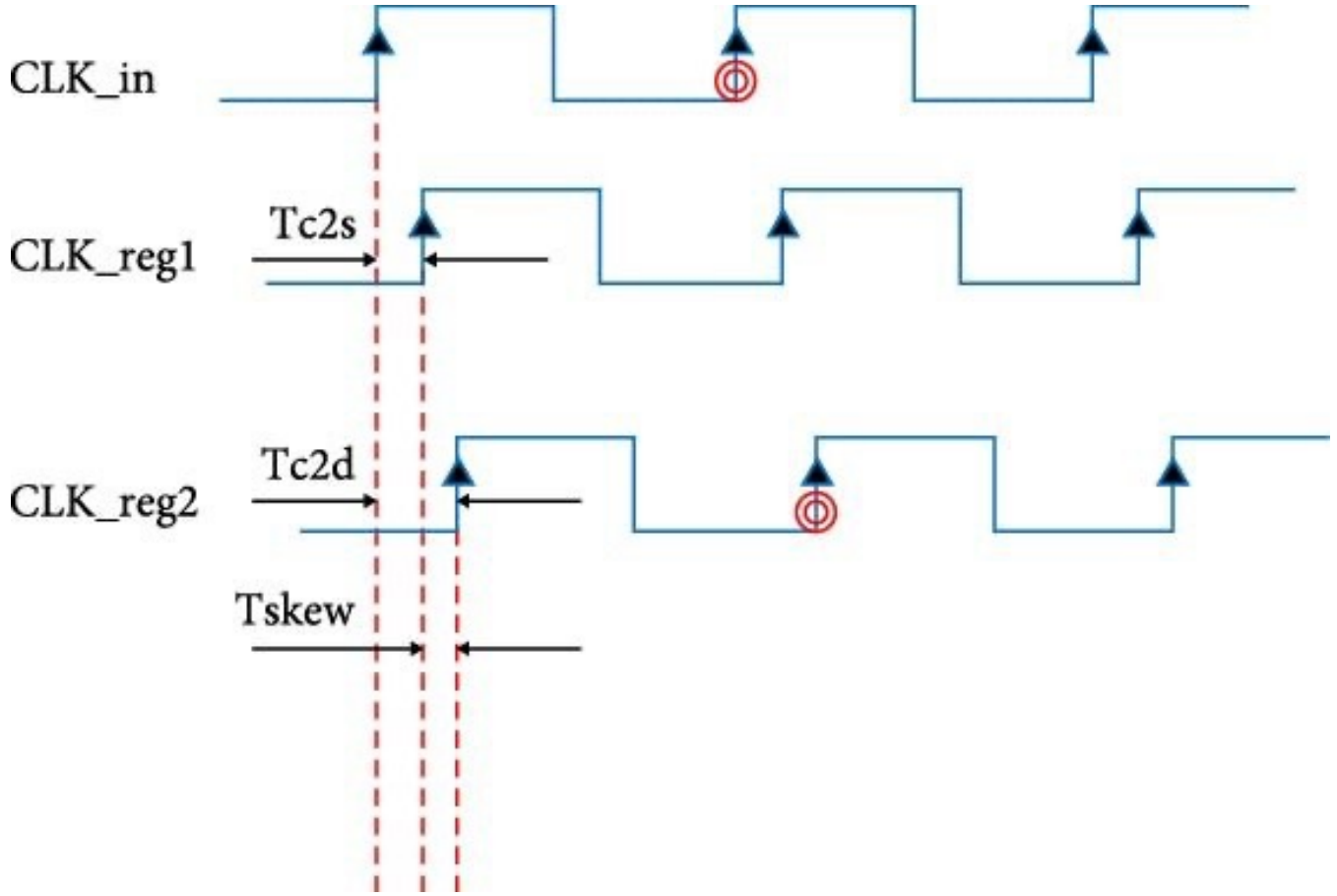
- **62.5 ns (16 MHz):** Arduino UNO
- **20.8 ns (48 MHz):** USB full-speed devices and many microcontrollers
- **10 ns (100 MHz):** Common system clock for digital logic and CPUs
- **1 ns (1 GHz):** High-speed processors and modern communication systems

Skew and Jitter



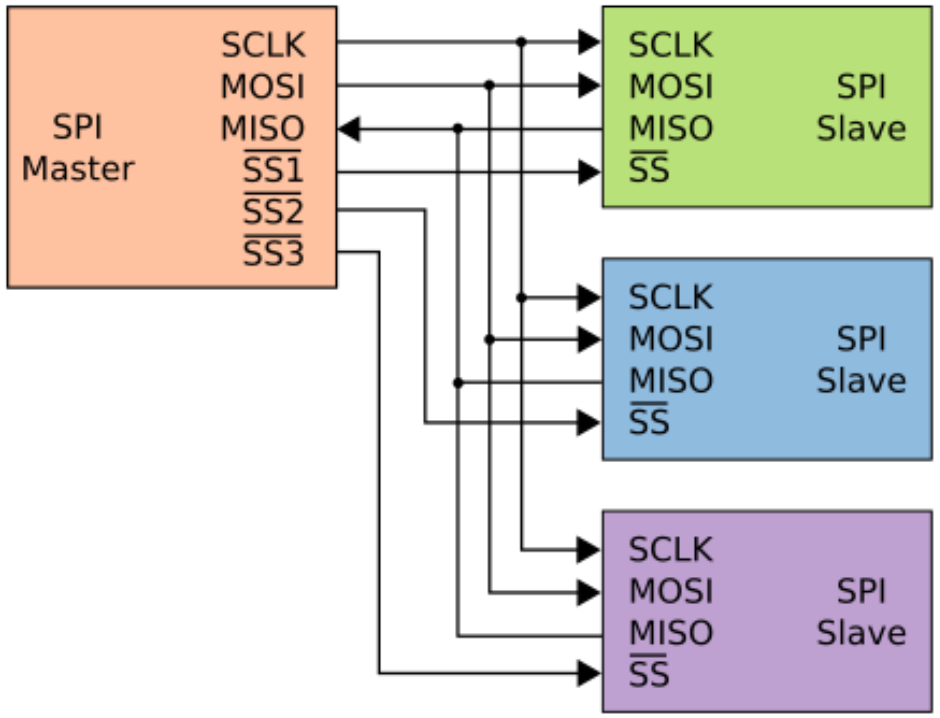
(a)

(b)



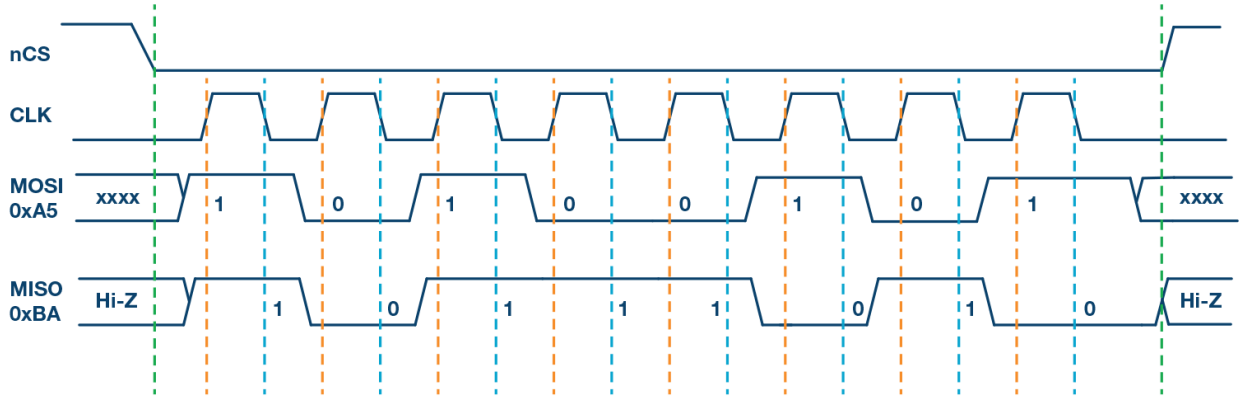
- **Skew** is the fixed or slowly varying difference in timing between related signals (for example, two clock or data lines) that are intended to arrive simultaneously but do not due to path length, component, or loading differences.
- **Jitter** is the short-term, random or deterministic variation in the timing of a signal's edges from their ideal positions, typically caused by noise, interference, or instability in the clock source.

SPI (Serial Peripheral Interface)



SPI bus speeds can reach 50 Mbps or even higher.

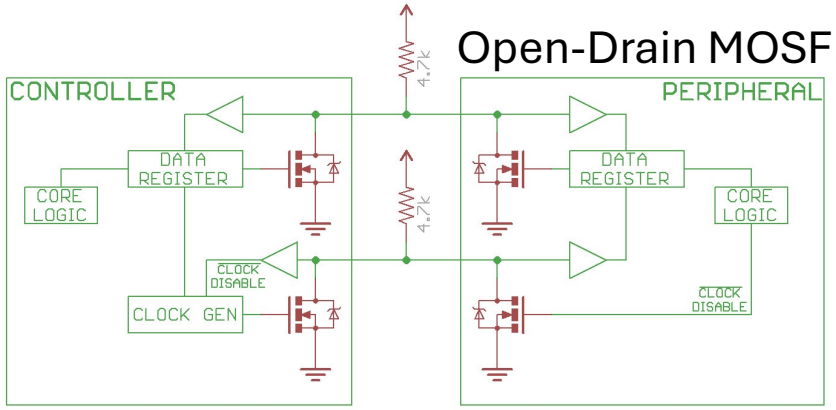
SPI is a synchronous serial communication protocol used for short-distance communication between a master and one or more slave devices. It uses separate lines for **data in**, **data out**, **clock**, and **chip select**, enabling high data rates and simple hardware. Communication is full-duplex.



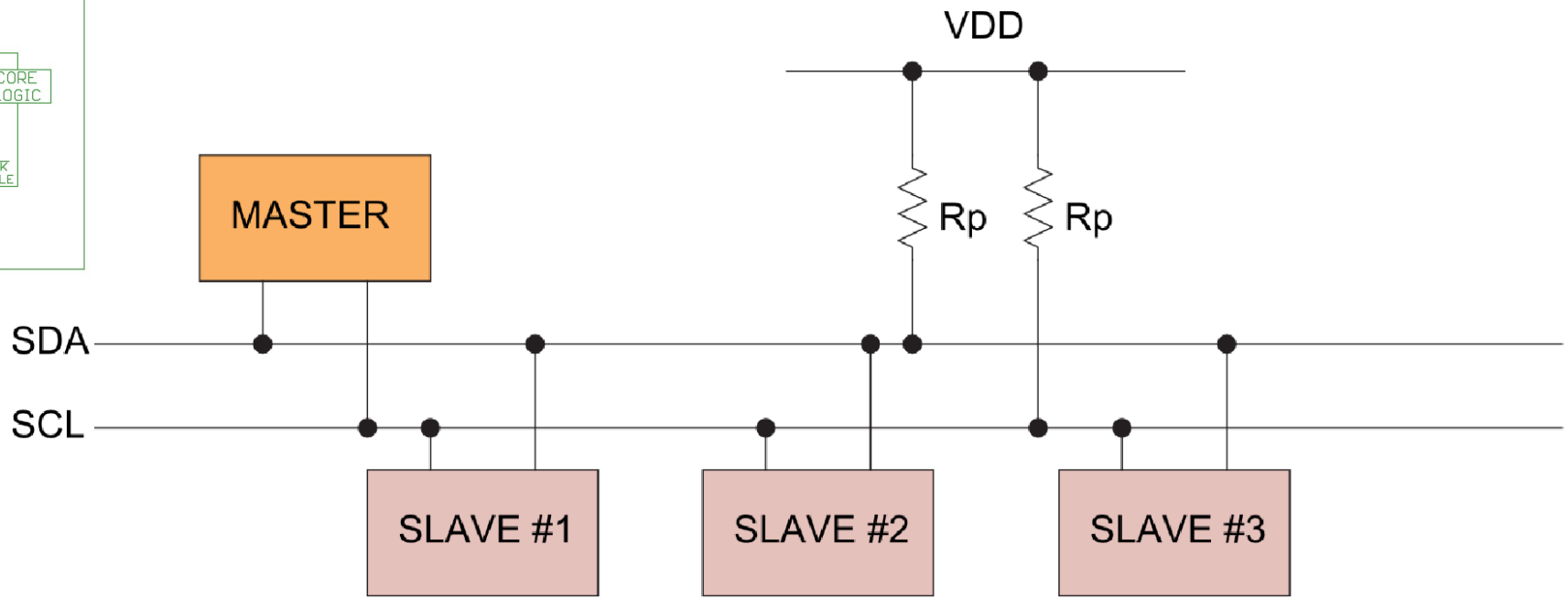
I2C (Inter-Integrated Circuit)

I2C is a synchronous, multi-master serial bus that uses only two lines: a **data line** and a **clock line**. Devices are addressed, allowing many peripherals to share the same bus. It is widely used for low-speed communication with sensors, EEPROMs, and configuration devices.

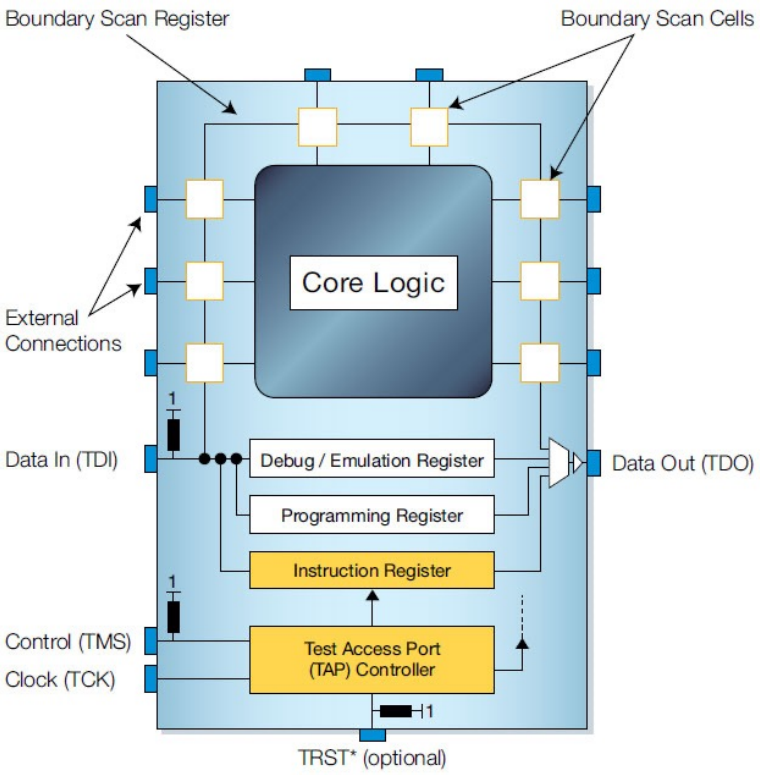
Open-Drain MOSFET configuration



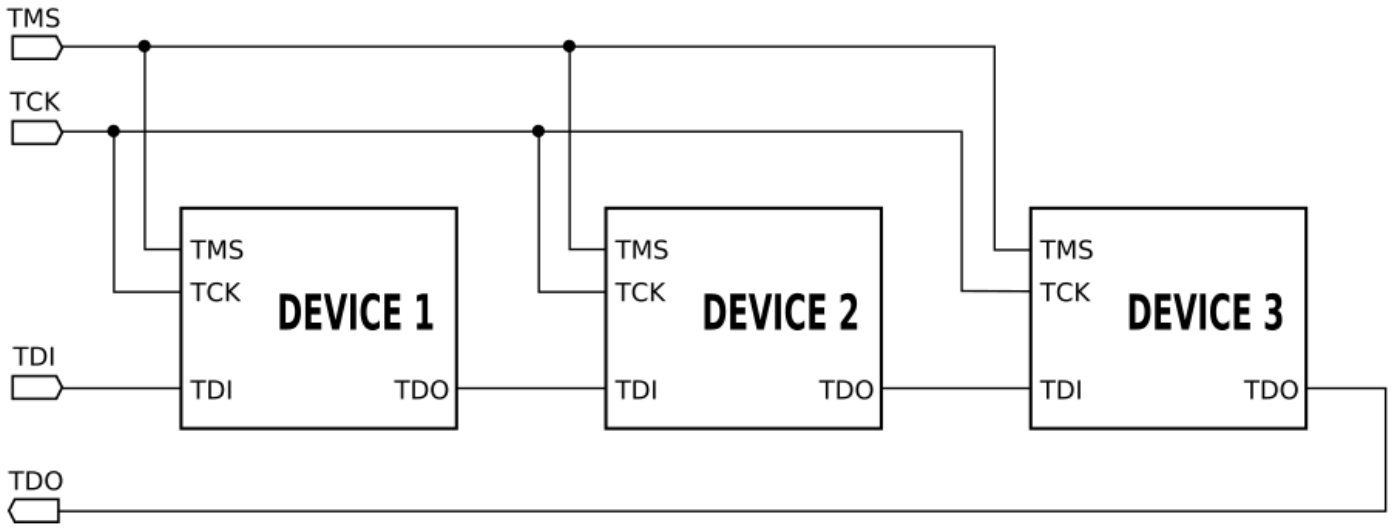
Common I²C bus speeds are the 100 kbit/s *standard mode* and the 400 kbit/s *fast mode*.



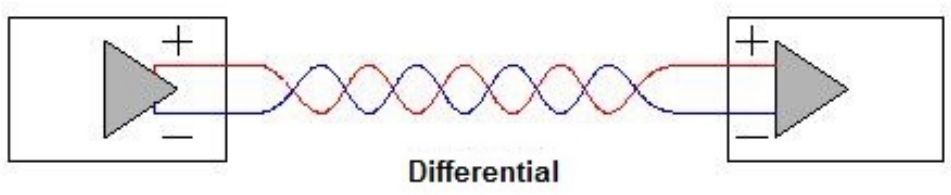
JTAG (Joint Test Action Group)

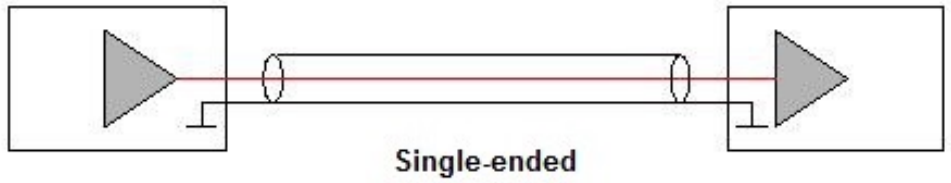


JTAG is a standard interface used for **testing, debugging, and programming electronic devices**. It provides access to internal registers and logic through a small set of dedicated pins. JTAG is commonly used for boundary-scan testing and in-system debugging of microcontrollers and FPGAs.

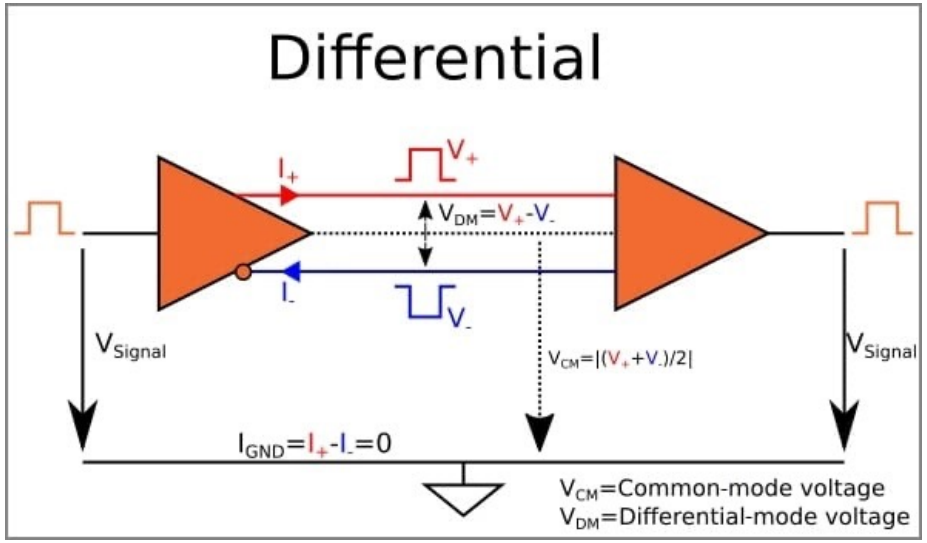


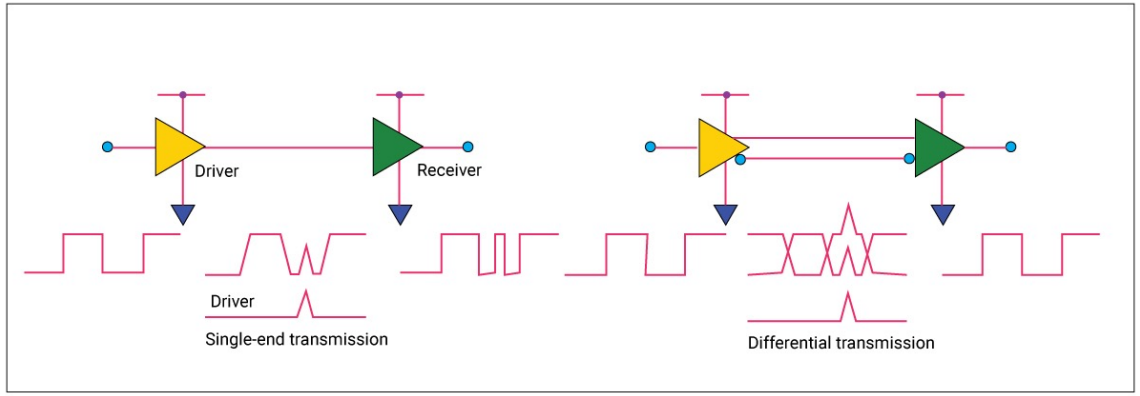
Single ended & Differential signaling





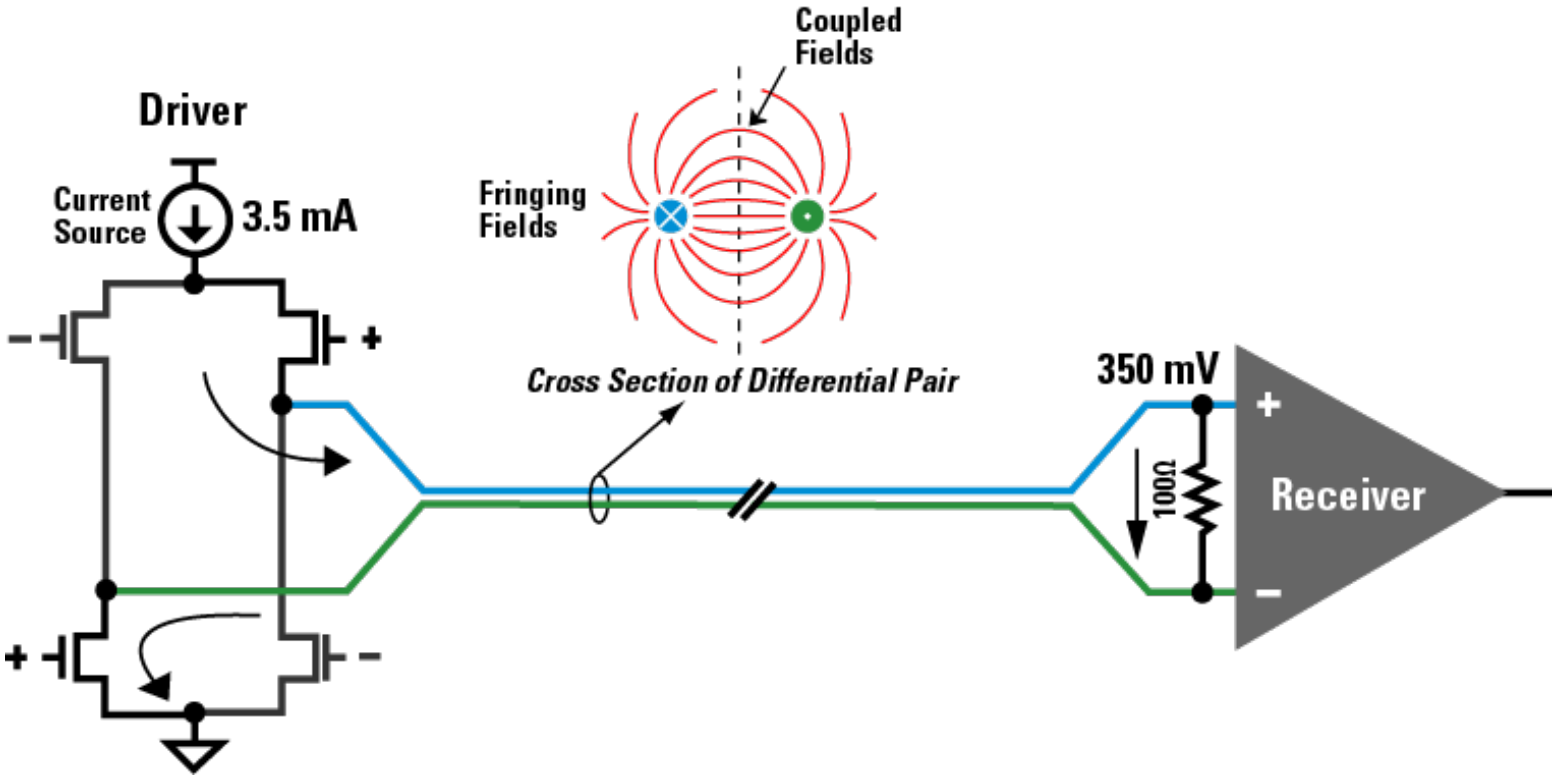
Single-ended signaling transmits information using one signal line referenced to a common ground. It is simple and low-cost but more susceptible to noise, ground offsets, and electromagnetic interference, especially at high speeds or long distances.



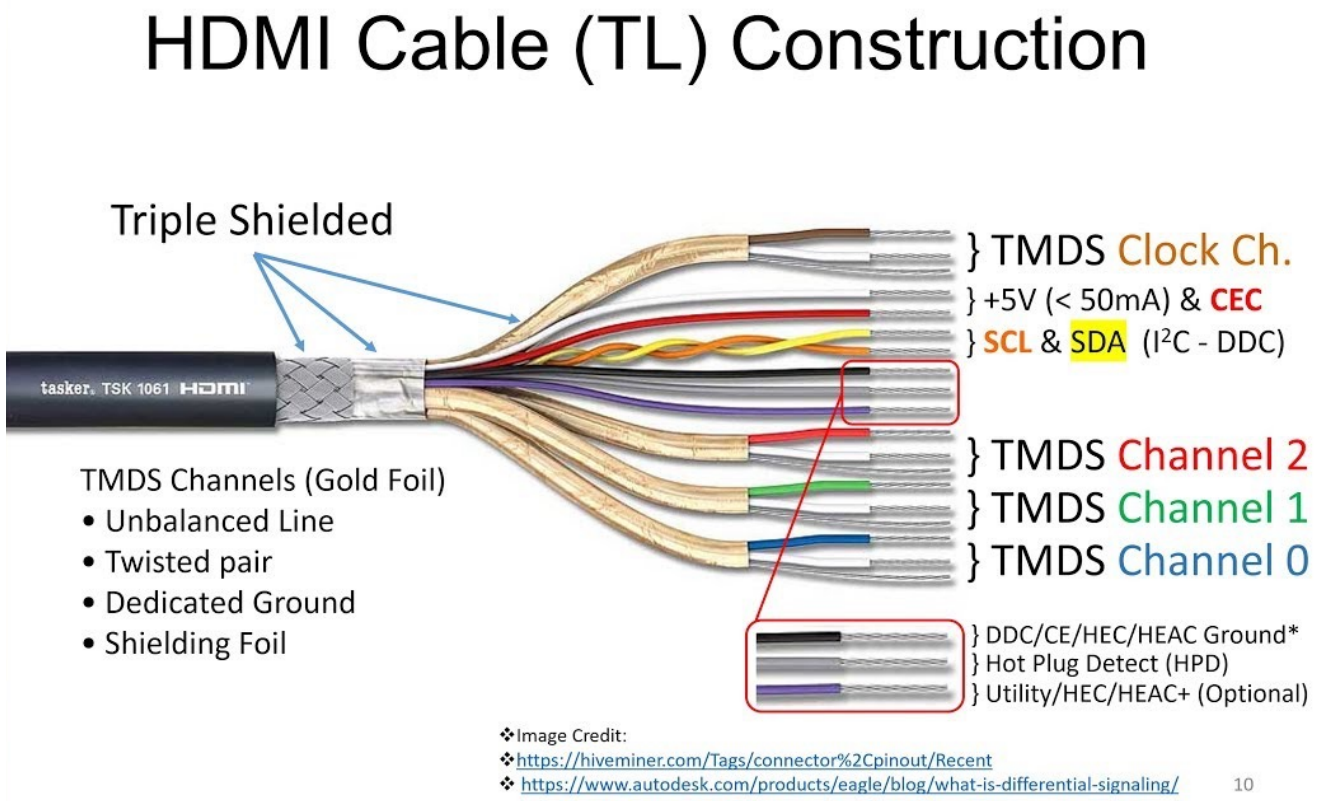


Differential signaling uses two complementary signal lines, and information is conveyed by the voltage difference between them.

LVDS (Low-voltage differential signaling)



TMDS (Transition-minimized differential signaling)



Signaling standards

SelectIO™ Interface DC Input and Output Levels

Table 7: **Recommended Operating Conditions for User I/Os Using Single-Ended Standards**

I/O Standard	V_{CCO} for Drivers(1)			V_{REF} for Inputs		
	V, Min	V, Nom	V, Max	V, Min	V, Nom	V, Max
LVTTL	3.0	3.3	3.45			
LVCMOS33	3.0	3.3	3.45			
LVCMOS25	2.3	2.5	2.7			
LVCMOS18	1.65	1.8	1.95			

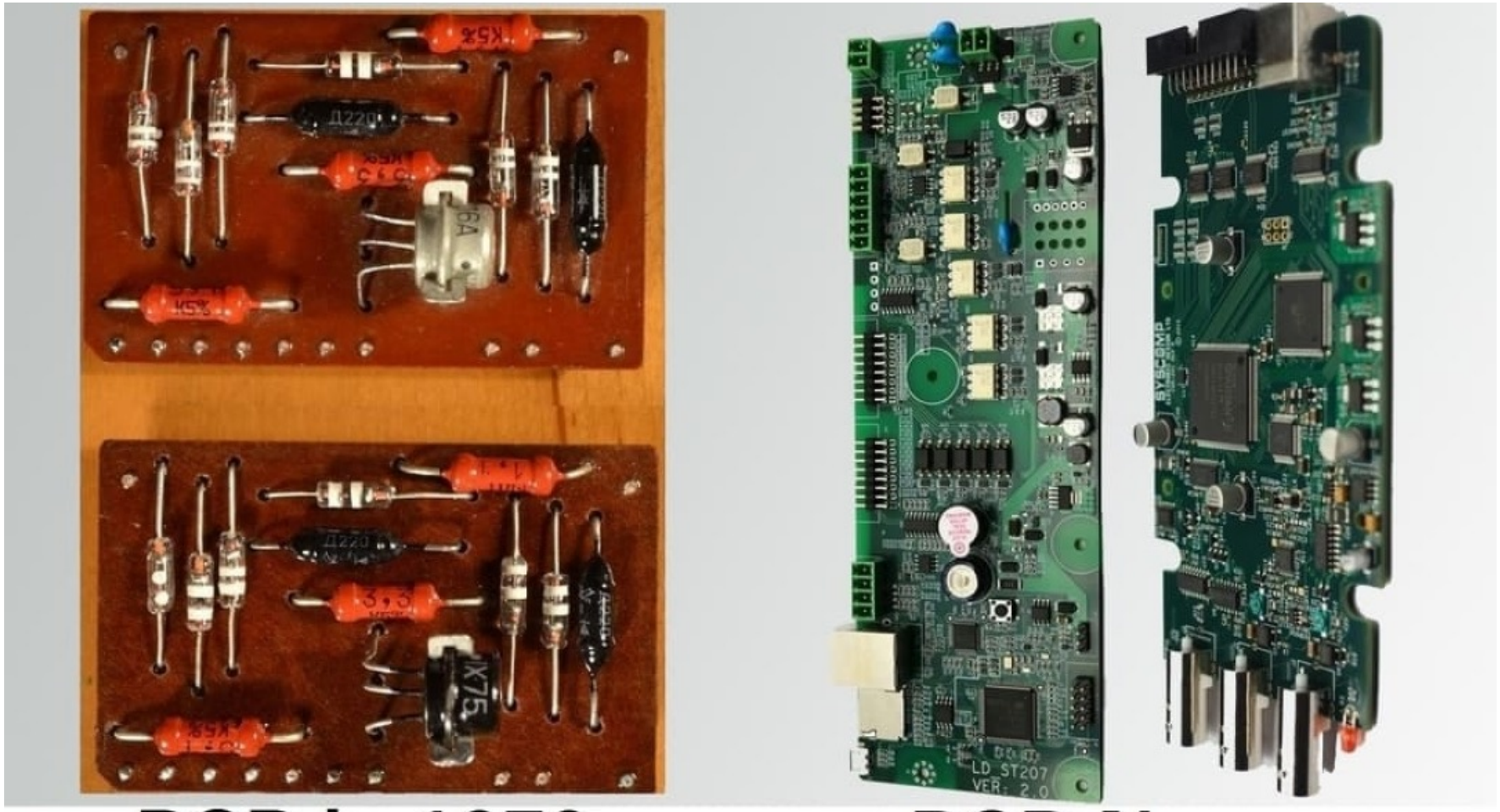
Table 9: **Single-Ended I/O Standard DC Input and Output Levels**

I/O Standard	V_{IL}		V_{IH}		V_{OL}	V_{OH}	I_{OL}	I_{OH}
	V, Min	V, Max	V, Min	V, Max	V, Max	V, Min	mA	mA
LVTTL	–0.5	0.8	2.0	4.1	0.4	2.4	Note 2	Note 2
LVCMOS33	–0.5	0.8	2.0	4.1	0.4	V_{CCO} – 0.4	Note 2	Note 2
LVCMOS25	–0.5	0.7	1.7	4.1	0.4	V_{CCO} – 0.4	Note 2	Note 2
LVCMOS18	–0.5	0.38	0.8	4.1	0.45	V_{CCO} – 0.45	Note 2	Note 2

Table 10: **Differential I/O Standard DC Input and Output Levels**

I/O Standard	V_{ID}		V_{ICM}		V_{OD}		V_{OCM}		V_{OH}	V_{OL}
	mV, Min	mV, Max	V, Min	V, Max	mV, Min	mV, Max	V, Min	V, Max	V, Min	V, Max
LVDS_33(2)(3)	100	600	0.3	2.35	247	454	1.125	1.375	–	–
LVDS_25(2)(3)	100	600	0.3	2.35	247	454	1.125	1.375	–	–

PCB development and history

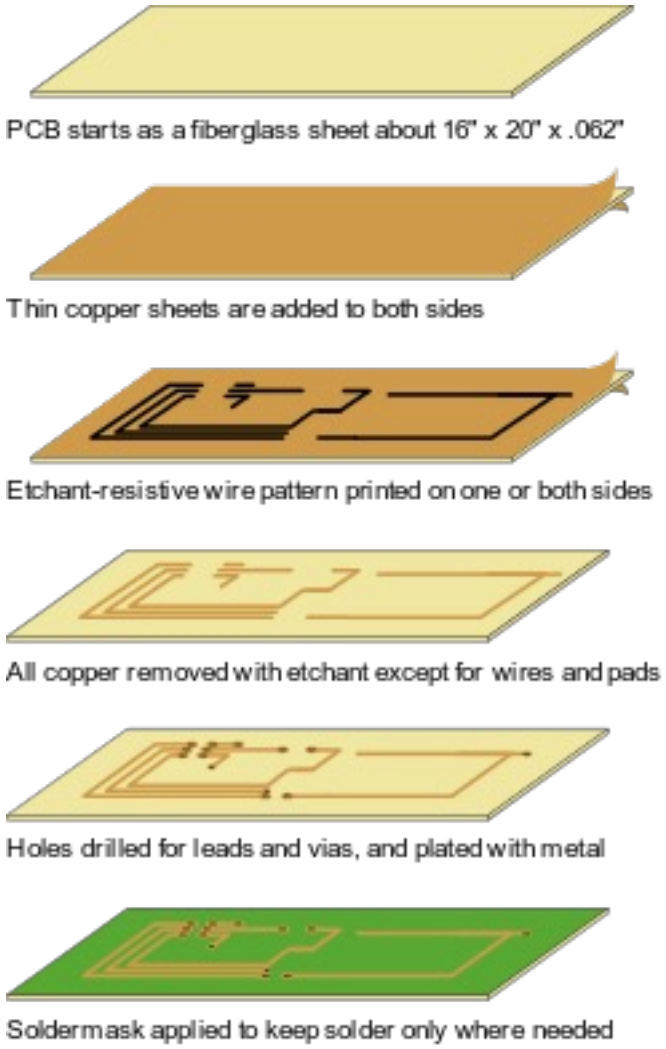


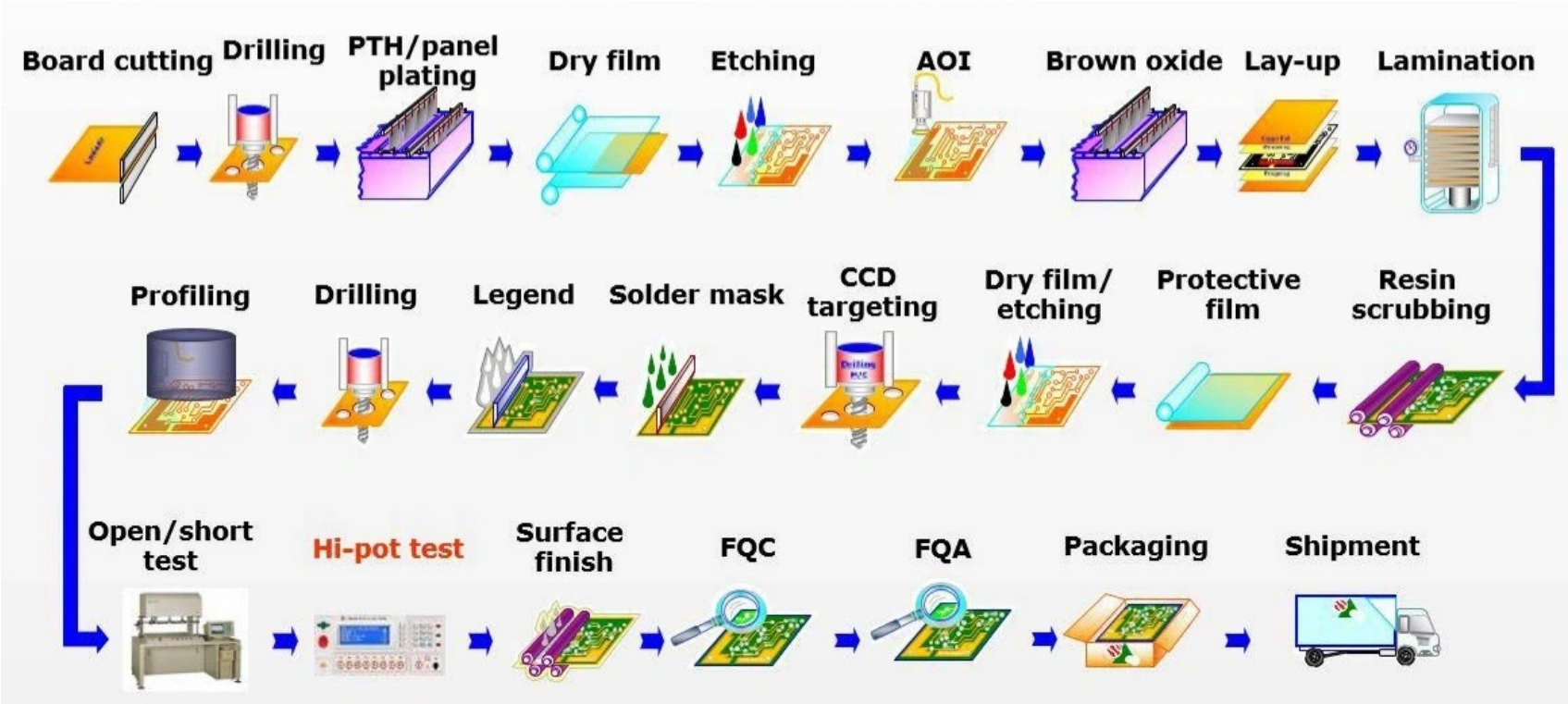
Why we need Printed Circuit Boards

An old Motorola television without PCB from 1948

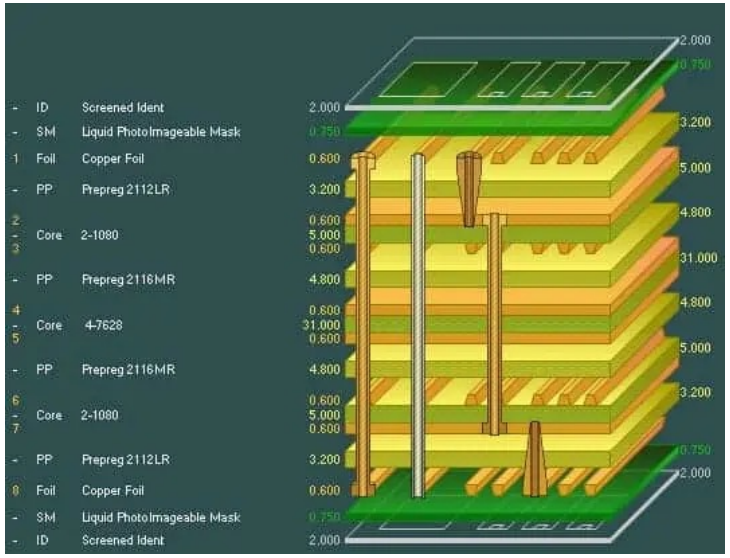
- Miniaturize devices
- Improve mass production
- Reduce cost
- Improve yeld
- Increase repeatability
- Reduced parasitics

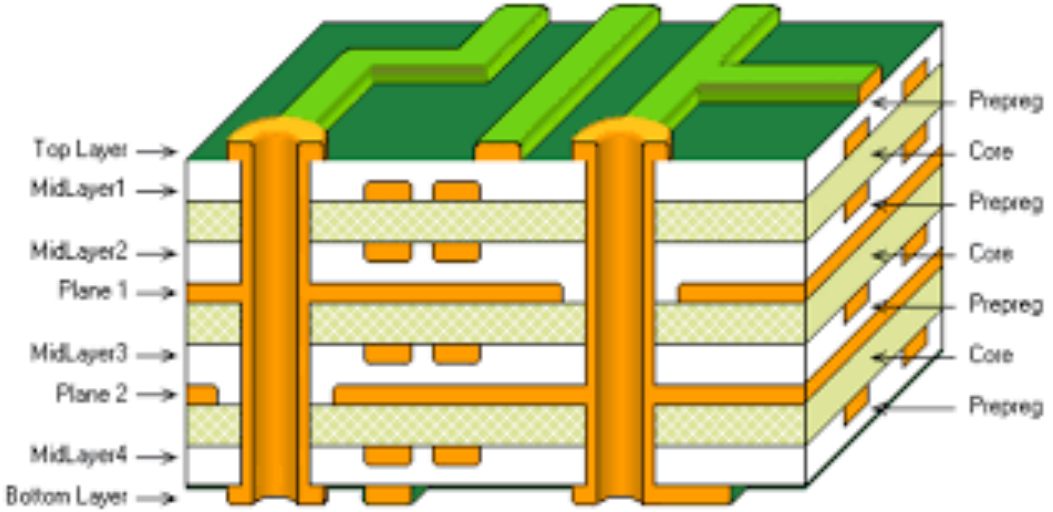
How a PCB is made?

