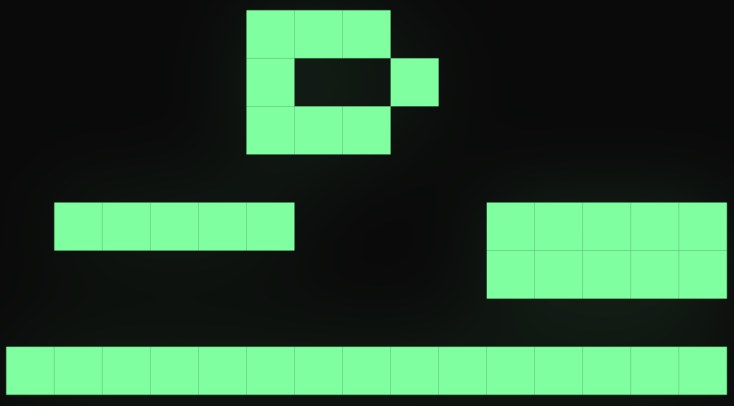
v3.0

Electrical Engineering for Computer Scientists



11	91	140	52	6
PHASES	MODULES	LESSONS	PROJECTS	CAPSTONES

BY SVX

Table of Contents

How to Use This Curriculum 11

The CS-Bridge Pattern . . . 12

The Project Model — Hardware + Simulation . . . 12

The Toolchain . . . 12

Pacing the 12 Months . . . 13

Prerequisites Checklist . . . 13

A Note on the Bench . . . 14

Symbols and Conventions . . . 14

Phase 1 — DC Circuit Analysis 15

Module 1.1 — Charge, Current, Voltage, Power, Energy . . . 15

Reference directions and the sign convention . . . 16

Power — sources vs. sinks . . . 16

Module 1.2 — Ohm's Law and Kirchhoff's Laws . . . 17

Ohm's law — the constitutive relation . . . 17

KCL — current conservation . . . 18

KVL — voltage conservation . . . 18

Solving circuits by hand: the systematic method . . . 18

Module 1.3 — Node Voltage & Mesh Current Methods . . . 19

Supernodes . . . 19

Supermeshes . . . 20

Dependent sources . . . 20

Module 1.4 — Thevenin & Norton Equivalent Circuits . . . 20

Finding V_th and R_th . . . 21

Maximum power transfer . . . 21

Module 1.5 — Superposition & Source Transformation . . . 22

Linearity is the whole game . . . 22

Module 1.6 — Capacitors, Inductors & Transient Analysis . . . 23

Capacitors and inductors as energy stores . . . 24

First-order transients — the RC and RL step response . . . 25

Second-order transients — the RLC response . . . 25

Phase 2 — AC Circuit Analysis 27

Module 2.1 — Sinusoidal Sources and Phasors . . . 27

The phasor transform . . . 28

Why it works — the eigenfunction property . . . 28

Module 2.2 — Impedance and Admittance . . . 29

Series and parallel combinations . . . 29

Module 2.3 — AC Power: Real, Reactive, Apparent, Power Factor . . . 30

The three powers . . . 30

Power factor correction . . . 31

Module 2.4 — Resonance, Bode Plots & Filters . . . 31

Bode plots — the frequency-domain picture . . . 33

Module 2.5 — Three-Phase Systems . . . 34

Module 2.6 — Mutual Inductance & Transformers . . . 35

The ideal transformer . . . 35

Real transformers . . . 35

Phase 3 — Analog Electronics 37

Module 3.1 — Diodes: Rectifiers, Clippers, Clampers . . . 37

Three classic diode circuits . . . 38

Module 3.2 — BJT Biasing and Small-Signal Model . . . 39
BJT regions of operation . . . 39
Biasing — setting the operating point . . . 40
Small-signal model — the hybrid- . . . 40
Module 3.3 — MOSFET Biasing and Small-Signal Model . . . 41
Small-signal model . . . 41
Why MOSFETs dominate modern electronics . . . 41
Module 3.4 — Operational Amplifiers: The Analog Swiss Army Knife . . . 42
The two golden rules . . . 43
Five canonical op-amp circuits . . . 44
Integrator and differentiator . . . 44
Real op-amp non-idealities . . . 44
Module 3.5 — Active Filters: Sallen-Key and MFB . . . 45
Sallen-Key low-pass . . . 45
Butterworth, Chebyshev, Bessel — choosing the response . . . 46
Module 3.6 — Oscillators and Timers: Wien Bridge, Phase-Shift, 555 . . . 47
Three classic oscillators . . . 47
The 555 timer — the most popular IC ever . . . 47
Module 3.7 — Linear Regulators and LDOs . . . 48
Module 3.8 — Phase 3 Capstone: Audio Amplifier . . . 49
Phase 4 — Digital Logic & Embedded Systems 51
Module 4.1 — Boolean Algebra and K-maps . . . 51
K-map mechanics . . . 52
Module 4.2 — Combinational Logic: Mux, Decoder, Adder . . . 53
Combinational hazards . . . 54
Module 4.3 — Sequential Logic: Flip-Flops, Counters, Registers . . . 54

Four flip-flop types . . . 55
Setup, hold, and metastability . . . 55
Counters and shift registers . . . 56
Module 4.4 — HDL Intro: Verilog and FSMs . . . 56
Verilog in one page . . . 57
A counter in Verilog . . . 57
Module 4.5 — FPGA Basics with Vivado and Artix-7 . . . 58
The FPGA design flow . . . 58
Module 4.6 — Microcontrollers: Arduino to STM32 . . . 59
The Arduino abstraction . . . 59
STM32 — the real-world MCU . . . 59
Clock trees and power management . . . 60
Module 4.7 — Interrupts, Timers, ADC, DAC, PWM . . . 60
Interrupts and priorities . . . 61
Timers — the workhorse peripheral . . . 61
ADCs — sampling the analog world . . . 61
Module 4.8 — Communication: UART, SPI, I2C, CAN . . . 62
I2C — addressing and arbitration . . . 63
Module 4.9 — ESP32 and Wireless Connectivity . . . 64
WiFi on ESP32 . . . 64
Over-the-air updates (OTA) . . . 64
Module 4.10 — Phase 4 Capstone: Digital FSM + MCU Driver . . . 65
Phase 5 — Signals, Systems & DSP 67
Module 5.1 — Continuous-Time Signals and Systems . . . 67
Impulse response and convolution . . . 67

Module 5.2 — LTI Systems: Convolution and Eigenfunctions . . . 69
Module 5.3 — Fourier Series and CTFT . . . 69
Fourier transform pairs you must know . . . 69
Properties that make Fourier useful . . . 70
Module 5.4 — Laplace Transform and the s-Domain . . . 70
Why Laplace? Differential equations become algebra . . . 71
Poles, zeros, and stability . . . 71
Laplace pairs you must know . . . 71
Module 5.5 — Sampling Theorem and Discrete-Time Signals . . . 72
Anti-alias filtering . . . 72
Reconstruction — the DAC and the zero-order hold . . . 72
Module 5.6 — Z-Transform and Discrete-Time Systems . . . 73
Difference equations and transfer functions . . . 74
Module 5.7 — DTFT, DFT, and FFT . . . 74
DFT definition . . . 75
FFT — the algorithm . . . 76
Module 5.8 — FIR and IIR Filter Design . . . 76
FIR design — the windowed sinc method . . . 76
IIR design — bilinear transform of analog prototypes . . . 77
Choosing FIR vs IIR . . . 77
Module 5.9 — Adaptive Filters (Intro) . . . 78
Module 5.10 — Phase 5 Capstone: Software-Defined Audio . . . 79

Phase 6 — Control Systems & Robotics 80

Module 6.1 — Feedback, Modeling, Transfer Functions . . . 80
Modeling physical systems . . . 81
Module 6.2 — Stability: Routh-Hurwitz and Pole Locations . . . 81

Routh-Hurwitz procedure . . . 81
Module 6.3 — Time-Domain Response and Steady-State Error . . . 82
Steady-state error and system type . . . 82
Module 6.4 — Root Locus Design . . . 83
Module 6.5 — Frequency Response: Bode, Nyquist, Margins . . . 84
Module 6.6 — PID Design and Tuning . . . 85
Ziegler-Nichols tuning . . . 85
Module 6.7 — State-Space: Controllability and Observability . . . 86
LQR — Linear Quadratic Regulator . . . 87
Module 6.8 — Digital Control: Sampled-Data Systems . . . 87
Module 6.9 — Robotics Kinematics . . . 88
Forward kinematics with DH parameters . . . 88
Inverse kinematics — the hard problem . . . 88
Module 6.10 — Phase 6 Capstone: Line-Following Robot . . . 89
Phase 7 — Power Electronics & Machines 91
Module 7.1 — Power Semiconductor Devices . . . 91
Why GaN and SiC are eating silicon . . . 92
Switching loss and gate drive . . . 92
Module 7.2 — Non-Isolated DC-DC Converters . . . 92
Buck converter — the canonical design . . . 93
Continuous vs discontinuous conduction . . . 93
Synchronous rectification . . . 93
Module 7.3 — Isolated Converters: Flyback, Forward, Bridge . . . 94
Three isolated topologies . . . 94
Transformer design — the hard part . . . 95

Module 7.4 — Inverters: Square, PWM, SPWM, SVPWM . . . 95
Space Vector PWM (SVPWM) — the three-level trick . . . 96
Module 7.5 — Rectifiers and Power Factor Correction . . . 97
Boost PFC . . . 97
Module 7.6 — DC Machines . . . 97
Four DC motor types . . . 97
Module 7.7 — Induction Machines . . . 98
Equivalent circuit . . . 98
Starting methods . . . 98
Module 7.8 — Synchronous Machines and BLDC/PMSM . . . 99
BLDC commutation . . . 99
PMSM and Field-Oriented Control (FOC) . . . 99
Module 7.9 — Motor Drives: V/f and FOC . . . 101
V/f control for induction motors . . . 101
FOC for induction and PMSM . . . 101
Module 7.10 — Phase 7 Capstone: BLDC FOC Driver . . . 101

Phase 8 — EM Fields, RF & Communications 103

Module 8.1 — Vector Calculus for EM . . . 103
Three operators, three theorems . . . 103
Module 8.2 — Electrostatics: Gauss, Boundary Conditions . . . 104
Capacitance and dielectrics . . . 104
Boundary conditions . . . 104
Module 8.3 — Magnetostatics: Biot-Savart, Ampère . . . 105
Inductance and magnetic energy . . . 105
Module 8.4 — Maxwell’s Equations and EM Waves . . . 105
Plane waves and polarization . . . 106

The EM spectrum . . . 106

Module 8.5 — Transmission Lines and the Smith Chart . . . 106

Reflections and matching . . . 107

The Smith chart — a graphical calculator for RF . . . 107

Module 8.6 — Antennas: Dipole, Patch, Gain, Pattern . . . 108

Antenna parameters . . . 108

The Friis transmission equation . . . 109

Module 8.7 — RF Front-End: LNA, Mixer, Oscillator, PLL . . . 109

The Friis cascade — first stage dominates . . . 109

Mixers — frequency translation . . . 110

Module 8.8 — Modulation: AM, FM, PM, ASK, FSK, PSK, QAM . . . 110

Analog modulation — AM and FM . . . 110

Digital modulation — constellation diagrams . . . 110

Module 8.9 — Phase 8 Capstone: FM Transmitter + SDR Receiver . . . 112

Phase 9 — VLSI & IC Design 114

Module 9.1 — CMOS Inverter: Static and Dynamic Behavior . . . 114

Voltage transfer characteristic (VTC) . . . 116

Dynamic behavior — propagation delay and power . . . 116

Module 9.2 — CMOS Layout and Fabrication Flow . . . 116

The CMOS fabrication flow . . . 117

Module 9.3 — Combinational CMOS Design and Gate Sizing . . . 117

Gate sizing and logical effort . . . 118

Module 9.4 — Sequential CMOS: Latches and Flip-Flops . . . 118

Setup, hold, and clock-to-Q . . . 118

Module 9.5 — Static Timing Analysis . . . 119

Timing corners . . . 119

Module 9.6 — Low-Power Design . . . 120

Module 9.7 — ASIC vs FPGA Design Flow . . . 120

Module 9.8 — Phase 9 Capstone: Pipelined Adder + Custom Inverter . . . 121

Phase 10 — Power Systems, Energy & Capstones 122

Module 10.1 — Power Generation . . . 122

Module 10.2 — Transmission and Distribution . . . 123

Why high voltage for transmission? . . . 123

HVDC — for very long distances . . . 123

Module 10.3 — Power System Protection . . . 124

Protection principles . . . 124

Common protection schemes . . . 124

Module 10.4 — Power Quality and Grid Stability . . . 124

Grid stability — three timescales . . . 125

Module 10.5 — Smart Grid, Microgrids, DER Integration . . . 125

DER integration challenges . . . 126

Module 10.6 — Energy Storage and Battery Management . . . 126

SOC and SOH estimation . . . 126

Module 10.7 — Solar PV Systems and MPPT . . . 126

PV system topologies . . . 128

Module 10.8 — Wind Energy Systems . . . 129

Three wind turbine types . . . 129

Module 10.9 — EV Charging Infrastructure . . . 129

Charging standards . . . 129

Grid impact of fast charging . . . 130

Module 10.10 — The Six Capstones 130

Closing the Curriculum 132

Appendix A — Toolchain Setup Guide 134

A.1 LTspice 134

A.2 KiCad 8 134

A.3 Arduino IDE 134

A.4 ESP32 (PlatformIO) 134

A.5 STM32CubeIDE 134

A.6 Python (NumPy/SciPy/matplotlib/control) 134

A.7 Octave 134

A.8 Vivado WebPACK 135

A.9 Magic VLSI 135

A.10 GNU Radio 135

Appendix B — Hardware Bill of Materials 136

B.1 Starter Kit (~$150) 136

B.2 Full Kit (~$600) 136

Appendix C — CS-to-EE Concept Map 138

Appendix D — Math Reference and Formula Sheets 139

D.1 Trigonometric identities 139

D.2 Derivatives and integrals used in EE 139

D.3 Laplace transform pairs 139

D.4 Z-transform pairs 139

D.5 Vector calculus operators 140

Appendix E — 12-Month Pacing Calendar 141

How to Use This Curriculum

This curriculum is a complete, rigorous, project-based pathway from a working knowledge of computer science to a working knowledge of electrical engineering. It is not a survey. It is not a list of topics to look up later. It is a 12-month plan that mixes mathematics, simulation, and bench work in roughly equal measure, so that by the time you finish Phase 10 you can specify, design, build, debug, and document a real electrical system — a motor controller, a solar inverter, a custom PCB product, a robotics platform, a smart-grid demo, or a software-defined radio front-end.