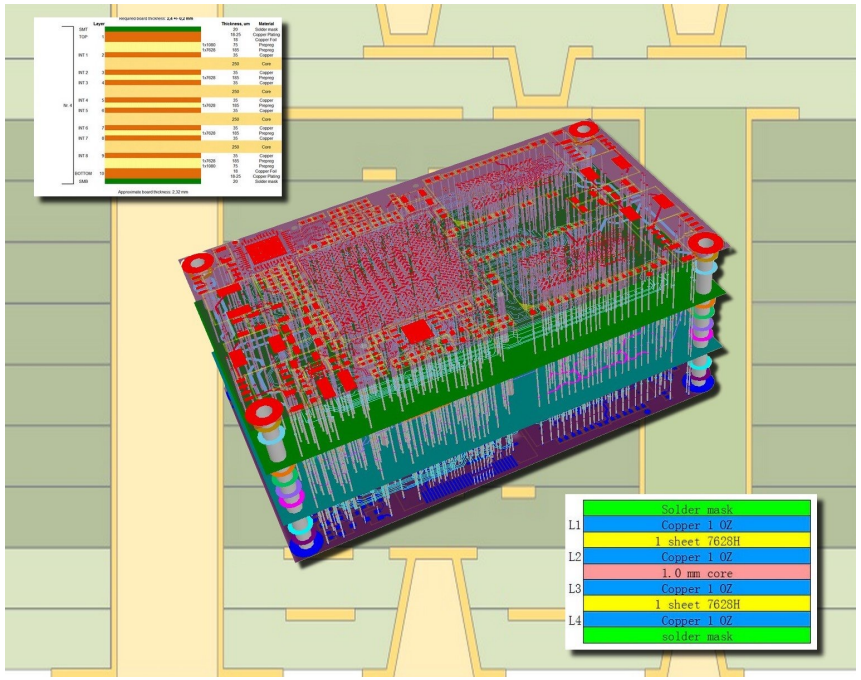


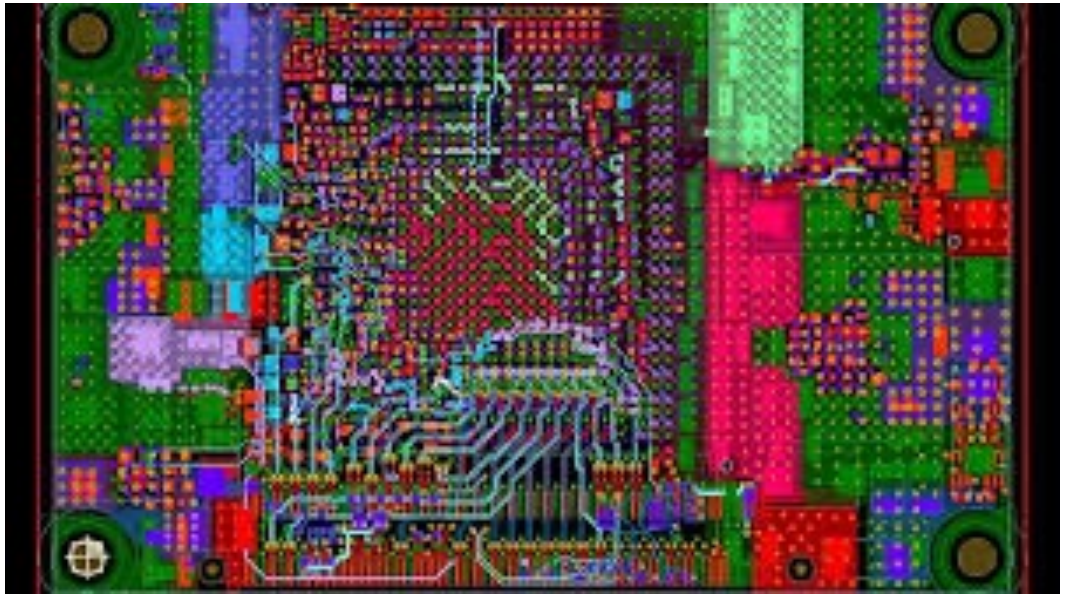


PCB stackup/1









PCB stackup/2

isola

185HR

High Reliability Epoxy Laminate and Prepreg

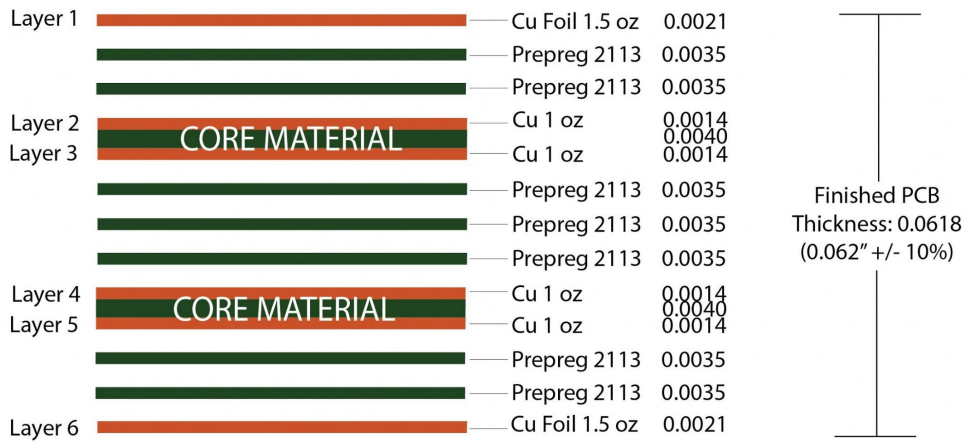
185HR laminate and prepreg materials are a proprietary, high-performance resin system with a Tg of 180°C for multilayer Printed Wiring Board (PWB) applications where maximum thermal performance and reliability are required.

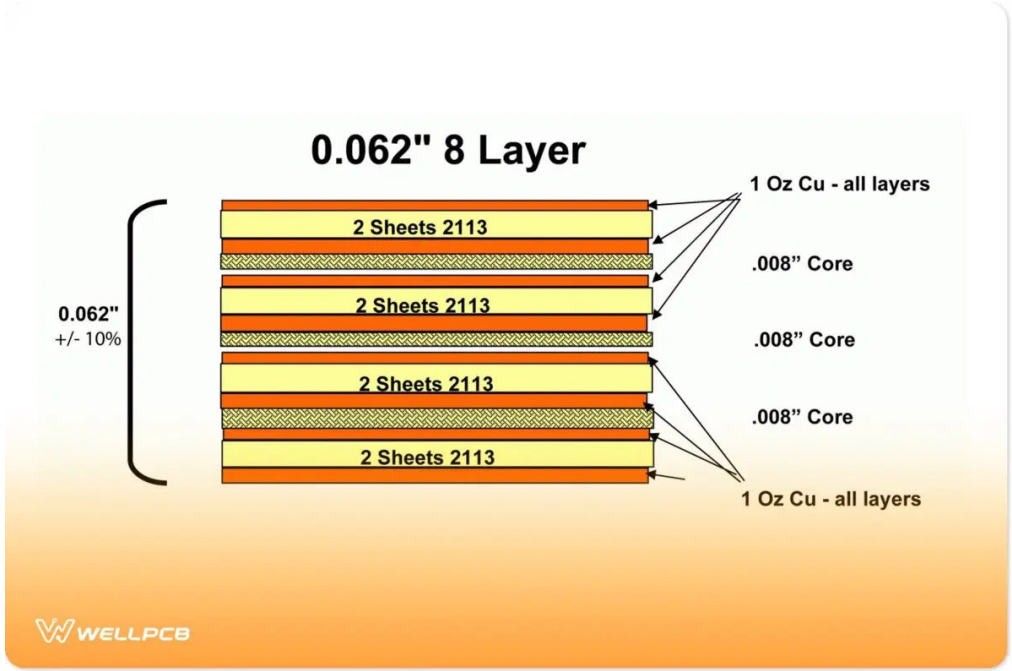
Constructions	Resin Content %	Standard/ Alternate	Thickness (inch)	Thickness (mm)	Dielectric Constant (DK)/ Dissipation Factor (DF)					
					100 MHz	500 MHz	1 GHz	2 GHz	5 GHz	10 GHz
1x1080	60%	Standard	0.0025	0.064	4.06 0.0160	3.98 0.0190	3.96 0.0200	3.93 0.0210	3.81 0.0250	3.80 0.0250
1x1067	73%	Alt/Spread	0.0025	0.064	3.84 0.0180	3.77 0.0210	3.74 0.0230	3.70 0.0240	3.55 0.0290	3.54 0.0290
1x1080	66%	Standard	0.0030	0.076	3.96 0.0170	3.89 0.0200	3.86 0.0220	3.82 0.0220	3.68 0.0270	3.68 0.0270
1x1086	61%	Alt/Spread	0.0030	0.076	4.04 0.0170	3.97 0.0190	3.94 0.0200	3.91 0.0210	3.78 0.0250	3.78 0.0250
1x3313	52%	Alternate	0.0035	0.089	4.20 0.0150	4.13 0.0170	4.11 0.0180	4.09 0.0190	3.97 0.0220	3.98 0.0220
1x2113	54%	Standard	0.0035	0.089	4.19 0.0160	4.10 0.0170	4.07 0.0190	4.04 0.0200	3.92 0.0230	3.92 0.0230
2x106	68%	Standard	0.0035	0.089	3.92 0.0140	3.88 0.0150	3.86 0.0180	3.84 0.0190	3.69 0.0190	3.69 0.0190
1x2116	48%	Standard	0.0040	0.102	4.28 0.0150	4.21 0.0160	4.20 0.0180	4.18 0.0180	4.05 0.0210	4.05 0.0210
1x3313	57%	Alternate	0.0040	0.102	4.11 0.0160	4.04 0.0180	4.03 0.0190	3.99 0.0200	3.86 0.0240	3.86 0.0240
2x106	72%	Standard	0.0040	0.102	3.86 0.0180	3.78 0.0210	3.76 0.0230	3.72 0.0240	3.57 0.0280	3.56 0.0290
1x2116	53%	Standard	0.0045	0.114	4.18 0.0150	4.11 0.0170	4.10 0.0190	4.07 0.0190	3.94 0.0230	3.94 0.0230
1x106 / 1x1080	65%	Alternate	0.0045	0.114	3.97 0.0170	3.90 0.0200	3.87 0.0210	3.84 0.0220	3.70 0.0260	3.70 0.0260
1x2116	57%	Standard	0.0050	0.127	4.11 0.0160	4.04 0.0180	4.03 0.0190	3.99 0.0200	3.86 0.0240	3.86 0.0240
2x1080	60%	Standard	0.0050	0.127	4.06 0.0160	3.98 0.0190	3.96 0.0200	3.93 0.0210	3.81 0.0250	3.80 0.0250
2x1067	73%	Alt/Spread	0.0050	0.127	3.84 0.0180	3.77 0.0210	3.74 0.0230	3.70 0.0240	3.55 0.0290	3.54 0.0290
2x1080	63%	Alternate	0.0055	0.140	4.00 0.0170	3.93 0.0190	3.91 0.0210	3.87 0.0220	3.74 0.0260	3.74 0.0260
1x106 / 1x2113	60%	Alternate	0.0055	0.140	4.06 0.0160	3.98 0.0190	3.96 0.0200	3.93 0.0210	3.81 0.0250	3.80 0.0250
1x1652	52%	Standard	0.0060	0.152	4.20 0.0150	4.13 0.0170	4.11 0.0180	4.09 0.0190	3.97 0.0220	3.98 0.0220
2x1080	65%	Standard	0.0060	0.152	3.97 0.0170	3.90 0.0200	3.87 0.0210	3.84 0.0220	3.70 0.0260	3.70 0.0260

Property		Typical Value	Units	Test Method
			Metric (English)	IPC-TM-650 (or as noted)
Test data generated from rigid laminate		50	%	2.3.16.2
Glass Transition Temperature (Tg) by DSC		180	°C	2.4.25C
Glass Transition Temperature (Tg) by DMA		185	°C	2.4.24.4
Decomposition Temperature (Td) by TGA @ 5% weight loss		340	°C	2.4.24.6
Time to Delaminate by TMA (Copper removed)	A. T260 B. T288	60 >15	Minutes	2.4.24.1
Z-Axis CTE	A. Pre-Tg B. Post-Tg C. 50 to 260°C, (Total Expansion)	40 220 2.7	ppm/°C ppm/°C %	2.4.24C
X/Y-Axis CTE	Pre-Tg	13/14	ppm/°C	2.4.24C
Thermal Conductivity		0.4	W/m·K	ASTM E1952
Thermal Stress 10 sec @ 288°C (550.4°F)	A. Unetched B. Etched	Pass	Pass Visual	2.4.13.1
Dk, Permittivity	A. @ 100 MHz B. @ 1 GHz C. @ 2 GHz D. @ 5 GHz E. @ 10 GHz	4.13 4.04 4.01 3.88 3.88	—	2.5.5.3 Bereskin Stripline Bereskin Stripline Bereskin Stripline Bereskin Stripline
Df, Loss Tangent	A. @ 100 MHz B. @ 1 GHz C. @ 2 GHz D. @ 5 GHz E. @ 10 GHz	0.0158 0.0192 0.0200 0.0235 0.0236	—	2.5.5.3 Bereskin Stripline Bereskin Stripline Bereskin Stripline Bereskin Stripline
Volume Resistivity	A. C-96/35/90 B. After moisture resistance C. At elevated temperature	— 3.0×10^8 7.0×10^8	MΩ-cm	2.5.17.1
Surface Resistivity	A. C-96/35/90 B. After moisture resistance C. At elevated temperature	— 3.0×10^6 2.0×10^8	MΩ	2.5.17.1
Dielectric Breakdown		>50	kV	2.5.6B
Arc Resistance		115	Seconds	2.5.1B
Electric Strength (Laminate & laminated prepreg)		54 (1350)	kV/mm (V/mil)	2.5.6.2A
Comparative Tracking Index (CTI)		3 (175-249)	Class (Volts)	UL 746A ASTM D3638
Peel Strength	A. Low profile copper foil and very low profile copper foil all copper foil >17 µm [0.669 mil] B. Standard profile copper 1. After thermal stress 2. At 125°C (257°F) 3. After process solutions	>0.79 (>4.5) >0.79 (>4.5) >0.79 (>4.5) >0.79 (>4.5)	N/mm (lb/inch)	2.4.8C 2.4.8.2A 2.4.8.3 2.4.8.3
Flexural Strength	A. Length direction B. Cross direction	669 (97.1) 373 (54.1)	MPa (kpsi)	2.4.4B

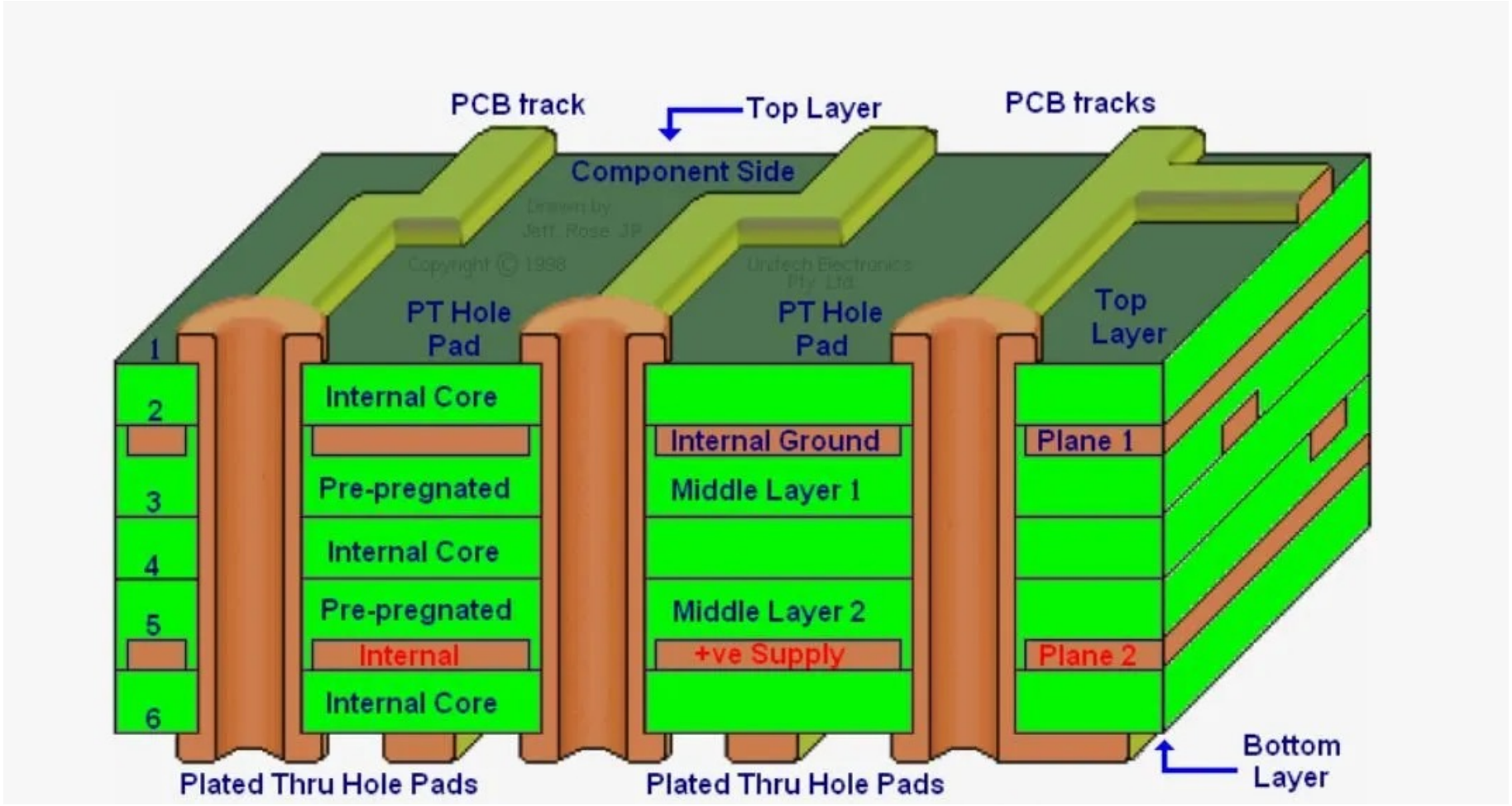
Nominal dimensions

Copper Weight	Copper Thickness
1 oz	1.37 mils/0.0348mm
2 oz	2.74 mils/0.0696mm
3 oz	4.11 mils/0.1044mm
4 oz	5.48 mils/0.1392mm

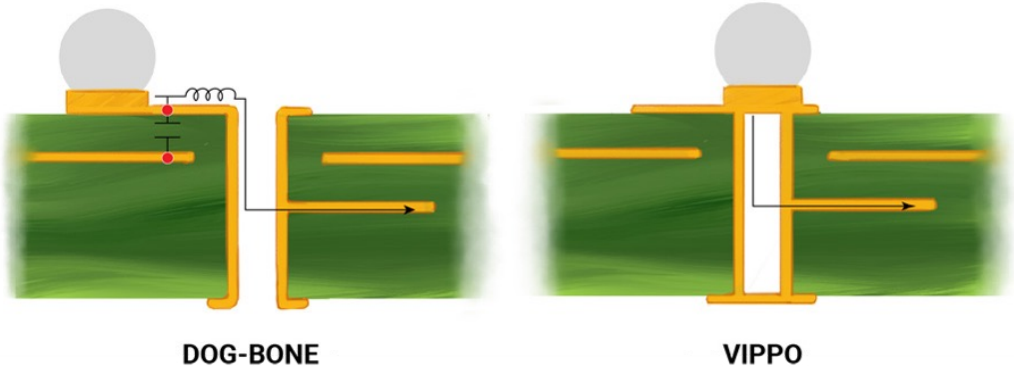


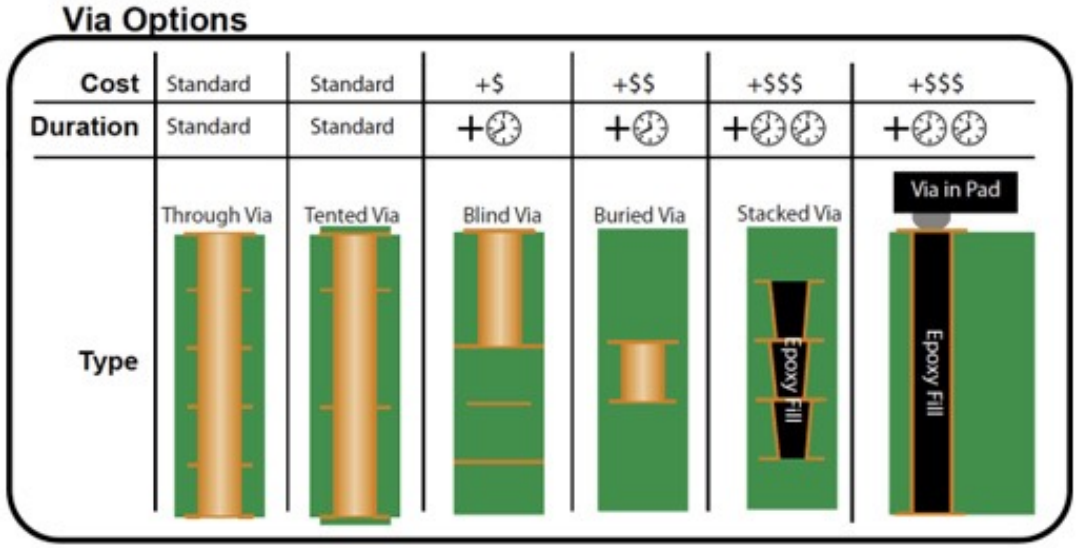


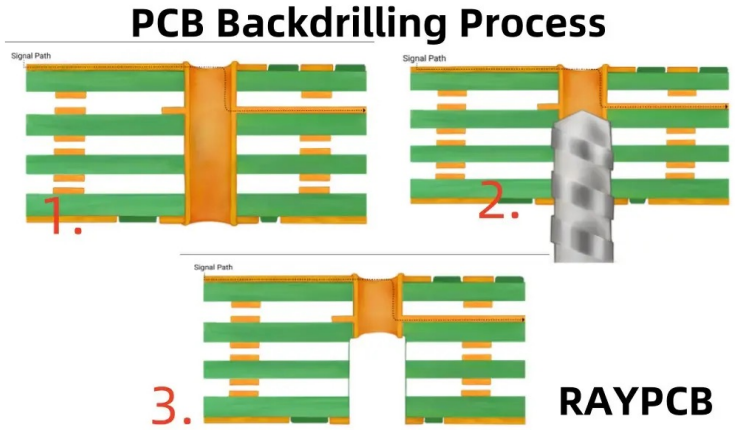
Some nomenclature

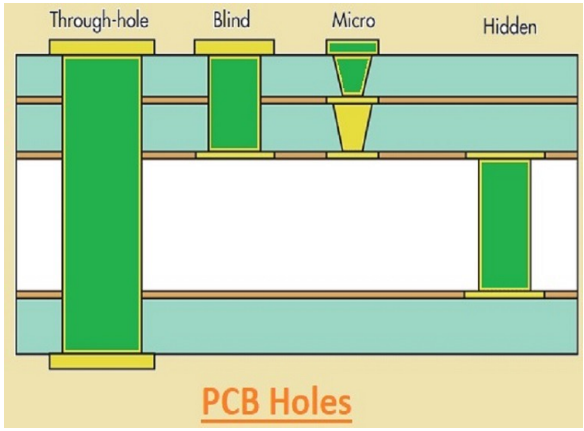


VIAs

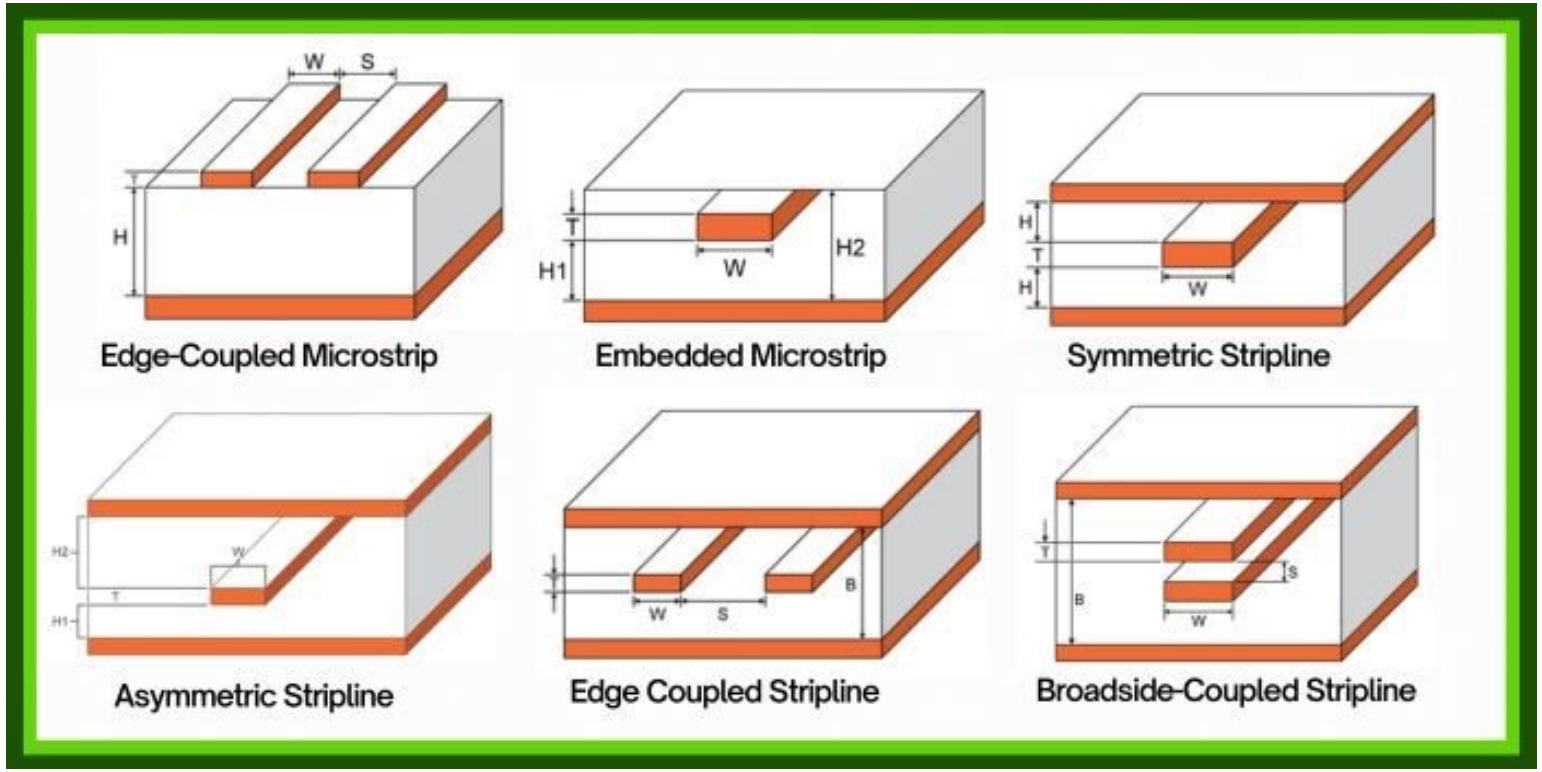

DOG-BONE
VIPPO


Via Options
Cost
Standard
Standard
+$
+$$
+$$$
+$$$
Duration
Standard
Standard
Type
Through Via
Tented Via
Blind Via
Buried Via
Stacked Via
Via in Pad
Epoxy Fill
Epoxy Fill


PCB Backdrilling Process
Signal Path
Signal Path
Signal Path
1.
2.
3.
RAYPCB

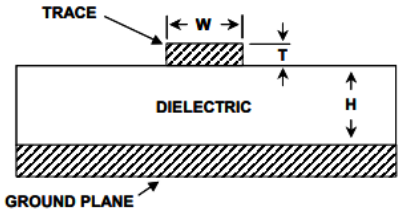

Through-hole
Blind
Micro
Hidden
PCB Holes

Tracks


W
S
H
Edge-Coupled Microstrip
T
H1
W
H2
Embedded Microstrip
H
T
H
W
Symmetric Stripline
H2
W
T
H1
Asymmetric Stripline
W
S
B
Edge Coupled Stripline
T
B
S
W
Broadside-Coupled Stripline

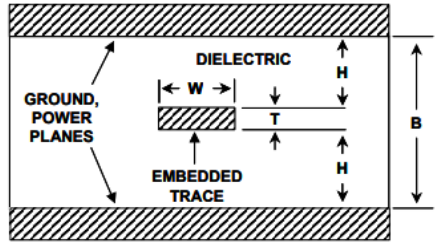
Track impedance

$$Z = \frac{87}{\sqrt{\varepsilon_r + 1.41}} \ln\left(\frac{5.98H}{0.8W + T}\right)$$



"In electrical engineering, impedance is the opposition to alternating current presented by the combined effect of resistance and reactance in a circuit."

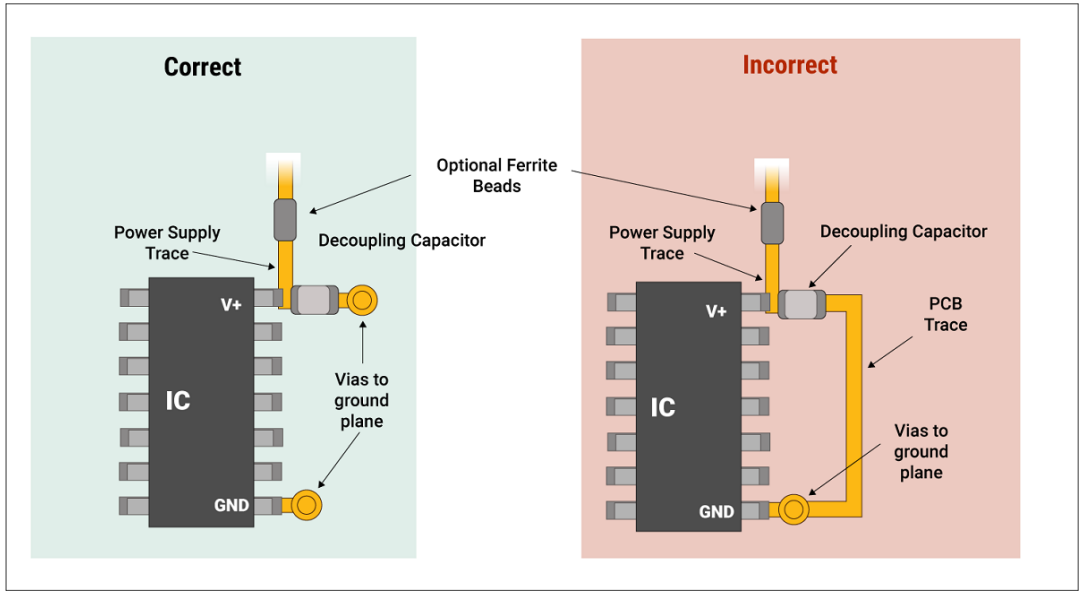
$$Z = \frac{60}{\sqrt{\varepsilon_r}} \ln\left(\frac{1.9B}{0.8W + T}\right)$$

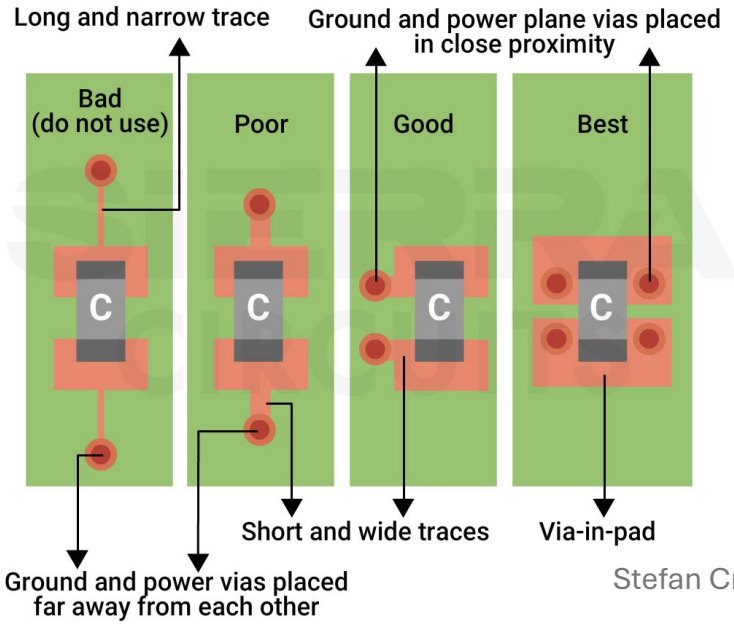


The characteristic impedance of a line is the impedance seen by a wave front traveling down the line

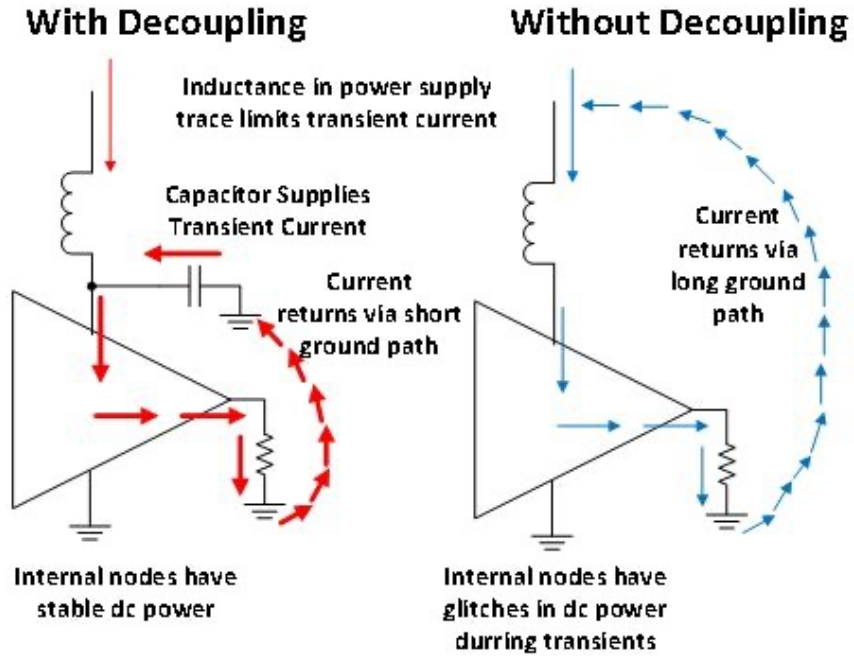
The trace width, dielectric thickness, and dielectric material are the only factors that set the impedance.

Decoupling capacitors

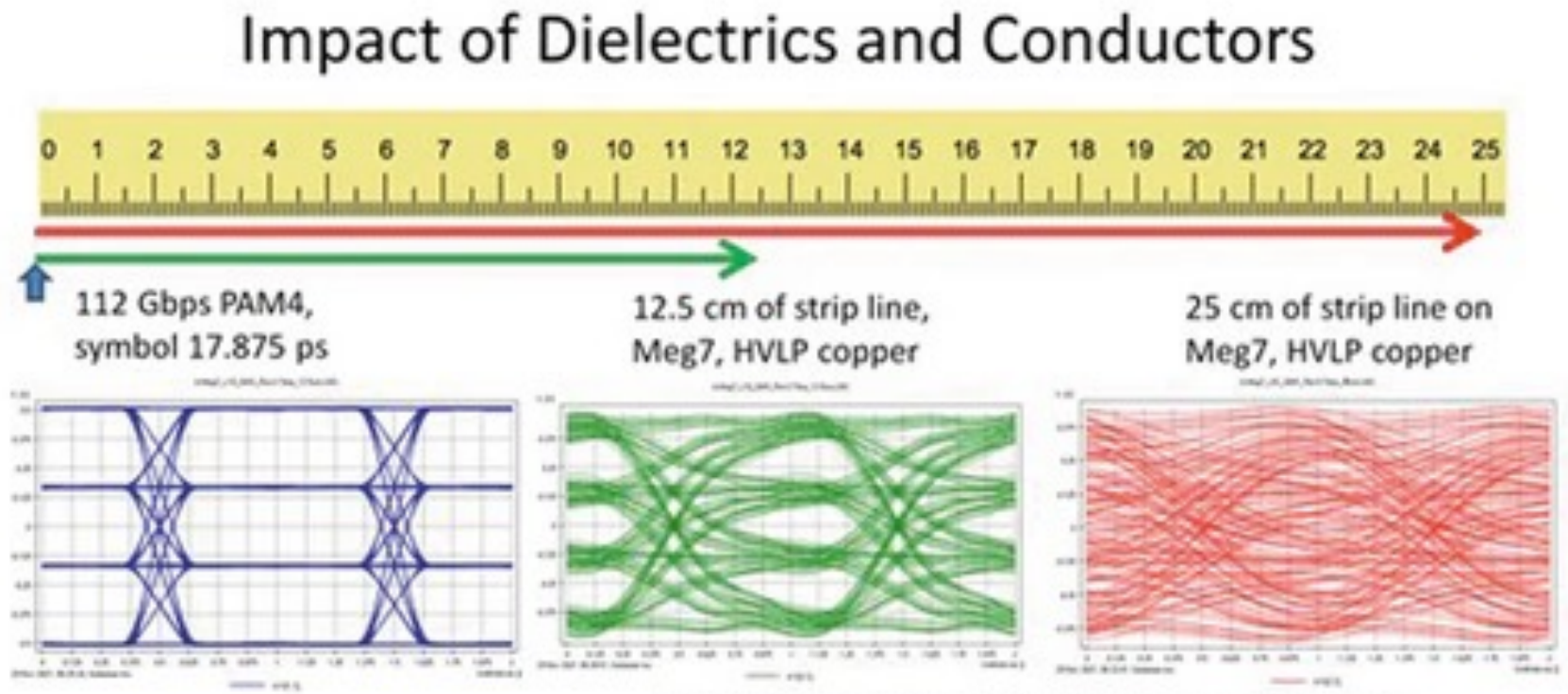




- **Provide a small storage of power for briefs transients of current**
- **A low impedance connection for high frequency noise to ground**



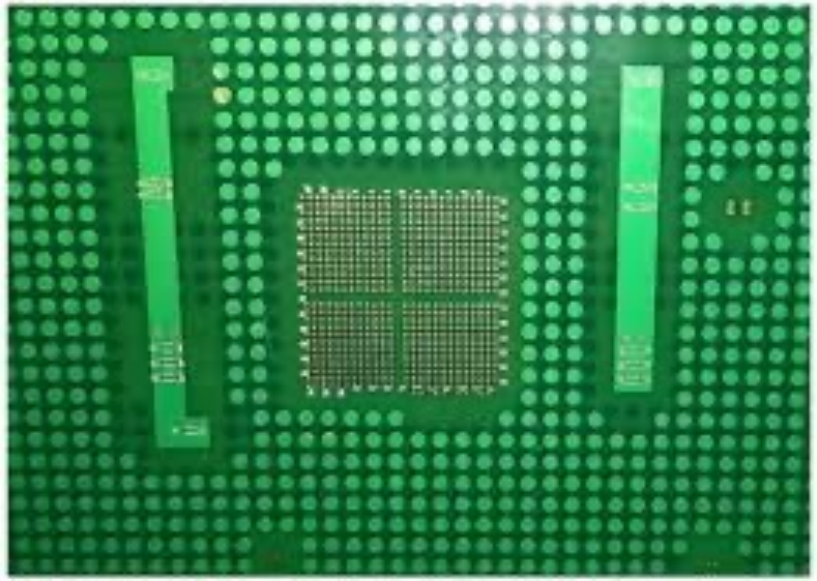
Attenuation and trace length

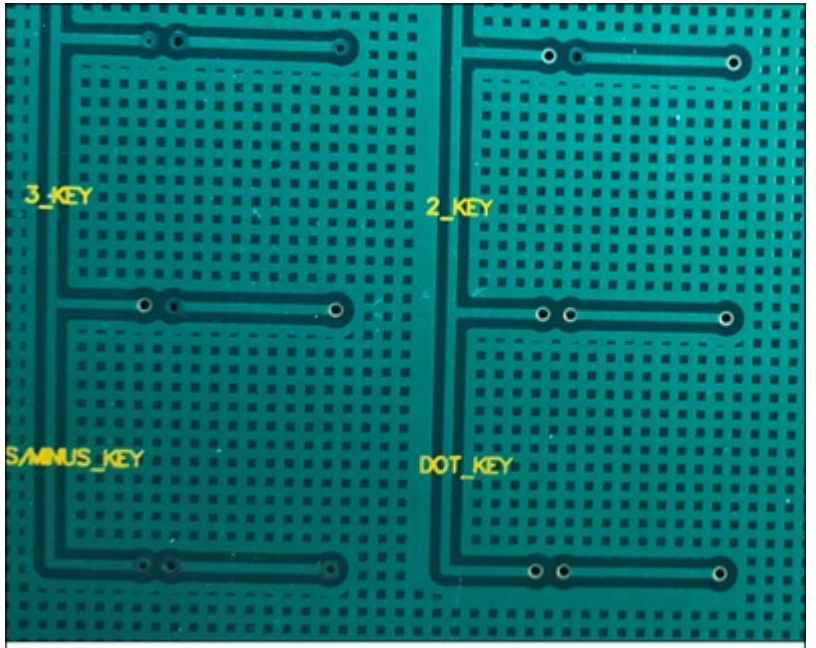


PAM4 -> 2 bit per symbol

Double data rate w.r.t NRZ at the same bandwidth.
More complexity and more sensible to noise.

Copper thieving

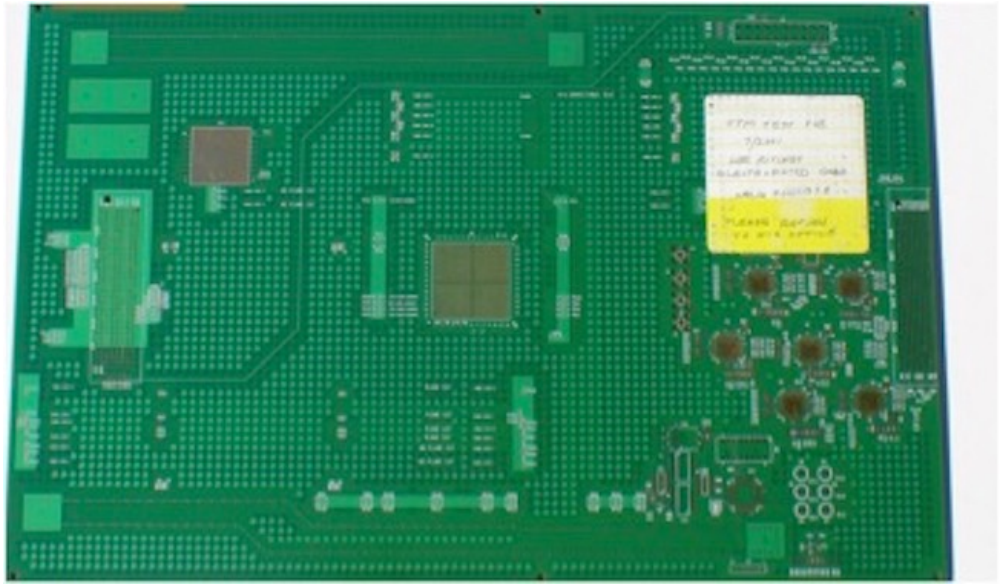




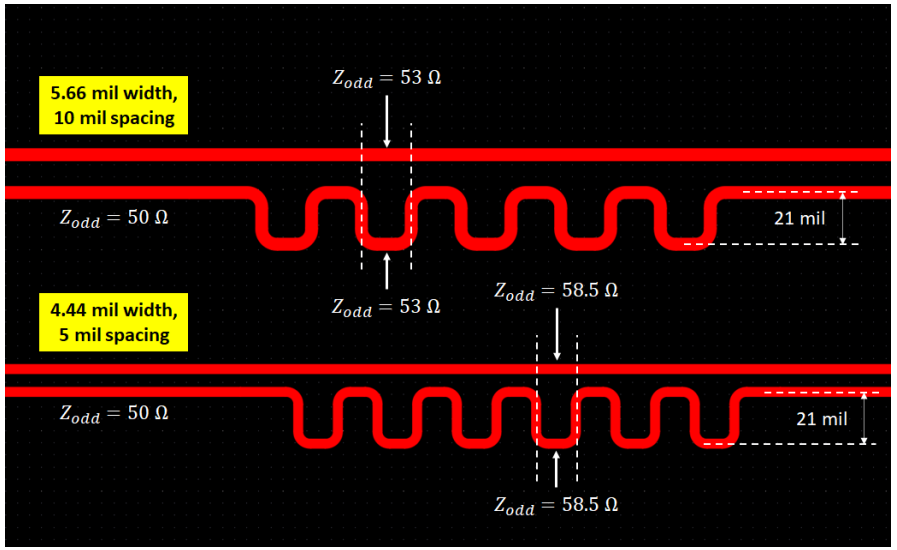
Copper Thieving Pattern

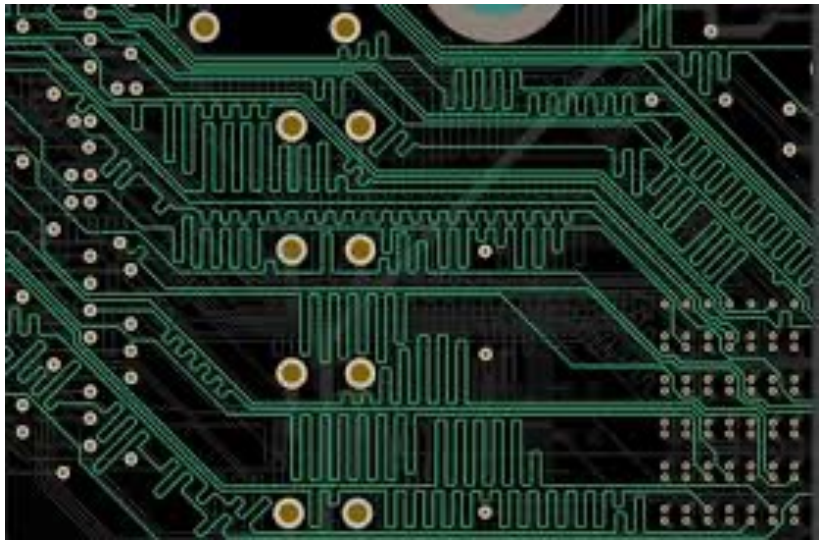
Used to even out the copper distribution across a copper layer. The added copper isn't connected to any nets on the board and should not affect the functionality.

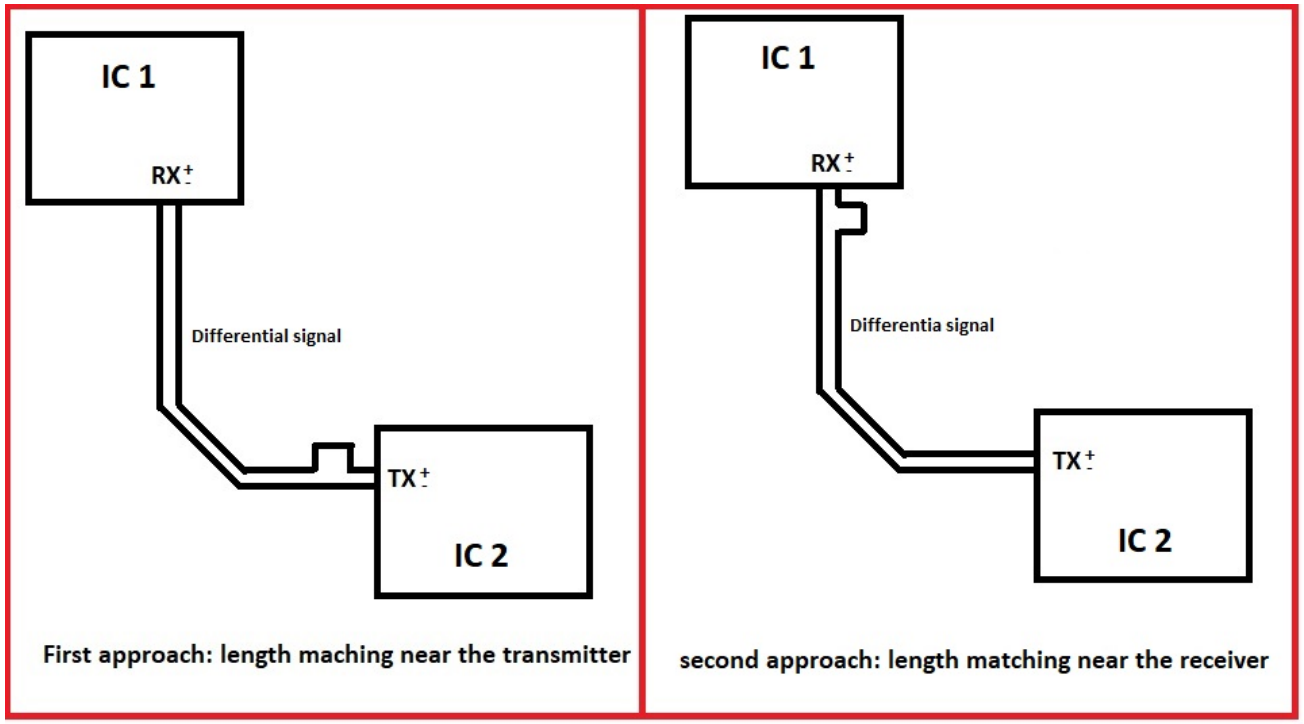
Care will be taken to ensure thieving will not interfere with any defined controlled impedance or RF signal lines the design may contain.

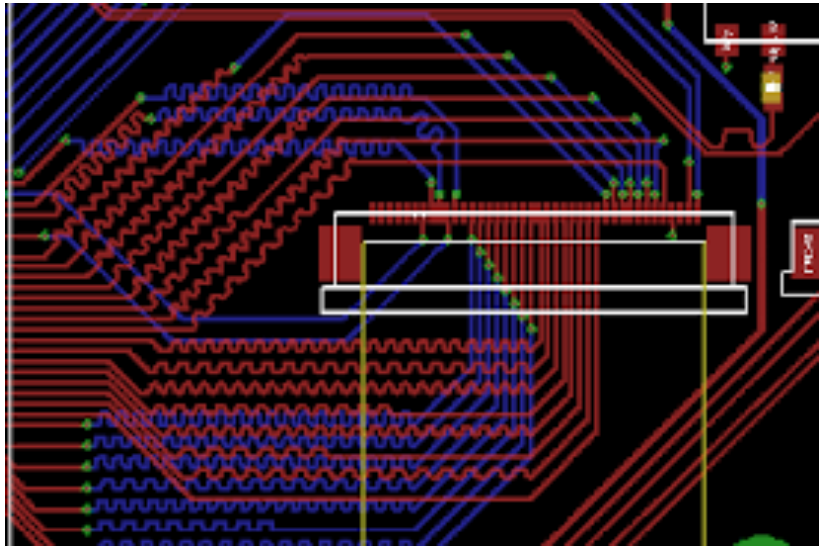


Length tuning

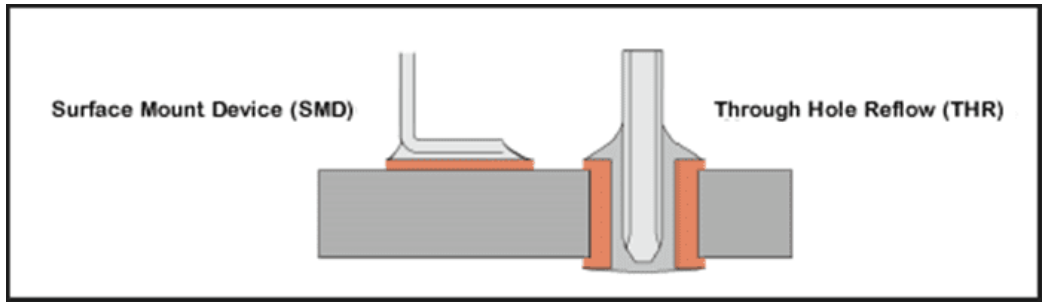








Components, SMD and THR

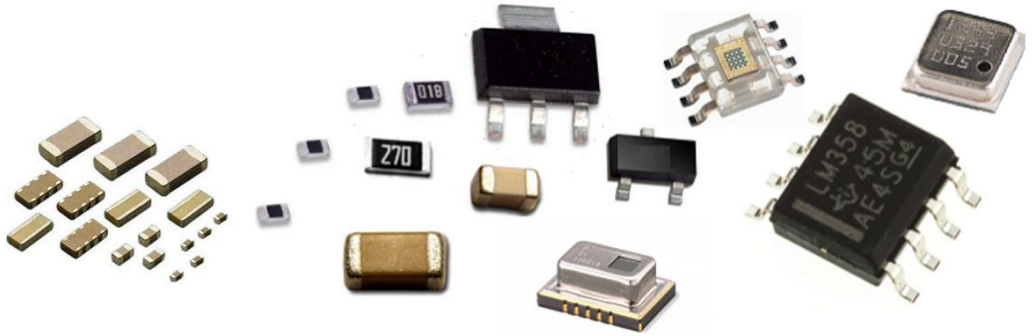


Surface Mount Device

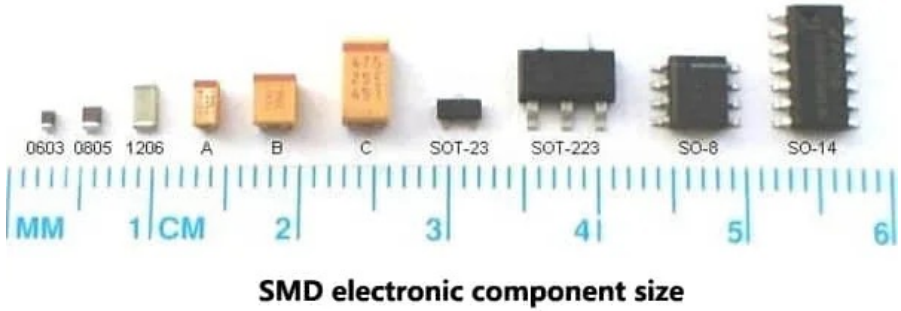
- Better for automatic assembly
- Generally lower parasitics, less exposed metal, better for high frequency applications
- Can have a much smaller footprint

Through Hole Reflow

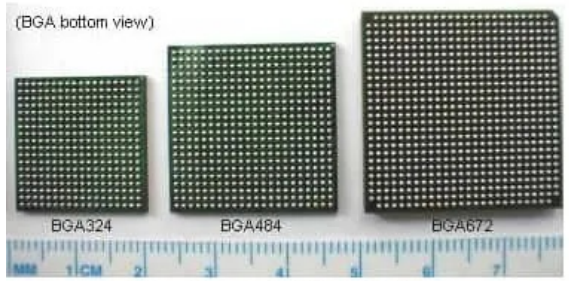
- Connection on all PCB layers
- Usable with breadboard and easier to hand solder
- Stronger mechanical connection