VOLT NARRATES

Hi, I'm Volt. I'll be your guide. You're a programmer. Now you want to learn circuits. Good news: you already know most of it. A circuit is just a data structure. A signal is a function. A feedback loop is a recurrence. Same ideas, different clothes. I'll point out the connections as we go. I'll also tell you when the analogy stops working. That's where the real learning lives. Read my callouts. They are not optional.



AMP ASKS

And I'm Amp. I'll ask the dumb questions so you don't have to.



VOLT SAYS

They're not dumb. They're the questions a programmer would actually ask.

The structure is hierarchical and consistent. **Phases** are the ten top-level blocks; each phase runs four to eight weeks. A phase contains several **modules**; each module is sized for roughly four to ten hours of focused study. Every module breaks down into **lessons** (the conceptual reading), one or more **hands-on projects** (simulation and/or hardware), and a **checkpoint** (self-test questions you must be able to answer cold before moving on). Six **capstone projects** in Phase 10 integrate everything you have learned into shipping-grade hardware.

A note on the visual style. Throughout the curriculum you will see small green pixel-art diagrams — Minecraft-style voxel renderings of resistors, capacitors, op-amps, Smith charts, Bode plots, PID loops, and so on. These are not decorations. They are schematic icons that give you a fast visual handle on the circuit or concept being discussed. Treat each one as a one-glance mental model: when you see the pixel rectifier, recall the diode+load topology; when you see the pixel Bode plot, recall the flat-then-rolloff shape. The cover uses the same voxel language at large scale; the body uses it as inline iconography.

> "A computer scientist learning EE is not starting from zero — you are starting from a vantage point most EE freshmen never reach: you already think in systems."
>
> — Volt, opening the curriculum

The CS-Bridge Pattern

You are a computer scientist. That is not a handicap — it is leverage. Almost every EE concept has a precise analog in computer science, and once you see the mapping, you can learn EE three to five times faster than a freshman would. Throughout this curriculum, every module opens with a small yellow **CS BRIDGE** callout that names the CS concept you already know and the EE concept it maps to. Use these bridges as your primary scaffolding — when an equation looks alien, ask first what data structure or algorithm it corresponds to in code.

CS BRIDGE

FSM sequential circuit. A finite-state machine in a compiler course and a Mealy/Moore machine built from flip-flops are the same mathematical object. Once you accept this, sequential circuit design becomes 'compile your state graph into gates and a register'. The same is true for FFT DCT (both are fast orthogonal transforms using butterfly factorization), for state-space linear dynamical systems (both are first-order matrix recurrences), for recursion RLC transients (both are solutions to a linear recurrence relation), and for pipelining CPU pipeline (both are registered data-flow graphs optimizing throughput at the cost of latency).



VOLT SAYS

You'll see yellow CS BRIDGE boxes at the start of each module. They are the most useful thing in this book. They take a CS idea you already know and pair it with an EE idea. When an equation looks weird, find the bridge first. The math is probably something you've already coded.



AMP ASKS

So like a Rosetta Stone? CS on one side, EE on the other?

The Project Model — Hardware + Simulation

Every module has at least one hands-on project, and the projects are deliberately split between simulation and hardware. Simulation lets you test ideas fast and without burning parts; hardware teaches you what simulation hides — parasitics, noise, grounding, probe loading, real-component tolerance, and the thousand small ways a bench can lie to you. A module that is simulation-only is a partial module. A module that is hardware-only is missing the math. Plan to spend roughly one-third of your weekly EE time at the bench.

The Toolchain

The curriculum standardizes on free or low-cost tools so nothing blocks you. **LTspice** is the SPICE simulator of record — free, fast, and the industry default for analog. **KiCad** is the open-source PCB design suite; we use it for schematic capture and PCB layout from Phase 3

onward. **Arduino IDE** introduces embedded work in Phase 4; **ESP32** adds wireless; **STM32** (via STM32CubeIDE or PlatformIO) is the real-world MCU you will use for motor control and capstones. **Python** with NumPy, SciPy, matplotlib, and control handles signal processing, control system design, and data analysis — leveraging what you already know. **Octave** (the free MATLAB clone) is used where the EE literature expects MATLAB syntax. **Vivado WebPACK** and a cheap Artix-7 board cover FPGA work. **Magic VLSI** introduces IC layout. Full installation instructions are in Appendix A.

Pacing the 12 Months

The curriculum is calibrated for roughly 12 months at 10–15 hours per week. If you can sustain 15 hours per week, you will finish in 12. At 10 hours per week, plan for 15–16 months. The pacing is also flexible: phases can be interleaved if you prefer to work on theory and bench in parallel. The 12-month calendar in Appendix E maps every module to a specific week, with review weeks and exam-style self-tests built in. Treat the calendar as a soft target, not a hard deadline — the only hard rule is that you must complete every checkpoint before advancing.

Phase	Topic	Weeks	Hours/Week	Cum.
0	Foundations & Math Bridge	4	12	48
1	DC Circuit Analysis	5	12	108
2	AC Circuit Analysis	5	12	168
3	Analog Electronics	6	12	240
4	Digital Logic & Embedded	6	12	312
5	Signals, Systems & DSP	7	12	396
6	Control Systems & Robotics	6	12	468
7	Power Electronics & Machines	7	12	552
8	EM Fields, RF & Communications	6	12	624
9	VLSI & IC Design	5	12	684
10	Power Systems, Energy & Capstones	8	15	804
—	Total	65	—	~804 h

That is roughly 800 hours of focused work. A four-year EE B.Sc. contains about 1,500 hours of in-major study; you are covering the practical core in a little over half the time, because you are skipping the general-education requirements and because your CS background does real work for you.

Prerequisites Checklist

Before you start Phase 0, you should be comfortable with the following. If any item is shaky, the Phase 0 math bootcamp will patch it, but expect to spend extra hours there.

- **Programming:** Python or C/C++ at an intermediate level. You should be able to write a 200-line program with no help.
- **Discrete math:** sets, functions, relations, induction, basic graph theory, modular arithmetic.
- **Linear algebra:** vectors, matrices, matrix multiplication, inverses, determinants, eigenvalues (conceptually).
- **Single-variable calculus:** derivatives and integrals of polynomials, exponentials, and trig functions; the chain rule; u-substitution.
- **Basic physics:** what charge, current, voltage, and a magnetic field are conceptually. We will formalize this in Phase 0.
- **Comfort at a terminal:** installing packages, editing config files, running scripts.

A Note on the Bench

Hardware work costs money. The starter bill of materials in Appendix B comes to about $150 — breadboard, DMM, resistors/caps/inductors, OpAmps, transistors, 555, regulators, Arduino Uno, ESP32 devkit, jumper wires, and a small bench supply. The full kit (~$600) adds an STM32 Nucleo, a BLDC motor with driver, an oscilloscope (or a DSO138 kit), an FPGA board, and a bench load. You can complete Phases 0–3 with the starter kit alone. From Phase 4 onward you will want to extend. Do not buy everything at once — buy what the current phase needs, and only after you have understood why you need it.

Symbols and Conventions

Throughout this curriculum, the following conventions are used. They are standard EE conventions, but stating them once prevents confusion later.

- **Units:** SI throughout. V (volt), A (ampere), (ohm), F (farad), H (henry), W (watt), J (joule), Hz (hertz), Wb (weber), T (tesla).
- **Lowercase v(t), i(t):** time-varying voltage and current. **Uppercase V, I:** DC or RMS values.
- **Boldface v, i, V, I:** phasor or vector quantities.
- **j** is the imaginary unit (not **i**, which is reserved for current).
- **Reference directions:** passive sign convention — current enters the positive voltage terminal of a device.
- **Ground symbol:** circuit common, not necessarily earth ground.
- **Arrow ↑/↓** in figures indicates assumed positive direction of current.
- **f' = df/dt, f" = d^2f/dt^2** — dot notation is used for time derivatives when the variable name is short.



Phase 1 — DC Circuit Analysis

Duration: 5 weeks · **Modules:** 7 · **Goal:** master the analytical machinery of DC circuits — Ohm, Kirchhoff, nodal/mesh analysis, Thevenin/Norton, superposition, and the transient behavior of energy-storage elements.

DC circuit analysis is the foundation of all EE. Every later topic — AC, analog electronics, power, even RF — is a generalization of DC analysis. The mental model is simple: charges move through conductors in response to electric fields, and the movement is constrained by the topology of the circuit. The math that emerges from this picture is graph theory plus linear algebra, which you already know. The work of this phase is to make that mapping automatic.



VOLT NARRATES

Welcome to circuits. You already know this stuff. You just don't know you know it. A wire is a variable. A resistor is a throttle. A capacitor is a buffer. By the end of this phase, circuits will feel as natural as code.

Module 1.1 — Charge, Current, Voltage, Power, Energy

Duration: 4–6 h · **Difficulty:** Foundation

CS BRIDGE

Charge packet, current packet rate, voltage pressure. Charge (coulomb, C) is the count of electrons that have moved past a point. Current (ampere, A = C/s) is the rate of that flow. Voltage (volt, V = J/C) is the energy per unit charge — the electrical analog of pressure. Power (watt, W = J/s = V·A) is the rate of energy delivery. Energy (joule, J = W·s) is the integral of power over time. These five quantities are the entire vocabulary of circuit analysis — memorize their definitions and units cold.

The passive sign convention is the rule that keeps power accounting consistent. For any two-terminal device, label one terminal + and the other −. The current i is defined as entering the + terminal. Then $p = v \cdot i$ is the power **absorbed** by the device. If $p > 0$, the device consumes power (resistor, charging capacitor, charging inductor). If $p < 0$, the device delivers power (battery, discharging cap, op-amp output). Sum the powers over all elements in a circuit and you must get zero — this is Tellegen's theorem, the circuit-level statement of energy conservation.



VOLT SAYS

Pick a direction for your current arrow. Any direction. It doesn't matter. If you guess wrong, the answer comes out negative. That's fine. Negative just means the current goes the other way. The math fixes your guess for you.



AMP ASKS

So I can pick directions randomly and the math sorts it out?



VOLT SAYS

Yes. As long as you're consistent. The only sin is not labeling at all.

$$Q = \int i\,dt \quad | \quad V = W/Q \quad | \quad P = V{\cdot}I = I^2{\cdot}R = V^2/R \quad | \quad W = \int P\,dt$$

> "Voltage is energy per unit charge. Current is charge per unit time. Power is energy per unit time. Everything else in circuit analysis is bookkeeping on these three definitions."
>
> — Volt, on the five fundamental quantities

Reference directions and the sign convention

Reference directions are arbitrary; the math tells you whether your guess was right. You label a current arrow in some direction and assign + and − to the device terminals. Then you solve. If the answer is positive, your arrow was pointing in the actual direction of positive charge flow. If negative, the actual flow is opposite. This freedom matters because complex circuits have dozens of unknown currents, and you cannot guess them all correctly. The convention is: pick a direction, write the equation, let the math correct you.

Power — sources vs. sinks

A voltage source delivering current from its + terminal is supplying power $p = -v{\cdot}i$ (with passive sign convention). A resistor always absorbs power $p = i^2{\cdot}R > 0$ — there is no such thing as negative resistance in a passive resistor (active circuits can synthesize negative resistance, and we will meet that idea in Phase 3 with op-amps). A capacitor stores energy $W = \tfrac{1}{2}{\cdot}C{\cdot}V^2$ in its electric field; an inductor stores energy $W = \tfrac{1}{2}{\cdot}L{\cdot}I^2$ in its magnetic field. These storage terms are the bridge to the transients you will study in Module 1.6.

HANDS-ON PROJECT — Bench: Measure Power in a 3-Resistor Circuit

Goal: verify power conservation by direct measurement.

Circuit: 5V supply 220 330 470 to ground (series string).

Measure: voltage across and current through each resistor with the DMM.

Compute: P = V·I for each resistor; sum. Compare to V_supply · I_total.

Pass: the two totals agree within DMM tolerance (2% error).

CHECKPOINT — Can you answer these?

- Define charge, current, voltage, power, and energy. Give the SI unit of each.
- State the passive sign convention. What does p > 0 mean? p < 0?
- How much energy is stored in a 100 F cap charged to 12 V? In a 10 mH inductor carrying 2 A?
- Why is the sum of powers in any circuit exactly zero?

Module 1.2 — Ohm's Law and Kirchhoff's Laws

Duration: 6–8 h · **Difficulty:** Foundation

CS BRIDGE

KCL conservation at a node in a flow network. Kirchhoff's current law (KCL) says the sum of currents into a node is zero. This is literally the same as flow conservation in a graph: edges have flows, internal nodes have zero net flow. Kirchhoff's voltage law (KVL) says the sum of voltages around any closed loop is zero — this is the discrete analog of E·dl = 0 in a conservative field. KCL and KVL together are a graph-theoretic system: nodes have potentials (unknowns), edges have currents (unknowns), and the two Kirchhoff laws plus the element constitutive laws (V=IR for resistors) close the system.

VOLT INSIGHT

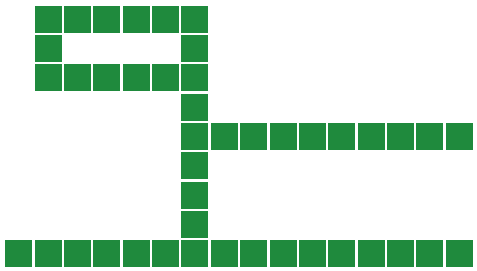
KCL is just flow conservation. Same math as a pipe network. Same math as a packet-switched router. KVL is the same idea, but for voltage. You already know this from graph theory. We're just applying it to electrons instead of packets.

AMP ASKS

So Kirchhoff is just graph theory?

VOLT SAYS

Yes. Conservation at the nodes. Conservation around the loops. That's it.



Voltage divider — V_in R1 midpoint R2 ground. V_out = V_in · R2/(R1+R2).

Ohm's law — the constitutive relation

Ohm's law V = I·R is the constitutive relation of an ideal resistor. It is linear, time-invariant, and memoryless — the resistor's voltage depends only on the current at this instant. Real resistors deviate: they have temperature coefficients (typically 50–200 ppm/°C for metal film), parasitic inductance (a few nH for a ¼W metal film), and parasitic capacitance to ground. For DC and low-frequency work, all of these are negligible. They will start to matter in Phase 2 (AC) and Phase 8 (RF).

$$V = I \cdot R \quad | \quad I = V / R \quad | \quad R = V / I$$



VOLT WARNS

The number one mistake beginners make: forgetting the ground reference. Without a ground, node voltages are undefined. The system has infinitely many solutions. Always pick one node as ground before writing any equation.



AMP ASKS

Like picking the origin in a coordinate system?



VOLT SAYS

Exactly. Without it, you can't say where anything is.

KCL — current conservation

At any node, the algebraic sum of currents is zero. A node is any point where two or more elements connect; in graph-theoretic terms, it is a vertex of the circuit graph. Pick a reference direction (in or out) for each current at the node. Sum to zero. If you have n nodes, KCL gives n−1 independent equations (the last is dependent — sum of all node equations is automatically zero). This is the same rank-deficiency you see in a graph incidence matrix; the null space is the trivial constant-potential vector, which is why you need a ground reference.