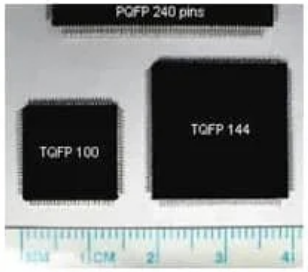
Footprint example for SMF and THR



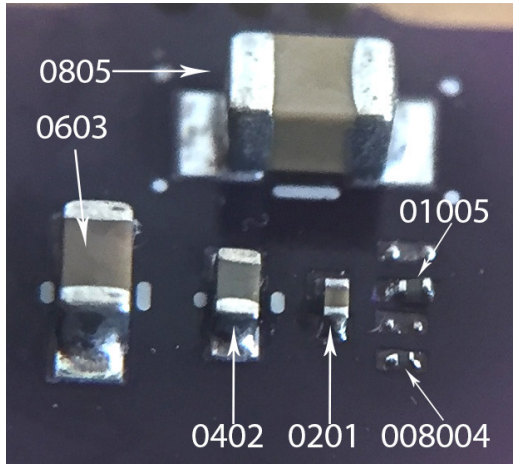
SMD electronic component size

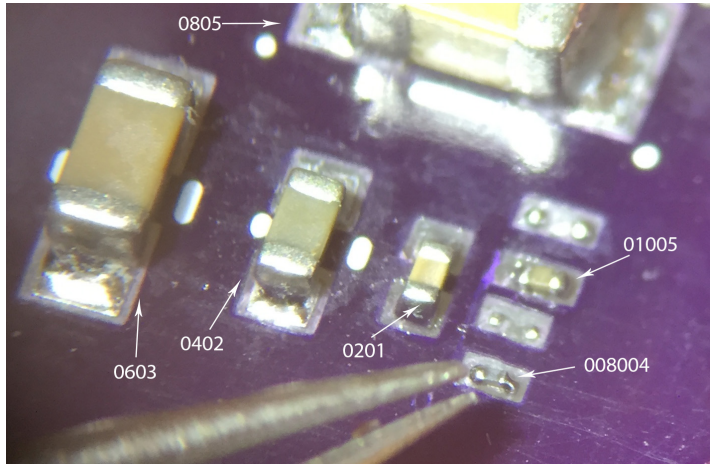


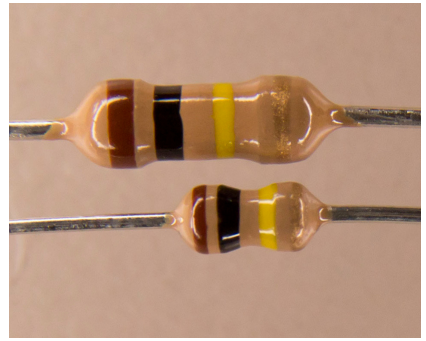
The Size of SMD BGA

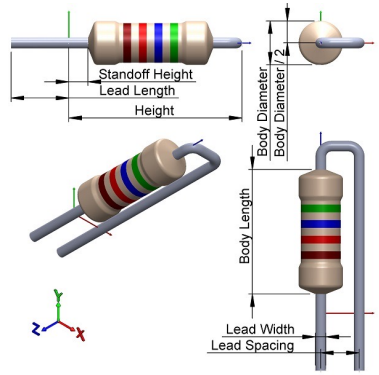


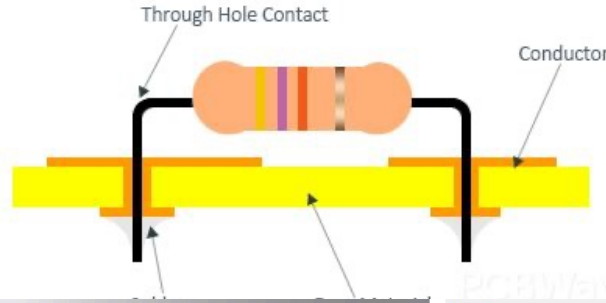
The Size of QFP

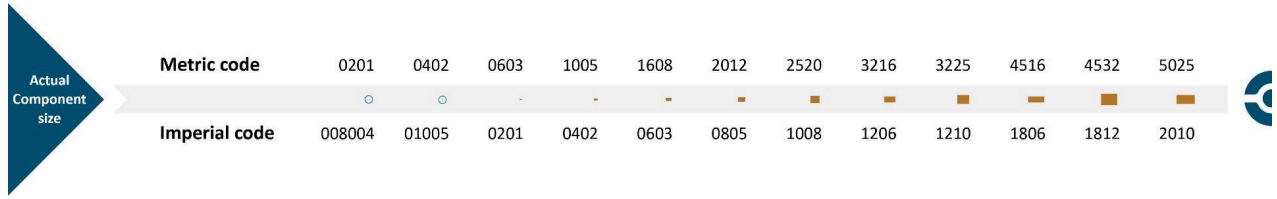


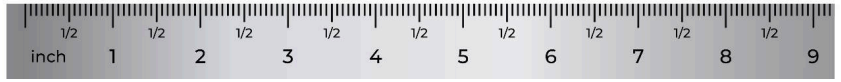




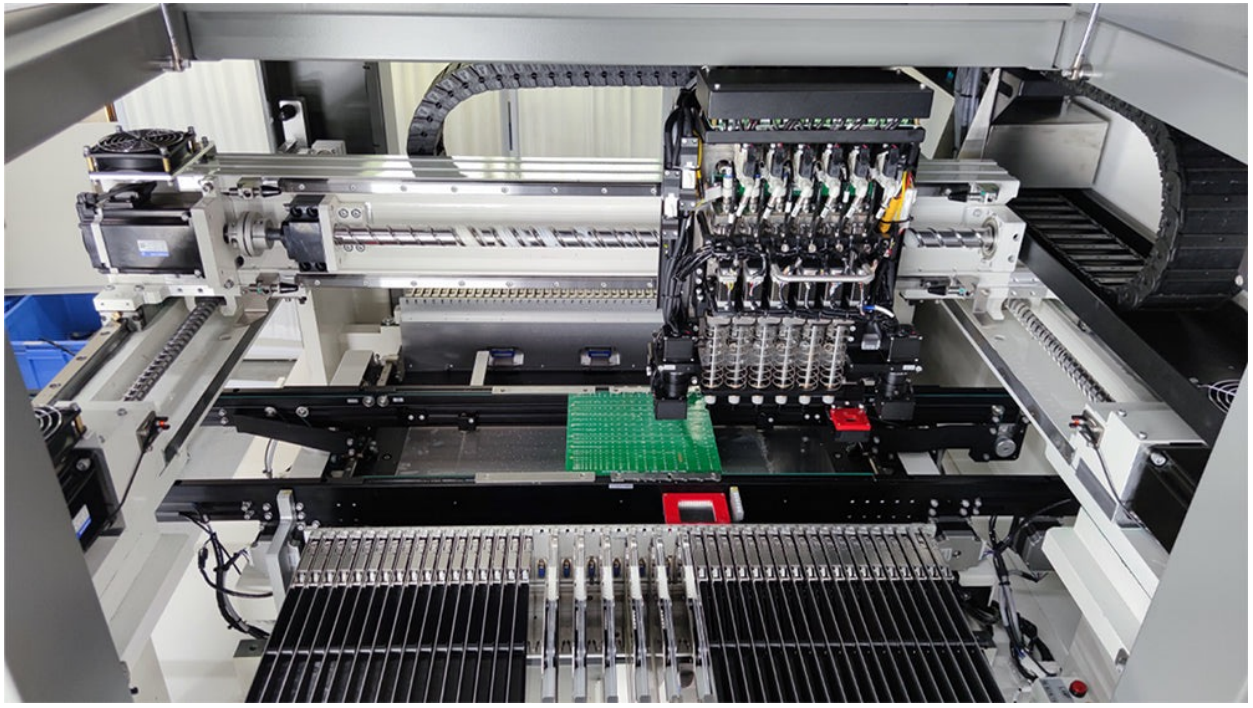


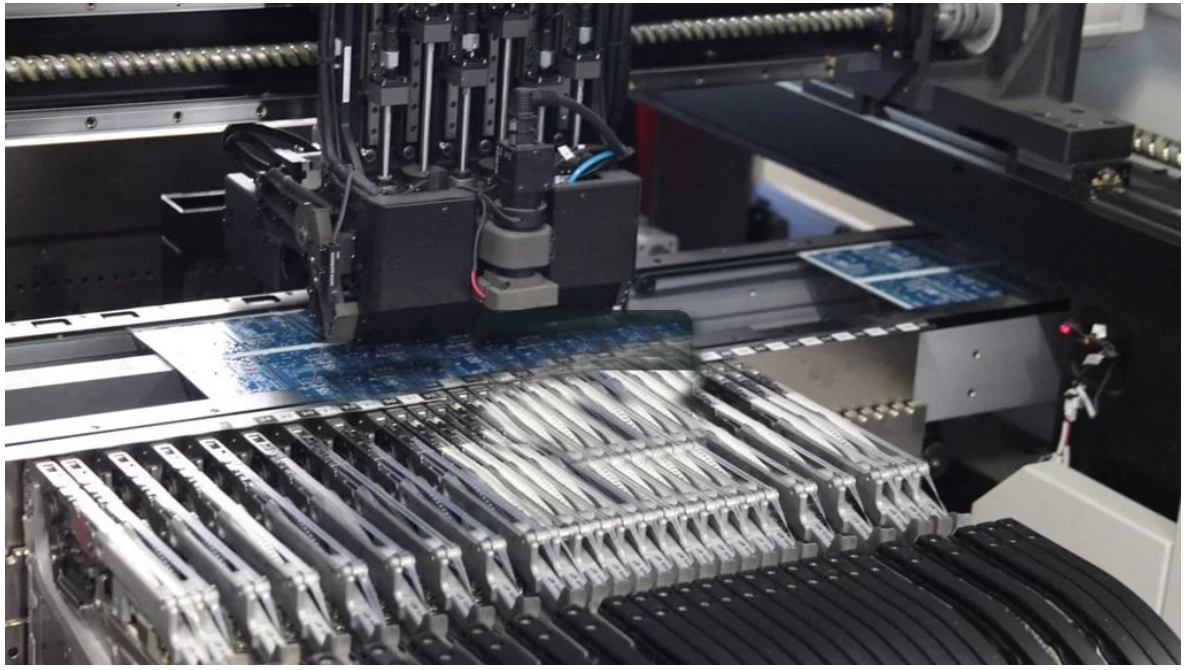




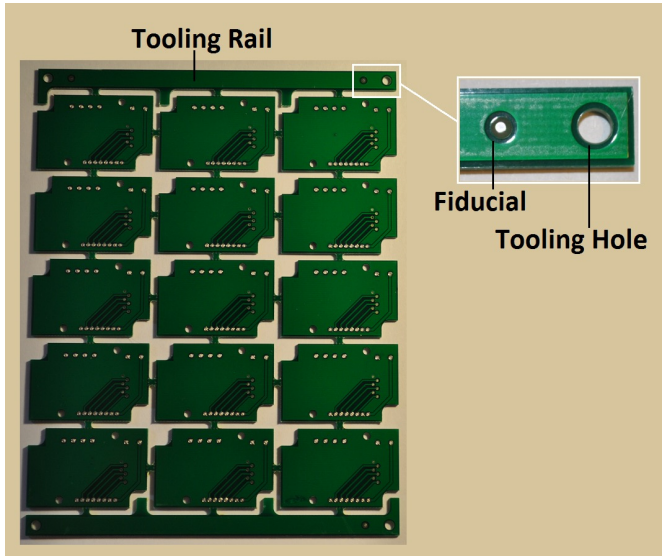


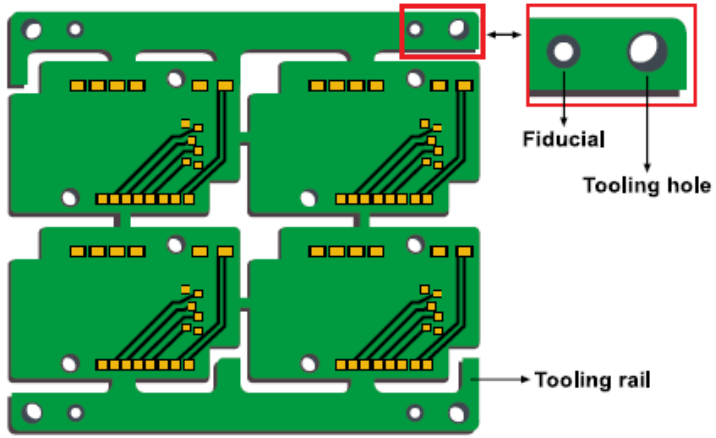
Pick and Place machine

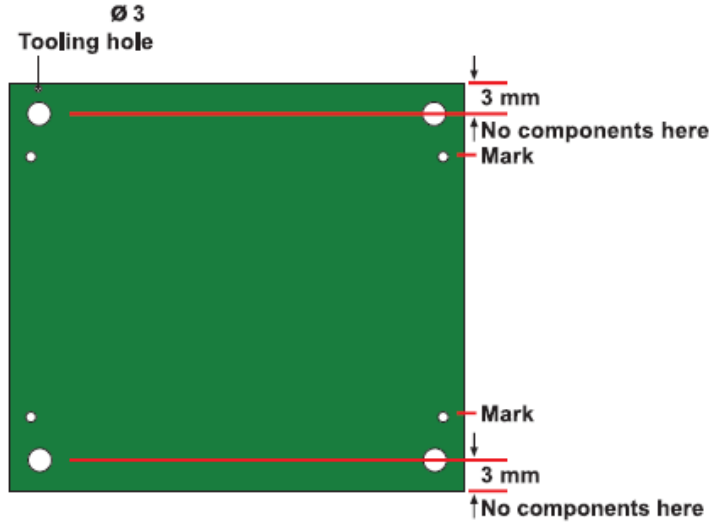


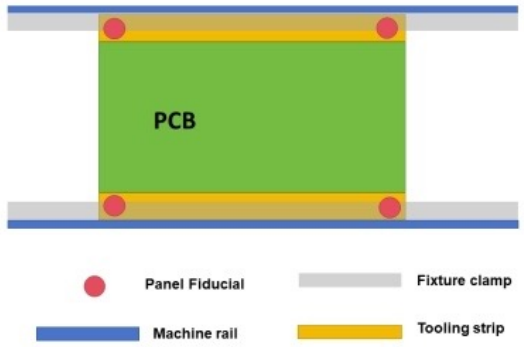


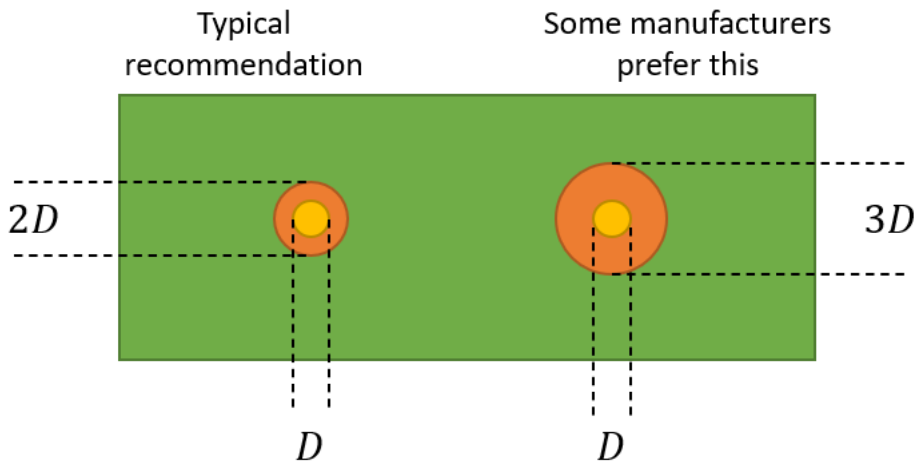
Fiducial and tooling holes



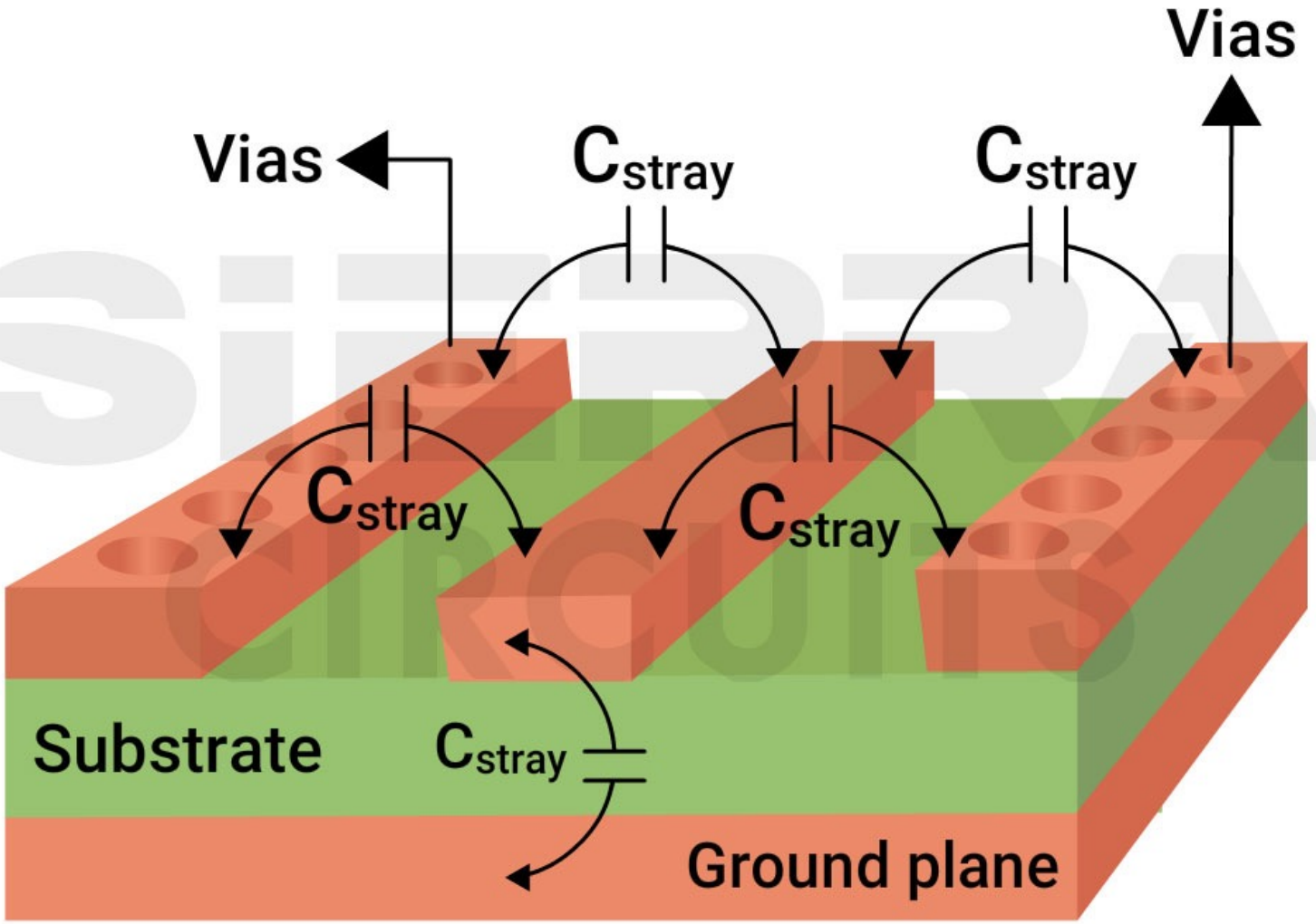




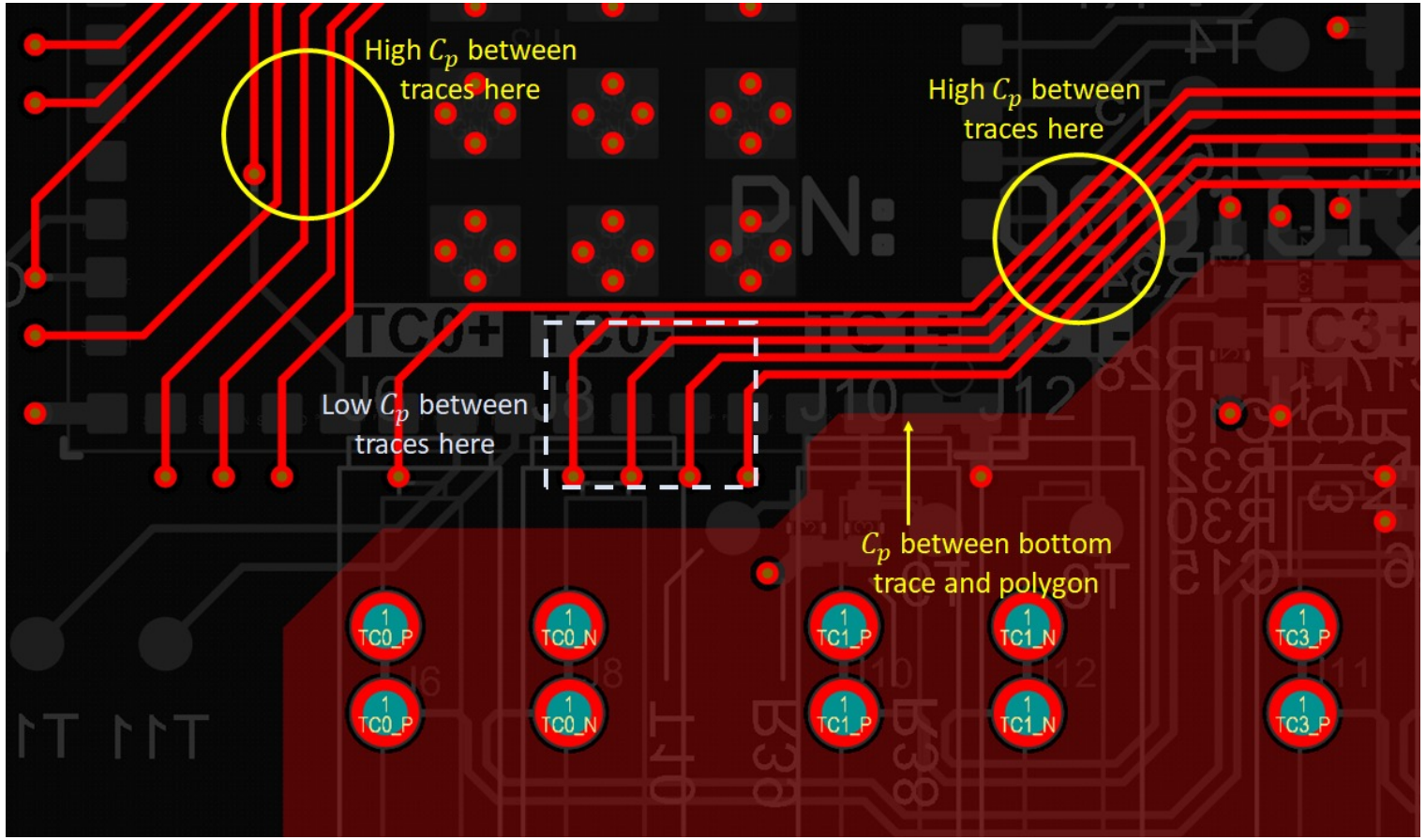


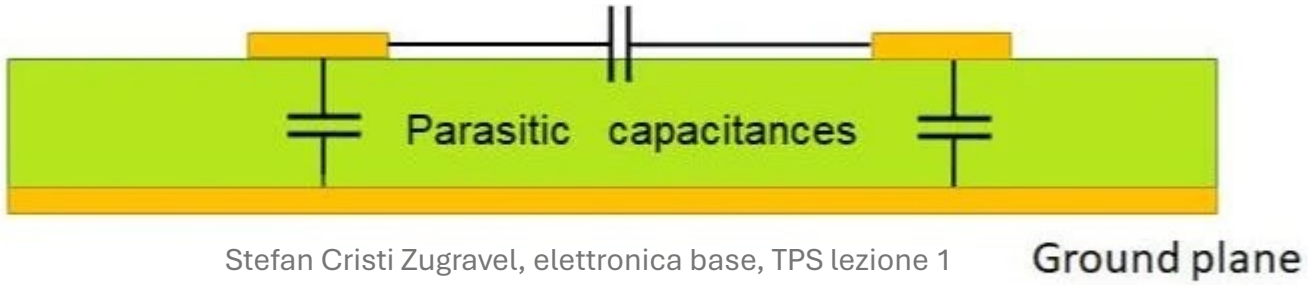


PCB parasitics/1

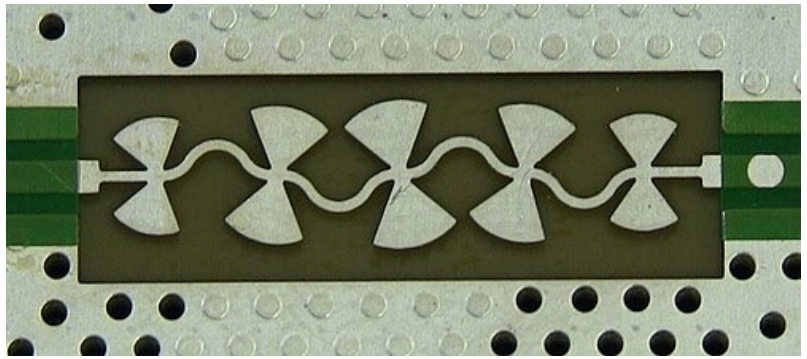


PCB parasitics/2

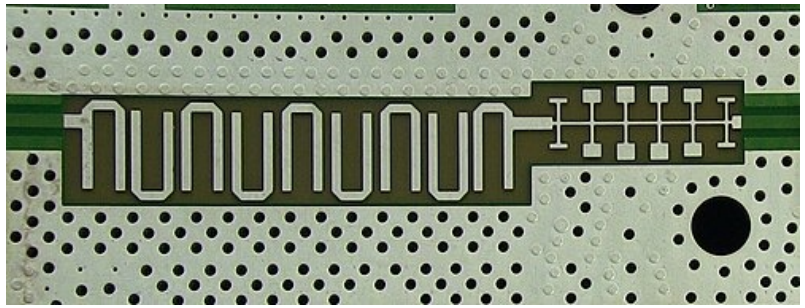




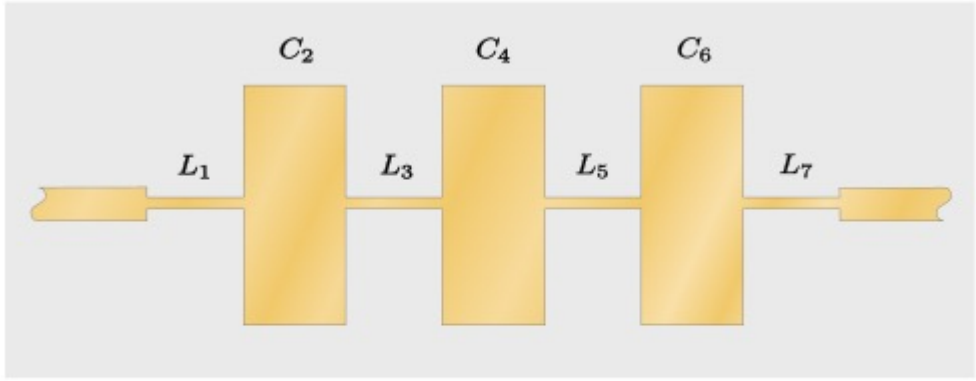
Distributed-element filter

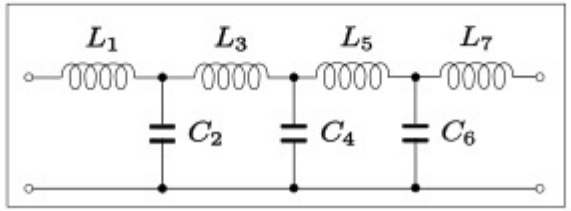


A microstrip low pass filter implemented with bowtie stubs inside a 20 GHz Agilent N9344C spectrum analyzer



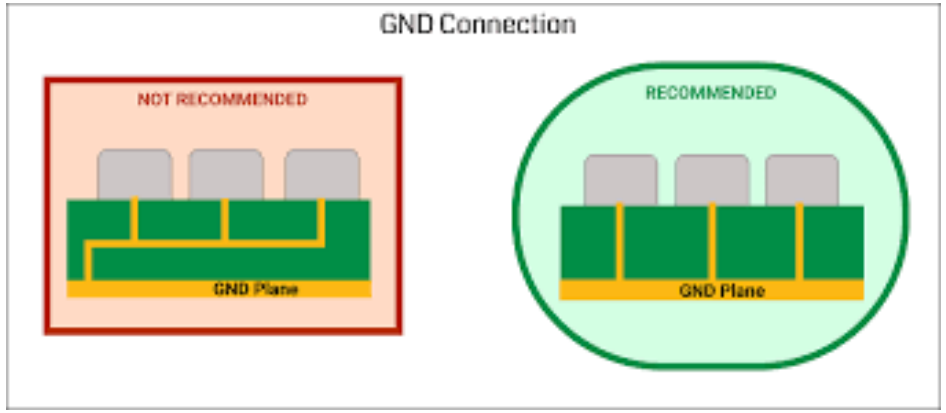
A microstrip hairpin filter followed by a low pass stub filter on a PCB in a 20GHz Agilent N9344C spectrum analyzer

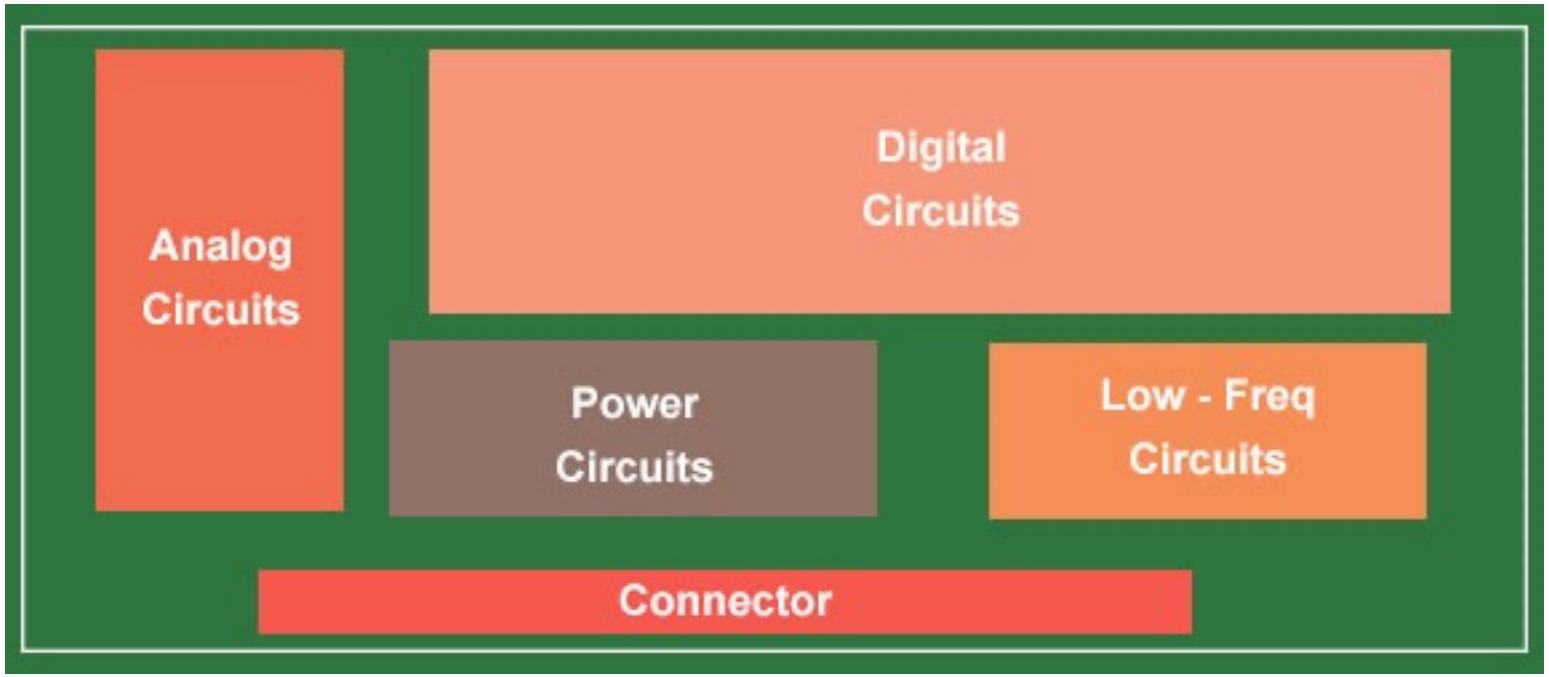


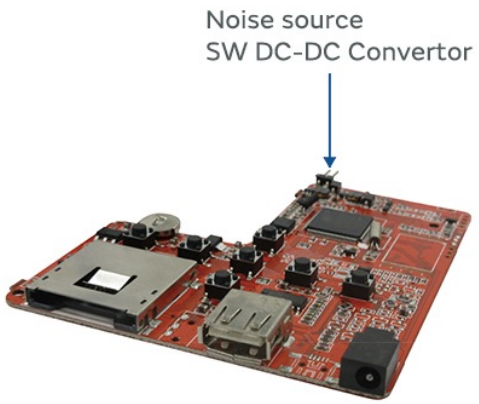


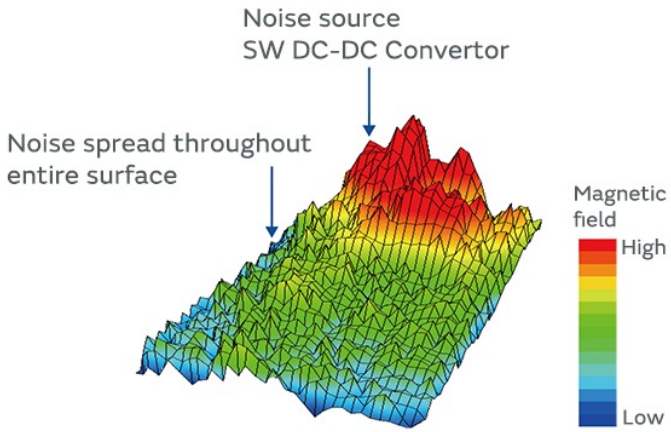
Stepped-impedance low-pass filter formed from alternate high and low impedance sections of line

PCB layouts and noise considerations



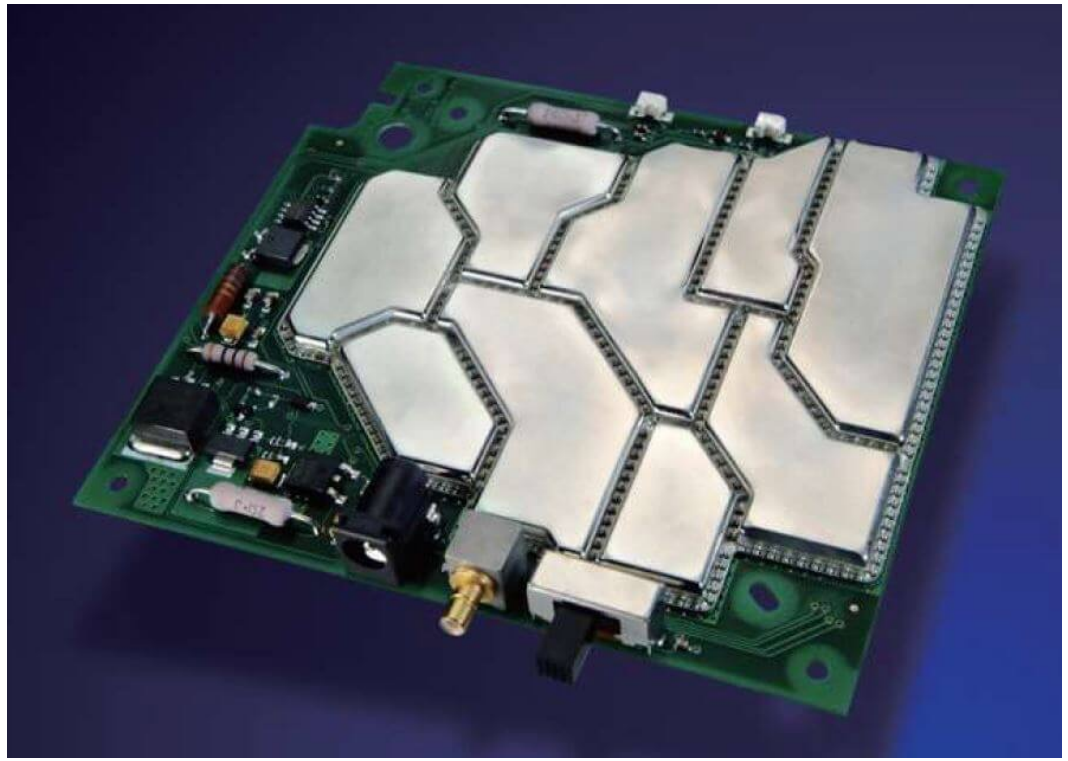


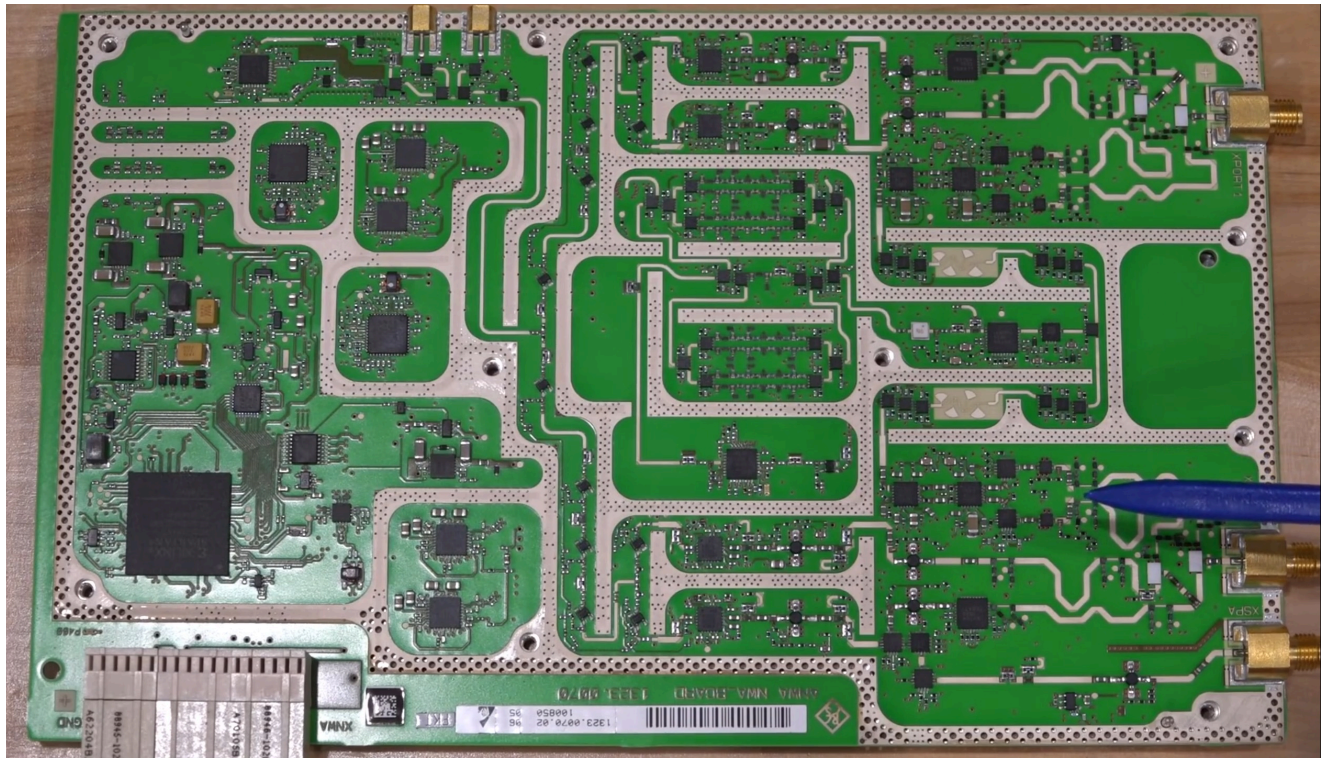




Caging

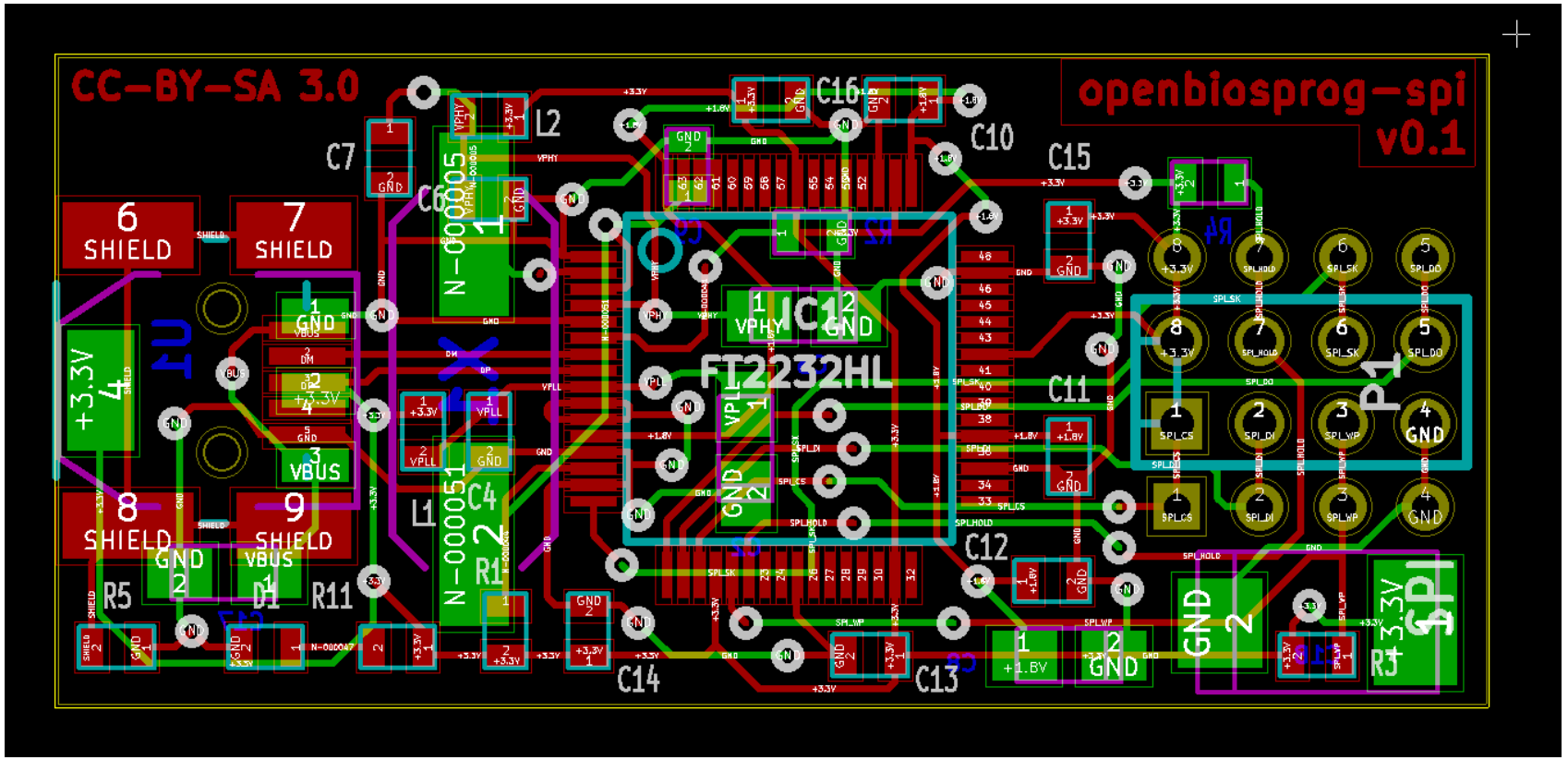
- Shielding cans or shielding cages are typically made of aluminum and are mounted on the PCB around specific groups of components.
- These small metal shields are then connected to ground, forming a Faraday cage around the components.
- These mechanical components are most often placed on a PCB as an EMI solution, either to help with passing susceptibility testing, radiated emissions testing, or both.





GERBER

“The Gerber format is an open vector format for printed circuit board (PCB) designs.”



Bill Of Materials

- **Reference Designator(s); i.e. “R1, R2, R3”**
- **Description; i.e. “10k 5% 0.125W Resistor”**
- *Manufacturer; i.e. “Vishay Dale”*
- *Manufacturers Part Number; it’s useful to link this to the datasheet. | i.e. “C0402C100J5GAC”.*
- *Distributor; once specified the distributor can be included. This will also help in tracking cost and leadtime as they may vary from one distributor to another. | i.e. “AVNet”*
- *Distributor Part Number; some PCBA manufacturing services require the customer to specify the distributor and orderable part number to save them time. It can take a significant amount of effort to shop around as to where to buy. | i.e. “C0402C100J5GACTU”*
- **Package; component size and shape | i.e. “SMD 0402”**
- **Qty; of each part per unit of product | i.e. “3”**

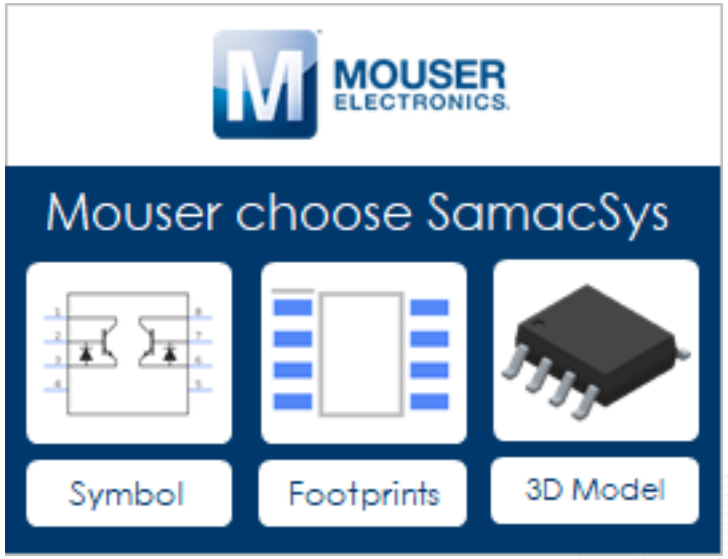
	A	B	C	D	E	H	I	J
3	Item #	Qty	Ref Des	Manufacturer	Manf part num	Description	Package	Type
4	1	1	U1	Atmel	ATMEGA328P-AUR	MCU	44-TQFP	SMT
5	2	1	U2	FTDI	FT231XS-R	USB to UART	20-SSOP	SMT
6	3	1	U2	Freescale	MMA8452QR1	Accelerometer	QFN16	SMT
7	4	1	Q1	Fairchild	KDT00030TR	Phototransistor	0603	SMT
8	5	1	Y1	Epson	NX5032GC-16MHZ-STD-CSK-6	16 MHz		SMT
9	6	3	D1, D2, D3	Osram	LY R976-PS-36	LED Yellow	0805	SMT
10	7	1	D4	Diodes, Inc	B0520LW-7-F	Schottky	0805	SMT
11	8	2	C1, C2	Yageo	CC0805DRNPO9BN8R0	8pf, 16V, 10%	0805	SMT
12	9	2	C3, C4	Yageo	CC0805KKX7R7BB105	1.0uf, 16V	0805	SMT
13	10	1	C5	Yageo	CC0805KRX7R9BB103	.01uf, 16V	0805	SMT
14	11	6	C6, C7, C8, C9, C13, C14	Samsung	CL21B104KOANNNC	.1uf, 16V	0805	SMT
15	12	2	C10, C11	Samsung	CL21C470JBANNNC	47pf, 16V	0805	SMT
16	13	1	C12	AVX	F931A476MAA	47uf, 16V		SMT
17	14	1	C26	Taiyo	LMK212BJ475KD-T	4.7uf, 10V	0805	SMT
18	15	2	L1, L2	Bourns	MH2029-300Y	30ohm ferrite	0805	SMT
19	16	1	R1	Stackpole	RMCF0805JT1M00	1M	0805	SMT
20	17	2	R2, R3	Stackpole	RMCF0805JT22R0	22 ohms	0805	SMT
21	18	3	R4, R5, R6	Stackpole	RMCF0805FT680R	680 ohm	0805	SMT
22	19	2	R7, R8	Rohm	MCR10ERTF2201	2.2K	0805	SMT
23	20	2	R9, R10	Stackpole	RMCF0805FT10K0	10K	0805	SMT
24	21	1	R11	Stackpole	RMCF0805JT4K70	4.7K	0805	SMT
25	22		S1, S2	C&K	PTS645SM43SMTR92 LFS	Momentary push		DNS
26	23	2	S3, S4	C&K	PTS645SM43SMTR92 LFS	Momentary push		SMT
27	24		J1	FCI	10118192-0001LF	USB microB		DNS
28	25	1	J2	FCI	10118192-0001LF	USB microB		SMT
29	26	1	B1	Linx	BC501SM	12mm Coin cell		SMT

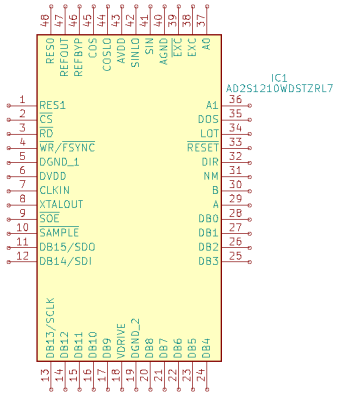
POS

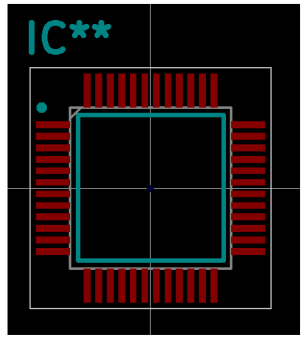
- Designator - Component Reference Designator (e.g. C1, L2, R3)
- Mid X/Mid Y - The X/Y coordinate of the component centroid. Generally in mm
- Layer - Top / Bottom, the board side where the component should be placed.
- Rotation - The rotation of the component given in degrees.
- Recommended File Format: .csv, .xls and .xlsx.