KVL — voltage conservation

Around any closed loop, the algebraic sum of voltages is zero. Walk around the loop, assign + when you go from − to + across an element, − when you go + to −. Sum to zero. The number of independent loops is b − n + 1, where b is the number of branches and n the number of nodes — this is the cycle rank of the graph, exactly the number of independent cycles in graph theory.

Solving circuits by hand: the systematic method

For small circuits (3–5 nodes), you can solve by inspection. For larger circuits, use a systematic method: assign node voltages (one node is ground, the rest are unknowns), write KCL at each non-ground node, solve the linear system. This is **nodal analysis**. Alternatively, assign mesh currents (one per independent loop), write KVL around each mesh, solve. This is **mesh analysis**. Nodal works for any circuit; mesh works only for planar circuits. SPICE uses a

generalization of nodal analysis called Modified Nodal Analysis (MNA) that also handles voltage sources cleanly.

Worked example (nodal): Circuit with three nodes — ground, V , V . V connected to a 5 V source. Between V and V : 1 k . Between V and ground: 2 k . KCL at V : (V −V)/1k + V /2k = 0 2·(V −5) + V = 0 3·V = 10 V = 3.33 V. Current through the 2 k is 1.67 mA. Total power delivered by source = 5 V · 1.67 mA = 8.33 mW. Power dissipated in the two resistors: (5−3.33)²/1k + 3.33²/2k = 2.78 + 5.55 = 8.33 mW. Conservation confirmed.

HANDS-ON PROJECT — LTspice + Bench: 4-Node Network

Goal: solve a 4-node circuit three ways — by hand, by LTspice, on the bench — and reconcile all three answers.

Circuit: voltage divider with a load. V = 12 V source; R1 = 1 k series; node A feeds R2 = 2 k to ground AND R3 = 4 k to node B; node B has R4 = 1 k to ground. Compute V_A and V_B.

By hand: nodal analysis. Should take 5 minutes.

LTspice: build the same circuit; run .op (operating point); record V_A and V_B.

Bench: build it on a breadboard with 5% resistors; measure V_A and V_B with DMM.

Reconcile: explain any discrepancy. The 5% resistor tolerance should account for most of it. Compute the worst-case range and confirm the bench measurement is inside it.

CHECKPOINT — Can you answer these?

- State KCL and KVL in your own words. What physical laws are they derived from?
- How many independent KCL equations does an n-node circuit have? KVL equations?
- Why does nodal analysis need a ground reference?
- When is mesh analysis more convenient than nodal? When is it impossible?
- Set up and solve a 3-node circuit with two voltage sources by nodal analysis.

Module 1.3 — Node Voltage & Mesh Current Methods

Duration: 6–8 h · **Difficulty:** Foundation

Module 1.2 introduced the systematic methods at a high level. This module deepens them to handle the cases that trip up beginners: voltage sources between two non-reference nodes (the supernode), current sources shared between two meshes (the supermesh), and dependent sources. Master these three cases and you can solve any linear DC circuit by hand.

Supernodes

When a voltage source connects two non-reference nodes, you cannot write a simple KCL at either node because the current through the source is unknown. The fix is to merge the two nodes into a supernode and write KCL for the supernode as a whole, then add the constraint

equation $V_j - V_k = V_{source}$. The supernode KCL plus the constraint give you the missing equation. Practice this on a 3-node circuit with a 5 V source between nodes 2 and 3 — without the supernode trick, you cannot solve it.

Supermeshes

The dual case in mesh analysis: a current source on the boundary between two meshes. The voltage across the current source is unknown, so you cannot write a clean KVL for either mesh. Merge the two meshes into a supermesh, write KVL around the outer perimeter (skipping the current source), and add the constraint $I_{mesh2} - I_{mesh1} = I_{source}$.

Dependent sources

Dependent sources model active devices — the output of a transistor depends on its input current or voltage. To handle them, treat the dependent source like an independent one in the KCL/KVL equations, then add the constraint equation that defines the dependency (e.g., $V_{dep} = 5 \cdot I_x$ where I_x is a current elsewhere in the circuit). The system stays linear; the only cost is one extra equation per dependent source.

HANDS-ON PROJECT — Practice Set: 10 Circuits by Hand

Goal: fluency with nodal/mesh on any linear circuit.

Tasks: solve 10 circuits of increasing difficulty (supernode, supermesh, dependent source, mixed sources) by hand. The problem set is in appendix/circuits_phase1.pdf — work them in order.

Tool: paper and pencil. Calculator allowed. No SPICE.

Pass: 9 of 10 correct (an answer is correct when the power balance closes — sum of all element powers is zero to within rounding).

CHECKPOINT — Can you answer these?

- What is a supernode and when do you need one?
- What is a supermesh and when do you need one?
- How do you handle a dependent source in nodal analysis?
- Solve a circuit with a CCVS (current-controlled voltage source) using nodal analysis.

Module 1.4 — Thevenin & Norton Equivalent Circuits

Duration: 5–7 h · **Difficulty:** Foundation

CS BRIDGE

Thevenin API encapsulation. A Thevenin equivalent is the circuit analog of an API: from the outside (the load), all that matters is V_th and R_th; everything inside the black box is hidden. Swap the box for any other box with the same V_th and R_th and the load cannot tell. This is the most useful circuit theorem in practice — it is how you reason about output impedance of op-amps, signal-source loading, sensor interfaces, and power-supply design.



VOLT INSIGHT

Thevenin's theorem is just encapsulation. Hide everything behind two numbers. V_th and R_th. The load can't tell what's inside. When you think about output impedance, sensor loading, or supply design, you're computing a Thevenin equivalent. You just don't call it that.



AMP ASKS

So it's like an interface in code? The rest of the world doesn't see the implementation?



VOLT SAYS

That's exactly what it is. Two-terminal interface. Two numbers.

Thevenin's theorem: any linear two-terminal circuit containing voltage/current sources and resistors can be replaced by an equivalent circuit consisting of a single voltage source V_th in series with a single resistor R_th. Norton's theorem is the dual: a current source I_N in parallel with R_N. The two are interchangeable via source transformation: V_th = I_N · R_th, R_N = R_th.

Finding V_th and R_th

Two methods, both must be in your toolkit:

- **Open-circuit / short-circuit method.** V_th = V_oc (open-circuit voltage at the terminals). I_N = I_sc (short-circuit current). R_th = V_oc / I_sc. Three measurements or computations, done.
- **Test-source method.** Turn off all independent sources (voltage sources short, current sources open; leave dependent sources alone). Apply a test voltage V_test at the terminals, compute I_test, R_th = V_test / I_test. Works even with dependent sources, which the open/short method struggles with.

Maximum power transfer

A classic theorem: if a source has Thevenin resistance R_th, the load that extracts maximum power from it is R_L = R_th. At that match, the load voltage is V_th/2, the load current is V_th/(2·R_th), and the load power is $V_{th}^2/(4 \cdot R_{th})$. The source dissipates an equal amount — efficiency is 50%. This is why RF systems are almost always impedance-matched (50 everywhere) and why power-grid transformers are not impedance-matched (you want efficiency, not maximum power transfer).

$$P_{L,max} = V_{th}^2 / (4 \cdot R_{th}) \text{ when } R_L = R_{th}$$

HANDS-ON PROJECT — Bench: Thevenize a Real Divider

Goal: verify Thevenin equivalence on the bench.

Circuit: 12V 1k node A 1k ground (a 1:1 divider). V_A open = 6V.

Compute: V_th = 6V, R_th = 1k 1k = 500 .

Verify load behavior: attach loads of 100 , 500 , 1k , 10k . Measure V_A under each load. It should match V_th · R_L / (R_th + R_L).

Replace: swap the divider for a 6V source in series with a 500 resistor. Repeat the load sweep. The voltages should match the original within tolerance.

Pass: all 4 load measurements agree between the original and the Thevenin equivalent, within 2% (resistor tolerance + DMM accuracy).

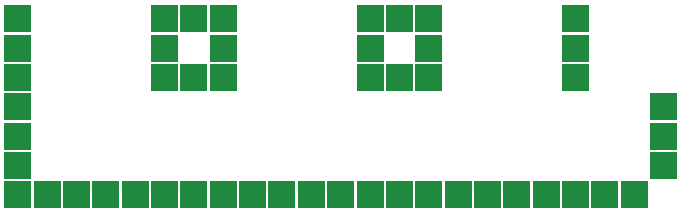
CHECKPOINT — Can you answer these?

- State Thevenin's and Norton's theorems.
- Two methods for finding R_th. When do you prefer the test-source method?
- What is the maximum-power-transfer condition? Why is it not used in power grids?
- Thevenize a voltage divider with a load. What is the practical implication for sensor readout?

Module 1.5 — Superposition & Source Transformation

Duration: 4–6 h · **Difficulty:** Foundation

Two more circuit-reduction techniques that work only on linear circuits. **Superposition**: in a circuit with multiple independent sources, the response (any voltage or current) is the sum of the responses to each source acting alone (others turned off: voltage sources shorts, current sources opens). **Source transformation**: a voltage source V in series with R is equivalent to a current source I = V/R in parallel with R, and vice versa. Both techniques are tools for simplifying a circuit before solving; they are not strictly necessary, but they often turn a 20-minute nodal analysis into a 2-minute back-of-envelope.



Superposition — turn off sources one at a time, sum the individual responses. Linearity is the whole game.

Linearity is the whole game

Every theorem in this module — Thevenin, Norton, superposition, source transformation — depends on the circuit being linear. If the circuit contains a diode, transistor, or any nonlinear element, these theorems do not apply directly. The standard workaround is to linearize around an operating point (small-signal model) — which is exactly what we will do in Phase 3 when we model transistors as small-signal equivalent circuits.

HANDS-ON PROJECT — LTspice: Superposition on a 2-Source Circuit

Goal: verify superposition numerically.

Circuit: two 5V sources and three resistors arranged so that V_mid depends on both.

Step 1: simulate with both sources on; record V_mid.

Step 2: turn off source 2 (set to 0); record V_mid1.

Step 3: turn off source 1; record V_mid2.

Pass: V_mid1 + V_mid2 = V_mid exactly (within numerical precision).

CHECKPOINT — Can you answer these?

- State the superposition principle. What kind of circuits does it apply to?
- How do you 'turn off' a voltage source? A current source?
- Why does superposition fail for nonlinear circuits?
- Demonstrate source transformation on a Thevenin equivalent to get a Norton equivalent.

Module 1.6 — Capacitors, Inductors & Transient Analysis

Duration: 8–10 h · **Difficulty:** Intermediate

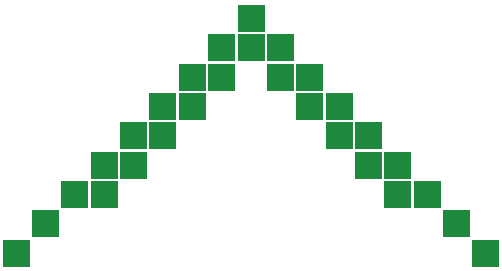
CS BRIDGE

RC transient exponential decay recursion. The ODE $RC \cdot dv/dt + v = V_s$ has solution $v(t) = V_s \cdot (1 - e^{-t/RC})$ for a step input. The same exponential decay structure appears in CS as the temperature schedule of simulated annealing, the convergence of gradient descent on a quadratic, and the cooling of a Markov chain to its stationary distribution. The time constant $\tau = RC$ is the half-life (more precisely, the 1/e time) of the decay.

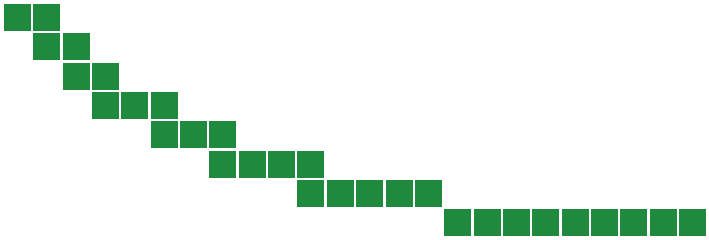


VOLT INSIGHT

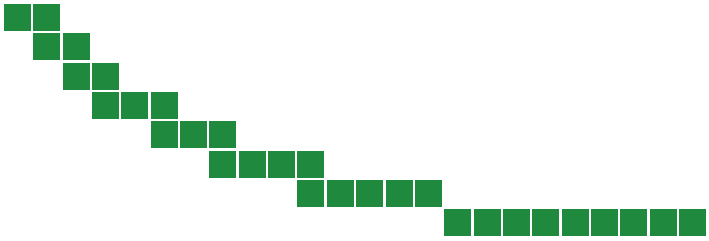
A capacitor isn't really a component. It's a promise. A promise that the voltage across it won't change instantly. An inductor makes the same promise about current. These two promises are the whole key to transient analysis.



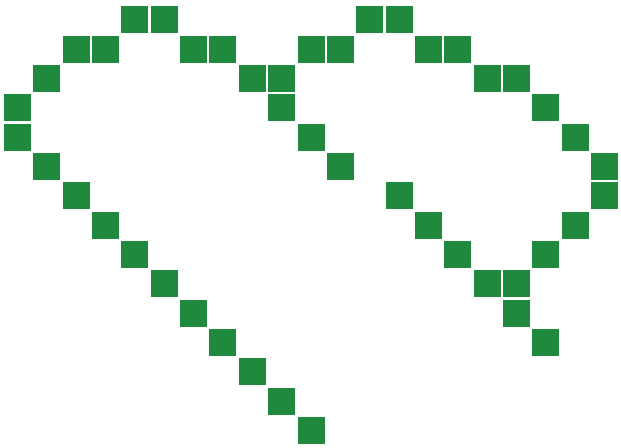
Wheatstone bridge — four resistors in a diamond. Tiny changes in R cause large changes in V_out (sensor readout).



RC charging curve — V_C(t) = V_s · (1 − e^(−t/)). After , 63% charged; after 5 , ~99%.



RL charging curve — I_L(t) = (V_s/R) · (1 − e^(−t/)), = L/R. Dual of the RC response.



RLC damped response — underdamped ringing decays exponentially. < 1 means complex poles.

Capacitors and inductors as energy stores

Resistors dissipate energy — they convert it to heat, irreversibly. Capacitors and inductors store energy — reversibly. A capacitor stores energy in its electric field: $W = \frac{1}{2} \cdot C \cdot V^2$. An inductor stores energy in its magnetic field: $W = \frac{1}{2} \cdot L \cdot I^2$. The constitutive laws are differential: $i = C \cdot dv/dt$ and $v = L \cdot di/dt$. Because these involve derivatives, the voltage across a cap cannot change instantaneously (it would require infinite current), and the current through an inductor cannot change instantaneously (it would require infinite voltage). These two continuity conditions are the key to transient analysis.

$$i_C = C \cdot dv_C/dt \quad | \quad v_L = L \cdot di_L/dt \quad | \quad W_C = \tfrac{1}{2} \cdot C \cdot V^2 \quad | \quad W_L = \tfrac{1}{2} \cdot L \cdot I^2$$



Resistor (American rectangle symbol) — $V = IR$, dissipates $P = I^2R$ as heat.



Capacitor (parallel-plate symbol) — $i = C \cdot dv/dt$, stores energy in E-field.



Inductor (coil symbol) — $v = L \cdot di/dt$, stores energy in B-field.

First-order transients — the RC and RL step response

An RC circuit driven by a step from 0 to V_s has the ODE $RC \cdot dv/dt + v = V_s$. The solution is $v(t) = V_s \cdot (1 - e^{-t/\ })$ with $\ = RC$. The current is $i(t) = (V_s/R) \cdot e^{-t/\ }$. After one time constant the voltage has reached 63% of its final value; after five time constants it is within 1%. The RL step response is identical in form: $i(t) = (V_s/R) \cdot (1 - e^{-t/\ })$ with $\ = L/R$.

$$v_C(t) = V_f + (V_0 - V_f) \cdot e^{-t/\ } \qquad = RC$$

Second-order transients — the RLC response

A series RLC has the ODE $L \cdot d^2i/dt^2 + R \cdot di/dt + (1/C) \cdot i = 0$ after the source is removed. Characteristic equation: $s^2 + (R/L) \cdot s + 1/(LC) = 0$. Compare to the canonical form $s^2 + 2\ _n \cdot s + \ _n^2 = 0$ and read off $\ _n = 1/\sqrt{LC}$ and $\ = (R/2) \cdot \sqrt{C/L}$. The three regimes — overdamped, critically damped, underdamped — appear physically:

- **Overdamped (> 1):** R is large. Two real, negative poles. The response is a slow exponential decay with no overshoot.
- **Critically damped (= 1):** R is exactly right. Repeated real pole. Fastest possible settling with no overshoot.

- **Underdamped (< 1):** R is small. Complex conjugate poles. The response rings — a decaying sinusoid at frequency _d = _n· (1– ²).

HANDS-ON PROJECT — Bench: Ring an LC Tank

Goal: see the three damping regimes physically.

Circuit: 10 mH inductor in series with a 100 nF cap and a variable resistor (1k pot).

Drive: charge the cap to 5V, then disconnect the supply and let the tank ring.

Measure: scope across the cap. You should see a decaying sinusoid.

Sweep R: start with R = 0 (just the inductor's DCR). Then increase R. Watch the ringing get less pronounced, then disappear at critical damping, then become a slow exponential rise.

Compute: verify = (R/2)· (C/L) matches the observed behavior. _n should be 1/ (LC) 10 rad/s = 1.6 kHz.

CHECKPOINT — Can you answer these?

• Why can't the voltage across a capacitor change instantaneously? What about the current through an inductor?
• What is the time constant of an RC circuit? An RL circuit?
• Write the general second-order ODE and the canonical form. Define and _n.
• What value of R makes a series RLC critically damped? Compute it for L = 10 mH, C = 100 nF.
• Sketch the step response of an underdamped second-order system. Label rise time, overshoot, settling time.

"The capacitor is not a component — it is a promise that the voltage across it will not change instantly. The inductor makes the dual promise about current."

— Volt, on continuity conditions



VOLT NARRATES

Phase 1 is done. You can solve any linear DC circuit. You understand the three passive components. You can predict what happens when you flip a switch. Phase 2 takes these tools into the AC domain. Every signal is a sine wave. Impedance replaces resistance. The math is the same. Only the costumes change.



Phase 2 — AC Circuit Analysis

Duration: 5 weeks · **Modules:** 6 · **Goal:** generalize DC analysis to sinusoidal steady-state via phasors and impedance; master AC power (real, reactive, apparent, power factor); meet resonance and three-phase systems.

AC analysis is the technique that makes sinusoidal circuits as easy as DC circuits. The trick is the phasor transform: a sinusoidal voltage $v(t) = V_m \cdot \cos(\ t + \ )$ is represented by a single complex number $\mathbf{V} = V_m \cdot e^{j\ }$. Differentiation becomes multiplication by j , integration becomes division by j , and the entire machinery of DC analysis (Ohm, Kirchhoff, Thevenin) carries over with R replaced by complex impedance Z. This single mathematical move turns a hard problem (solving ODEs with sinusoidal forcing) into a routine linear algebra problem.



VOLT NARRATES

Phase 2 is the payoff for all the complex-number work in Phase 0. Once you see that a sine wave is a single complex number, and that differentiation is just multiplication by j , every AC circuit becomes a DC circuit with complex resistors. The whole chapter is one trick. Applied a hundred ways.



AMP ASKS

So a capacitor becomes a complex resistor?



VOLT SAYS

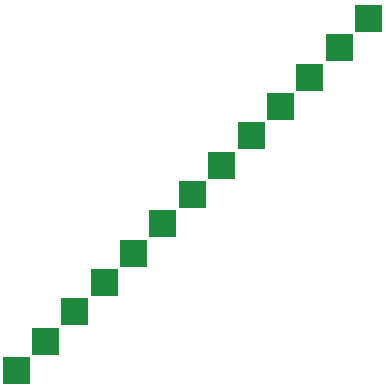
Yes. With a value of 1/(j C). One number. That's the trick.

Module 2.1 — Sinusoidal Sources and Phasors

Duration: 5–7 h · **Difficulty:** Foundation

CS BRIDGE

Phasor hashing a function to a number. A sinusoid has three degrees of freedom (amplitude, frequency, phase). At a fixed frequency, two remain (amplitude, phase) — exactly the modulus and argument of a complex number. The phasor transform is a lossless hash of {amplitude, phase} . Addition of sinusoids at the same frequency is complex addition; differentiation is multiplication by j . This is the engineer's version of representing a 2D vector as a complex number — you already do this in graphics code.



AC phasor — a rotating complex number. At a fixed , only amplitude and phase matter; the e^(j t) factor is common and dropped.

The phasor transform

A sinusoid v(t) = $V_m \cdot \cos(\ t + \ )$ is the real part of $V_m \cdot e^{j(\ t + \ )} = V_m \cdot e^{j\ } \cdot e^{j\ t}$. The factor $e^{j\ t}$ is common to every signal at frequency , so we drop it and keep only $\mathbf{V} = V_m \cdot e^{j\ } = V_m\ \ $. To recover the time-domain signal: v(t) = Re{$\mathbf{V} \cdot e^{j\ t}$}. The transform is bijective at fixed frequency; you lose nothing.

$$v(t) = V_m \cdot \cos(\ t + \ ) \qquad \mathbf{V} = V_m\ \ = V_m \cdot e^{j}$$

Why it works — the eigenfunction property

Sinusoids are eigenfunctions of linear time-invariant (LTI) systems. If the input is a sinusoid at frequency , the steady-state output is a sinusoid at the same frequency, scaled and phase-shifted. The scaling and phase shift are encoded in a single complex number — the system's frequency response **H**(j). For a single LTI system, $\mathbf{V_{out}} = \mathbf{H}(j\ ) \cdot \mathbf{V_{in}}$. This is the entire foundation of frequency-domain analysis, and it generalizes to the Fourier transform (Phase 5) and the Laplace transform (Phase 5).

HANDS-ON PROJECT — LTspice: Phasor Addition

Goal: verify that adding two sinusoids in time domain equals adding their phasors.

Set up: two AC sources v1 = 5·cos(1000t) V and v2 = 3·cos(1000t + 60°) V, in series.

Predict: phasor sum: 5 + 3 60° = 5 + 1.5 + j·2.598 = 6.5 + j·2.598 = 7.0 21.8°. So v_total(t) = 7.0·cos(1000t + 21.8°).

Simulate: run transient sim in LTspice; measure amplitude and phase of the sum.

Pass: measured amplitude within 1% of 7.0 V; phase within 1° of 21.8°.

CHECKPOINT — Can you answer these?

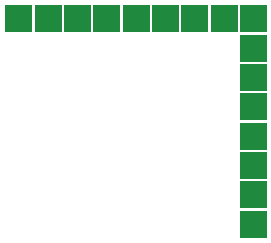
- What is a phasor? How is it related to the time-domain sinusoid?
- Why are sinusoids eigenfunctions of LTI systems?
- Convert v(t) = 10·sin(100t + 30°) to a phasor. (Hint: sin = cos shifted by −90°.)
- Add 5 0° and 3 90° in both rectangular and polar form. Confirm they match.

Module 2.2 — Impedance and Admittance

Duration: 5–7 h · **Difficulty:** Foundation

The impedance **Z** of a two-terminal element is the ratio of phasor voltage to phasor current: **Z** = **V**/**I**. For a resistor: $\mathbf{Z}_R$ = R (purely real). For an inductor: $\mathbf{Z}_L$ = j L (purely imaginary, positive — voltage leads current by 90°). For a capacitor: $\mathbf{Z}_C$ = 1/(j C) = −j/(C) (purely imaginary, negative — current leads voltage by 90°). These three elements, in any combination, give any passive impedance.

$$Z_R = R \quad | \quad Z_L = j\ L \quad | \quad Z_C = 1 / (j\ C) = -j / (\ C)$$



Impedance triangle — R (real) horizontal, X (reactive) vertical, Z (magnitude) hypotenuse. |Z| = (R²+X²).

Admittance **Y** = 1/**Z** is the dual. $\mathbf{Y}_R$ = 1/R = G (conductance). $\mathbf{Y}_L$ = 1/(j L) = −j/(L). $\mathbf{Y}_C$ = j C. Admittance is more convenient than impedance when elements are in parallel (admittances add). The real part of admittance is conductance G; the imaginary part is susceptance B. The real part of impedance is resistance R; the imaginary part is reactance X. **Z** = R + jX, **Y** = G + jB.

Series and parallel combinations

Impedances combine exactly like resistors: series impedances add, parallel impedances combine by reciprocal-sum. The arithmetic is now complex, but the structure is identical. Practice: a 1 k resistor in series with a 100 mH inductor at 1 kHz has impedance **Z** = 1000 + j·2 ·1000·0.1 = 1000 + j628 . Magnitude 1185 , phase +32°. The voltage across the inductor leads the current by 90°, and the total voltage leads the current by 32°.

$$Z_{series} = Z_1 + Z_2 + \ldots \quad | \quad 1/Z_{parallel} = 1/Z_1 + 1/Z_2 + \ldots$$

HANDS-ON PROJECT — LTspice: RC Impedance Sweep

Goal: measure the impedance of an RC parallel combo across frequency.

Circuit: 1 k resistor in parallel with 1 F cap, driven by a 1 V AC source.

Compute: Z(j) = R 1/(j C) = R / (1 + j RC). At = 1/(RC) = 1000 rad/s = 159 Hz, Z = R/2 = 500 , phase −45°.

Simulate: LTspice .ac sim from 10 Hz to 10 kHz. Plot |Z| and Z vs frequency.

Pass: the magnitude and phase curves match the predicted formula at the 3 marked frequencies (159 Hz, 1.59 kHz, 15.9 Hz).

CHECKPOINT — Can you answer these?

- What is the impedance of a 10 mH inductor at 60 Hz? At 1 MHz?
- What is the impedance of a 1 F capacitor at 60 Hz? At 1 MHz?
- Combine a 1 k resistor in series with a 0.1 F cap at 1 kHz. What is |Z|? What is Z?
- Why does an inductor's impedance grow with frequency while a cap's shrinks?

Module 2.3 — AC Power: Real, Reactive, Apparent, Power Factor

Duration: 6–8 h · **Difficulty:** Intermediate

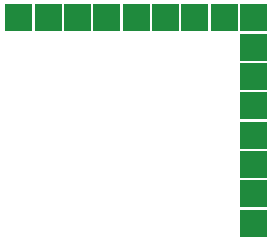
CS BRIDGE

Power factor load matching in distributed systems. A load with PF = 1 is 'resistive' — current and voltage are in phase, all delivered energy is consumed. A load with PF < 1 is 'reactive' — some energy sloshes back and forth every cycle, like a chatty protocol with high acknowledgment overhead. The grid bills you for the energy you actually use (real power), but the wires and transformers must be sized for the apparent power. Power-factor correction is the engineering practice of adding caps or inductors to bring PF close to 1, reducing the apparent power the grid must deliver.

The three powers

For a sinusoidal voltage $V = V_m\cos(t)$ across a load drawing current $I = I_m\cos(t -)$, the instantaneous power is p(t) = v(t)·i(t). Averaging over a cycle gives the **real power** $P = V_{rms}\cdot I_{rms}\cdot\cos( )$, in watts. The **reactive power** $Q = V_{rms}\cdot I_{rms}\cdot\sin( )$, in VAR (volt-amperes reactive). The **apparent power** $|S| = V_{rms}\cdot I_{rms}$, in VA. The relationship is S = P + jQ — apparent power is the complex sum of real and reactive. The **power factor** PF = P / |S| = cos( ).

$$P = V_{rms}\cdot I_{rms}\cdot\cos( ) \text{ [W]} \quad | \quad Q = V_{rms}\cdot I_{rms}\cdot\sin( ) \text{ [VAR]} \quad | \quad |S| = V_{rms}\cdot I_{rms} \text{ [VA]} \quad | \quad PF = \cos( ) = P / |S|$$



Power triangle — P (real) horizontal, Q (reactive) vertical, S (apparent) hypotenuse. PF = P/|S| = cos( ).

RMS (root-mean-square) is the equivalent DC value that would deliver the same average power to a resistor. For a sinusoid, $V_{rms} = V_m/\sqrt{2}$. For a waveform of arbitrary shape, $V_{rms} = \sqrt{(1/T \cdot \int v(t)^2\,dt)}$. The 230 V or 120 V at your wall outlet is RMS — the peak is $\sqrt{2}$ times higher (325 V or 170 V).

Power factor correction

Industrial loads are typically inductive (motors, transformers) and have PF around 0.7–0.85 lagging. To correct, add a capacitor in parallel sized so that $Q_{cap} = Q_{load}$. The cap draws leading current that cancels the lagging inductive current, bringing the net PF to 1. The size of the cap: $C = Q / (\cdot V^2)$. For a 5 kW load at 230 V, 50 Hz, PF = 0.8 lagging: Q = P·tan( ) = 5000·tan(36.87°) = 3750 VAR. C = 3750 / (2 ·50·230²) = 225 F.

HANDS-ON PROJECT — LTspice: PF Correction

Goal: design a PFC cap and verify it brings PF to 1.

Load: R = 23 in series with L = 50 mH (a model of a motor) at 230 V, 50 Hz.