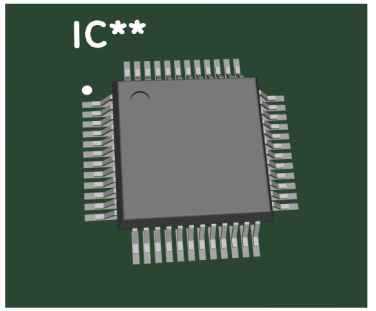
```
### Module positions - created on Monday, January 02, 2023 at 10:32:14 AM ###
### Printed by Pcbnew version (6.0.0-0)
## Unit = mm, Angle = deg.
## Side : bottom
# Ref     Val                  Package                          PosX       PosY       Rot  Side
BT1       Battery_Cell         CR2032-LCSC-C964760           -105.9500   -94.1500  135.0000  bottom
C1        0.1uF                C_0805                         -93.3000   -94.8000  135.0000  bottom
D1        Schottky             D_SOD-323F                     -97.7000  -110.9000 -135.0000  bottom
D2        LED_RGBA             LED_Reverse_XZMDKCBDDG45S-9    -96.9000  -100.3000  125.0000  bottom
D4        LED_RGBA             LED_Reverse_XZMDKCBDDG45S-9    -92.9000  -102.8000  125.0000  bottom
D6        LED_RGBA             LED_Reverse_XZMDKCBDDG45S-9    -89.0000  -105.6000  125.0000  bottom
D8        LED_RGBA             LED_Reverse_XZMDKCBDDG45S-9    -85.0000  -108.5000  125.0000  bottom
J1        USB_C_Receptacle_USB2.0  USB_C_Power_Only_C2927027  -100.1000  -119.2000    0.0000  bottom
R1        5.1_kΩ               R_0805                        -101.4000  -108.4000 -135.0000  bottom
R2        5.1_kΩ               R_0805                        -103.4000  -111.1000  -90.0000  bottom
R3        100_Ω                R_0805                         -83.1000  -101.7000  135.0000  bottom
R4        100_Ω                R_0805                         -92.0000  -111.1000  -55.0000  bottom
R5        100_Ω                R_0805                         -90.4000  -112.8000  -55.0000  bottom
SW1       SW_SPDT              Switch,_Side_-_LCSC_C963207   -110.1491  -110.6595   50.0000  bottom
SW2       SW_Push              500-0001-TS1187ACGB-(2_Pin)   -115.3000   -82.9000   45.0000  bottom
U1        ATtiny1614-SS        SOIC-14_3.9x8.7mm_P1.27mm      -88.6500   -90.1000  135.0000  bottom
## End
```

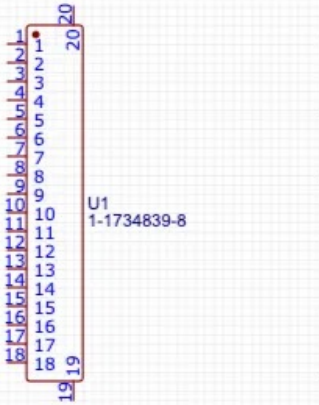
Symbol – Footprint and 3D model

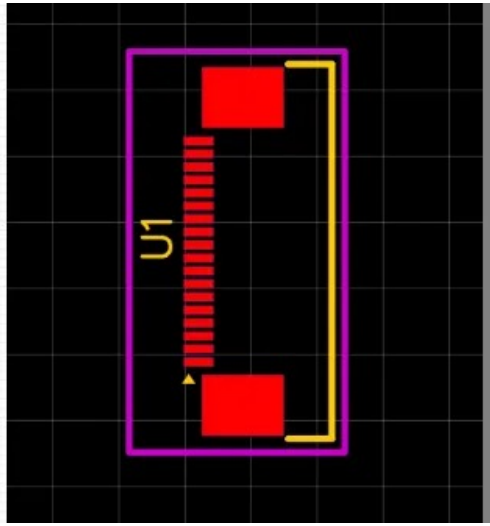


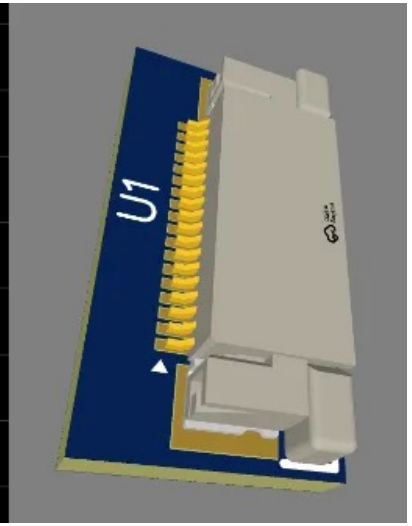






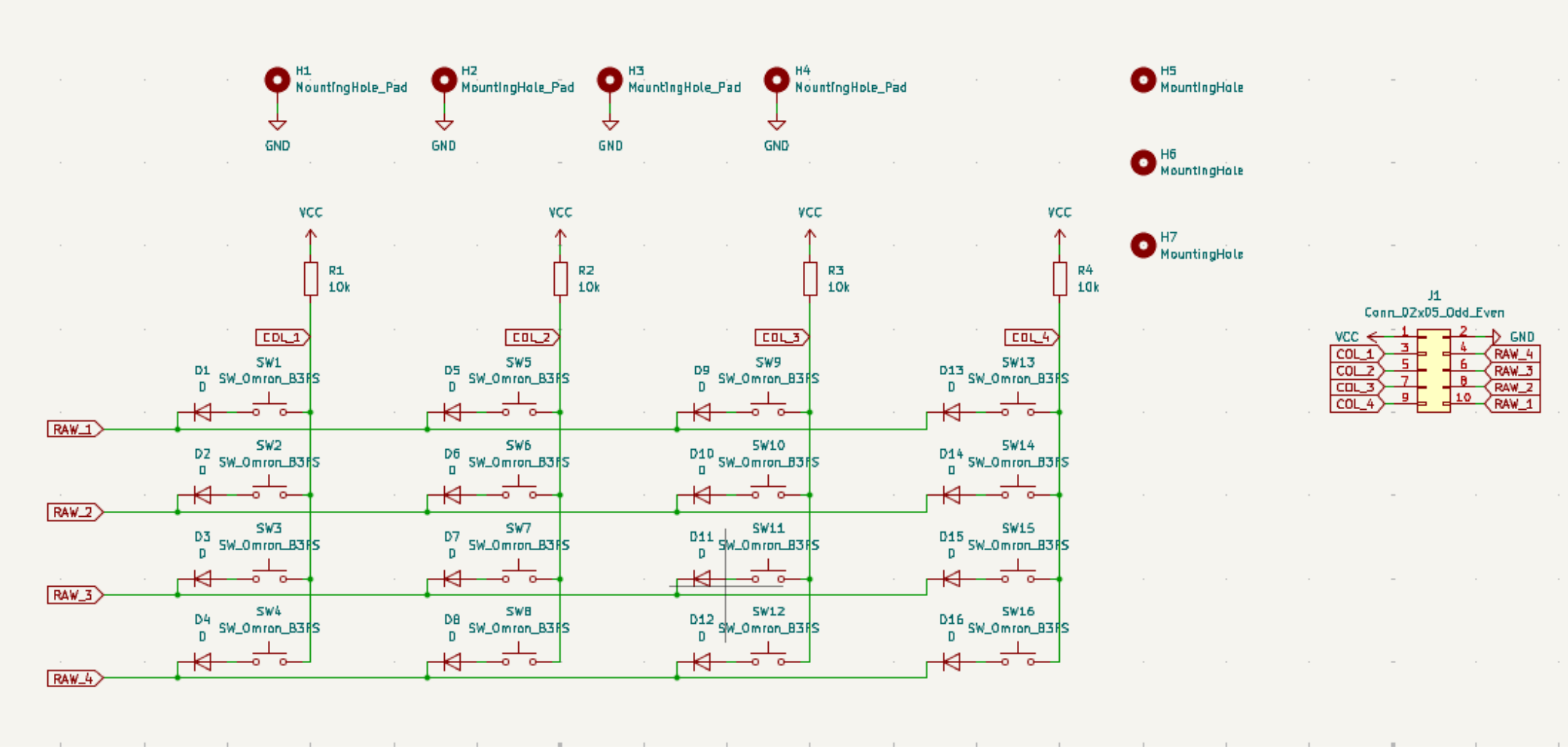






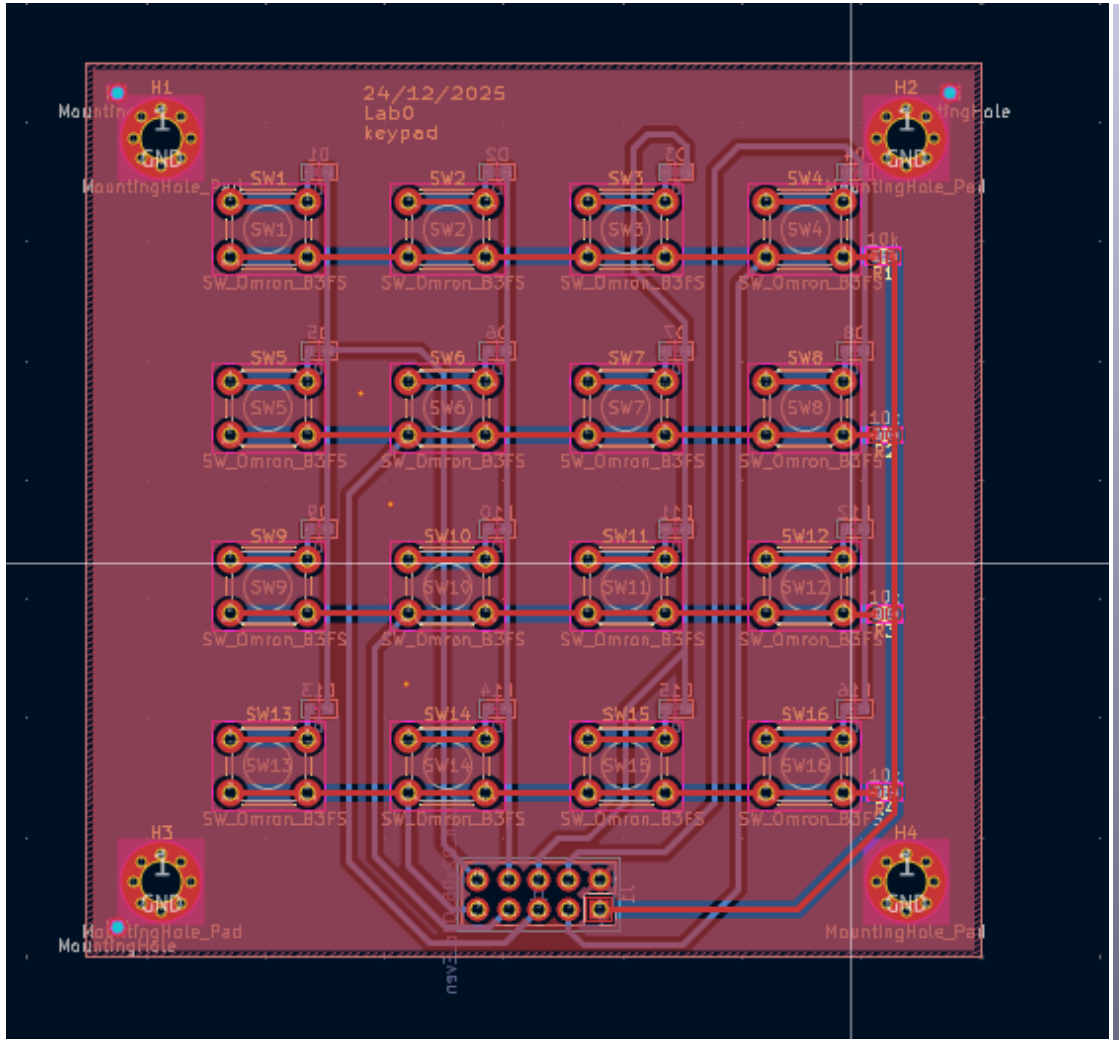
Lab 1 - 1

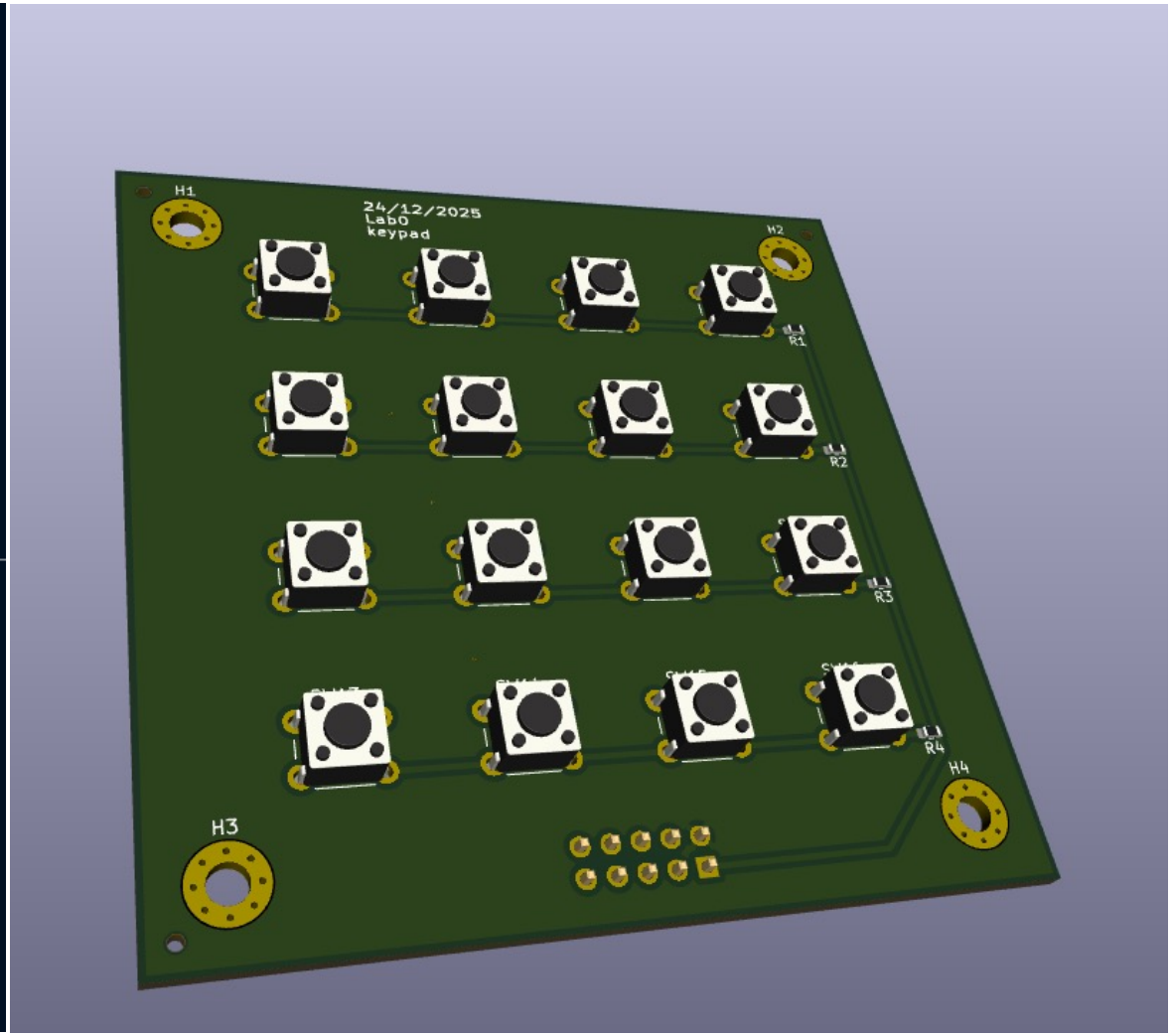
A simple keypad PCB with KiCAD - schematic



Lab 1 - 2

A simple keypad PCB with KiCAD - layout



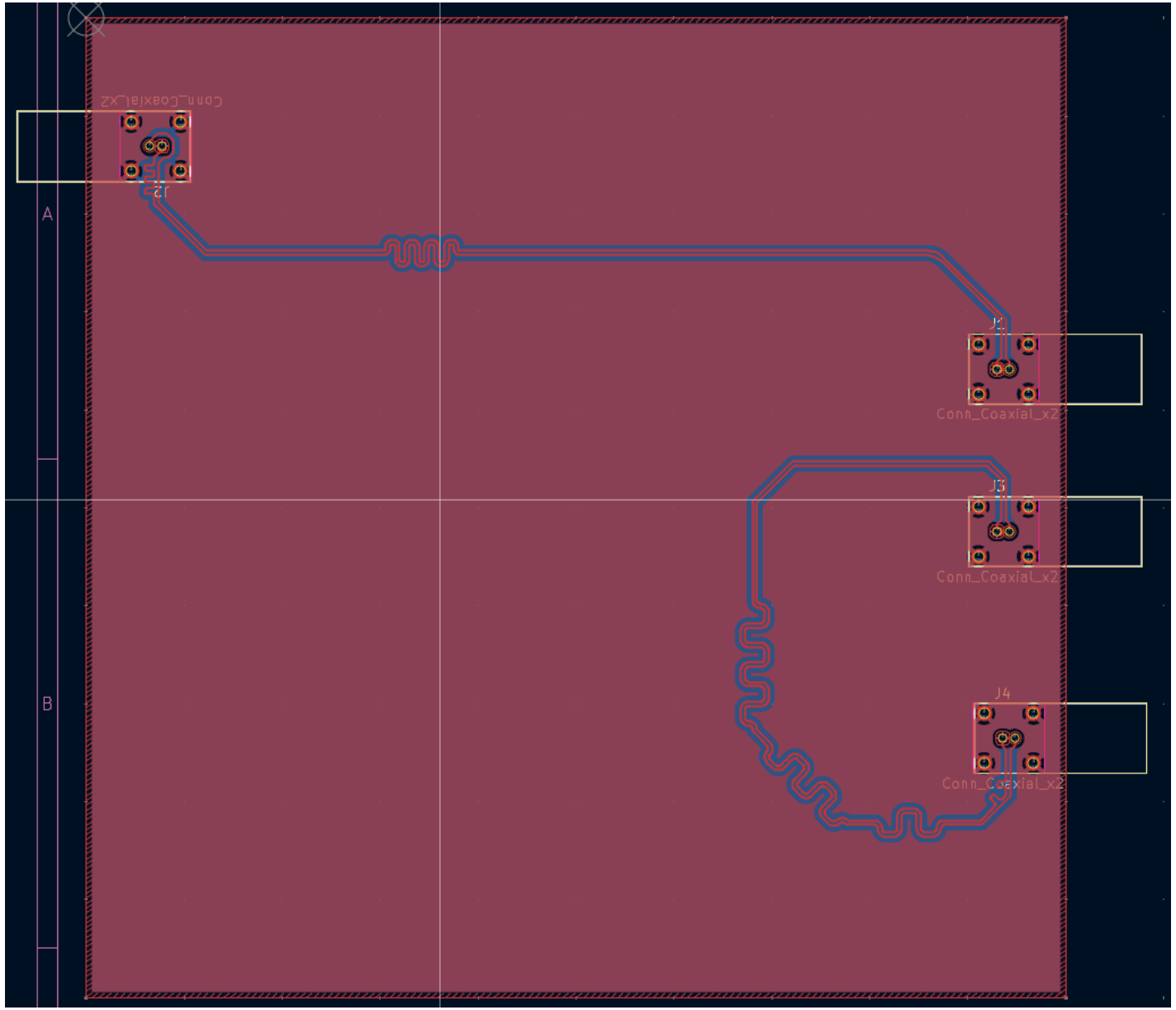


Lab 2

Length tuning and de-skewing tools in KiCAD

https://www.youtube.com/watch?v=z8syQI2KPbM

https://www.youtube.com/watch?v=KgOQaBUPiBo



Lab 3 - 1

A current monitor device with KiCAD

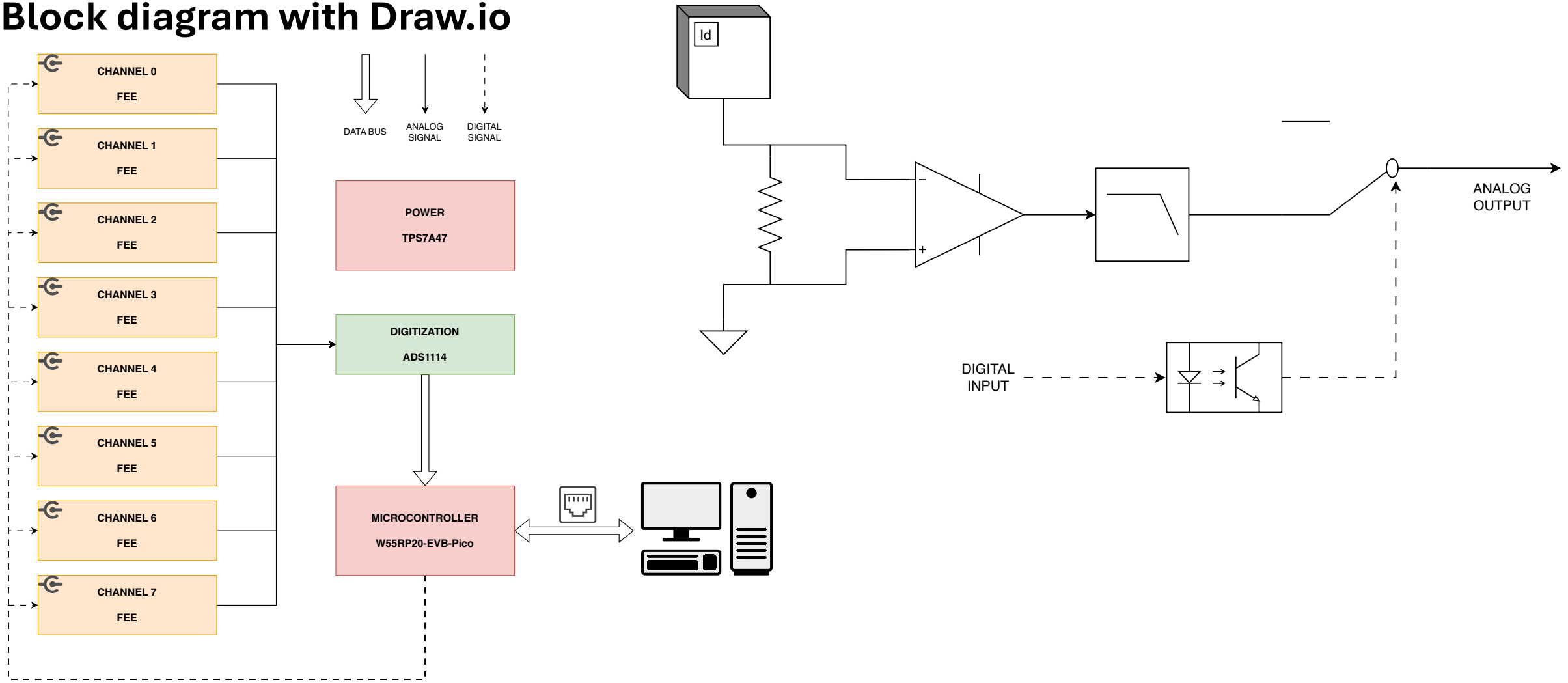
Requirements and specifications

Develop a custom module used to detect currents from 5 uA to 700 uA.

- **8 inputs connected to a 16 bit ADC (8 to 860 SPS)**
- **Shunt resistor with Current-Sense Amplifier**
- **20 KHz -3 dB FEE bandwidth low pass filtering**
- **Microcontroller data taking with of the shelf module -> why? FPGA?**
- **Low noise**
- **FEE insulated from digital section - Galvanic isolation**

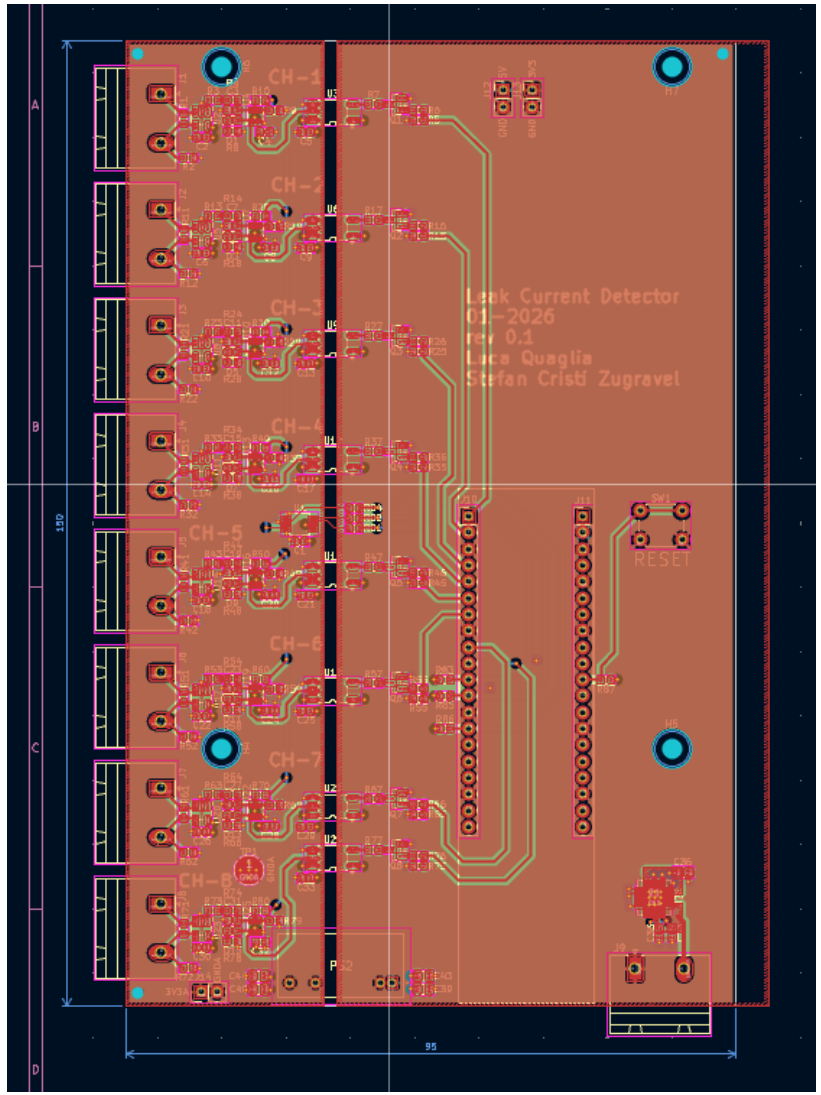
Lab 3 - 2

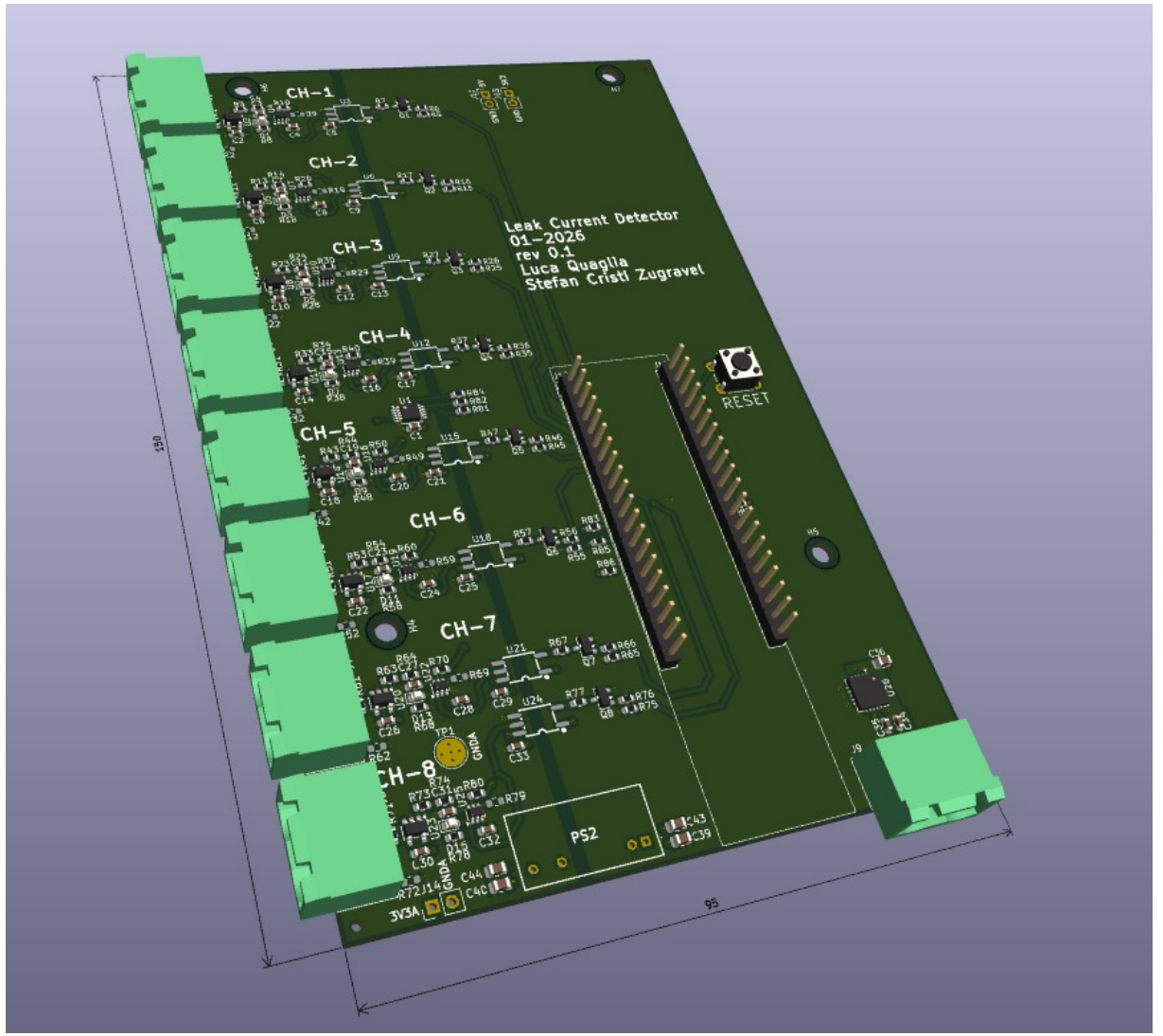
Block diagram with Draw.io



Lab 3 - 3

Layout and 3D model





Lab 4

Add an FPGA in your KiCAD design

https://www.youtube.com/watch?v=THLdycw9-Vs

https://www.youtube.com/watch?v=DKRtVnDbTgk

Suggested as last lab
-> Read the documentation!! <-

Keep calm

Lab 5

Use the LTspice simulator

lab5_0.asc Basic resistor divider (.op and .dc sweep)

lab5_1.asc Non inverting op-amp (.ac and .tran)

lab5_2.asc Inverting op-amp (.ac and .tran)

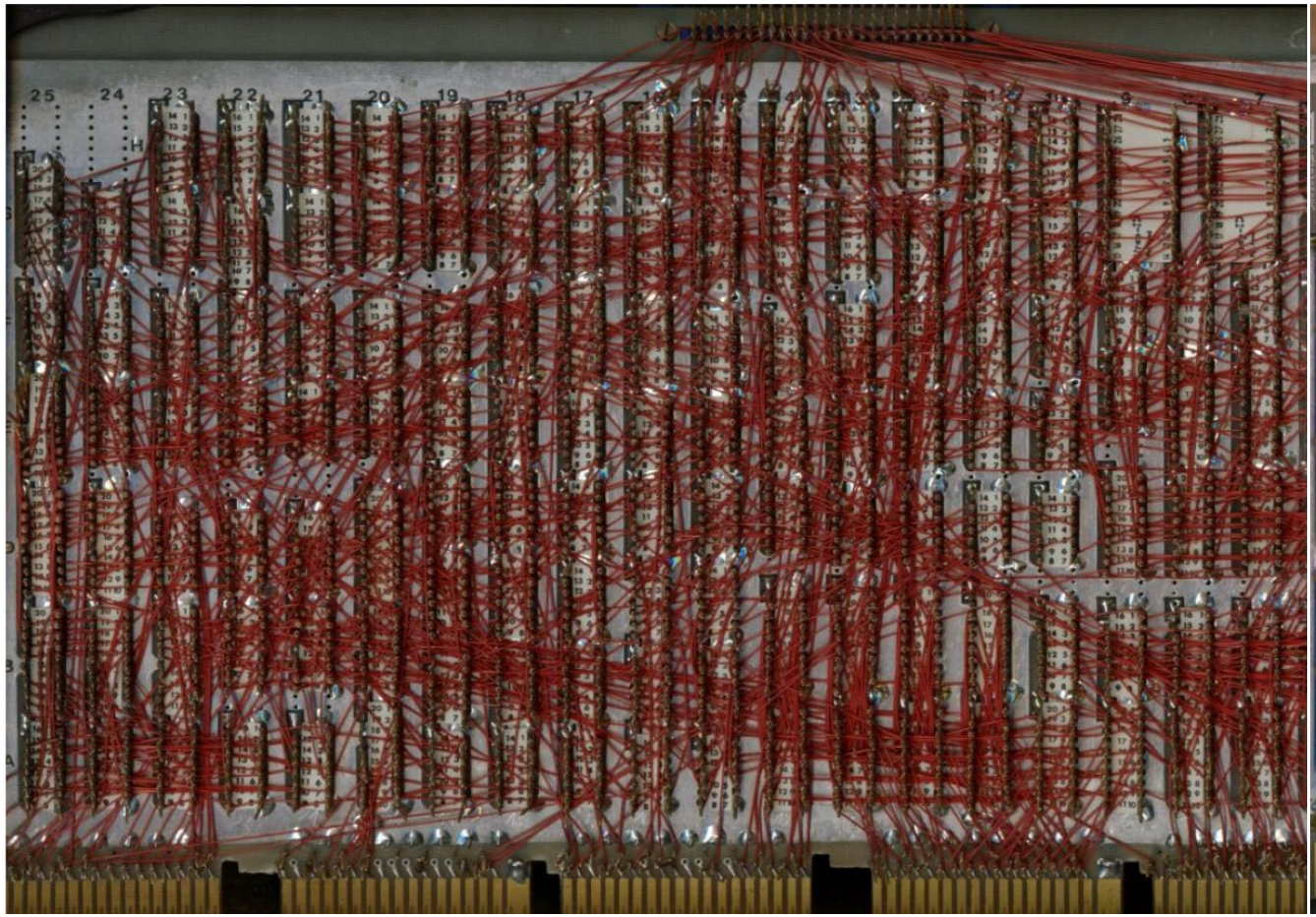
lab5_3.asc Simple MOSFET characterization

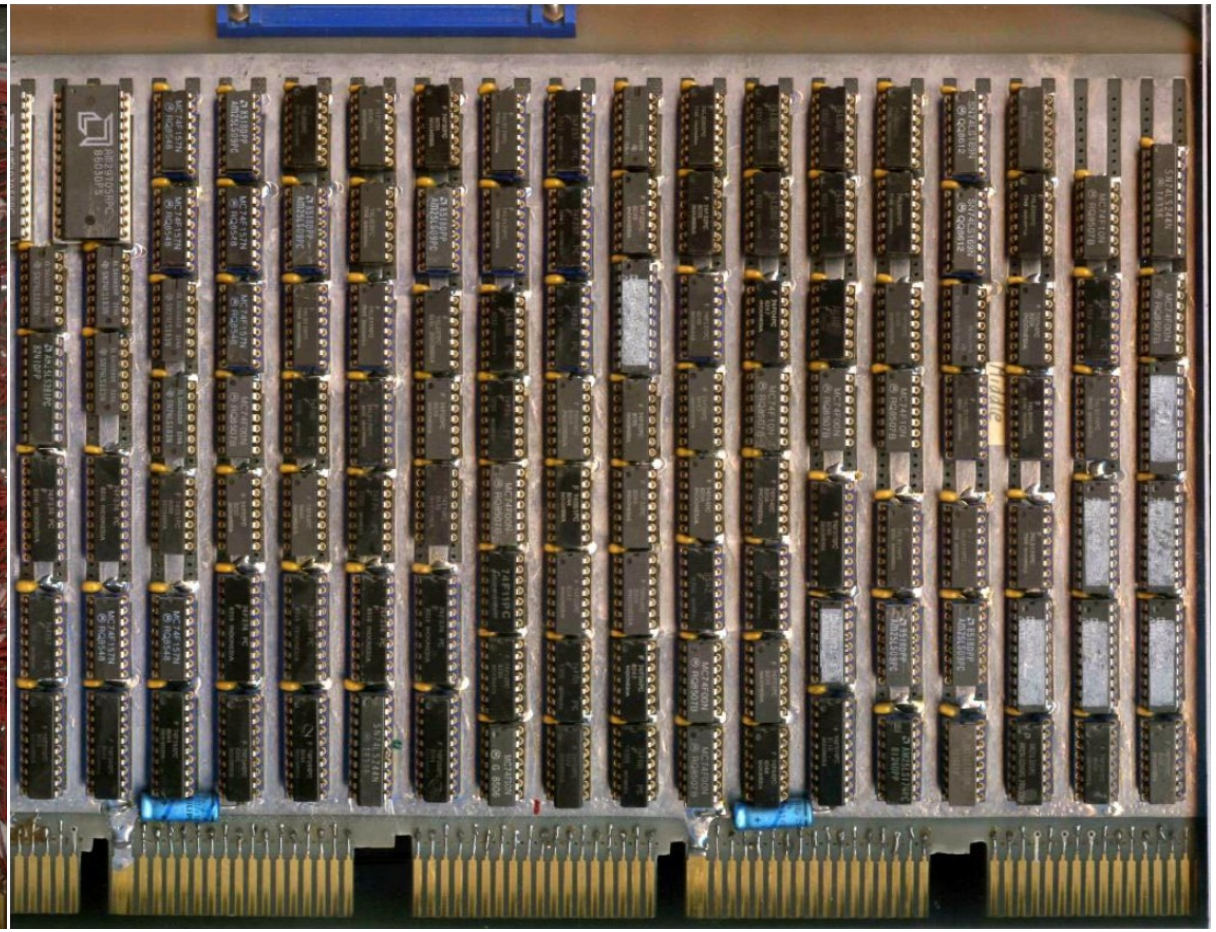
lab5_4.asc Op-amp importing external lib

lab5_5.asc Op-amp with capacitive feedback

PART 2 – introduction to FPGA

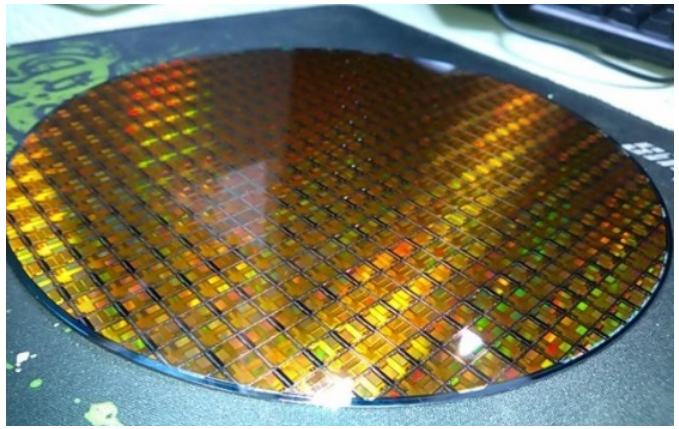
Early times of digital electronics

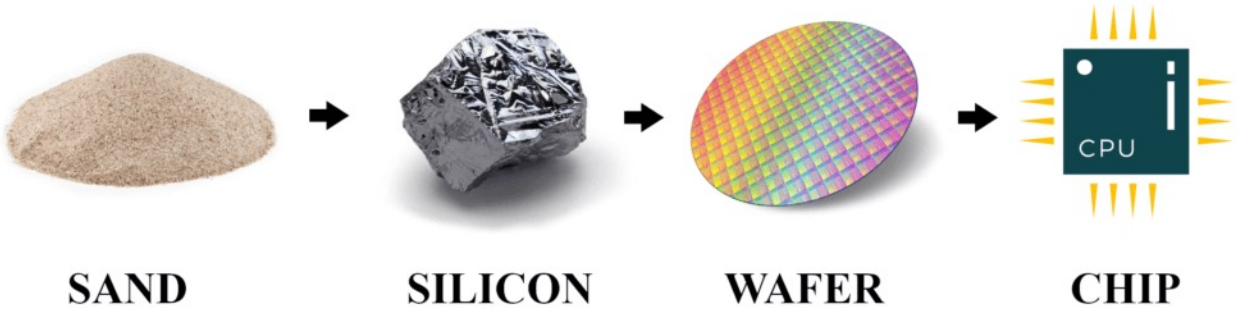




Why programmable ICs?

- Producing a Chip costs much more than producing a PCB
- Using one Chip on many PCBs increases Number of Chips and reduces Costs per Chip
- If a Chip is programmable, it can be used on many PCBs in different situations
- Producing one wafer costs ~ 10k€
  - Not included manpower to design wafer & packaging
- Saving space on a PCB





What are FPGAs?