Without PFC: Z_load = 23 + j·15.7 . |Z| = 27.9 , I = 8.24 A, = 34.4°. PF = cos(34.4°) = 0.825. P = 230·8.24·0.825 = 1563 W. Q = 230·8.24·sin(34.4°) = 1068 VAR.

Compute C: C = Q / (·V²) = 1068 / (2 ·50·230²) = 64.3 F.

Verify in LTspice: add the cap in parallel; re-sim. The source current should drop to ~6.8 A (P/230), in phase with the voltage.

CHECKPOINT — Can you answer these?

- Define real, reactive, and apparent power. Give the unit of each.
- What is power factor? How is it related to the angle between voltage and current?
- Compute the RMS value of a square wave that swings between +5V and −5V.
- Design a PFC capacitor for a 10 kW load at 400 V, 50 Hz, PF = 0.7 lagging.

Module 2.4 — Resonance, Bode Plots & Filters

Duration: 6–8 h · **Difficulty:** Intermediate

A series RLC circuit has impedance **Z**(j) = R + j L + 1/(j C) = R + j·(L − 1/(C)). At $_0$ = 1/ (LC), the imaginary part vanishes — **Z** is purely real and equal to R. This is **series**

resonance: the inductor and capacitor cancel each other, and the only thing limiting the current is R. The current is maximum at resonance. The same circuit in parallel gives **parallel resonance**: at $_0$, the impedance is maximum (purely real, equal to R, or in the ideal case infinite).

$$_0 = 1 / (LC) \quad | \quad Q = _0 \cdot L / R = 1 / (_0 \cdot R \cdot C) \quad | \quad BW = _0 / Q$$

VOLT INSIGHT

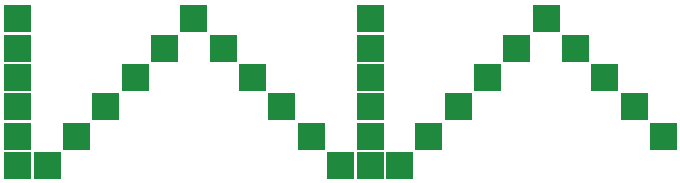
Resonance is when inductive and capacitive reactance cancel out. At one frequency. = 1/ (LC). The imaginary part of the impedance goes to zero. The circuit acts like it's purely resistive. This is the heart of every radio. Tune the L and C, and you tune which frequency you hear.

AMP ASKS

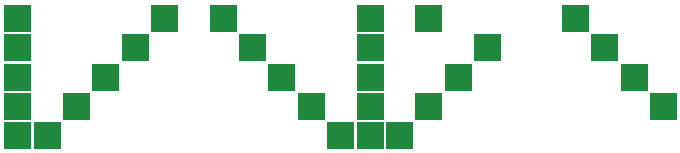
Like tuning a guitar string to one note?

VOLT SAYS

Yes. Same idea. Resonance picks out one frequency and amplifies it.

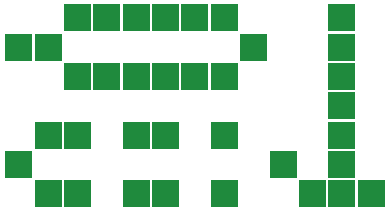


Series resonance — at , L and C cancel. Current peaks. Used in tuners and notch filters.

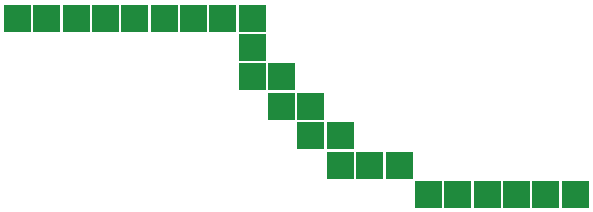


Parallel resonance (tank) — at , impedance peaks. The basis of every LC oscillator.

The quality factor Q measures how sharp the resonance is. Q = $_0$/BW where BW is the −3 dB bandwidth. A high-Q resonator passes a narrow band of frequencies and rejects the rest — the basis of every radio receiver and every analog filter. A low-Q resonator passes a wide band. The damping ratio = 1/(2Q).



Series RLC resonator — R, L, C in a string. At = 1/ (LC), XL and XC cancel.



Bode magnitude of an RC low-pass — flat to f_c, then −20 dB/decade rolloff.

Bode plots — the frequency-domain picture

A Bode plot is the magnitude (in dB) and phase (in degrees) of a transfer function H(j) versus log frequency. Straight-line Bode approximations are a staple of EE: each pole contributes −20 dB/decade rolloff starting at the pole frequency and −45° phase shift at the pole, growing to −90° a decade above. Each zero contributes +20 dB/decade and +90°. Second-order poles/zeros have twice the slope and contribute up to ±180°.

Decibels: dB = $20 \cdot \log_{10}(|H|)$. The factor 20 (not 10) comes from power being proportional to voltage squared, so $10 \cdot \log(P) = 10 \cdot \log(V^2) = 20 \cdot \log(V)$ when the load is held constant. −3 dB is the half-power point: 10·log(0.5) −3. −20 dB/decade = −6 dB/octave.

HANDS-ON PROJECT — Python: Bode Plotter for Transfer Function

Goal: build a reusable Bode plotter in Python.

Tasks: write a function bode(num, den, f_range) that uses scipy.signal.freqs to compute H(j) and plots magnitude (dB) and phase (deg) vs frequency (log).

Test: plot $H(s) = 1/(s^2 + s + 1)$ — second-order low-pass at _n=1, =0.5. Verify the peak at _n and the −40 dB/decade rolloff above.

Bench: build the same filter with R=1k , L=1H (use a gyrator or active inductor simulator), C=1 F. Sweep frequency from 10 Hz to 10 kHz, measure |H|, overlay on the Python plot. They should match.

CHECKPOINT — Can you answer these?

- Define series and parallel resonance. Where do they occur?
- What is Q? How is it related to damping ratio and bandwidth?
- Sketch the Bode magnitude plot of a first-order RC low-pass. Mark the cutoff frequency.
- How many dB per decade does a 4th-order low-pass roll off above cutoff?
- Convert 0.707 to dB. Why is this number special?

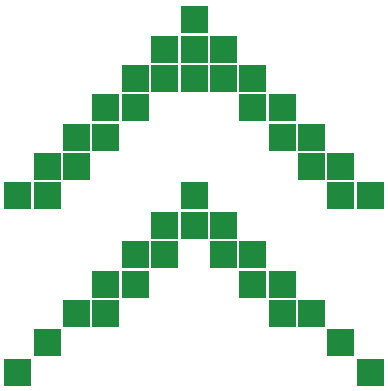
Module 2.5 — Three-Phase Systems

Duration: 4–6 h · **Difficulty:** Intermediate

Three-phase power is the workhorse of electrical generation, transmission, and industrial distribution. Three sinusoidal voltages at the same frequency and amplitude, 120° apart: $V_a = V_m\cos(\ t)$, $V_b = V_m\cos(\ t - 120°)$, $V_c = V_m\cos(\ t + 120°)$. Their sum is always zero — this means a three-phase system with balanced loads carries no neutral current, halving the conductor material needed compared to three single-phase systems.

$$V_L = \sqrt{3} \cdot V_{ph} \quad \text{(line-line vs phase-neutral)} \quad | \quad P = \sqrt{3} \cdot V_L \cdot I_L \cdot PF$$

Two standard connections: **wye (Y)** — three phases plus neutral, common in distribution (230/400 V in Europe, 120/208 V or 277/480 V in US). **Delta ( )** — three phases only, no neutral, common in transmission and industrial motors. A balanced load draws $\sqrt{3}$ times the line current of an equivalent Y load at the same line voltage. Three-phase rectifiers produce smoother DC than single-phase — six pulses per cycle instead of two.



Three-phase phasor diagram — phases A, B, C separated by 120°. V_a + V_b + V_c = 0.

HANDS-ON PROJECT — Python: Three-Phase Phasor Visualizer

Goal: see three-phase phasors rotating.

Tasks: animate three phasors (red, yellow, blue) at 50 Hz, 120° apart. Show their sum is always zero. Animate a rotating three-phase voltage space vector — it traces a circle of radius V_m.

Tool: matplotlib FuncAnimation.

Extension: add an unbalanced case (V_a reduced by 20%) and show the sum is no longer zero — this is why PFC and load balancing matter.

CHECKPOINT — Can you answer these?

- Why does a balanced three-phase system need no neutral?
- Convert 400 V line-line to phase voltage.
- Compute the real power delivered to a balanced Y load with R=10 per phase at 400 V line-line.
- Why is three-phase transmission more efficient than single-phase?

Module 2.6 — Mutual Inductance & Transformers

Duration: 4–6 h · **Difficulty:** Intermediate

Two coils sharing magnetic flux have mutual inductance M. The voltage across coil 1 is $v_1 = L_1 \cdot di_1/dt + M \cdot di_2/dt$, and similarly for coil 2. The coupling coefficient $k = M/\sqrt{(L_1 \cdot L_2)}$ ranges from 0 (no coupling) to 1 (perfect coupling, all flux shared). Real transformers achieve k = 0.95–0.99 with ferromagnetic cores.

The ideal transformer

In the ideal-transformer limit (k = 1, no losses, infinite inductance), the voltage and current ratios are set by the turns ratio $n = N_1/N_2$: $v_2/v_1 = N_2/N_1 = 1/n$, $i_2/i_1 = N_1/N_2 = n$. Power is conserved: $v_1 \cdot i_1 = v_2 \cdot i_2$. The impedance seen at the primary is the secondary load scaled by n^2: $Z_1 = n^2 \cdot Z_2$. This is the impedance-reflection property that makes transformers so useful — you can match any load to any source by choosing n correctly.

$$V_2/V_1 = N_2/N_1 \quad | \quad I_2/I_1 = N_1/N_2 \quad | \quad Z_{reflected} = (N_1/N_2)^2 \cdot Z_{load}$$

Real transformers

Real transformers have winding resistance, leakage inductance (flux that doesn't link both coils), magnetizing inductance (the primary inductance that draws magnetizing current even with no load), core losses (hysteresis + eddy currents), and inter-winding capacitance. The equivalent circuit is a T or network with these parasitics added to the ideal transformer. For power-grid transformers the parasitics are small (97–99% efficiency). For signal transformers (audio, RF) they shape the frequency response.

HANDS-ON PROJECT — LTspice: Step-Down Transformer

Goal: simulate a 10:1 step-down transformer and verify power transfer.

Model: ideal transformer with n = 10. Primary: 230 V RMS at 50 Hz. Secondary: 23 V RMS into a 10 load.

Predict: I_primary = (V_secondary / R_load) · (N_2/N_1) = (23/10) · 0.1 = 0.23 A. P_load = V^2/R = $23^2/10$ = 52.9 W. P_primary = 230 · 0.23 = 52.9 W.

Simulate: verify both powers match.

Extension: add leakage inductance of 1 mH on primary and 10 H on secondary. How does the regulation (V_load vs. load current) change?

CHECKPOINT — Can you answer these?

- What is mutual inductance? Define the coupling coefficient k.
- State the ideal transformer equations for voltage, current, and impedance.
- A 230 V:12 V transformer has a 100 load on the secondary. What is the primary impedance?
- Why don't transformers work with DC?
- Name three parasitic effects in real transformers and how each manifests in circuit behavior.

"An ideal transformer is a lossless impedance converter. The turns ratio does not just scale voltage — it scales impedance by the square, and that is the reason the entire power grid can move gigawatts across a thousand kilometers."

— Volt, on transformers



VOLT NARRATES

Phase 2 is done. You can solve any AC circuit with phasors. You understand impedance. You understand resonance. You know why the grid runs on three phases. Phase 3 introduces the nonlinear parts. Diodes. Transistors. Op-amps. These are what make circuits do useful things. Amplify. Switch. Modulate. The linear math ends here. The small-signal approximation begins.

Phase 3 — Analog Electronics

Duration: 6 weeks · **Modules:** 8 · **Goal:** master the nonlinear workhorses of analog — diodes, BJTs, MOSFETs, op-amps — and build a complete audio amplifier by the end of the phase.

Phase 3 is where the rubber meets the road. Every element so far has been linear; now we meet diodes, transistors, and op-amps — devices whose V-I curves are nonlinear. The trick that makes them tractable is the **small-signal model**: linearize around a DC operating point, treat the nonlinear device as a linear element for AC analysis, then add the DC and AC solutions. This is the same linearization you do in optimization (Newton's method) and in machine learning (backprop through a nonlinearity).

VOLT NARRATES

Nonlinear doesn't mean scary. Diodes, transistors, op-amps. They all follow V-I curves you can look up. The trick is to bias them to a known DC operating point. Then linearize around it. The AC behavior near that point is just another linear circuit. Same math as Phase 1. One extra step at the start.

AMP ASKS

Like linearizing a function around a point in calculus? Tangent line approximation?

VOLT SAYS

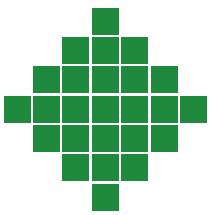
Exactly. Same idea. Different domain.

Module 3.1 — Diodes: Rectifiers, Clippers, Clampers

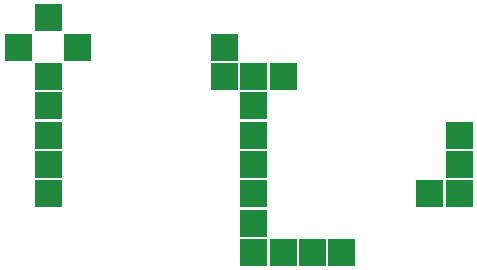
Duration: 5–7 h · **Difficulty:** Intermediate

A diode is a one-way valve for current. The ideal diode conducts with zero resistance when forward-biased (anode > cathode) and blocks completely when reverse-biased. The real diode (Shockley equation) has a forward voltage drop of about 0.7 V (silicon) or 0.3 V (Schottky) and a small reverse leakage. $I = I_S \cdot (e^{V/V_T} - 1)$ where $V_T = kT/q$ 26 mV at room temperature and I_S is the saturation current (typically 10^{-15}–10^{-12} A).

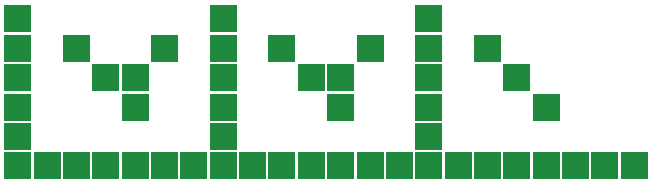
$$I = I_S \cdot (e^{V/(nV_T)} - 1) \quad | \quad V_T = kT/q \quad 26\ \text{mV}$$



Diode (anode left, cathode right) — one-way current valve. V_f 0.7 V (Si).



Half-wave rectifier — AC source diode load ground. Passes only positive half-cycles.



Full-wave bridge rectifier — four diodes route both half-cycles to the load. Output average = 2·V_m/ . Ripple freq = 2× line.

Three classic diode circuits

- **Half-wave rectifier:** a single diode in series with the load. Passes only the positive half-cycles. Output average = V_m/ . Ripple frequency = line frequency.
- **Full-wave bridge rectifier:** four diodes arranged so both half-cycles contribute. Output average = 2·V_m/ . Ripple frequency = 2× line frequency. The classic front-end of every linear power supply.
- **Peak rectifier:** a bridge followed by a capacitor. The cap charges to V_m and holds; the diode only conducts briefly at the peak. Output is roughly DC with ripple V = I/(f·C). This is the front-end of every linear and switch-mode supply.

Two more diode applications to know: **clippers** (limit a signal's voltage swing — used for input protection) and **clampers** (add a DC offset to a signal — used to restore DC levels after AC coupling). Zener diodes are designed to operate in reverse breakdown safely and provide a stable reference voltage — every linear regulator uses one.

HANDS-ON PROJECT — Bench: Build a 12V Linear Supply Front-End

Goal: rectify a 12 VAC wall transformer to DC.

Circuit: 12 VAC secondary 1N4007 bridge rectifier 1000 F filter cap 12 V Zener clamp.

Measure: scope at the cap: see the ripple. Compute V = I_load / (f·C) where f = 100 Hz (full wave) and I_load = 50 mA. V 0.5 V.

Verify: the measured ripple matches the formula. Add load, watch ripple grow linearly.

Pass: the supply delivers 12 V ±0.6 V at loads up to 100 mA. The scope shows ripple matching the formula within 20%.

CHECKPOINT — Can you answer these?

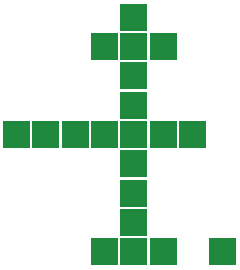
- State the Shockley diode equation. What is V_T at room temperature?
- Why does a full-wave rectifier have half the ripple of a half-wave at the same load?
- Compute the ripple of a 1000 F cap delivering 100 mA at 100 Hz ripple frequency.
- What is a Zener diode and how does it differ from a regular diode in operation?

Module 3.2 — BJT Biasing and Small-Signal Model

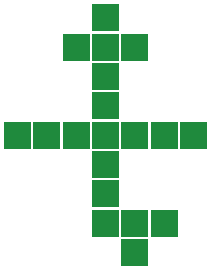
Duration: 8–10 h · **Difficulty:** Intermediate

CS BRIDGE

BJT current-controlled current source, MOSFET voltage-controlled current source. A BJT's collector current is proportional to base current (I_C = ·I_B, 100). A MOSFET's drain current is controlled by gate-source voltage (I_D = ½·k·(V_GS − V_th)² in saturation). Conceptually, a BJT is a current amplifier; a MOSFET is a transconductance amplifier. Both have a small-signal model that linearizes them around a DC operating point — same idea as Newton's method.



NPN BJT — base (left), collector (top), emitter (bottom). I_C = · I_B.



Common-emitter amplifier — BJT with R_C and R_E. Inverts the input; voltage gain A_v = −g_m · R_C.

BJT regions of operation

Region	Condition	Behavior
Cut-off	V_BE < 0.6 V	I_C 0. Off switch.

Region	Condition	Behavior
Active	V_BE 0.7 V, V_CE > V_CE(sat)	I_C ·I_B. Amplifier region.
Saturation	V_BE 0.7 V, V_CE 0.2 V	I_C < ·I_B. On switch.
Reverse active	V_EB 0.7 V	Inverted operation. Rarely used.

Biasing — setting the operating point

A BJT amplifier must be biased to a stable quiescent (Q) point in the active region. The four-resistor bias network (R1, R2 from V_CC to ground at the base; R_E at the emitter; R_C at the collector) is the standard. The base voltage is set by R1/R2 divider; the emitter voltage follows minus 0.7 V; the emitter current is V_E/R_E; the collector current is approximately equal. The design rule: make the divider current 10× the base current so the bias is stable against variation.

$$V_B \quad V_{CC}\cdot R_2/(R_1+R_2) \quad | \quad V_E = V_B - 0.7 \quad | \quad I_E \quad I_C = V_E/R_E \quad | \quad V_C = V_{CC} - I_C\cdot R_C$$

Small-signal model — the hybrid-

Linearize the BJT around its Q point. The small-signal model has: input resistance r_ = /g_m where g_m = I_C/V_T is transconductance (~40 mA/V per mA of I_C), output resistance r_o = V_A/I_C (Early voltage V_A ~ 100 V), and a current source g_m·v_be from collector to emitter. With this model you can compute voltage gain, input impedance, and output impedance of any BJT amp using the linear-circuit techniques of Phase 1.

$$g_m = I_C / V_T \quad 40\cdot I_C\ [A/V] \quad | \quad r = / g_m \quad | \quad A_v = -g_m\cdot R_C \text{ (common-emitter, unloaded)}$$

HANDS-ON PROJECT — Bench: Common-Emitter Audio Preamp

Goal: build a CE amplifier with gain 20, verify on bench.

Specs: V_CC = 12V, I_C = 1 mA, V_CE = 6V (mid-supply for max swing).

Design: R_E = 1 k (sets I_C 1 mA with V_E 1V). R_C = 5 k (sets V_C 7V, V_CE 6V). R1 = 50 k , R2 = 10 k (sets V_B 2V). bypass R_E with a 100 F cap for AC gain.

Compute: g_m = 40 mA/V. A_v = −g_m · R_C = −200. But with R_E bypassed only partially (use 100 unbypassed for stability), A_v −R_C / R_E_unbypassed = −50.

Build & measure: drive with 10 mV sine at 1 kHz. Measure V_out. Compute gain. Compare to prediction. Sweep frequency from 20 Hz to 200 kHz; find the −3 dB bandwidth.

CHECKPOINT — Can you answer these?

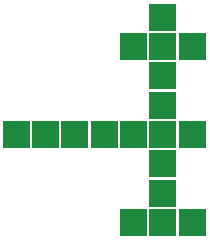
- Name the four regions of BJT operation. Which is used for amplification? For switching?
- Design a four-resistor bias network for I_C = 2 mA, V_CC = 12 V, = 100.
- Define g_m, r_ , r_o. How do they depend on I_C?
- Compute the voltage gain of a CE amp with g_m = 40 mA/V and R_C = 5 k .
- Why do we bypass R_E with a capacitor? What is the trade-off?

Module 3.3 — MOSFET Biasing and Small-Signal Model

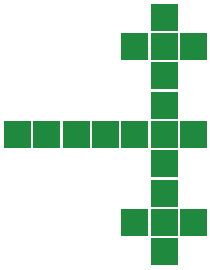
Duration: 6–8 h · **Difficulty:** Intermediate

The MOSFET has three regions: cut-off (V_GS < V_th, I_D 0), triode (V_GS > V_th and V_DS < V_GS − V_th, behaves as a resistor), and saturation (V_GS > V_th and V_DS > V_GS − V_th, behaves as a current source). The saturation-region equation is I_D = ½·k_n·(V_GS − V_th)²·(1 + ·V_DS), where k_n = _n·C_ox·W/L and is the channel-length modulation parameter (analogous to the BJT Early voltage).

$$I_D = \tfrac{1}{2}\cdot k_n\cdot(V_{GS} - V_{th})^2\cdot(1 + \ \cdot V_{DS}) \quad \text{(saturation)}$$



n-channel MOSFET — gate (left), drain (top), source (bottom). Voltage-controlled current source.



Common-source amplifier — MOSFET with R_D. Inverts the input; gain A_v = −g_m · (R_D r_o). The CMOS analog workhorse.

Small-signal model

Linearize around the Q point: g_m = I_D/ V_GS = k_n·(V_GS − V_th) = (2·k_n·I_D). Output resistance r_o = 1/(·I_D). The small-signal model has infinite input resistance at the gate (no DC current flows), a voltage-controlled current source g_m·v_gs from drain to source, and r_o in parallel. The CS amplifier voltage gain is A_v = −g_m·(R_D r_o).

$$g_m = \ (2\cdot k_n\cdot I_D) \quad | \quad A_v = -g_m\cdot(R_D \ \ r_o)$$

Why MOSFETs dominate modern electronics

The MOSFET's gate draws no DC current — input impedance is essentially infinite at DC and low frequencies. This is why MOSFETs (specifically CMOS) dominate digital electronics: a billion CMOS gates draw nearly zero static current. BJTs always draw base current, which is

fine for analog but catastrophic for high-density digital. MOSFETs also scale better — smaller geometries work, which is why Moore's law has been a MOSFET story for 50 years. You will see the consequences in Phase 9 (VLSI).

HANDS-ON PROJECT — LTspice: MOSFET CS Amp

Goal: design and simulate a CS MOSFET amplifier.

Specs: V_DD = 5V, I_D = 100 A, V_DS = 2.5V.

MOSFET: use 2N7002 model in LTspice.

Design: R_D = (5 − 2.5)/100 = 25 k . Find V_GS for I_D = 100 A from the datasheet curve (typically 1.5 V for 2N7002). Bias the gate via R1/R2 divider with high values (1 M each) to preserve input impedance.

Simulate: verify DC operating point. Then run AC sim. Compute gain and bandwidth.

Bench (optional): build with discrete 2N7002. Note: discrete MOSFETs have huge V_th variation; you will need to tune the bias.

CHECKPOINT — Can you answer these?

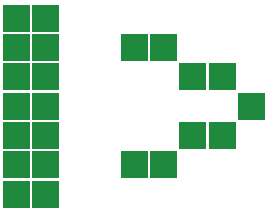
- Name the three regions of MOSFET operation. Which is used for amplification?
- Why is MOSFET input impedance nearly infinite at the gate?
- Compute g_m for k_n = 1 mA/V², I_D = 100 A.
- Why do digital circuits use MOSFETs (CMOS) rather than BJTs?

Module 3.4 — Operational Amplifiers: The Analog Swiss Army Knife

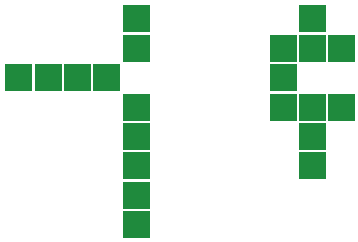
Duration: 8–10 h · **Difficulty:** Intermediate

CS BRIDGE

Op-amp functional-programming pure function. An ideal op-amp has infinite gain, infinite input impedance, zero output impedance, infinite bandwidth. Its output is determined entirely by the feedback network — the op-amp itself is a pure gain block, like a pure function. The feedback network is the 'program'; the op-amp is the 'runtime'. This is why op-amp circuits are so composable: swap the feedback network and you get a different function — inverting amp, integrator, differentiator, summer, subtractor, filter, oscillator.



Op-amp triangle — inverting input (−) top, non-inverting input (+) bottom, output right.



Inverting amplifier — R_f feedback, R_in input. V_out = −(R_f/R_in)·V_in.



VOLT INSIGHT

Op-amps are the analog swiss army knife. The feedback network sets the behavior. Not the op-amp. Swap the feedback and the same chip becomes an amplifier. Or an integrator. Or a filter. Or an oscillator. The op-amp is the runtime. Your feedback network is the program.



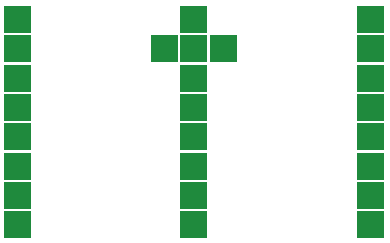
AMP ASKS

Like how a CPU just executes instructions, and the program makes it do different things?



VOLT SAYS

That's a perfect analogy. The op-amp is the CPU. The feedback network is the code.



Op-amp integrator — cap in feedback. V_o = −(1/RC)· V_in dt. The analog computer of the 1960s; still everywhere today.

The two golden rules

For an ideal op-amp with negative feedback, two rules suffice to analyze any circuit: (1) the voltages at the + and − inputs are equal (virtual short); (2) no current flows into either input (infinite input impedance). These two rules plus Kirchhoff give you every op-amp circuit.

Five canonical op-amp circuits

Circuit	Configuration	Gain / Function
Inverting	Input through R1 to −, R_f from − to output, + grounded	A_v = −R_f/R_1
Non-inverting	Input to +, − tied to output through R_f and to ground through R_1	A_v = 1 + R_f/R_1
Voltage follower	Output tied directly to −, input to +	A_v = 1 (buffer)
Summing	Multiple inputs through R_1, R_2, … to −, R_f feedback	V_o = −R_f·(V_1/R_1 + V_2/R_2 + …)
Difference	V_1 to − through R_1, V_2 to + through R_g; matched feedback	V_o = (R_f/R_1)·(V_2 − V_1)

Integrator and differentiator

Replace R_f in an inverting amp with a capacitor: you get an integrator, V_o = −(1/RC)· V_in dt. Replace R_1 with a cap: differentiator, V_o = −RC·dV_in/dt. These are the analog computers of the 1960s — and they are still the basis of every active filter and every analog control loop. The differentiator is noise-amplifying and rarely used in pure form; the integrator is everywhere (PID controllers, active filters, charge amplifiers).

$$\text{Integrator: } V_o = -(1/RC)\cdot \int V_{in}\,dt \quad \big| \quad \text{Differentiator: } V_o = -RC\cdot dV_{in}/dt$$

Real op-amp non-idealities

Real op-amps have finite open-loop gain (10 –10), finite input impedance (M to G), non-zero output impedance (~50 open-loop, much less closed-loop), finite bandwidth (gain-bandwidth product GBW — typically 1–10 MHz), input offset voltage (1–10 mV), input bias current (pA for FET-input, A for bipolar), slew rate (V/ s), and noise. The two that bite most often are GBW (a 1 MHz op-amp at gain 100 has bandwidth of only 10 kHz) and slew rate (a 0.5 V/ s op-amp distorts a 20 V_pp sine above ~4 kHz).

HANDS-ON PROJECT — Bench: Build an Audio Mixer

Goal: sum three audio sources with individual level controls.

Circuit: three inverting-amp inputs through 10k pots summing node op-amp with R_f = 10k output. Add a follower at the output to drive low-impedance headphones or a power amp.

Op-amp: TL072 (low-noise JFET input, GBW = 3 MHz, slew rate = 13 V/ s).

Verify: feed 1 kHz sine to each input at 100 mV. Output should be sum × gain. Add three different frequencies; check the output spectrum has all three peaks with the right amplitudes.

Extension: add a one-pole low-pass at 20 kHz at the output to suppress out-of-band noise.

CHECKPOINT — Can you answer these?

- State the two golden rules of ideal op-amp analysis.
- Design an inverting amp with gain −10. What are R_1 and R_f?
- What is the gain-bandwidth product? Compute the bandwidth of an amp with GBW = 1 MHz at gain 100.
- Why is a differentiator rarely used in pure form?
- What is slew rate, and how does it limit a 20 V_PP sine at 100 kHz?