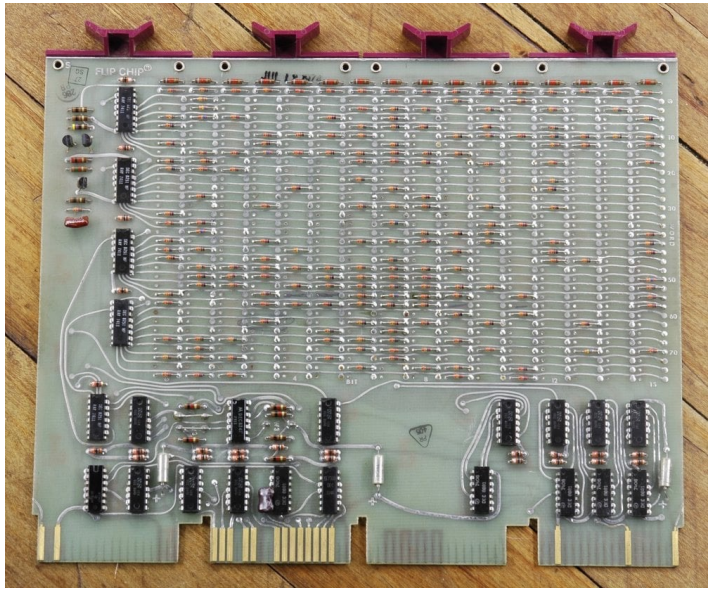
Field Programmable Gate Array

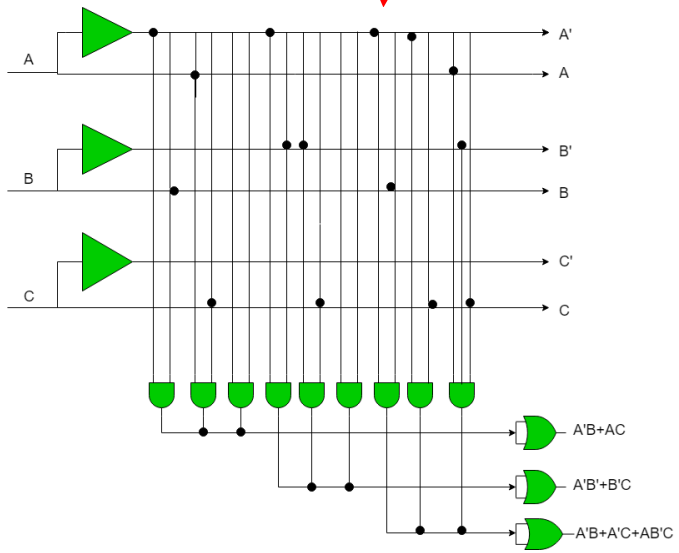
- Simple:
  - Programmable Logic Devices

- More detailed:
  - Programmable Logic Devices
    - With lot of additional function-block (e.g. RAM, DSP)
    - Flexible configuration
    - Can host microcontroller

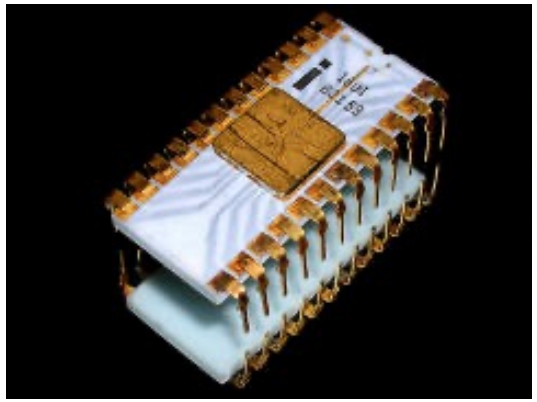
History of Programmable Logic Devices /1

- **1960s:** Fuse Configurable Diode Matrix
  - First form of programmable logic
  - One-Time Programmable (OTP)
- **1971:** Programmable ROM (**PROM**)
  - Logic functions stored as truth tables
  - Still limited flexibility and efficiency
- **PAL** (Programmable Array Logic):
  - Fixed AND/OR logic structure
  - Programmable interconnections
  - OTP technology



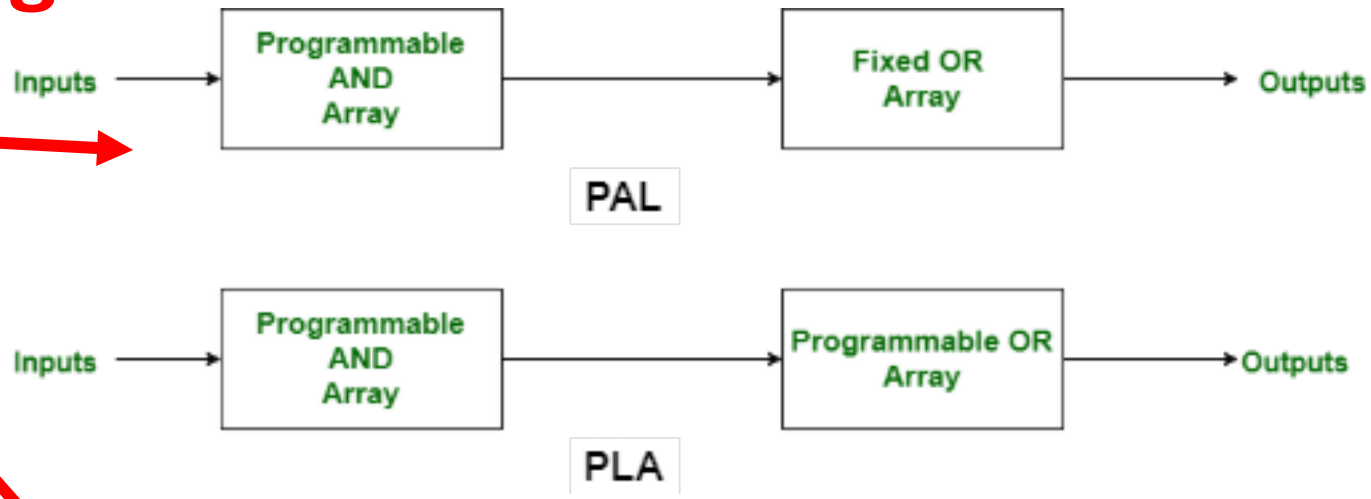


256 x 8-bit Static/Dynamic PROM



History of Programmable Logic Devices /2

- **1978: PLA (Programmable Logic Array)**
  - Programmable AND plane and OR plane
- **CPLD (Complex Programmable Logic Device)**
  - Multiple PAL-like blocks
  - In-system programmable
  - Configuration **retained after power-off**



The most noticeable difference between a large CPLD and a small FPGA is the presence of on-chip non-volatile memory in the CPLD, which allows CPLDs to be used for bootloader functions

Comparison with other Programmable Logic Devices

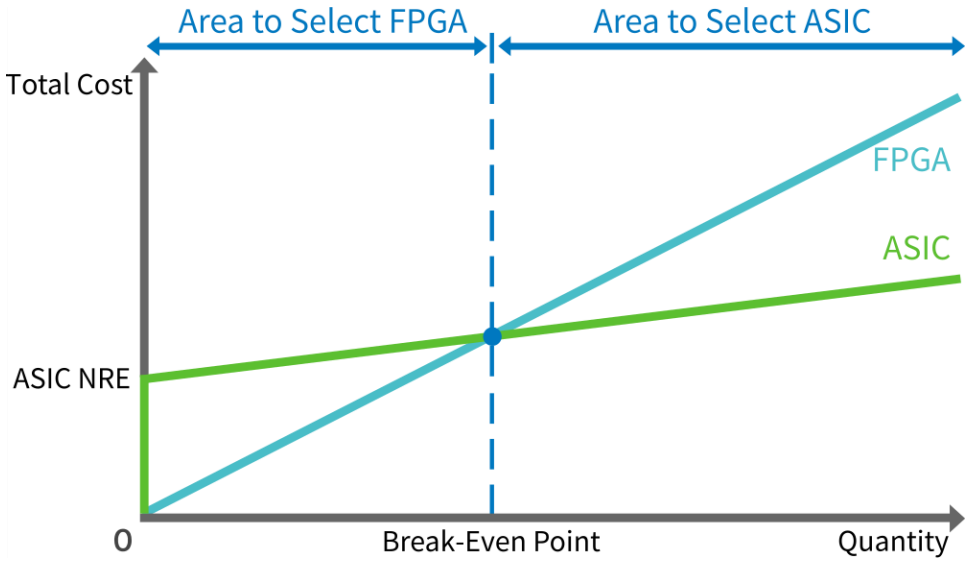
- PLA has a programmable AND gate array and programmable OR gate array.
- PAL has a programmable AND gate array but a fixed OR gate array.
- ROM has a fixed AND gate array but programmable OR gate array.

History of Programmable Logic Devices /3

- **1989:** First Field Programmable Gate Array (FPGA)
  - Configurable Logic Blocks (CLBs)
  - Programmable routing fabric
  - Massive parallelism
- Compared to CPLDs:
  - Much higher logic density
  - SRAM-based configuration (loaded at startup)
  - Ideal for complex systems

Price point

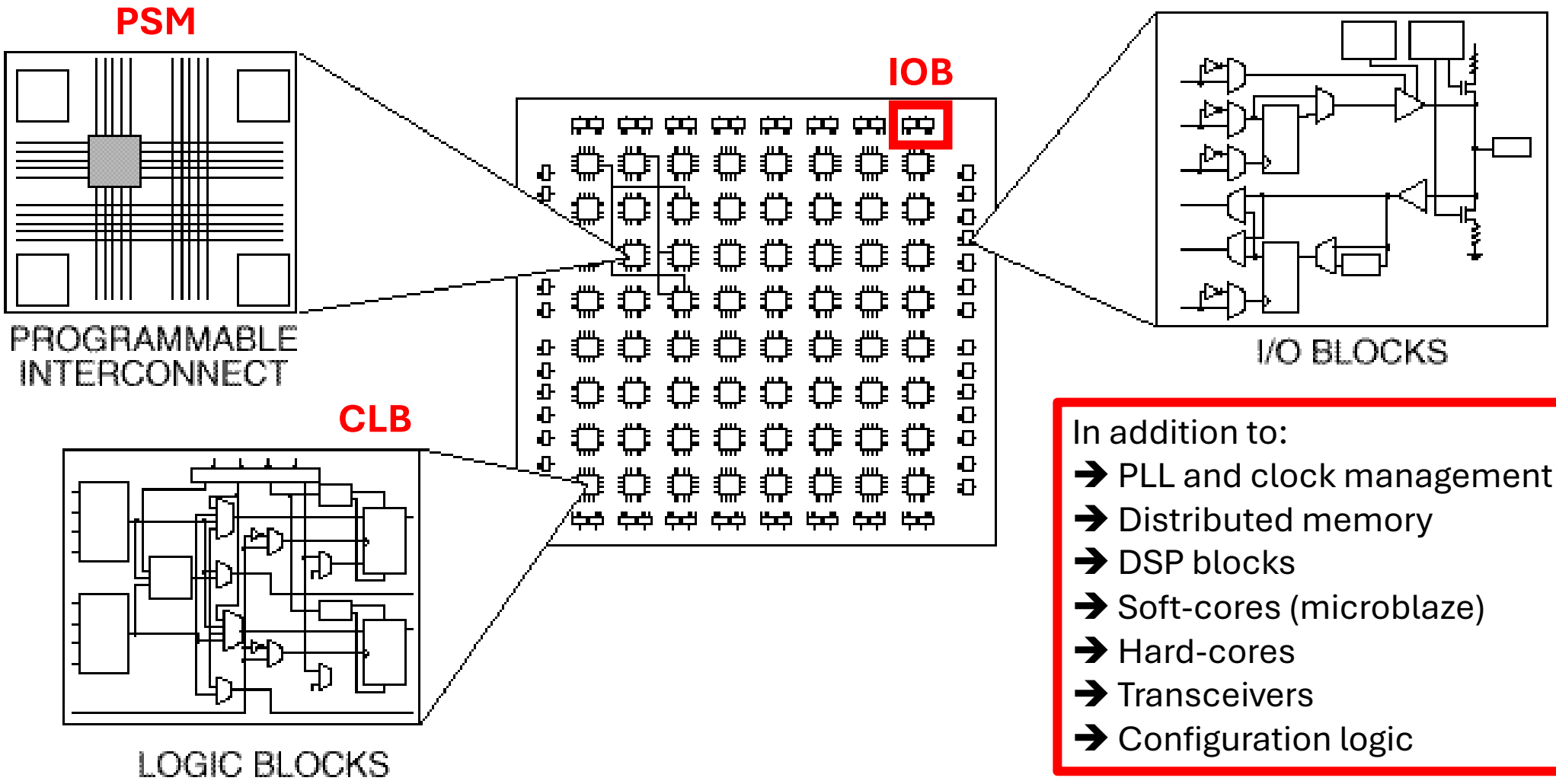
	**Produttore Codice prodotto** ICE40UL640-CM36AI **Codice Mouser** 842-ICE40UL640CM36AI	Lattice	FPGA - Array di gate programmabile su campo FPGA iCE40-UltraLite 1.2V CBGA PKG Maggiori informazioni	Scheda dati	243 A magazzino Confezione alternativa	1 2,58 € 25 2,24 € 100 2,12 €
	**Produttore Codice prodotto** ICE40UL1K-CM36AI **Codice Mouser** 842-ICE40UL1K-CM36AI	Lattice	FPGA - Array di gate programmabile su campo FPGA iCE40-UltraLite 1.2V CBGA PKG Maggiori informazioni	Scheda dati	722 A magazzino Confezione alternativa	1 2,67 € 25 2,32 € 100 2,24 €
	**Produttore Codice prodotto** ICE40LP384-CM49 **Codice Mouser** 842-ICE40LP384CM49	Lattice	FPGA - Array di gate programmabile su campo iCE40LP Ultra LowPwr 384 LUTs 1.2V Maggiori informazioni	Scheda dati	836 A magazzino Confezione alternativa	1 2,67 € 25 2,32 € 100 2,24 €
	**Produttore Codice prodotto** ICE40LP384-CM36TR **Codice Mouser** 842-ICE40LP384CM36TR	Lattice	FPGA - Array di gate programmabile su campo iCE40LP Ultra LowPwr 384 LUTs 1.2V Maggiori informazioni	Scheda dati	Tempo di consegna, se non a magazzino 24 settimane Confezione alternativa	Nastro continuo 20.000 2,68 €
			Array di gate mmabile su FPGA iCE40- 1.2V WLCS ri informazioni	Scheda dati	832 A magazzino Confezione alternativa	Nastro continuo 1 2,71 € 25 2,37 € 100 2,24 € Nastrati 1.000 2,24 €
	**Produttore Codice prodotto** AGIA035R39A2I3V **Codice Mouser** 989-AGIA035R39A2I3V	Altera	FPGA - Array di gate programmabile su campo	Scheda dati	Tempo di consegna diretta 16 settimane	3 38.140,14 €
	**Produttore Codice prodotto** 1ST210EU2F50I1VG **Codice Mouser** 989-1ST210EU2F50I1VG	Altera	FPGA - Array di gate programmabile su campo	Scheda dati	Tempo di consegna diretta 16 settimane	3 37.496,86 €
	**Produttore Codice prodotto** AGIA035R39A2E2VB **Codice Mouser** 989-AGIA035R39A2E2VB	Altera	FPGA - Array di gate programmabile su campo	Scheda dati	Tempo di consegna diretta 16 settimane	3 37.080,62 €
	**Produttore Codice prodotto** AGIA035R39A2E3E **Codice Mouser** 989-AGIA035R39A2E3E	Altera	FPGA - Array di gate programmabile su campo	Scheda dati	Tempo di consegna diretta 16 settimane	3 37.080,62 €
	**Produttore Codice prodotto** XCVU7P-L2FLVB2104E **Codice Mouser** 217-VU7P-L2FLVB2104E	AMD / Xilinx	FPGA - Array di gate programmabile su campo XCVU7P-L2FLVB2104E EAR	Scheda dati	Tempo di consegna, se non a magazzino 37 settimane	1 38.475,97 €
	**Produttore Codice prodotto** XCVU7P-2FLVB2104I **Codice Mouser** 217-CVU7P-2FLVB2104I	AMD / Xilinx	FPGA - Array di gate programmabile su campo XCVU7P-2FLVB2104I EAR	Scheda dati	Tempo di consegna, se non a magazzino 37 settimane	1 38.475,97 €



Comparing FPGA and Dedicated ICs

	FPGA	Dedicated IC
Up-front cost	Smaller	Larger
Per-unit cost	Larger	Smaller
Time to market	Shorter	Longer
Speed	Slower	Faster
Power consumption	Greater	Smaller
Update in the field	Straightforward	Very difficult
Density	Lesser	Greater
Design Flow	Simpler	More complex
Granularity	Logic blocks	Individual cells
Need for gate-level verification	Little	Much
Technology upgrade path	Easier	Harder
Additional features	Many	Fewer

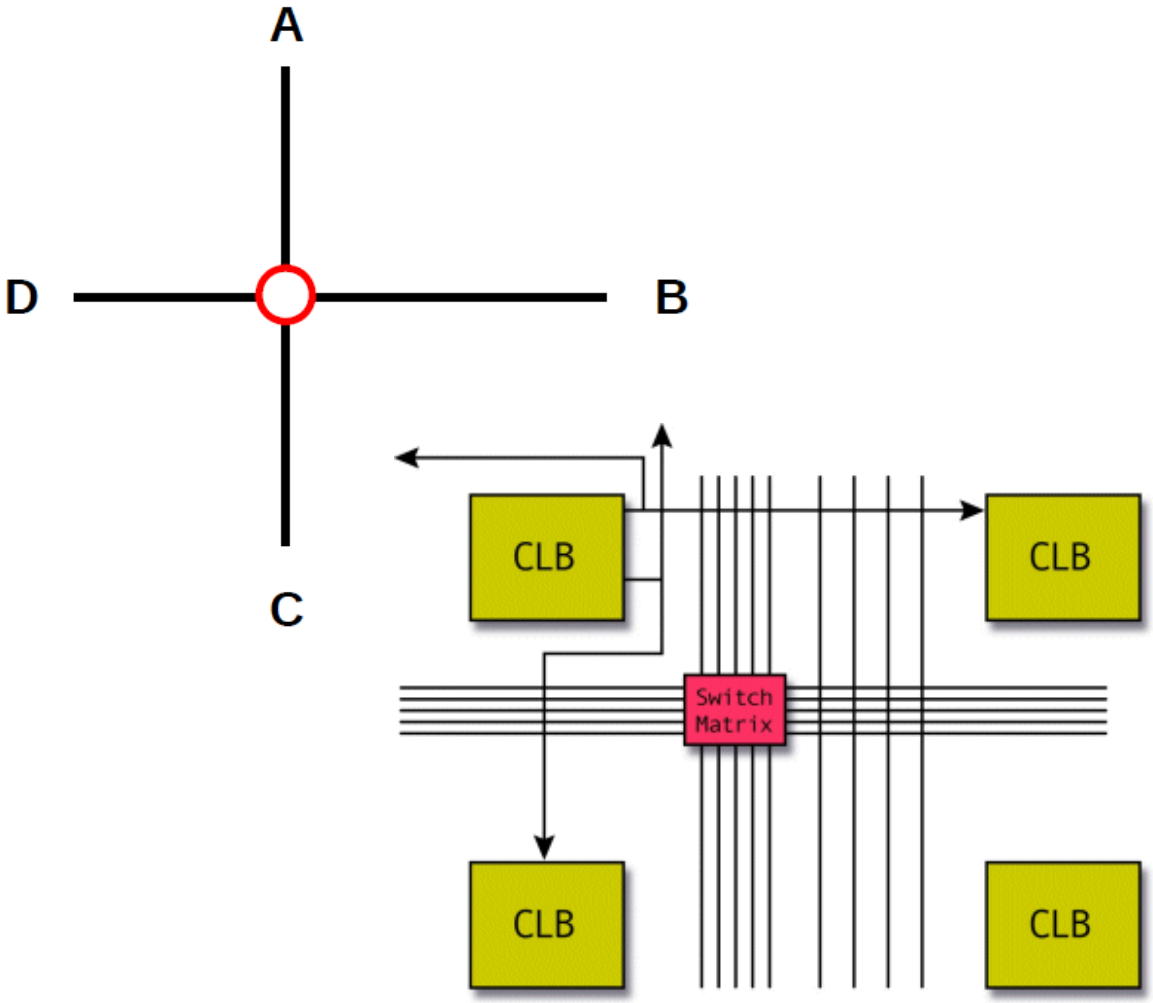
Fundamental FPGA components

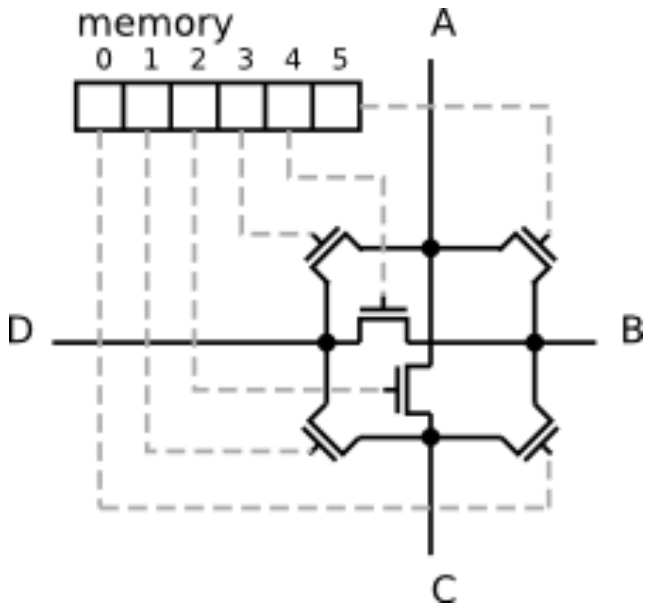


In addition to:

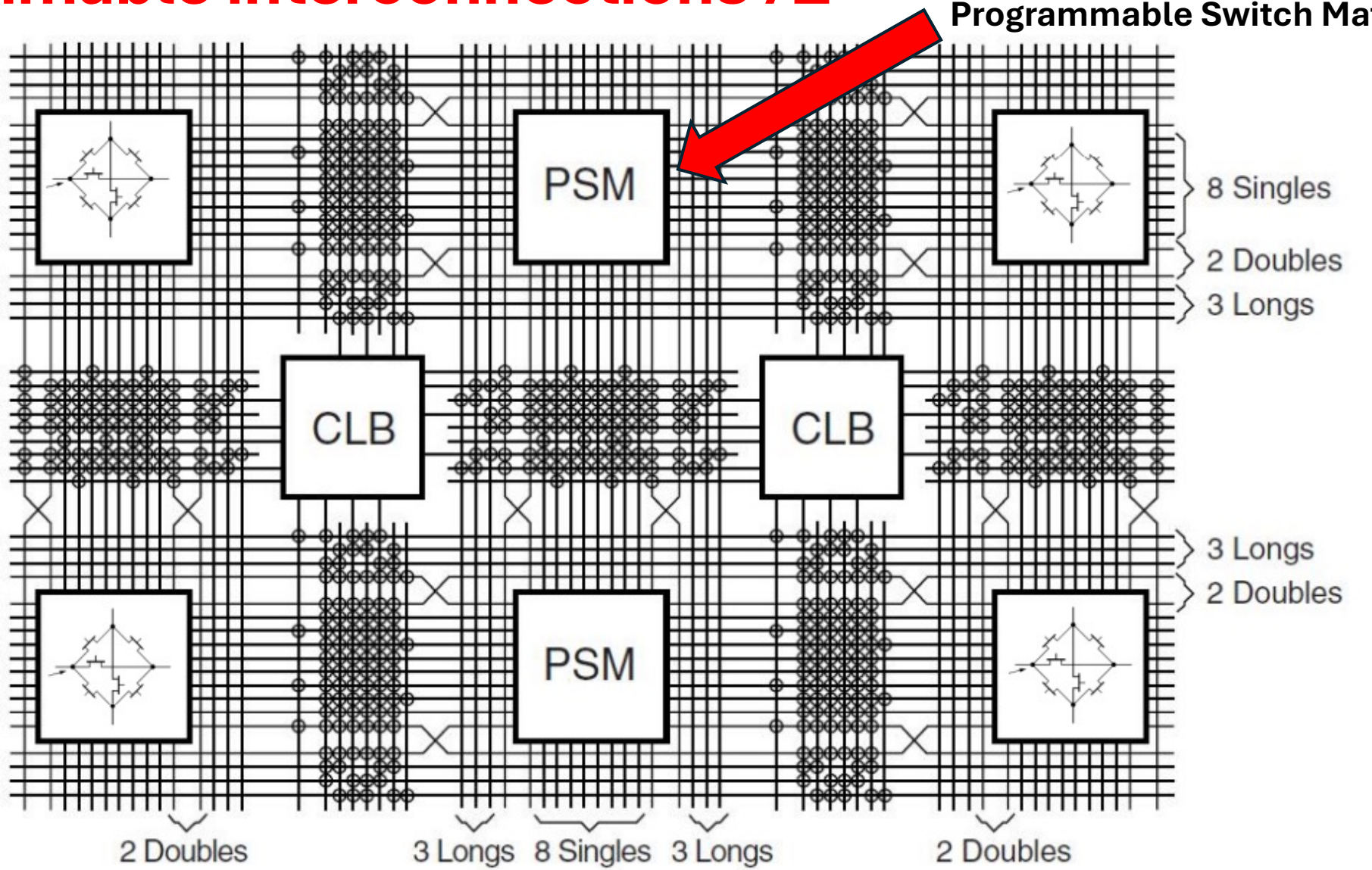
- ➔ PLL and clock management
- ➔ Distributed memory
- ➔ DSP blocks
- ➔ Soft-cores (microblaze)
- ➔ Hard-cores
- ➔ Transceivers
- ➔ Configuration logic

Programmable interconnections /1



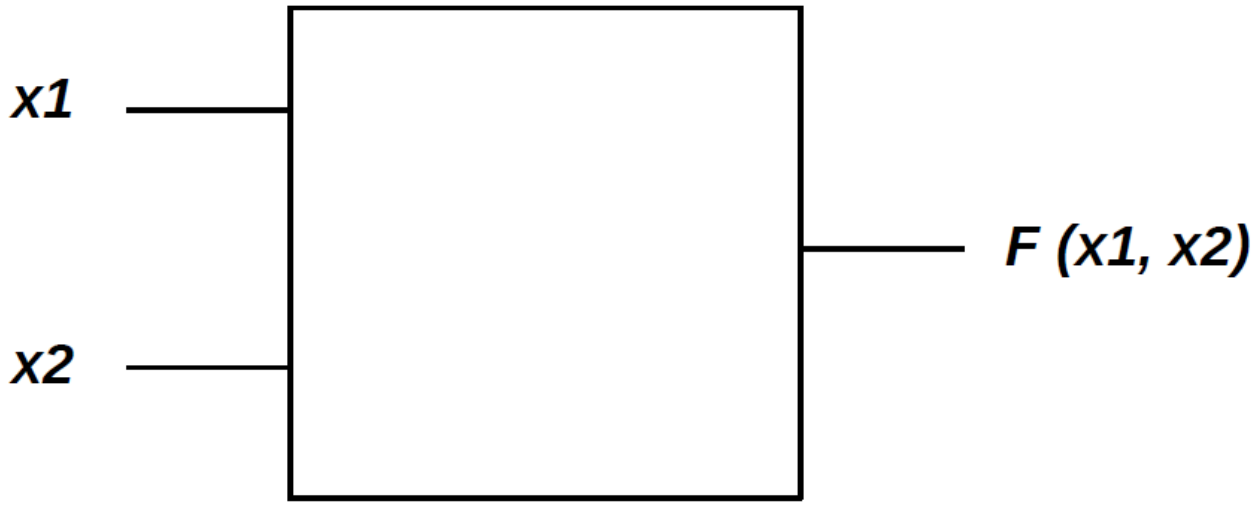


Programmable interconnections /2



Look-up tables (LUT) /1

How to implement a 2-bit truth table (boolean logic) ?

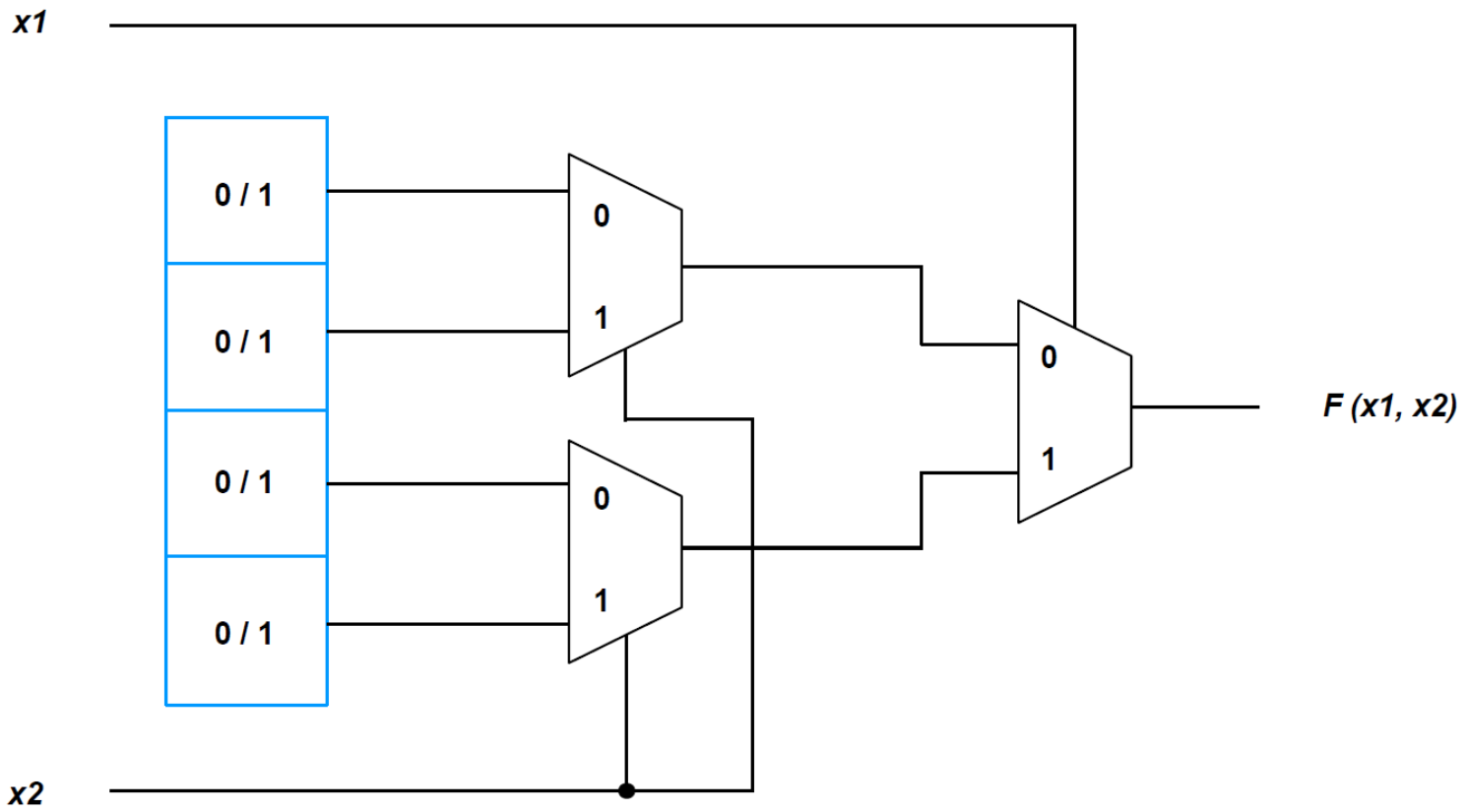


XNOR

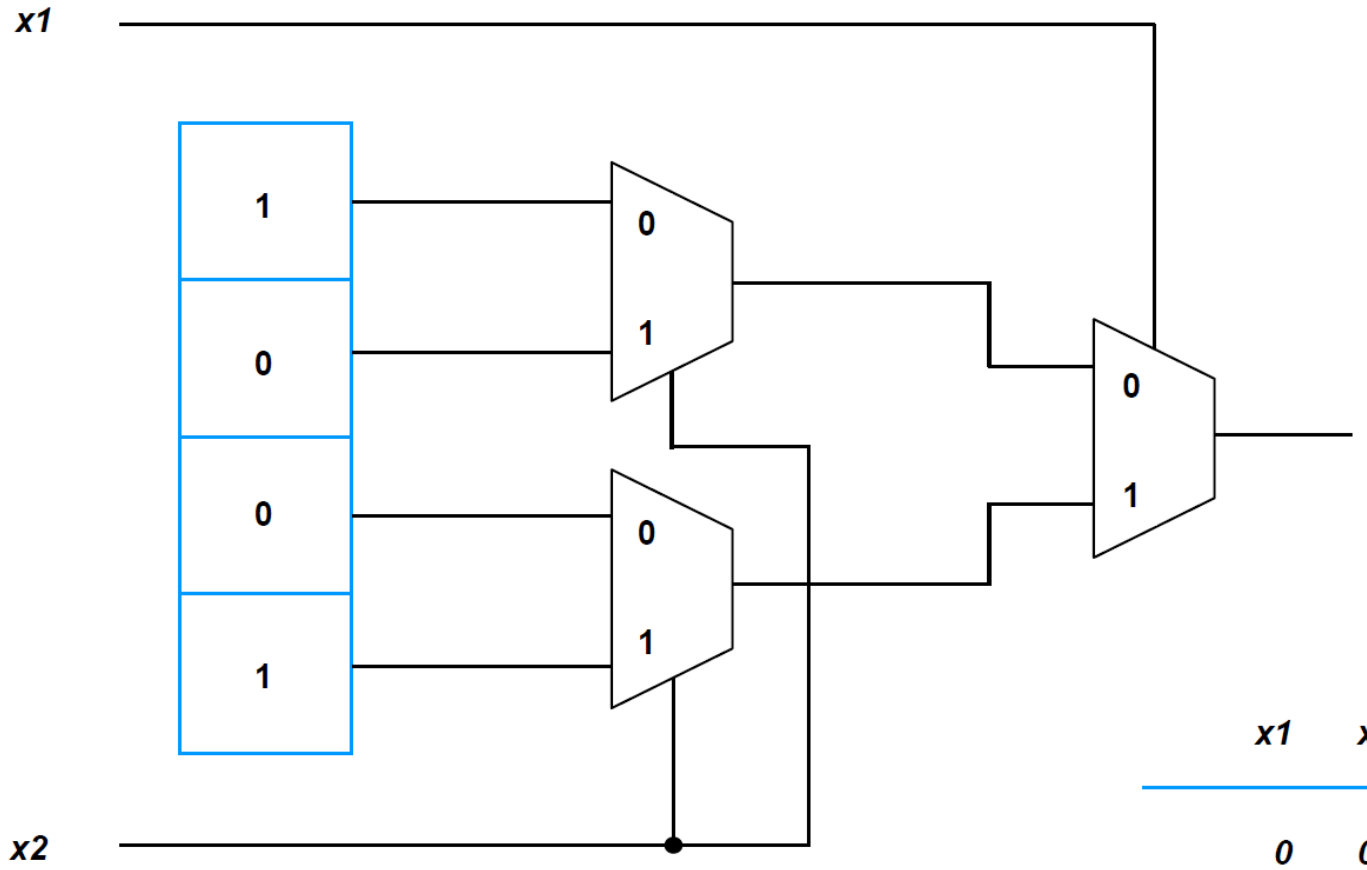
x1	x2	F
0	0	1
0	1	0
1	0	0
1	1	1

Look-up tables (LUT) /2

FPGA solution: write the truth table into a RAM !