Module 3.5 — Active Filters: Sallen-Key and MFB

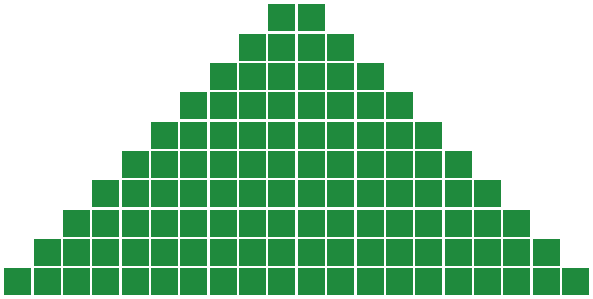
Duration: 6–8 h · **Difficulty:** Intermediate

Passive filters (RC, LC) work, but inductors at audio frequencies are bulky, lossy, and hard to tune. Active filters use op-amps to synthesize the inductor behavior with capacitors and resistors only. The two most common topologies are **Sallen-Key** (non-inverting, low output impedance, easy to cascade) and **Multiple-Feedback (MFB)** (inverting, better high-frequency performance, more component-efficient for high-Q). Both are second-order; cascade them to build higher-order filters.

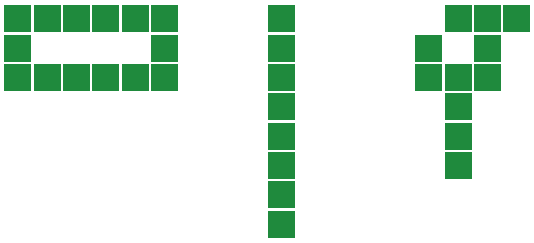
Sallen-Key low-pass

A Sallen-Key low-pass uses two RC sections followed by an op-amp voltage follower. The transfer function is H(s) = ²/(s² + (/Q)·s + ²) where = 1/ (R R C C) and Q = (R R C C)/(R C + R C + R C (1−K)) with K = 1 for the unity-gain version. Choose C = C = C, R = R = R for simplicity: = 1/(RC), Q = 1/(2 − K) = 0.5 for K=1 (Butterworth needs Q = 0.707, achieved by setting K = 1.586).

$$H(s) = {}_0{}^2 / (s^2 + ({}_0/Q)\cdot s + {}_0{}^2) \quad | \quad {}_0 = 1/\ (R_1R_2C_1C_2)$$



Band-pass magnitude response — peaks at , falls off on both sides at ±20 dB/dec.



Sallen-Key 2nd-order low-pass — two RC sections + unity-gain buffer. The classic active-filter building block.

Butterworth, Chebyshev, Bessel — choosing the response

Three classic filter responses, each optimizing a different property. **Butterworth** is maximally flat in the passband — the smoothest magnitude response, at the cost of a relatively slow rolloff (−20n dB/decade for n-th order) and mediocre phase linearity. **Chebyshev** trades passband ripple for steeper rolloff — useful when you need to reject a nearby unwanted signal. **Bessel** maximizes phase linearity (constant group delay) — best for time-domain pulses and data transmission, but the slowest rolloff. Choose based on what matters most in your application: flatness (Butterworth), selectivity (Chebyshev), or pulse fidelity (Bessel).

HANDS-ON PROJECT — Bench: 4th-Order Butterworth Low-Pass at 1 kHz

Goal: cascade two Sallen-Key 2nd-order sections to build a 4th-order filter.

Specs: Butterworth, f_c = 1 kHz, gain = 1 in passband.

Design: each stage: Q = 0.541 (stage 1) and Q = 1.306 (stage 2) for 4th-order Butterworth. Use C = 10 nF, R = 1/(2 ·1000·C) = 15.9 k . Tune the gain of each stage to achieve the required Q (gain K = 3 − 1/Q).

Build & measure: sweep frequency 100 Hz to 100 kHz. Measure magnitude. Verify: flat to 1 kHz, −3 dB at 1 kHz, −80 dB/decade rolloff.

Compare to LTspice: simulate the same circuit; overlay the measured response.

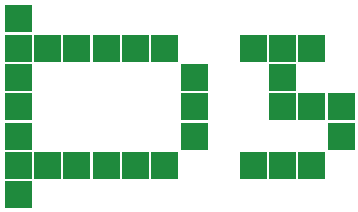
CHECKPOINT — Can you answer these?

- Why use active filters instead of passive LC filters at audio frequencies?
- Design a Sallen-Key low-pass with f_c = 1 kHz using C = 10 nF. What is R?
- Compare Butterworth, Chebyshev, Bessel — what does each optimize?
- How do you build a 6th-order filter from 2nd-order sections? What Q values does each stage need (Butterworth)?

Module 3.6 — Oscillators and Timers: Wien Bridge, Phase-Shift, 555

Duration: 5–7 h · **Difficulty:** Intermediate

An oscillator is an amplifier with positive feedback at a specific frequency. The Barkhausen criterion: at the oscillation frequency, the loop gain must be exactly 1 ($|A| = 1$) and the loop phase must be 0° (or 360°). If $|A| < 1$ the oscillation decays; if $|A| > 1$ it grows until nonlinearity limits it. Real oscillators start with $|A|$ slightly > 1 (so noise can build up) and rely on nonlinearity (clipping, AGC) to stabilize at $|A| = 1$.



555 timer — internal block view: comparators + flip-flop + discharge transistor. The most-sold IC ever.

Three classic oscillators

- **Wien bridge:** RC bandpass in positive feedback, $R_f/R_1 = 2$ in negative feedback. Clean sine output. The classic variable-frequency audio oscillator.
- **Phase-shift:** three RC high-pass sections give 180° phase shift at one frequency; inverting amp gives the other 180°. Simple, fewer components than Wien.
- **Colpitts / Hartley:** LC oscillators for RF (1 MHz+). Use a tank circuit for the frequency-selective element. We will meet these in Phase 8.

The 555 timer — the most popular IC ever

The 555 is an 8-pin oscillator/timer that has been in continuous production since 1972. Two modes: **astable** (free-running oscillator) and **monostable** (one-shot). In astable: frequency $f = 1.44/((R_1 + 2 \cdot R_2) \cdot C)$. Duty cycle $= (R_1 + R_2)/(R_1 + 2 \cdot R_2)$. It is the go-to for cheap square-wave clocks, PWM generators, debouncers, and tone generators.

HANDS-ON PROJECT — Bench: Variable Wien Bridge Sine Source

Goal: build a 100 Hz–10 kHz sine source with under 1% THD.

Circuit: Wien bridge with dual-gang pot for frequency control, JFET for AGC.

Op-amp: TL072 (low distortion, sufficient GBW).

Verify: measure THD with a sound-card spectrum analyzer (e.g., REW or RightMark). Should be under 1% across the frequency range.

Application: use as the input source for the audio amplifier capstone at end of Phase 3.

CHECKPOINT — Can you answer these?

- State the Barkhausen criterion. What happens if |A | > 1? < 1?
- Design a 555 astable for f = 1 kHz with 50% duty cycle. Pick R_1, R_2, C.
- Why does a Wien bridge need an amplitude stabilization mechanism?
- Why are LC oscillators preferred over RC at RF frequencies?

Module 3.7 — Linear Regulators and LDOs

Duration: 4–6 h · **Difficulty:** Intermediate

A linear regulator is a closed-loop amplifier that holds its output voltage constant despite variations in input voltage and load current. It works by dissipating the excess voltage as heat — efficiency is V_out / V_in. A 5 V regulator from 12 V is 42% efficient; the other 58% becomes heat. LDOs (low-dropout regulators) are a subset that can regulate with V_in as little as 100 mV above V_out, useful for battery-powered devices.

$$\text{Efficiency} = V_{out} / V_{in} \quad | \quad P_{diss} = (V_{in} - V_{out}) \cdot I_{load}$$

The classic fixed regulators are LM78xx (positive) and LM79xx (negative); the LM317 is an adjustable version. Modern LDOs (AMS1117, MCP1700, LT3045) target low noise and low quiescent current. All share the same internal topology: a voltage reference (Zener or bandgap), an error amplifier, and a pass transistor (BJT or MOSFET) in a feedback loop. Spec parameters: dropout voltage, line regulation, load regulation, quiescent current, PSRR (power supply rejection ratio), output noise.

HANDS-ON PROJECT — Bench: Adjustable Bench Supply with LM317

Goal: build a 1.25–20 V, 1 A bench supply.

Circuit: LM317 with R1 = 240 , R2 = 5k pot V_out = 1.25·(1 + R2/R1). Add input and output caps (0.33 F and 0.1 F) for stability.

Verify: sweep R2 from 0 to 5k. V_out should track 1.25 to 27 V linearly. Add 1 A load (e.g., 20 power resistor at 20V); verify V_out holds within load regulation spec (typically 0.1%).

Heatsink: at 1 A and 20 V drop, P_diss = 20 W. Heatsink required (thermal resistance < 5°C/W).

Pass: supply holds V_out within 50 mV across the full load range at all output voltages.

CHECKPOINT — Can you answer these?

- Compute the efficiency of a 5V LDO dropping 12V to 5V at 500 mA. What is the dissipated power?
- Why can't a linear regulator boost voltage?
- What is dropout voltage? Why does it matter for battery-powered devices?
- What is PSRR? Why does it matter for analog circuits powered from switching regulators?

Module 3.8 — Phase 3 Capstone: Audio Amplifier

Duration: 8–12 h · **Difficulty:** Intermediate

HANDS-ON PROJECT — Audio Power Amplifier (Class-AB)

Goal: design and build a complete audio power amp: 20 W into 8 , 20 Hz–20 kHz, THD < 1%.

Topology: three-stage: (1) differential input pair (long-tailed pair) for low offset, (2) common-emitter voltage-amp stage with Miller compensation, (3) class-AB output stage (NPN-PNP Darlington or MOSFET follower) for current drive.

Design choices: V_CC = ±25 V rails. Quiescent current 20 mA in output stage (class-AB bias). Feedback factor: closed-loop gain = 1 + R_f/R_in = 23 (28 dB voltage gain), with input sensitivity 700 mV RMS for full output.

Schematic: use KiCad. Typical reference: Douglas Self's Audio Power Amplifier Design Handbook, Chapter 2 (the 'blameless' amplifier).

Build: solder on perfboard or design a PCB in KiCad and order from JLCPCB (~$10 for 5).

Test: drive with the Wien bridge sine source from Module 3.6. Measure THD vs frequency and vs output level. Measure bandwidth. Measure output impedance by loading with 8 and 4 and computing the change.

Document: schematic, PCB layout, BOM, build photos, measurement plots. File in a project notebook — you will use this amp as a test load in later phases.

CHECKPOINT — Can you answer these?

- Sketch the three-stage topology of a class-AB audio amp. What does each stage do?
- Why class-AB and not class-A or class-B? What is the trade-off?
- What is Miller compensation and why is it needed?
- Compute the efficiency of a class-B amp at full output sine. (Hint: 78.5%.)
- How would you protect the output stage against a short circuit?

> "Small-signal linearization is the bridge between the nonlinear physical world and the linear analytical world. Every transistor model — every backprop gradient, every Newton step — is the same trick: zoom in until the curve looks like a line."
>
> — Volt, on linearization



VOLT NARRATES

Phase 3 is done. You've built real amplifiers from diodes, transistors, and op-amps. The audio amplifier capstone is the first project big enough to put on a resume. Phase 4 crosses into the digital domain. The signals are 0s and 1s. The gates are logic. Your CS background starts doing the heavy lifting. The fun part is just beginning.



Phase 4 — Digital Logic & Embedded Systems

Duration: 6 weeks · **Modules:** 10 · **Goal:** take the digital abstraction from gates to a working FPGA prototype and a wireless MCU telemetry demo.

Phase 4 is where your CS background pays the biggest dividends. Boolean algebra, FSMs, K-maps, register files, and pipelining are all things you already know — we just rename them and apply them to hardware. The new material is the physical side: setup and hold constraints, metastability, clock distribution, asynchronous signals, and the dozen other ways real hardware differs from an idealized HDL simulation.



VOLT NARRATES

This is the phase where you stop translating and start recognizing. Boolean algebra here is the same algebra you use in code. FSMs are the same FSMs from your compiler course. Pipelines are the same pipelines from your CPU class. The only new things are setup and hold times. Metastability. Clock domains. The physical reality that occasionally flips the clean abstraction into chaos.

Module 4.1 — Boolean Algebra and K-maps

Duration: 4–6 h · **Difficulty:** Foundation

CS BRIDGE

Boolean algebra bitwise operations in C. AND, OR, NOT, XOR are the same operators whether they appear in C code or as physical gates. De Morgan's laws (¬(A B) = ¬A ¬B) are universal. A sum-of-products (SOP) expression is a disjunction of minterms — a Boolean expression in DNF. K-maps are a visual trick for minimizing SOP expressions; algorithmically they are a special case of the Quine–McCluskey algorithm, which is itself a graph-based prime-implicant search.



VOLT INSIGHT

When you write a & b in C, the compiler emits an AND instruction. The CPU runs it in one cycle. It uses a few hundred transistors. CMOS NAND plus an inverter. The hardware IS the boolean algebra. The translation layer is zero.



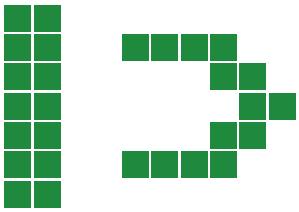
AMP ASKS

So when I write code, I'm literally just describing transistor layouts?

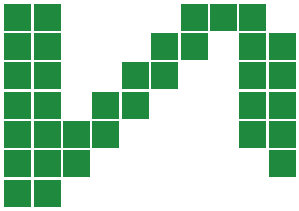


VOLT SAYS

Yes. At the bottom, it's all just switches. Code is a high-level view of switches.



NAND gate — AND followed by an inverter bubble. Universal: any logic function can be built from NANDs alone.



NOR gate — OR followed by an inverter bubble. Also universal; the natural CMOS gate (series PMOS, parallel NMOS).

Every Boolean function can be written as a sum of products (OR of ANDs) or a product of sums (AND of ORs). Both forms can be implemented directly in two-level logic. K-maps minimize these expressions by grouping adjacent 1-cells (for SOP) or 0-cells (for POS) into power-of-two rectangles. The minimized expression has fewer literals, which means fewer gates and less propagation delay.

De Morgan: ¬(A B) = ¬A ¬B | ¬(A B) = ¬A ¬B

K-map mechanics

A K-map is a 2D grid where each cell corresponds to a minterm. Adjacent cells differ in exactly one variable (Gray code ordering). Group adjacent 1-cells in rectangles of 1, 2, 4, 8, 16. Each group corresponds to one product term with the variables that don't change within the group. The minimum cover (fewest groups, each as large as possible) gives the minimum SOP. K-maps work up to 4 variables by hand; for 5+ variables use Quine–McCluskey or, in practice, let the synthesizer (Yosys, ABC) do it.