Look-up tables (LUT) /3



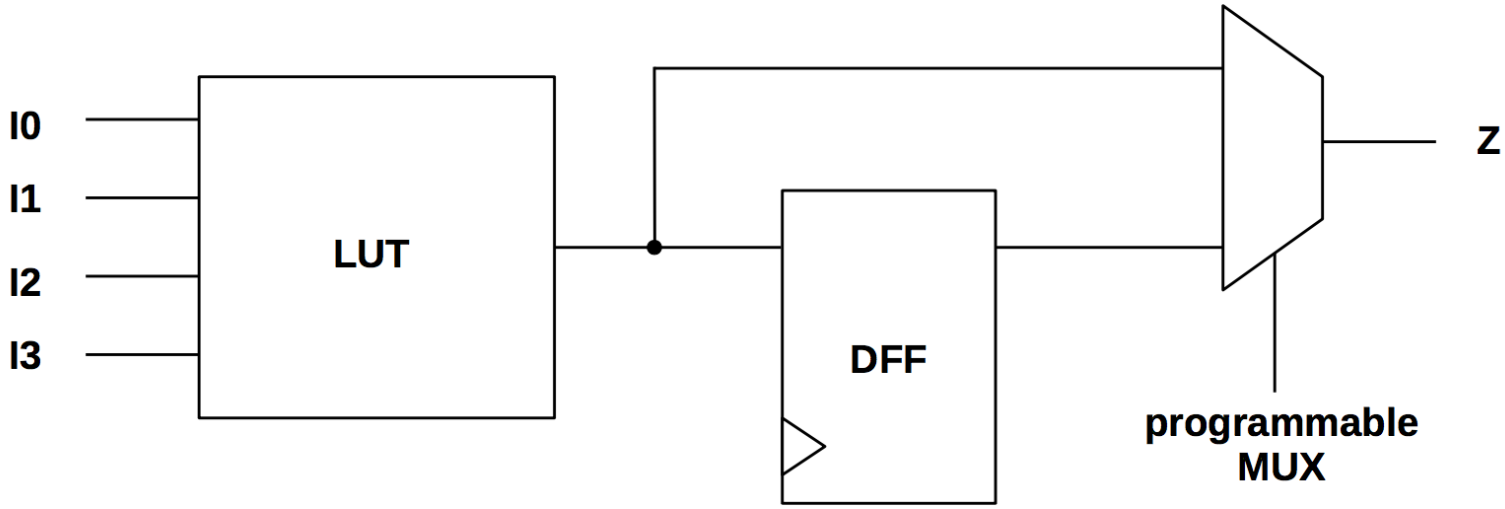
x1	x2	F
0	0	1
0	1	0
1	0	0
1	1	1

Logic Cell (LC)

Xilinx terminology :

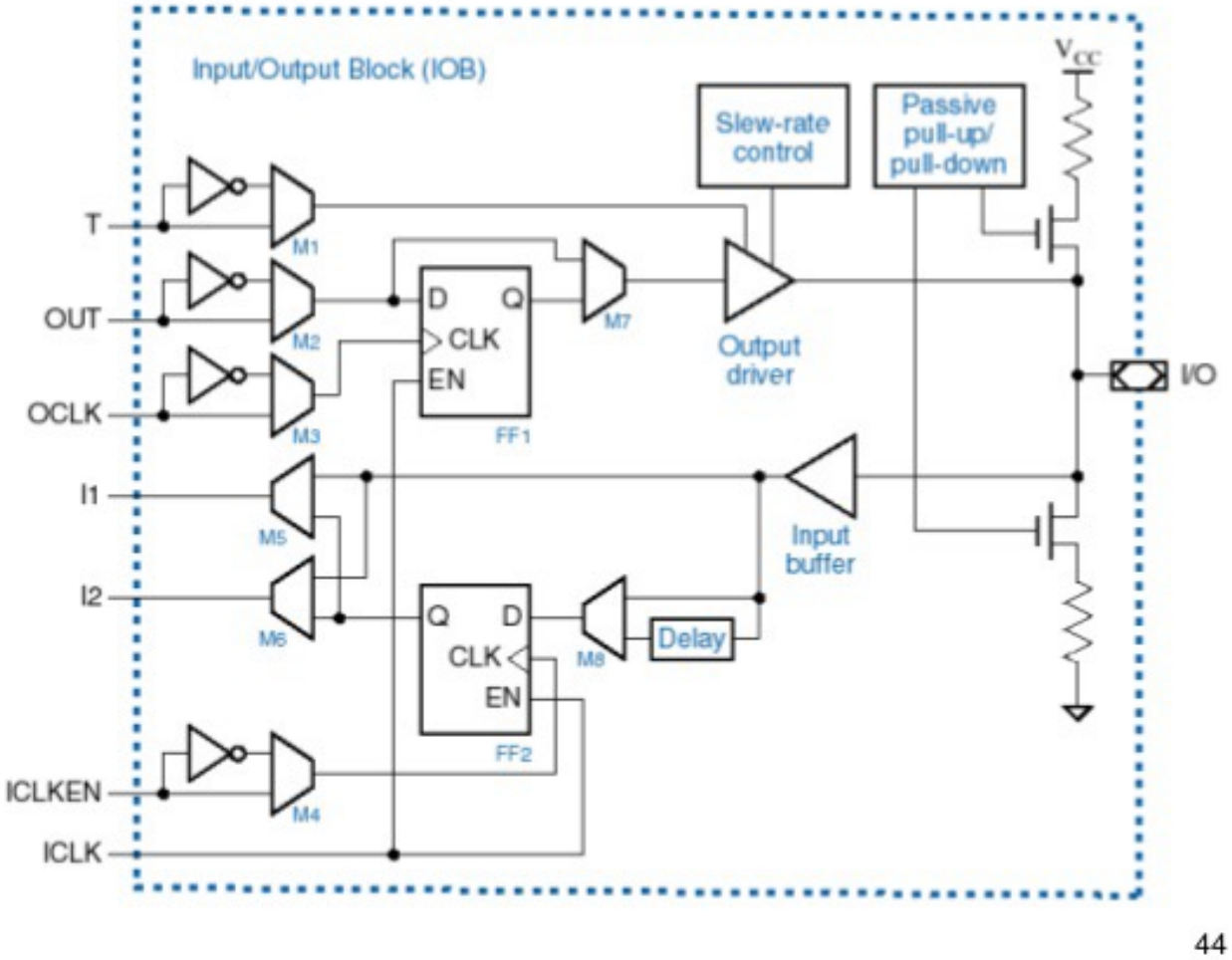
- **Logic Cell (LC)** = 1x LUT + 1 register (FF) + programmable MUX
- **LUT** implemented as 6-inputs, 64-bit RAM **WHY?**
- slice = 4x LUTs + 8 registers (FFs) + MUXs + fast-carry chain (CLA)
- **Congurable Logic Block (CLB)** = 2x slices
- Basic Element (BEL) = any device primitive

carry-lookahead adder



Programmable I/O pads

I/O BLOCK



Programmable pad

Xilinx FPGA line-up

Xilinx 7-series family comparison

Different size, number of programmable logic cells, I/O capabilities, performance and cost depending on the chosen device family :

- Spartan 7 : lowest-cost, lowest-power, good I/O performance, lowest number of resources
- Artix 7 : low-power, medium number of resources and I/O, good choice for DSP
- Kintex 7 : best compromise between number of resources, I/O, speed, power and cost
- Virtex 7 : highest system performance (resources, speed, I/O) but also largest power consumption and highest cost

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	–	16	32	96
Transceiver Speed	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	–	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	–	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

FPGA advantages and disadvanges

Advantages

- ➔ Cheap
- ➔ Do anything!
- ➔ Super fast
- ➔ Fiel programmable
- ➔ Massively parallel
- ➔ High I/O count

Disadvantages

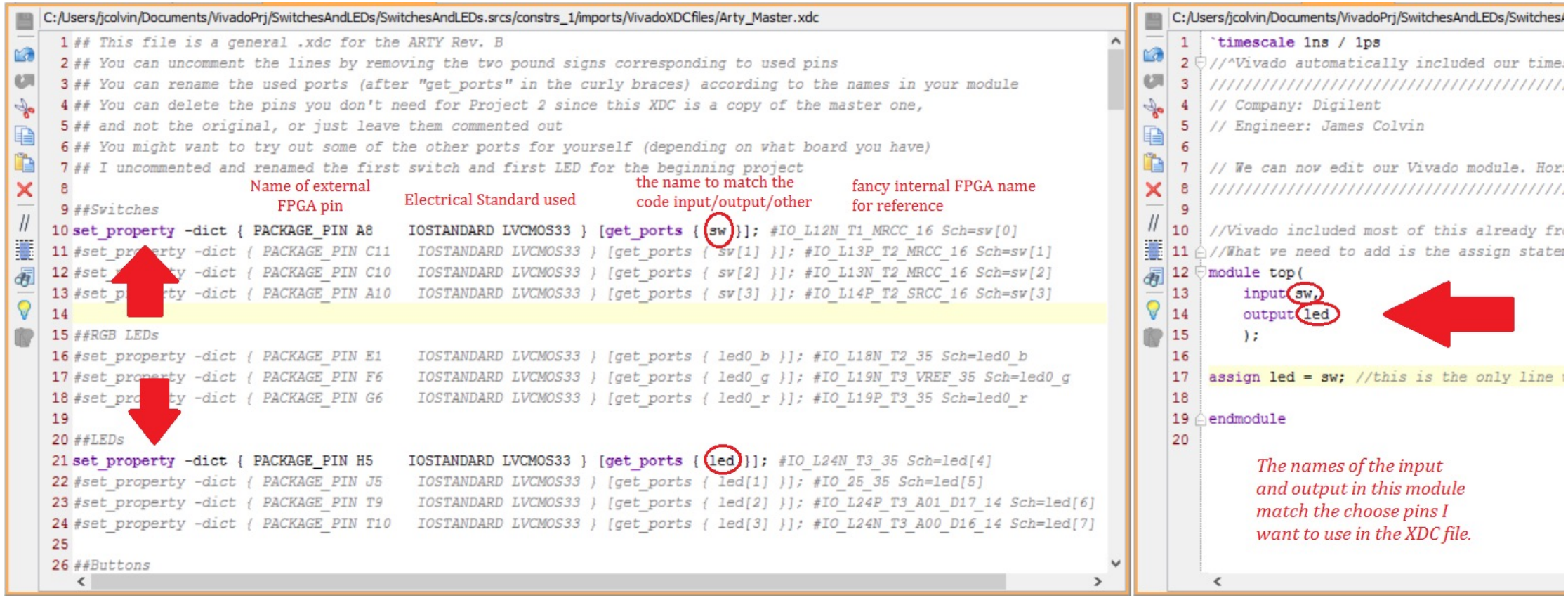
- ➔ Expensive
- ➔ High power
- ➔ Volative / BOOT time
- ➔ High pin count / BGA
- ➔ Complicated
- ➔ Many traps
- ➔ Complex tools
- ➔ Hard to chose / compare
- ➔ HDL “not easy” / intuitive

How “programming” a FPGA?

Hardware Description HDL	Describe what the hardware should do.
Syntax Check	Check Syntax
Synthesis	“compiles” the design to transform HDL source into an architecture-specific design netlist. (connections)
Translate	Merges incoming netlists and constraints into a Xilinx design file → connections are merged with timing
Map	Fits the design into the available resources → tell the FPGA which gate structures should be used
Place & Route	Places and routes the design to the timing constrains → decide, which gate is used for which function
Programming	Write configuration into the FPGA or configuration memory

Xilinx XDC – Xilinx Design Constraints file

XDC constraints are based on the standard **Synopsys Design Constraints (SDC)** format. SDC has been in use and evolving for more than 20 years, making it the most popular and proven format for describing design constraints.



XDC key concepts

1. Create clock
2. Define clocks interactions
3. Set input an output pins (and delays)
4. Set timing exceptions

1. Set False Path
2. Set Clock Groups
3. Set Multicycle Path
4. Set Max Skew/Set Max Delay -datapath_only

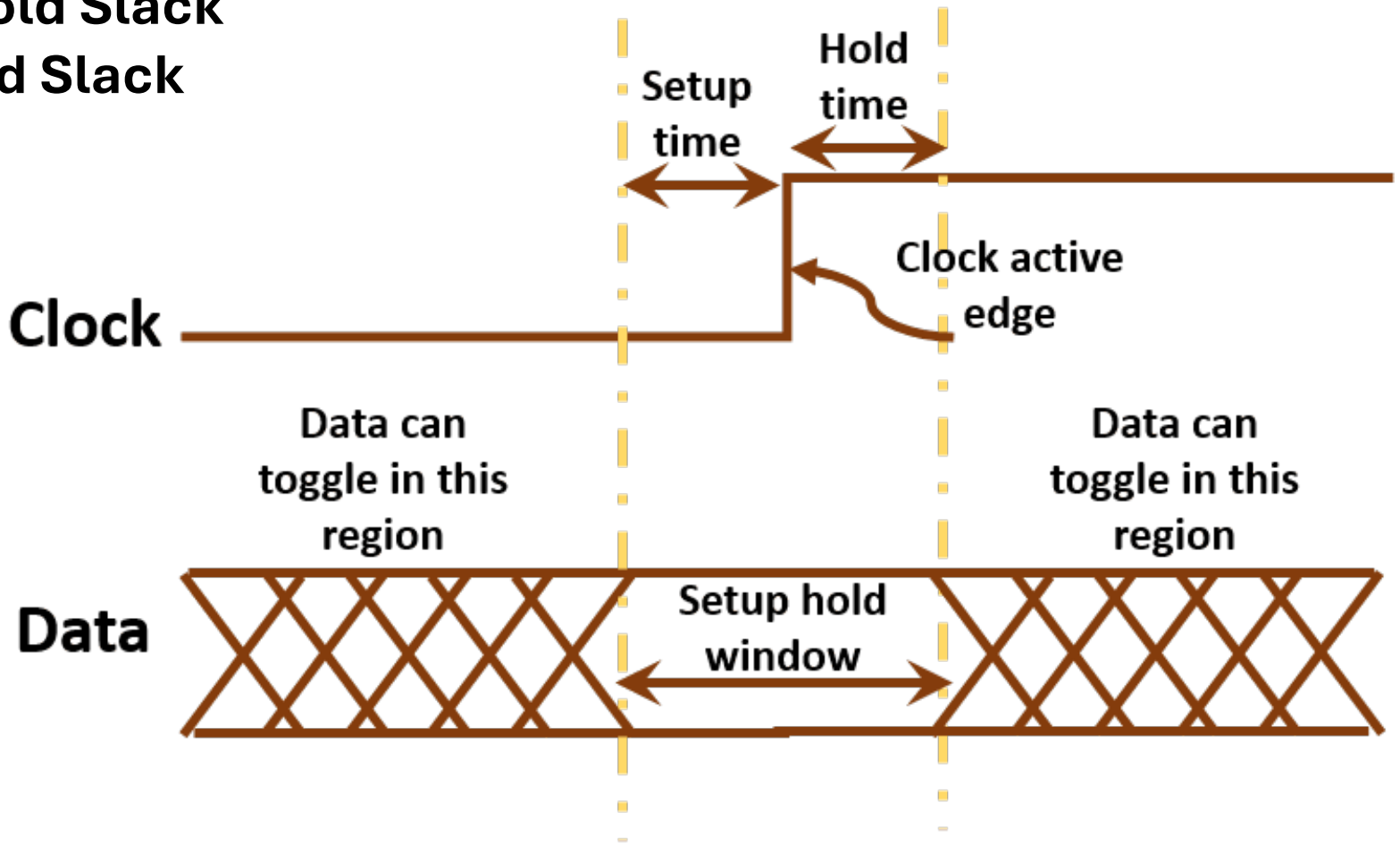
a timing constraints which is not required to be optimized for timing

The main difference between a **multicycle path** with many available cycles (*large n*) versus a **false path** is that the **multicycle path** will still be checked against setup and hold requirements and will still be included in the timing analysis.

```
set_multicycle_path <path_multiplier> [-setup|-hold] [-start|-end]
[-from <startpoints>] [-to <endpoints>] [-through <pins|cells|nets>]
```

WNS TNS WHS THS

1. Worst Negative Slack
2. Total Negative Slack
3. Works Hold Slack
4. Total Hold Slack



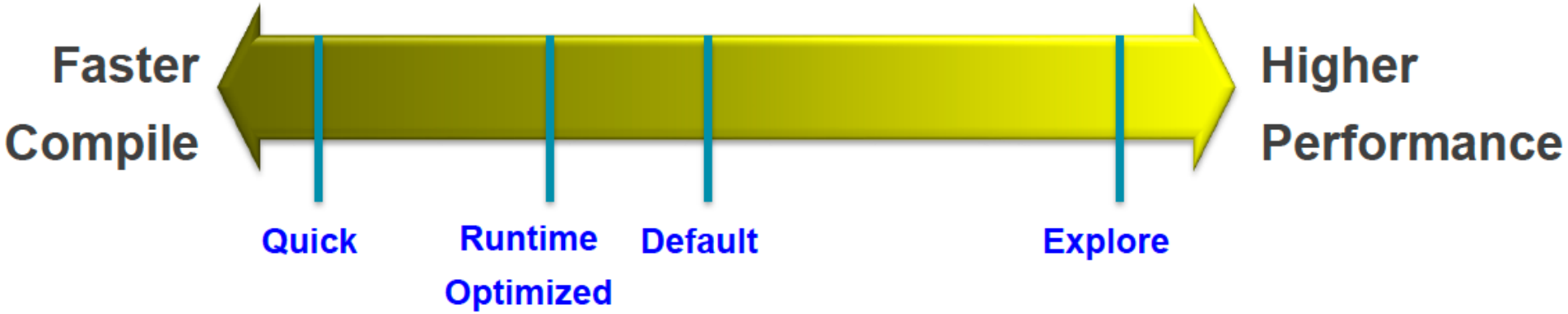
Vivado Implementation Strategies and Directives

Directive: “directs” command behavior to try alternative algorithms

- Enables wider exploration of design solutions
- Applies to opt_design, place_design, phys_opt_design, route_design

Strategy: combination of implementation commands with directives

- Performance-centric: all commands use directives for higher performance
- Congestion-centric: all commands use directives that reduce congestion
- Flow-centric: modifies the implementation flow to add steps to Defaults
  - power_opt_design
  - post-route phys_opt_design



Recommended Constraints Sequence

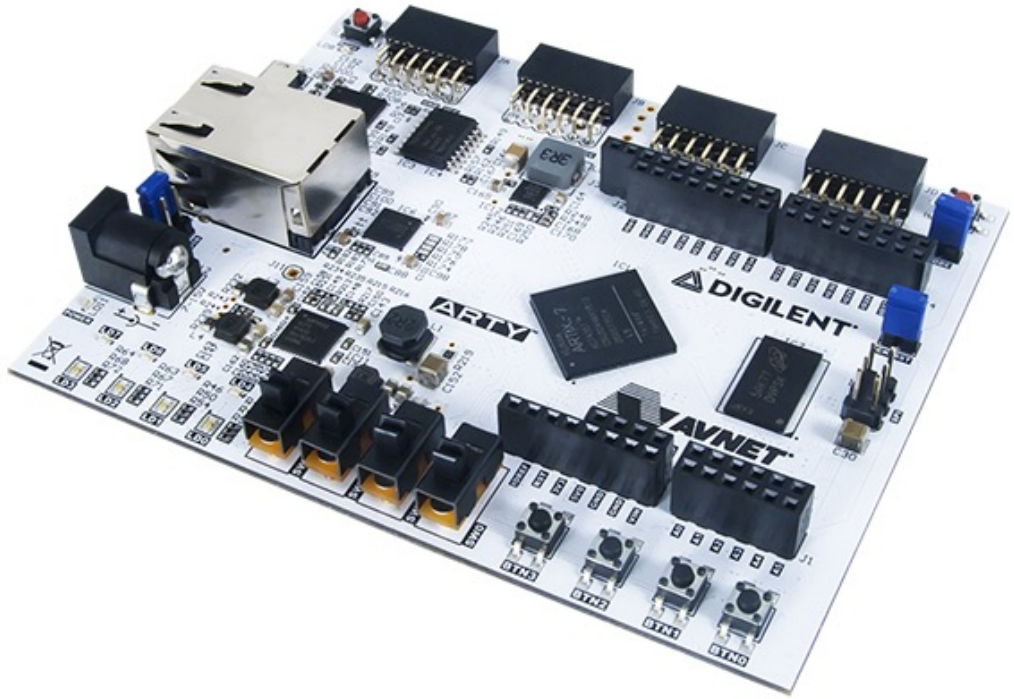
```
## Timing Assertions Section # Primary clocks
# Virtual clocks
# Generated clocks # Clock Groups
# Bus Skew constraints
# Input and output delay constraints
```

```
## Timing Exceptions Section # False Paths
# Max Delay / Min Delay # Multicycle Paths
# Case Analysis # Disable Timing

## Physical Constraints Section
# located anywhere in the file, preferably before or after the timing
constraints # or stored in a separate constraint file
```

Example: Digilent Arty-A7 evaluation board

Device: Xilinx Artix-7



PART 3 – Hardware Description Language - VHDL

Digital systems

analog signals *:*

- *continuous in both time and amplitude*
- *usually a voltage v(t) or a current i(t) as a function of time*
- *reach of information (e.g. frequency spectrum, FFT)*

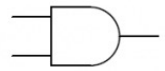
digital signals *:*

- *usually continuous in time but discrete in amplitude*
- *only two possible values e.g. high/low voltage levels, true/false, on/ff, closed/open etc.*
- *less information, but more robust against noise*
- *can be either asynchronous signals or synchronous signals*

Classification of digital circuits

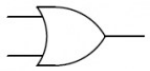
Digital circuits are classified as :

- *combinational circuits*
- *sequential circuits*
  - *asynchronous*
  - *synchronous*



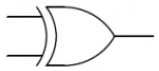
AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



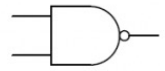
OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



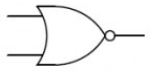
XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



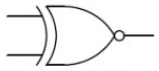
NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



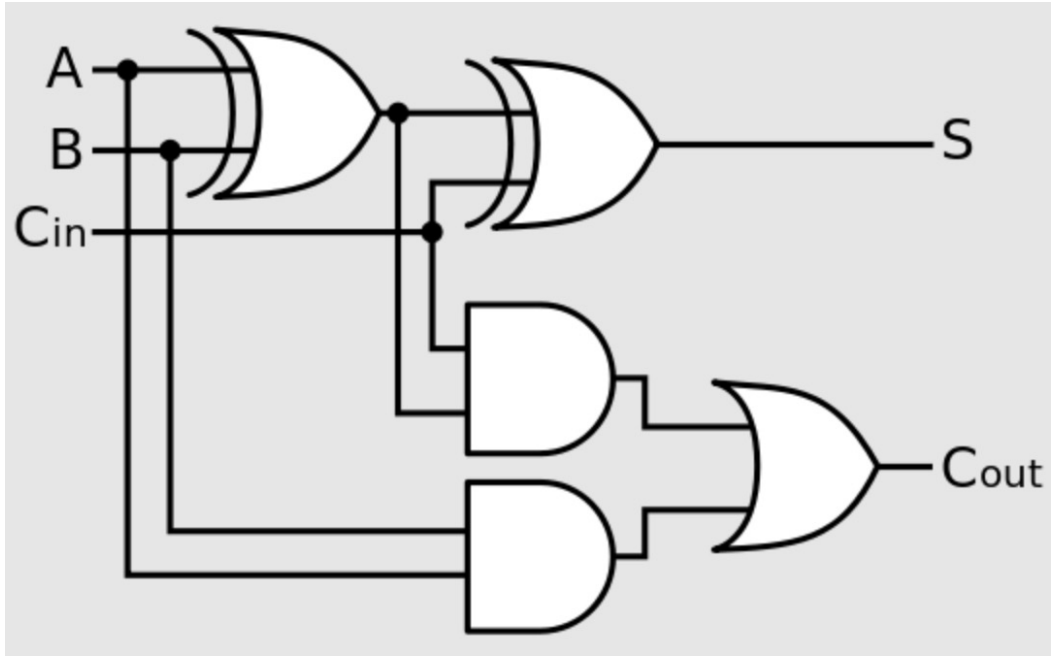
NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

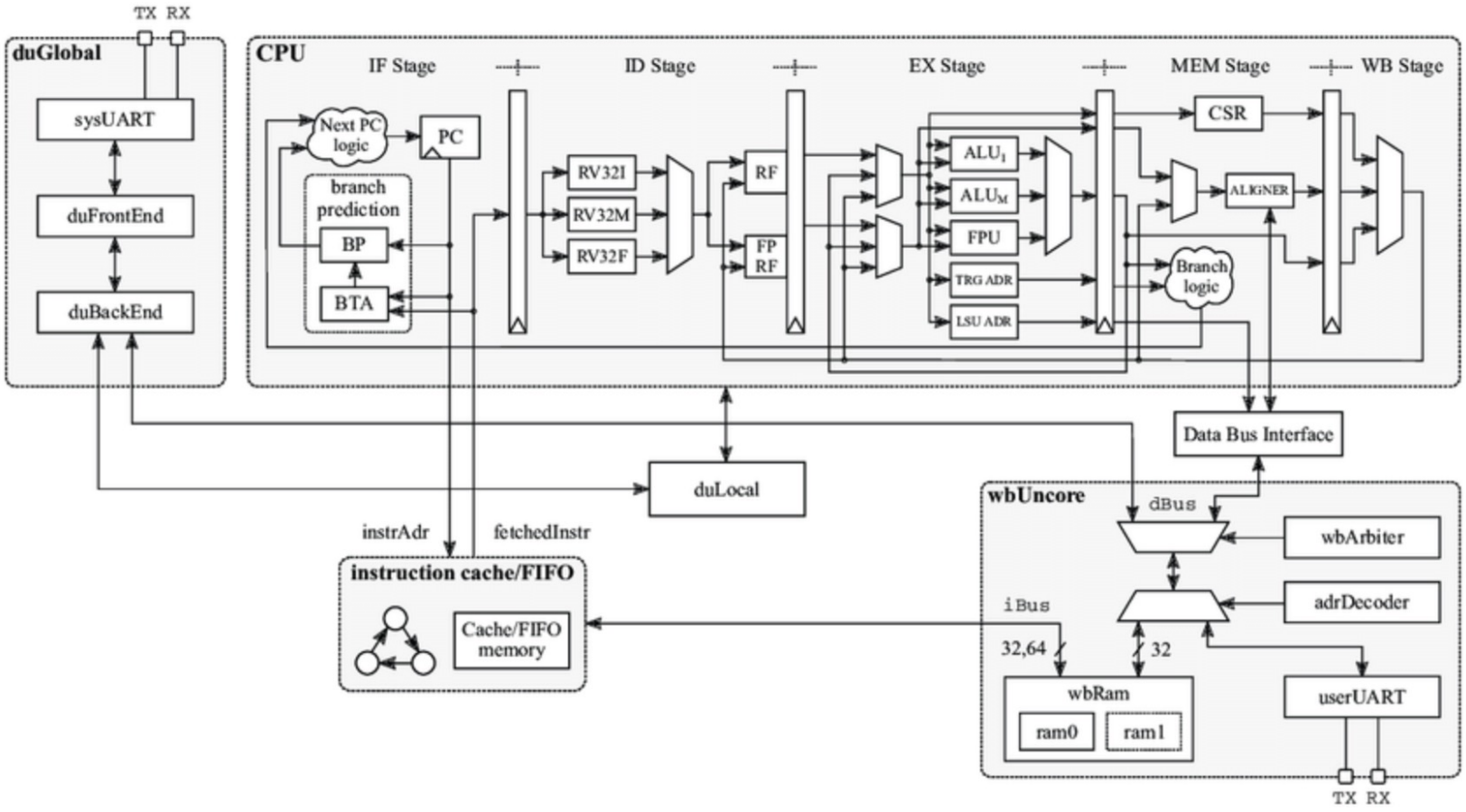


XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1



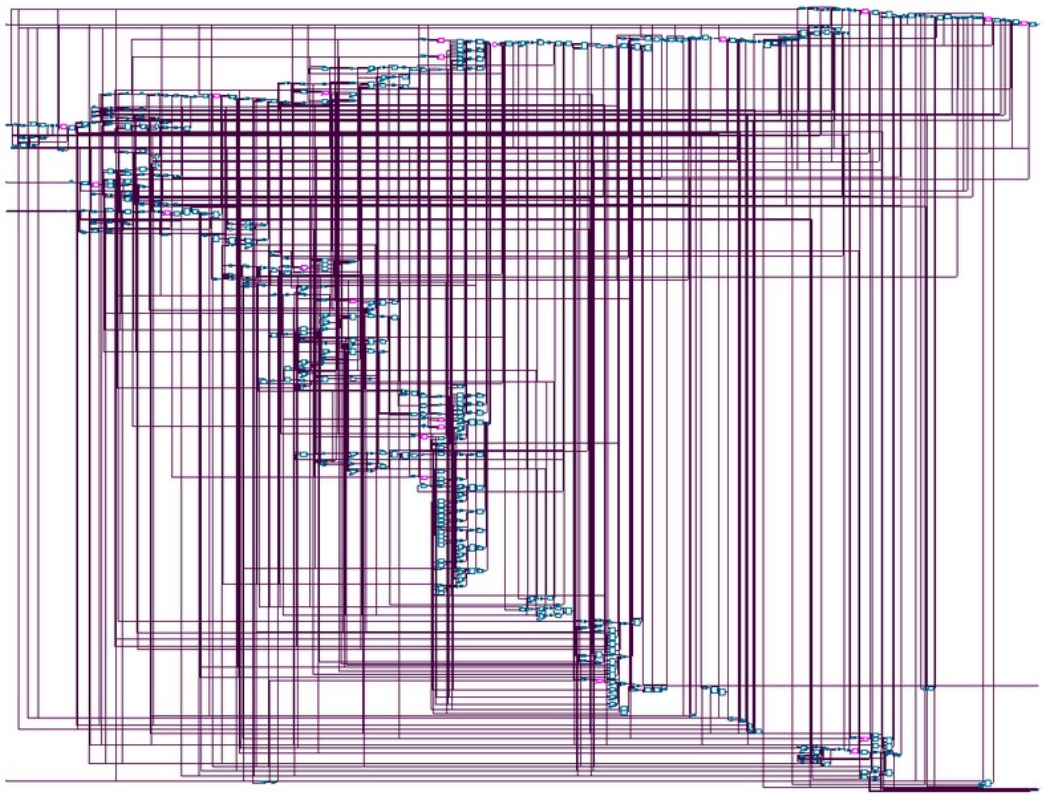
Hardware Description Languages (HDL)



Hardware Description Languages (HDL)

COMPLEX *digital system (e.g. micro-processor) are designed and simulated writing code!*
a ***SYNTHESIS*** *tool generates the real* ***HARDWARE*** *for you !*

```
17  localparam S2 = 2'b11;
18  localparam S3 = 2'b10;
19
20  reg [1:0] pState, nState;
21
22  always @ (pState, start, stop, increment)
23  begin
24      case (pState)
25          S0:begin
26              if (start == 1'b0 && increment ==
27                  nState = S0;
28              else if (increment == 1'b1 && sta:
29                  nState = S2;
30              else
31                  nState = S1;
32              end
33          S1:
34          begin
35              if (stop == 1'b1)
36                  nState = S0;
37              else
38                  nState = S1;
39           end
40          S2:
41              nState = S3;
42          S3:
43          begin
44              if (increment == 1'b1)
45                  nState = S3;
46              else
47                  nState = S0;
48           end
49          default:
50              nState = S0;
51      endcase
```



Hardware Description Languages (HDL)

Hardware Description Languages (HDLs) :

- they are NOT "programming" languages !
- provide constructs for both implementation (describe the functionality of digital circuits) and verification (simulations, assertions etc.)
- synthesis (first step of modern digital design
- "synthesizable" constructs are only a minimal subset of the full language

Two main options on the market :

- VHDL
- Verilog

advanced options :

- high-level synthesis (HLS) using C/SystemC
- SystemVerilog
- analog and mixed-signal simulations (Verilog-A, Verilog-AMS, VHDL-AMS)

VHDL

- VHDL = VHSIC-HDL (Hardware Description Language)
- VHSIC = Very High Speed Integrated Circuit
- born in 1983 as a project from US Department of Defense (DoD)
- syntax derived from the **ADA programming language** (similar to PASCAL) as requested by DoD
- initially foreseen as a language used to describe and simulate digital integrated circuits
- later used also for digital synthesis
- 1987: first standardization as **IEEE Std. 1076-1987** (aka "VHDL 87")
- 1993: **first major revision** of the language as **IEEE Std. 1076-1993** (aka "VHDL 93")
- 2008: **second major revision** of the language as **IEEE Std. 1076-2008** (aka "VHDL 2008")
- provides constructs for both physical implementation (synthesizable) and simulation
- very reach syntax
- extremely **verbose** and **strongly typed** !

Verilog

- created by P. Goel, P. Moorby, C.L. Huang and D. Warmke between late 1983 and early 1984
- syntax derived from the **C programming language**
- introduced by Gateway Design Automation, later purchased by Cadence Design Systems in 1990
- born as a **verification** language for logic designs, intended to describe and simulate digital integrated circuits similar to VHDL
- initially a **proprietary** and **closed** language owned by Cadence
- later released by Cadence as an open language in order to cope with the increasing popularity of VHDL (already standardized by IEEE in 1987)
- 1995: first standardization by IEEE as **IEEE Std. 1364-1995** (aka "Verilog 95")
- 2001: first major revision of the language with extensions and new language features known as **IEEE Std. 1364-2001** (aka "Verilog 2001")
- 2006: second major revision with minor changes as **IEEE Std. 1364-2006** (aka "Verilog 2006")
- finally merged as a sub-set of the **SystemVerilog** HDL as **IEEE Std. 1800-2009**

Verilog vs. VHDL

Verilog and VHDL are the two most widespread HDLs in the world :

- approx. 50% market each one
- Verilog more popular in US and Japan, VHDL in Europe
- Verilog more used (and integrated with professional CAD tools) to design **Application-Specific Integrated Circuits** (ASICs)
- traditionally VHDL more used for **FPGA programming** instead
- both **equally** and well **supported** by **Xilinx**
- exception: gate-level timing simulations only supports Verilog netlists
- individual preference, usually mostly historical (that is, first language learned…)
- Verilog is **easier to learn** (C-like syntax), but also potentially more error-prone due to its relaxed data typing
- that is… you can make real disasters on silicon with Verilog, while using VHDL is pretty impossible
- starting alone from scratch… learn VHDL first, then move to Verilog once really annoyed with VHDL "verbose" coding

Digital simulators

Many different digital simulators available on the market.
From the "big threes" :

- QuestaSim (the professional version of ModelSim) by Mentor (now Siemens)
- Xcelium (legacy Incisive) by Cadence
- VCS by Synopsys

Integrated with FPGA programming suites :

- XSim as part of the Vivado design suite by Xilinx
- ISim as part of the legacy ISE design suite by Xilinx
- ModelSim Altera as part of the Quartus design suite by Altera (now Intel)

Other solutions :

- Aldec HDL by Aldec
- open source simulators (FreeHDL, Icarus, GHDL etc,)

VHDL fundamentals

Case sensitivity, statements, white spaces

*VHDL is **case insensitive** (on the contrary, Verilog is case sensitive) :*

```
Z <= A and B ;                    z <= a AND b ;
```

every VHDL statement is terminated with a semicolon ;

VHDL is also not sensitive to blanks (white spaces) between statements and empty lines are ignored by the compiler

Comments

in VHDL comments starts with two dashes :

```
-- this is a single-line VHDL comment

-- this is a multiple-lines
-- VHDL
-- comment
```

VHDL lacks the possibility to quickly comment large sections of code (on the contrary, Verilog provides C-style block-comments enclosed within / and */)*

always put a lot of appropriate comments to document your code !

VHDL core elements

libraries and packages
- *import all useful data types and functions*

ENTITY declaration
- *interface of the digital block*
- *optional parameters*

ARCHITECTURE declaration
- *actual implementation of the block*

```
library ieee ;
use ieee.std_logic_1164.all ;

entity Inverter is
    port (
        X  : in  std_logic ;
        ZN : out std_logic
    ) ;
end entity Inverter ;

architecture rtl of Inverter is

begin

    ZN <= not X ;

end architecture rtl ;
```

Entity declaration

*Any **digital block** implementing some functionality in VHDL is called entity :*

the VHDL entity has I/O ports that can be declared as :

- *in*
- *out*
- *inout*
- *buffer*

optionally, the block can also include parameters (generics)

```
entity BlockName is

   -- optional generic list
   [generic( ... ) ; ]

   -- port list
   port( ... ) ;

end [entity] BlockName ;
```

Port list

Input ports :

```
clk : in std_logic,
rst : in std_logic, ...
```

Output ports :

```
busy : out std_logic,
LED  : out std_logic_vector(3 downto 0)
```

Inout (bidirectional) ports :

```
sda : inout std_logic
```

Architecture declaration

The actual block implementation is coded in the architecture instead :

```
architecture archName of BlockName is

   -- initial declarations
   ...

begin

   -- actual block implementation
   ...
   ...

end [architecture] archName ;
```

Testbench

*In order to **simulate** the functionality of the digital block we also need a **testbench** that generates **stimuli** fed to input ports of our **Device Under Test (DUT):***

```
entity tb_BlockName is
end entity ;

architecture testbench of tb_BlockName is
    ...
    ...
begin
    ...

    -- Device Under Test (DUT)
    DUT : BlockName port map (....) ;
    ...
end architecture ;
```

*The module under test is always **instantiated** inside the testbench module, which contains **non-synthesizable code**.*

VHDL data types

Logic constants

Fundamental logic constants in VHDL are ‘1’ and ‘0’ :

***single quotes** for 1-bit (scalar) logic constants*

```
'1'    '0'
```

***double quotes** for multiple-bits (buses) logic values*

```
"0010"    "1010"
```

Built-in logic values

original VHDL-87 logic values :

- *bit for 1-bit (scalar) signals*
- *bit_vector for multiple-bits (buses) signals*

```
signal clock   : bit ;
signal oneByte : bit_vector(7 downto 0) ;
```

unresolved data types…..

not enough to describe actual digital circuits ! (e.g. high-impedance for 3-state buffers)

Extended logic values

extended VHDL logic values :

- *std_logic for single-bit signals*
- *std_logic_vector for multiple-bits signals (buses, see later)*

require external library as defined by IEEE Std. 1164

```
-- include resolved types
library IEEE ;
use IEEE.std_logic_1164.all ;

...
...

signal clock   : std_logic ;
signal oneByte : std_logic_vector(7 downto 0) ;
```

- 1 logic one
- 0 logic zero
- Z high-impedance
- U uninitialized (sim. only)
- X unknown (driven)
- - don't care
- H weak high
- L weak low
- W weak signal

Buses and endianess

Most Significant Bit (MSB) vs. Least Significant Bit (LSB)
the ordering of bits within a multi-bit string is referred to as ***endianess***
big endian :

```
signal someSignal : std_logic_vector(11 downto 0)
```

little endian :

```
signal someSignal : std_logic_vector(0 to 4)
```

Sum between vectors (counting...)

In order to use the + operator between std_logic_vector you must use specific libraries:

- *IEEE.std_logic_unsigned*
- *IEEE.numeric_std*

```
library IEEE ;
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_unsigned.all ;
```

IEEE.std_logic_unsigned is now DEPRECATED since:

- *despite the prefix IEEE they are NOT IEEE standard, but **Synopsys proprietary** !*
- *since 1995, IEEE defines **numeric types** and **arithmetic functions** in the standard package numeric_std to outline a **true standard approach** for use arithmetic operator with synthesis tools*

Signed/unsigned vectors

Extension of std_logic_vector :

- *signed*
- *unsigned*

```
library IEEE ;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all ;

...
...

signal count_s : signed(7 downto 0) := (others => '0') ;
signal count_u : unsigned(7 downto 0) := (others => '0') ;
```

The IEEE.numeric_std provides also provides functions to perform type-casting :
<= std_logic_vector(<unsigned signal>)
<= std_logic_vector(<signed signal>)

Binary codes

- *"straight" binary (the usual "power of 2")*
- *Gray code*
- *thermometer*
- *one-hot / one-cold*

Many many others : e.g. Hamming (for SEU protection)

Straight binary code

Usual binary strings interpreted as usual as "power of 2" integer numbers :

```
"000"   -- 0
"001"   -- 1
"010"   -- 2
"011"   -- 3
"100"   -- 4
"101"   -- 5
"110"   -- 6
"111"   -- 7
```

$$x = b_0 \times 2^0 + b_1 \times 2^1 + b_2 \times 2^2 + \ldots + b_{N-1} 2^{N-1} = \sum_{i=0}^{N-1} b_i \, 2^i$$

Gray code

Only one bit change from a code to the next one :

```
"000"   -- 0
"001"   -- 1
"011"   -- 2
"010"   -- 3
"110"   -- 4
"111"   -- 5
"101"   -- 6
"100"   -- 7
```

Thermometer code

Continue to "padd" the code with a 1 on the right :

```
"0000000"  -- 0
"0000001"  -- 1
"0000011"  -- 2
"0000111"  -- 3
"0001111"  -- 4
"0011111"  -- 5
"0111111"  -- 6
"1111111"  -- 7
```

One-hot code

Only one bit set to one, then moving to the left :

```
"0000000"  -- 0
"0000001"  -- 1
"0000010"  -- 2
"0000100"  -- 3
"0001000"  -- 4
"0010000"  -- 5
"0100000"  -- 6
"1000000"  -- 7
```

The complementary is sometimes called "one cold"

Integer numbers

VHDL provides 32-bit integer numbers, can be signed or unsigned :

- *integer*
- *natural*
- *positive*

```
entity Counter is

   generic (
      N : natural := 2
   ) ;

   port (
      clk   : in  std_logic ;
      count : out std_logic_vector(N-1 downto 0)
   ) ;

end entity Counter ;
```

Real

real 64-bit double-precision real numbers :
Example :
Useful for Real-Number Modeling (RNM)

```
constant VDD : real := 1.2 ;
```

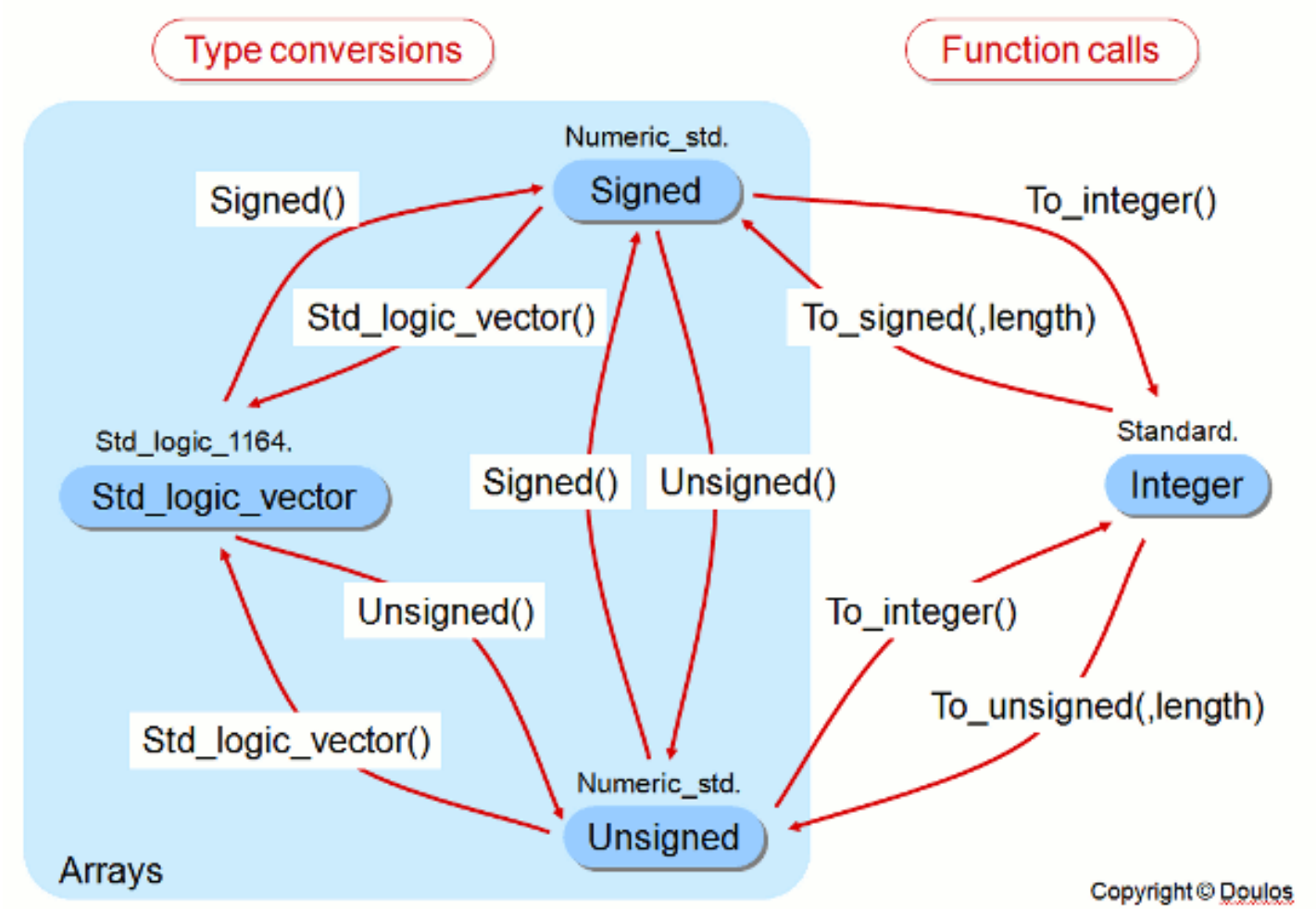
Physical data types

```
constant PERIOD : time := 10 ns ;
signal clk : std_logic ;

...

clock : process
begin
   clk <= '0' ;
   wait for PERIOD/2 ;
   clk <= '1' ;
   wait for PERIOD/2 ;
end process ;
```

Numeric Std Conversions



Composite type

A composite type is a datatype consisting of several subelements.

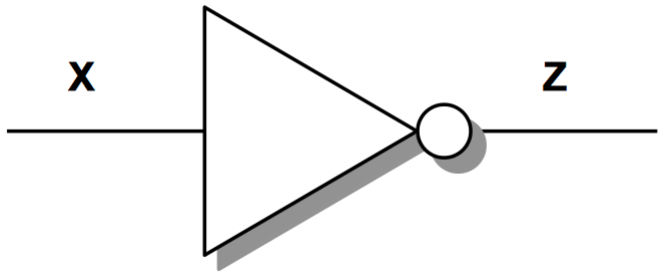
```
1  type arr_type is array (0 to 255) of real;
2
3  type rec_type is record
4    int_element : integer;
5    slv_element : std_logic;
6  end record;
7
8  signal my_arr : arr_type;
9  signal my_rec : rec_type;
10 signal my_slv : std_logic_vector(31 downto 0);
```

```
1 type example_fsm_t is (idle, starting, running, stopping);
```

```
1 -- Declare a signal which uses our custom type
2 signal example : example_fsm_t;
3 -- Example of assigning a value to our signal
4 example <= stopping;
```

Boolean algebra

NOT gate

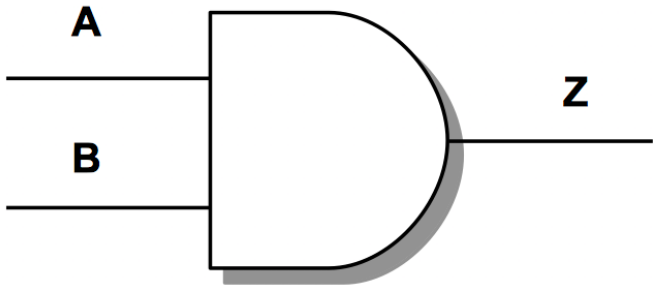


$$z = \bar{X}$$

VHDL syntax :

```
Z <= not X ;
```

AND gate

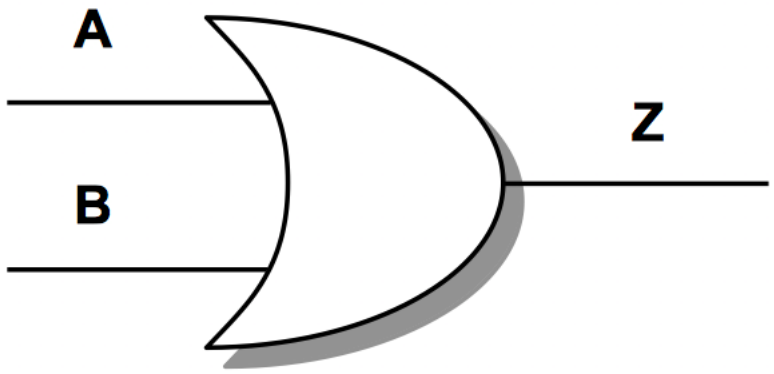


$$Z = A \cdot B$$

VHDL syntax :

```
Z <= A and B ;
```

OR gate

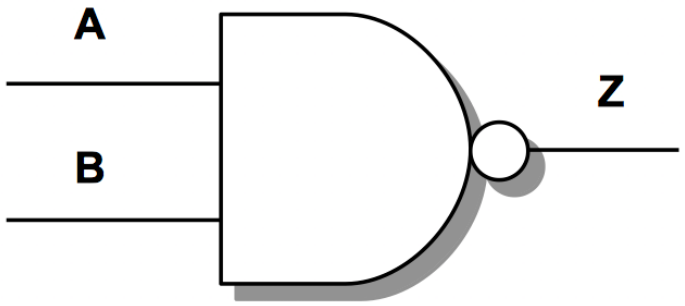


$$Z = A + B$$

VHDL syntax :

```
Z <= A or B ;
```

NAND gate

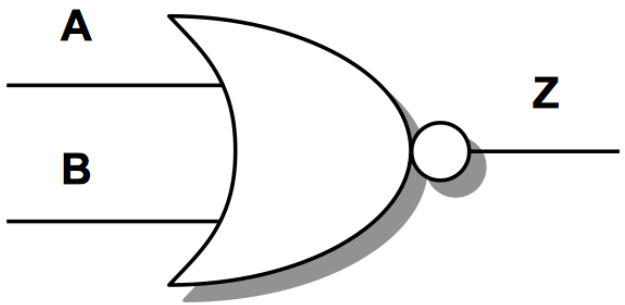


$$Z = \overline{A \cdot B}$$

VHDL syntax :

```
Z <= A nand B ;
```

NOR gate

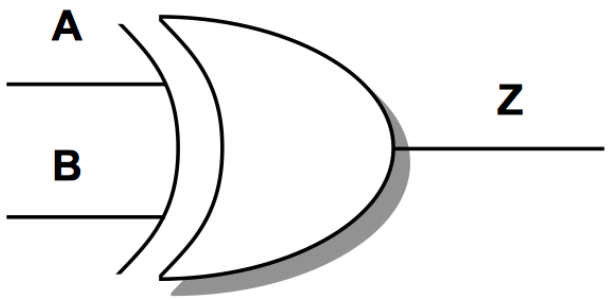


$$Z = \overline{A + B}$$

VHDL syntax :

```
Z <= A nor B ;
```

XOR gate

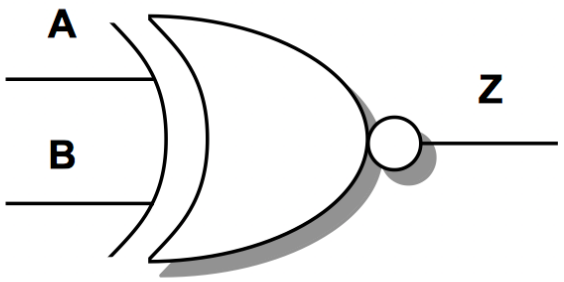


$$Z = A \oplus B = (A \cdot \overline{B}) + (\overline{A} \cdot B)$$

VHDL syntax :

```
Z <= A xor B ;
```

XNOR gate



$$Z = \overline{A \oplus B} = (A \cdot B) + (\overline{A} \cdot \overline{B})$$

VHDL syntax :

```
Z <= A xnor B ;
```

- *not implemented in the original VHDL-87*
- *later added as part of VHDL-93 revision upon designers request*
- *detects if two bits are identical*
- *fundamental component for binary comparators*

Propagation delays

VHDL syntax :

```
SUM   <= A xor B after 5 ns ;
CARRY <= A and B after 3 ns ;
```

Only for simulation purposes !

Multiple-inputs gates

```
Z <= X(0) and X(1) and X(2) and ... and X(N-1) ;
```

```
Z <= X(0) or X(1) or X(2) or ... or X(N-1) ;
```

```
Z <= X(0) nand X(1) nand X(2) nand ... nand X(N-1) ;
WRONG !
```

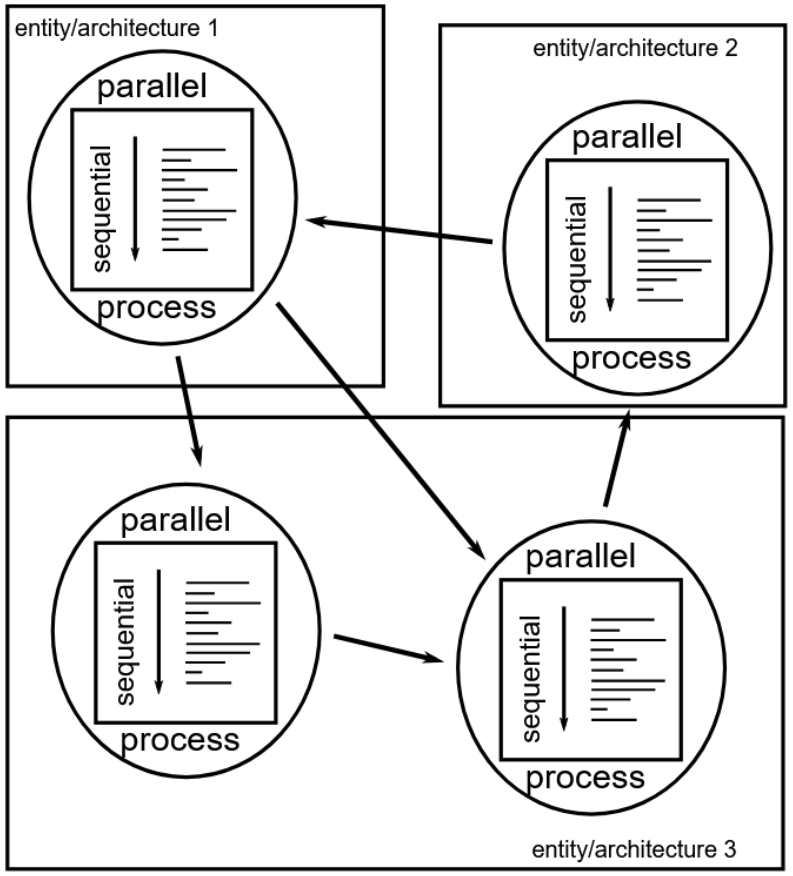
```
Z <= not( X(0) and X(1) and X(2) and ... and X(N-1)) ;
```

when/else conditional signal assignment

- A **concurrent conditional assignment** used outside processes.
- Evaluates conditions in order and assigns the first matching value.
- Commonly used for **simple combinational logic** and multiplexers.

```
b <= "1000" when a = "00" else
     "0100" when a = "01" else
     "0010" when a = "10" else
     "0001" when a = "11" ;
```

VHDL ‘process’ structure



“In VHDL, the process statement contains **sequential** statements. Processes are only permitted inside an architecture. The statements within processes execute sequentially, not concurrently.”

Combinational logic

```
1  MUX_2_1 : process(sel, in_a, in_b)
2  begin
3    case sel is
4      when '0' =>
5        out_q <= in_a;
6      when '1' =>
7        out_q <= in_b;
8      when others =>
9        out_q <= 'X';
10   end case;
11 end process;
```

Sequential logic

```
1  DELAY_PROC : process(clk)
2  begin
3    if rising_edge(clk) then
4      if rst = '1' then
5        sig_p1 <= '0';
6
7      else
8        sig_p1 <= sig;
9
10     end if;
11   end if;
12 end process;
```

If/elsif/else statement

- Used for conditional decisions with **priority**.
- Conditions are evaluated **top-to-bottom**, and the first true condition is executed.
- Commonly used in **clocked logic** and control paths.

```
31
32  entity XuLA_2 is
33      Port (  PB1             : in  STD_LOGIC;
34              PB2             : in  STD_LOGIC;
35              PB3             : in  STD_LOGIC;
36              PB4             : in  STD_LOGIC;
37              LED1            : out  STD_LOGIC;
38              LED2            : out  STD_LOGIC;
39              LED3            : out  STD_LOGIC;
40              LED4            : out  STD_LOGIC);
41  end XuLA_2;
42
43  architecture Behavioral of XuLA_2 is
44  begin
45
46     process (PB1)
47     begin
48        if PB1 = '1' then
49           LED1 <= '1';
50           LED2 <= '0';
51           LED3 <= '1';
52           LED4 <= '0';
53        else
54           LED1 <= '0';
55           LED2 <= '1';
56           LED3 <= '0';
57           LED4 <= '1';
58        end if;
59   end process;
60
61  end Behavioral;
```

Case/when statement

- Used for selecting **one of many mutually exclusive options**.
- All choices are evaluated in parallel, **no priority** between branches.
- Ideal for **state machines** and multiplexers..

```
31
32  entity XuLA_2 is
33      Port (  PB1             : in  STD_LOGIC;
34              PB2             : in  STD_LOGIC;
35              PB3             : in  STD_LOGIC;
36              PB4             : in  STD_LOGIC;
37              LED1            : out  STD_LOGIC;
38              LED2            : out  STD_LOGIC;
39              LED3            : out  STD_LOGIC;
40              LED4            : out  STD_LOGIC);
41  end XuLA_2;
42
43  architecture Behavioral of XuLA_2 is
44  begin
45
46     process (PB1)
47     begin
48        case PB1 is
49           when '1' => LED1 <= '1';
50                       LED2 <= '0';
51                       LED3 <= '1';
52                       LED4 <= '0';
53           when others => LED1 <= '0';
54                       LED2 <= '1';
55                       LED3 <= '0';
56                       LED4 <= '1';
57        end case;
58   end process;
59
60  end Behavioral;
61
```

For loop statement

- Repeats a set of sequential statements for a fixed range.
- Used for **iterating over arrays or buses**.
- Loops are **unrolled in hardware**, not executed over time.

```
1  entity T04_ForLoopTb is
2  end entity;
3
4  architecture sim of T04_ForLoopTb is
5  begin
6
7      process is
8      begin
9
10         for i in 1 to 10 loop
11             report "i=" & integer'image(i);
12         end loop;
13         wait;
14
15     end process;
16
17 end architecture;
```

Lab 6 – Inverter, logic gates and MUX

And …

test bench concept

Synthesis

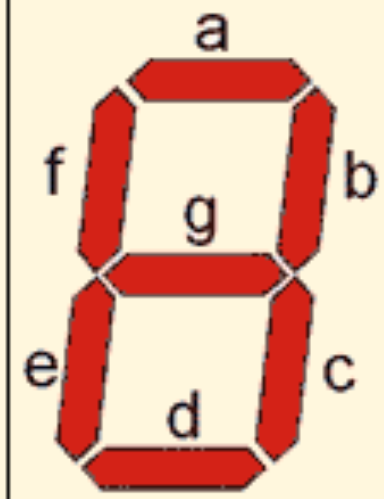
Implementation

Constraints

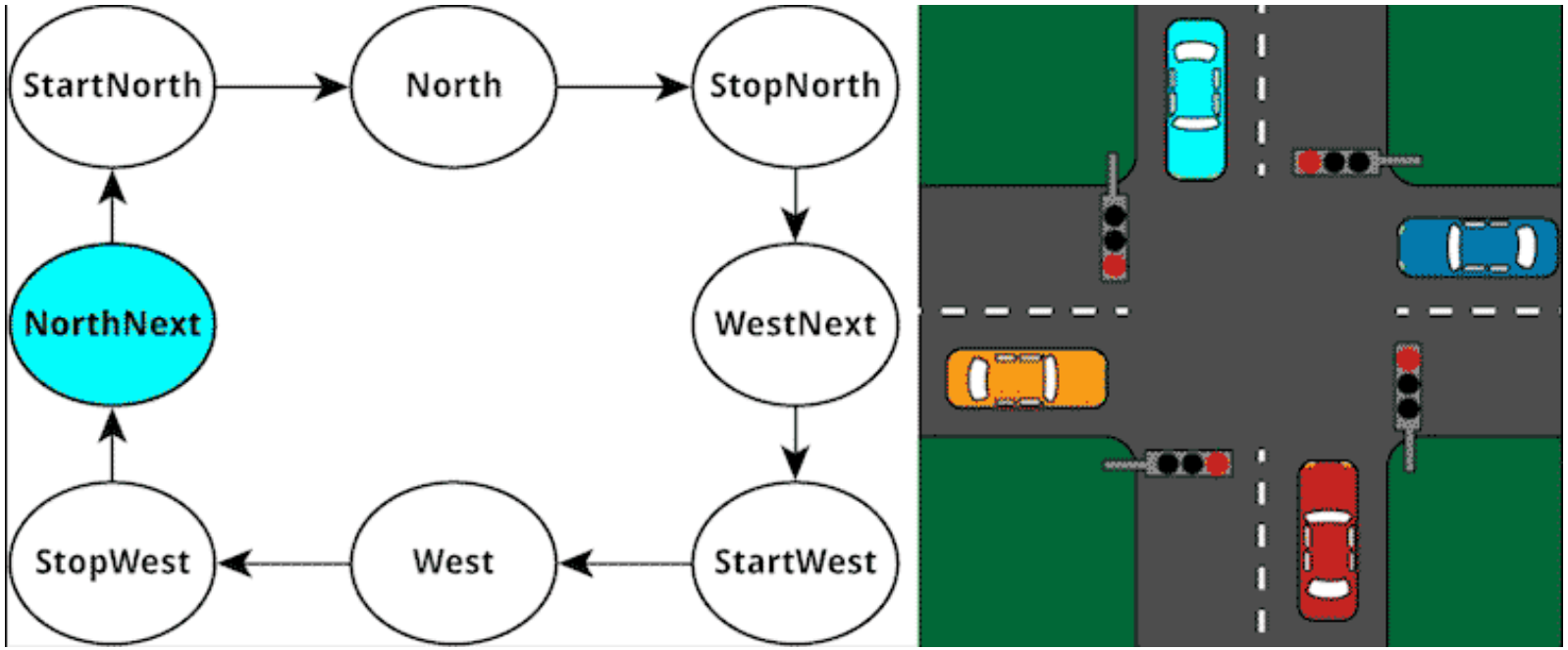
Bitfile generation

Lab 7 – BCD counter and clock divider

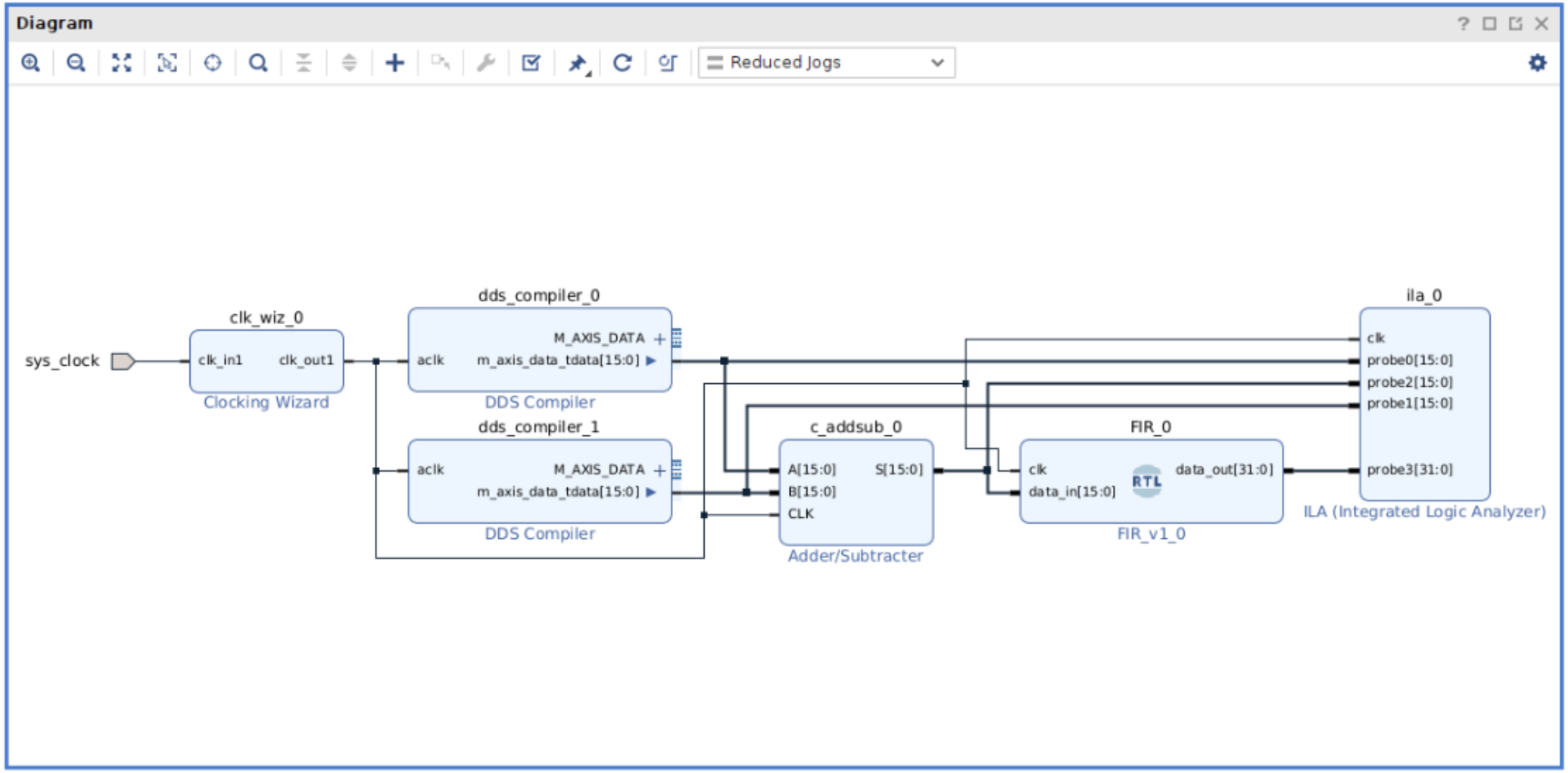
Binary Coded Decimal counter

Decimal digit	Input D C B A	Output a b c d e f g	Digit presentation
0	0 0 0 0	0 0 0 0 0 0 1	
1	0 0 0 1	1 0 0 1 1 1 1	
2	0 0 1 0	0 0 1 0 0 1 0	
3	0 0 1 1	0 0 0 0 1 1 0	
4	0 1 0 0	1 0 0 1 1 0 0	
5	0 1 0 1	0 1 0 0 1 0 0	
6	0 1 1 0	1 1 0 0 0 0 0	
7	0 1 1 1	0 0 0 1 1 1 1	
8	1 0 0 0	0 0 0 0 0 0 0	
9	1 0 0 1	0 0 0 1 1 0 0	

Lab 8 – A simple FSM

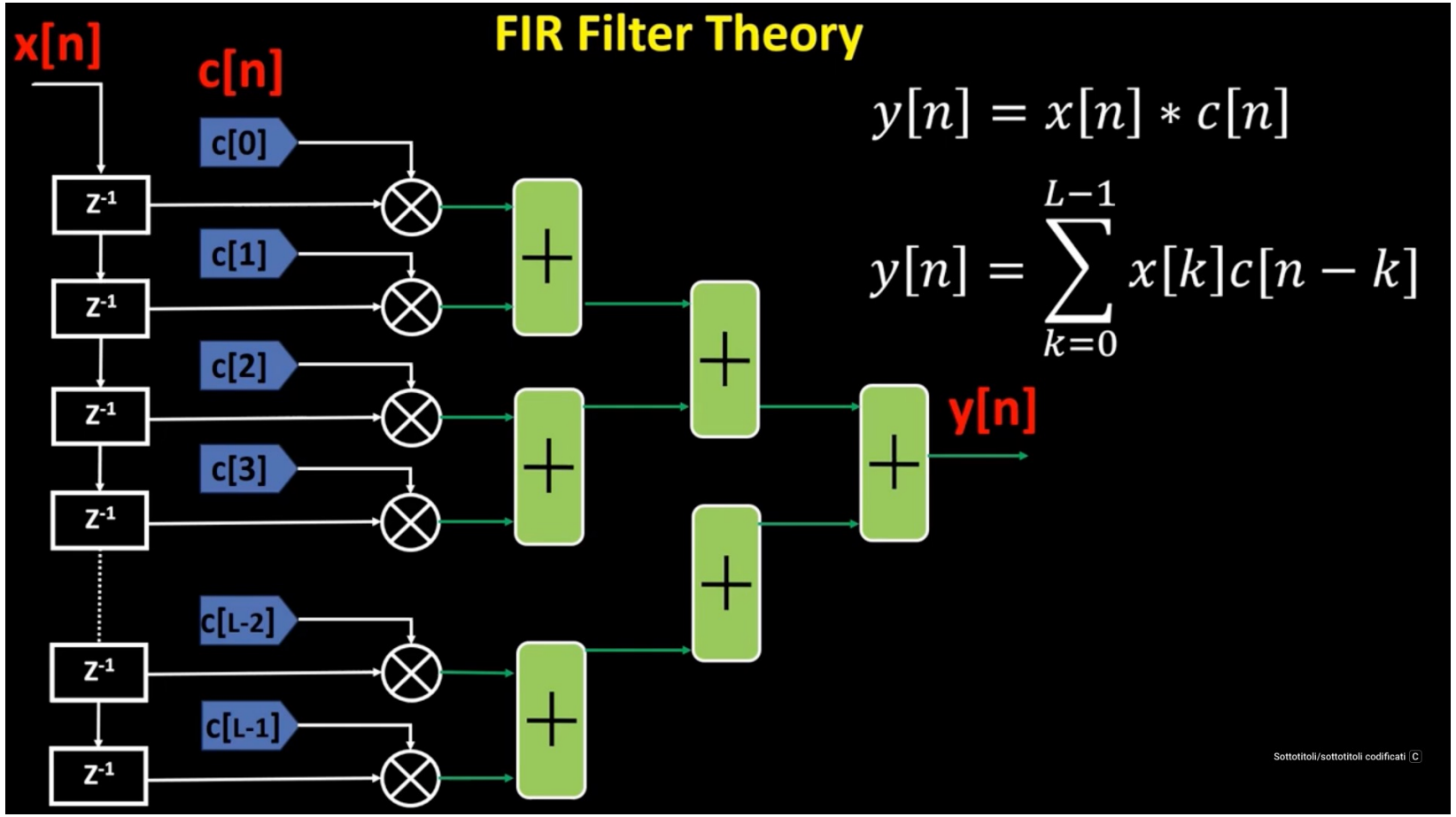


Lab 9 – FIR, DDS, ILA and Block Design



Lab 9 – FIR module

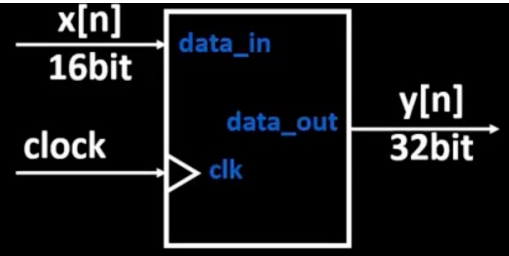
Finite impulse response filter



Lab 9 – FIR module

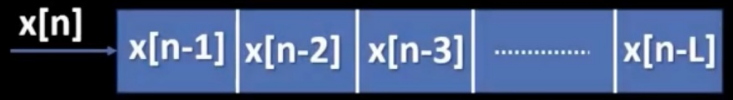
```vhdl
entity FIR_Filter is
    Port (
        clk         : in  std_logic;
        data_in     : in  std_logic_vector(15 downto 0);
        data_out    : out std_logic_vector(31 downto 0)
    );
end FIR_Filter;

architecture Behavioral of FIR_Filter is
```



➢Implement "delayed line"

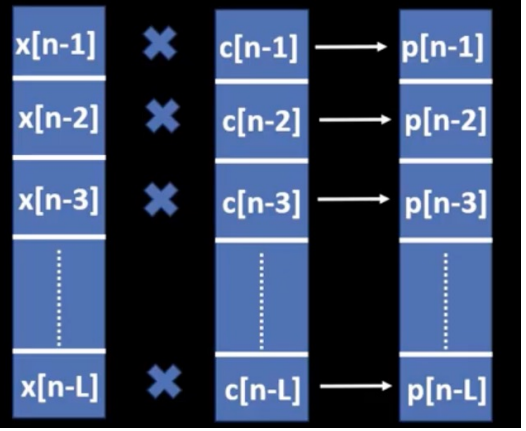
```vhdl
-- ================================================
-- Delay Line: Shifts input samples through the buffer
-- Each new input moves the existing samples one stage down
-- ================================================
process(clk)
begin
    if rising_edge(clk) then
        for i in 31 downto 1 loop
            delay_line(i) <= delay_line(i - 1);
        end loop;
        delay_line(0) <= signed(data_in);
    end if;
end process;
```



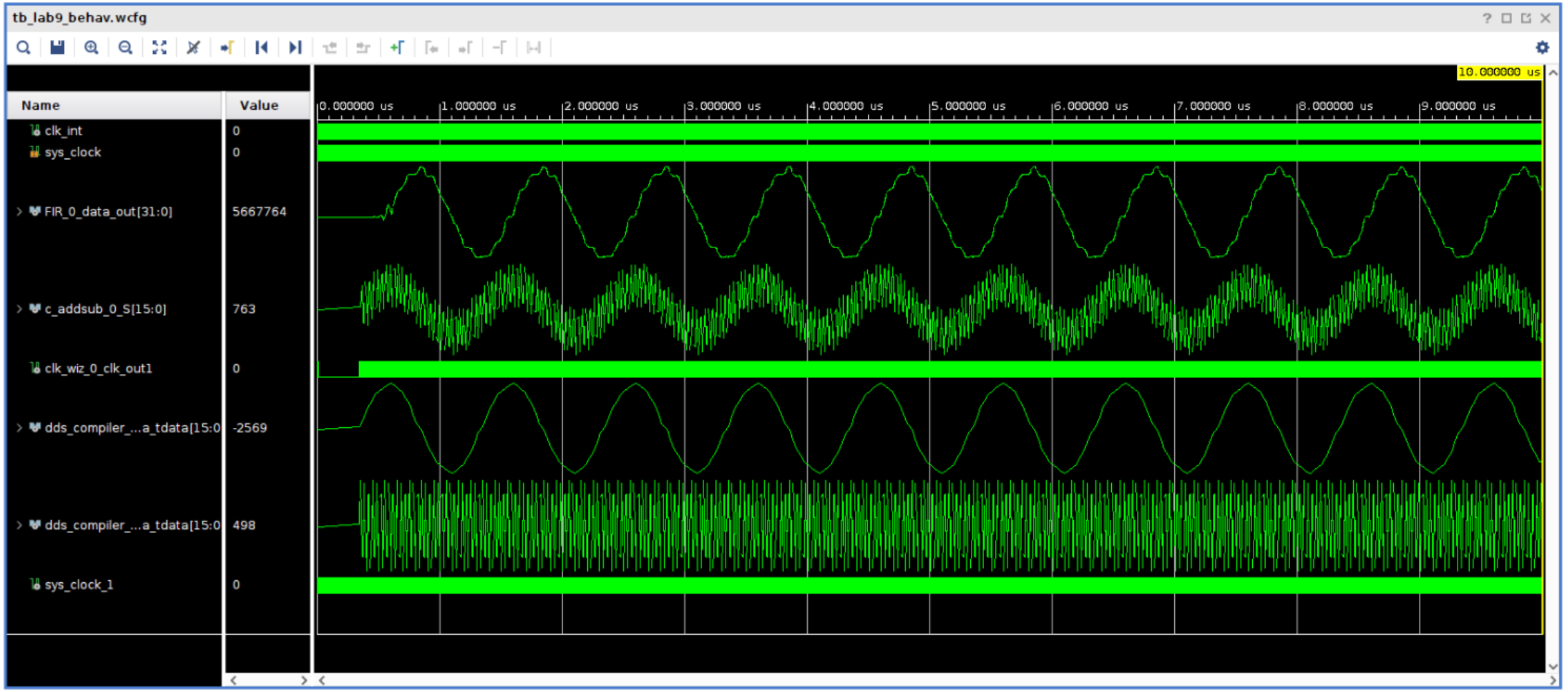
Provided VHDL code

➢Next process implements the multiplier in parallel

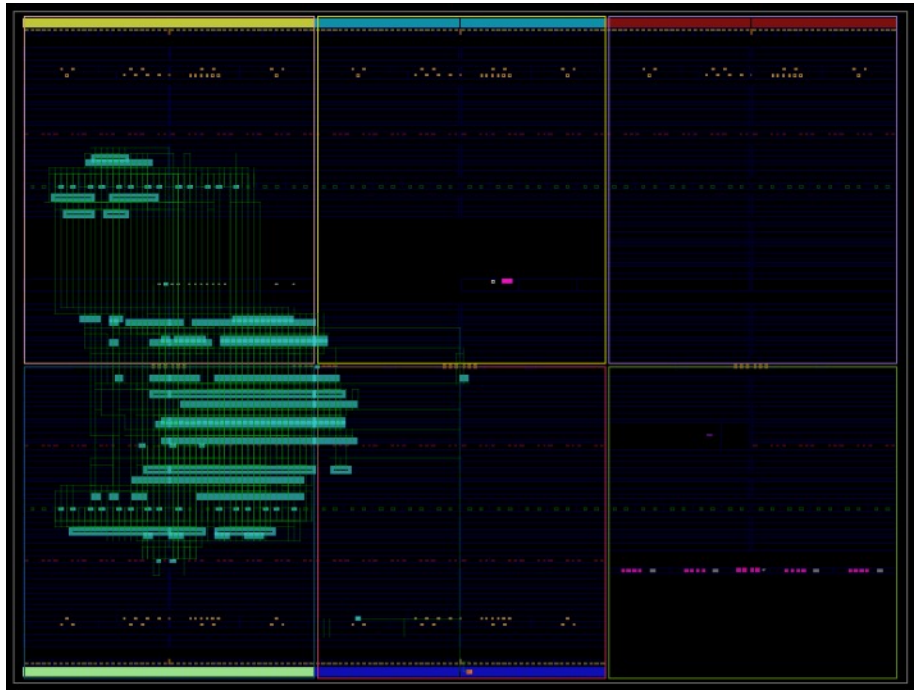
```vhdl
-- ================================================
-- Multiply Stage: Multiplies each delayed sample with its coefficient
-- Produces 32 parallel product values for summing
-- ================================================
process(clk)
begin
    if rising_edge(clk) then
        for i in 0 to 31 loop
            products(i) <= delay_line(i) * coeffs(i);
        end loop;
    end if;
end process;
```



Lab 9 – Simulation



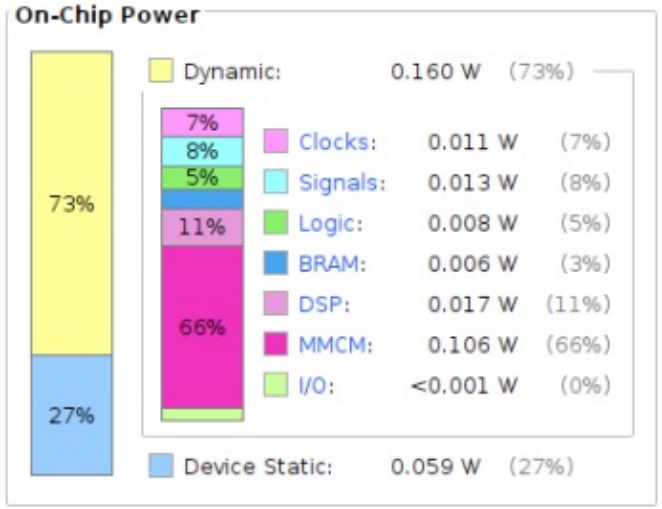
Lab 9 – Implementation and Utilization



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.219 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.0°C**
Thermal Margin:	74.0°C (15.4 W)
Effective θJA:	4.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



Utilization

Hierarchy

- Hierarchy
- Summary
- Slice Logic
  - Slice LUTs (9%)
    - LUT as Memory (2%)
      - LUT as Shift Register
      - LUT as Distributed RAM
    - LUT as Logic (8%)
  - F7 Muxes (<1%)
  - Slice Registers (7%)
    - Register as Flip Flop (7%)
- Slice Logic Distribution
  - Slice (9%)
    - SLICEM

Name	Slice LUTs	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	DSPs (90)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (5)	BSCA
lab9_block_design_wrapper	1806	2913	3	761	1603	203	4.5	28	1	3	1	
lab9_block_design_i (lab9_block_design)	1358	2172	3	555	1179	179	4.5	28	0	2	1	
ila_0 (lab9_block_design_ila_0_0)	863	1502	3	393	724	139	4.5	0	0	0	0	
FIR_0 (lab9_block_design_FIR_0_0)	437	559	0	143	397	40	0	28	0	0	0	
dds_compiler_1 (lab9_block_design_dds	22	47	0	14	22	0	0	0	0	0	0	
dds_compiler_0 (lab9_block_design_dds	20	48	0	12	20	0	0	0	0	0	0	
c_addsub_0 (lab9_block_design_c_adds	16	16	0	4	16	0	0	0	0	0	0	
clk_wiz_0 (lab9_block_design_clk_wiz_0_(	0	0	0	0	0	0	0	0	0	2	1	
dbg_hub (dbg_hub)	448	741	0	214	424	24	0	0	0	1	0	

utilization_1