HANDS-ON PROJECT — Practice: Minimize 10 Boolean Functions

Goal: fluency with K-maps.

Tasks: minimize 10 functions of 3–4 variables by hand using K-maps.

Verify: cross-check each result against Python's sympy.logic.boolalg.simplify_logic.

Pass: 8 of 10 match the sympy result. (Discrepancies are usually equally minimal alternatives — list both.)

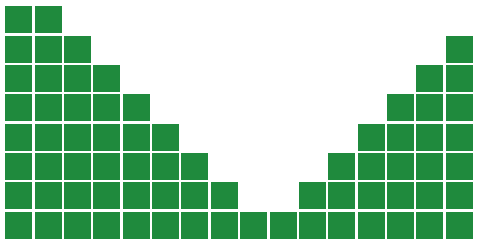
CHECKPOINT — Can you answer these?

- State De Morgan's laws. Convert (A B) (¬A C) to NAND-only form.
- Minimize F(A,B,C) = m(1,3,5,7,9) using a K-map. (Should be C.)
- What is a minterm? A maxterm? How are they related?
- When does Quine–McCluskey beat K-maps? When does Espresso beat both?

Module 4.2 — Combinational Logic: Mux, Decoder, Adder

Duration: 5–7 h · **Difficulty:** Foundation

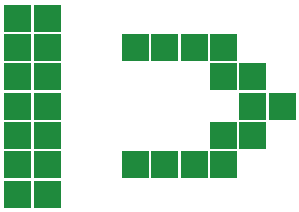
Three workhorse combinational building blocks. **Multiplexer (mux)**: 2 inputs, n select lines, one output — picks one input based on select. A LUT in an FPGA is just a mux. **Decoder**: n inputs, 2 outputs — asserts one output per input combination. Used for address decoding, 7-segment display driving. **Adder**: ripple-carry (slow, simple), carry-lookahead (fast, more gates), carry-skip and carry-select (in between). Modern FPGAs and CPUs use prefix adders (Kogge–Stone, Brent–Kung) for log-time addition.



Multiplexer (mux) — 2 inputs, n select lines, one output. A FPGA LUT is a mux with a truth table stored in SRAM.



Decoder — n inputs, 2 outputs (one-hot). The address decoder in every memory and peripheral.



AND gate — two inputs (left) feed the D-shape, output (right) is high only when both inputs are high.

Combinational hazards

A static-1 hazard occurs when an input change should leave the output at 1 but momentarily drops it to 0 due to unequal path delays. The fix is to add a consensus term — an extra product term that covers the hazardous transition. In practice, hazards rarely matter for synchronous designs (the clock masks them), but they bite hard in asynchronous logic and clock-domain crossings.

HANDS-ON PROJECT — Bench: 7-Segment Decoder on 74HC

Goal: build a BCD-to-7-segment decoder from discrete gates.

Circuit: 74HC08 (AND), 74HC32 (OR), 74HC04 (NOT). 4-bit input from DIP switches, output to common-cathode 7-segment via 330 resistors.

Design: write the truth table, K-map each segment, minimize, draw schematic.

Build & verify: all 10 BCD values (0–9) display correctly. The 6 invalid codes may show garbage or be blanked — your choice.

Extension: replace the gates with a single 74HC47 or 74HC4511 decoder IC. Compare chip count, board area, and power.

CHECKPOINT — Can you answer these?

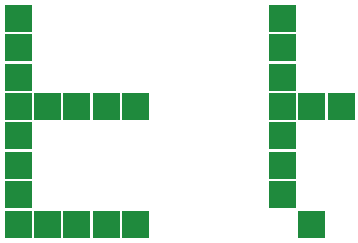
- Design a 4:1 mux using only 2:1 muxes.
- Compare ripple-carry and carry-lookahead adders. What is the delay of each for n bits?
- What is a static-1 hazard? How do you eliminate it?
- Why do modern FPGAs use LUTs instead of muxes?

Module 4.3 — Sequential Logic: Flip-Flops, Counters, Registers

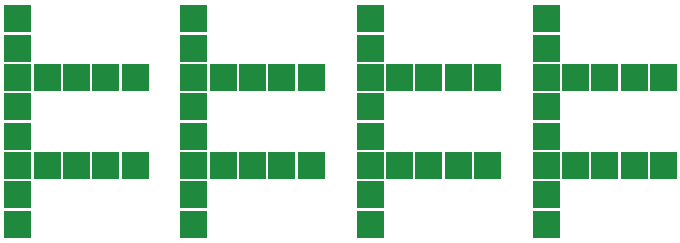
Duration: 6–8 h · **Difficulty:** Intermediate

CS BRIDGE

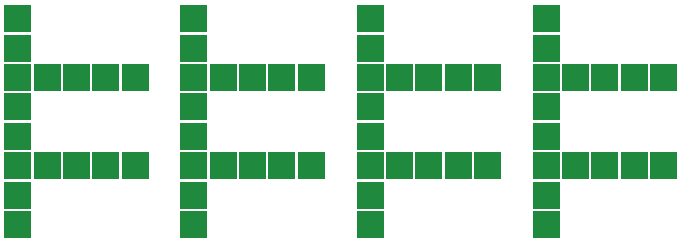
Flip-flop register in a CPU. A D flip-flop is a 1-bit register — it samples the input on the clock edge and holds it until the next edge. A shift register is a queue of flip-flops; a counter is a register that increments. A register file is a bank of flip-flops with read/write ports — exactly the register file in a CPU. The only thing that is new in EE is the timing: setup, hold, and clock-to-Q constraints.



D flip-flop — D input on left, CLK input middle, Q output right. Edge-triggered.



4-bit synchronous counter — four D flip-flops sharing a clock, with combinational next-state logic.



Shift register — chain of D flip-flops passing data on each clock. The hardware behind UART, SPI, I2C.

Four flip-flop types

Type	Behavior	Use
D	Q follows D on clock edge	Universal — most logic uses D
T	Q toggles when T=1 on clock edge	Counters
JK	J=K=1 toggles; J=1,K=0 sets; J=0,K=1 resets; J=K=0 holds	General-purpose (legacy)
SR	S=1 sets, R=1 resets, S=R=1 forbidden	Basic latch (rarely used standalone)

Setup, hold, and metastability

Every flip-flop has two timing constraints: **setup time** t_su (input must be stable for at least t_su before the clock edge) and **hold time** t_h (input must remain stable for at least t_h after the edge). Violate either and the flip-flop may go **metastable** — its output hovers between 0 and 1 for an unbounded time before resolving. Metastability is unavoidable when crossing asynchronous clock domains; the standard fix is a two-flop synchronizer, which reduces the failure probability to (MTBF) of billions of years.

Counters and shift registers

A ripple counter (each flop clocks the next) is slow — the count ripples through n flops. A synchronous counter (all flops share the clock; combinational logic computes the next count) is fast. A ring counter has one flop set, the rest clear; the '1' circulates. A Johnson counter feeds back the complemented output and has 2n states from n flops. Shift registers move data left or right; they are the basis of serial communication (UART, SPI, I2C).

HANDS-ON PROJECT — Bench: Traffic-Light Controller FSM on Arduino

Goal: implement a 6-state traffic-light controller as a Moore FSM.

States: NS_GREEN (8s) NS_YELLOW (2s) EW_GREEN (8s) EW_YELLOW (2s) PED_WALK (10s) FLASH_RED (5s) back to NS_GREEN.

Implementation: use enum for state, switch-case for next-state logic and output logic. Drive 6 LEDs (NS_R, NS_Y, NS_G, EW_R, EW_Y, EW_G).

Extension: add a pedestrian button input. Pressing it during NS_GREEN inserts PED_WALK after NS_YELLOW. Add a debounce circuit on the button.

Tool: Arduino IDE. (We will do the same FSM on an FPGA in Module 4.5.)

CHECKPOINT — Can you answer these?

- Define setup time, hold time, clock-to-Q. What is a timing violation?
- What is metastability? How does a two-flop synchronizer mitigate it?
- Design a 4-bit synchronous up-counter using D flip-flops and combinational logic.
- Why is a ripple counter slower than a synchronous counter?
- Convert a Moore FSM to a Mealy FSM. When does Mealy use fewer states?

Module 4.4 — HDL Intro: Verilog and FSMs

Duration: 6–8 h · **Difficulty:** Intermediate

Hardware Description Languages (HDLs) describe hardware, not programs. The biggest mental shift from software: every assignment happens concurrently — the entire circuit runs in parallel, all the time. There is no 'first this line, then the next' — only 'these signals flow continuously through these combinational clouds, sampled by these registers on these clock edges'. Two HDLs dominate: Verilog (C-like syntax, popular in US/India) and VHDL (Ada-like

syntax, popular in Europe and defense). We will use Verilog.

Verilog in one page

Verilog has two main abstractions: **structural** (instantiating submodules — like netlist) and **behavioral** (always blocks — like code, but every line describes hardware). The two essential constructs:

- **assign** for continuous assignment (combinational logic). Example: assign y = a & b;
- **always @(posedge clk)** for sequential logic. The body describes what happens on each clock edge.
- **always @(*)** for combinational logic in procedural form (preferred over assign for complex logic).

The non-blocking assignment **<=** is used inside clocked always blocks; it samples the RHS at the clock edge and updates the LHS 'simultaneously' with all other non-blocking assignments. The blocking assignment **=** is used in combinational always blocks. Mixing them up is the #1 source of Verilog bugs.

A counter in Verilog

```
module counter(
  input wire clk, rst,
  output reg [3:0] count
);
  always @(posedge clk or posedge rst) begin
    if (rst) count <= 4'b0;
    else     count <= count + 1;
  end
endmodule
```

HANDS-ON PROJECT — Simulate the Counter in Icarus Verilog

Goal: write a testbench and verify the counter.

Tasks: write a testbench that toggles clk every 5 ns, asserts rst for 10 ns at start, then runs for 200 ns. Dump waveforms with $dumpfile / $dumpvars.

Tool: Icarus Verilog (iverilog) + GTKWave. Both free.

Verify: open the .vcd in GTKWave. Confirm count goes 0,1,2,...,15,0,1,...

Synthesize: feed the same module to Vivado. Confirm it synthesizes to a 4-bit counter with no warnings.

CHECKPOINT — Can you answer these?

- What is the difference between = and <= in Verilog?
- What does always @(posedge clk) describe? always @(*)?
- Write a Verilog module for a 4:1 mux.
- What is a testbench? Why do you need one in HDL but not in C?
- Why is Verilog not 'just another programming language'?

Module 4.5 — FPGA Basics with Vivado and Artix-7

Duration: 6–8 h · **Difficulty:** Intermediate

An FPGA (Field-Programmable Gate Array) is a sea of configurable logic blocks (CLBs), each containing a small LUT (typically 6-input) and a flip-flop, interconnected by a programmable routing network. You describe your circuit in Verilog; the synthesizer maps it to LUTs and flops; the place-and-route tool fits them onto the chip; the bitstream is downloaded and the chip becomes your circuit. The Artix-7 (e.g., Nexys A7 or Basys 3 board, ~$150) is the standard student FPGA.

The FPGA design flow

- **Design Entry:** write Verilog. Specify IOs in an .xdc constraints file (pin numbers, voltage standards).
- **Simulation:** testbench in iverilog or Vivado simulator. Verify behavior before touching hardware.
- **Synthesis:** Vivado maps RTL to LUTs, flops, DSPs, BRAMs. Produces a netlist.
- **Implementation:** place-and-route. Fits the netlist onto the physical chip.
- **Bitstream generation:** produces a .bit file.
- **Programming:** download the bitstream to the FPGA over JTAG.
- **On-chip debug:** Vivado Logic Analyzer (ILA) captures internal signals for debugging.

HANDS-ON PROJECT — FPGA: Implement the Traffic-Light FSM from 4.3

Goal: port the Arduino traffic-light FSM to the Artix-7.

Tool: Vivado WebPACK + Nexys A7 (or Basys 3).

Tasks: same FSM as 4.3. Map the 6 outputs to 6 LEDs on the board. Use the 100 MHz on-board clock; divide down to 1 Hz with a counter.

Constraints: write a .xdc file mapping each port to the right pin.

Verify: the LEDs cycle through the FSM exactly as the Arduino did. Congratulations — you have just implemented hardware from scratch.

Debug: insert an Integrated Logic Analyzer (ILA) on the state register. Trigger on state transition. Capture the state sequence.

CHECKPOINT — Can you answer these?

- What is a CLB? A LUT? A slice?
- List the steps of the FPGA design flow.
- What is in an .xdc file? Why do you need it?
- How would you debug an FPGA design that works in simulation but not on hardware?
- Compare FPGA and ASIC design flows. When would you choose each?

Module 4.6 — Microcontrollers: Arduino to STM32

Duration: 8–10 h · **Difficulty:** Intermediate

CS BRIDGE

MCU tiny computer with memory-mapped I/O. A microcontroller is a CPU core + RAM + flash + peripherals (GPIO, ADC, timers, UART, SPI, I2C, PWM, DMA) on one chip. To a CS engineer, the model is: each peripheral is a set of memory-mapped registers; you write device drivers by reading and writing those registers. Arduino abstracts this away behind friendly APIs; STM32 makes you deal with it directly via the CMSIS layer or via STM32 HAL. Bare-metal STM32 is closer to embedded Linux driver development than to Arduino.

The Arduino abstraction

Arduino wraps the Atmel AVR (or now also ARM) MCU in a friendly C++ API: pinMode, digitalWrite, digitalRead, analogRead, analogWrite, Serial.begin, delay. Under the hood these are macros that write to specific AVR registers. Great for prototyping, but the abstraction hides timing — digitalWrite takes ~5 s because of the lookup table that maps Arduino pin numbers to AVR ports. For sub-microsecond timing, you must bypass the abstraction.

STM32 — the real-world MCU

The STM32 family (ST Microelectronics, ARM Cortex-M cores) is the workhorse of modern embedded systems. Common parts: F103 (cheap, popular in Blue Pill boards), F446 (168 MHz, FPU, used in Nucleo-F446RE), H7 (480 MHz, top-end). The development environment is STM32CubeIDE (Eclipse-based, free). Configure peripherals via the .ioc graphical tool, then write code in C using the HAL (Hardware Abstraction Layer) or the lower-level LL (Low-Layer) libraries. For real-time work, you often bypass HAL and write directly to registers.



VOLT SAYS

When you drop from Arduino to STM32, you cross the same line as moving from Python to C. The HAL is friendly but slow. The bare registers are fast but unforgiving. Real products live in between. HAL for setup. Direct register access for the hot paths.



AMP ASKS

Like using a high-level API for the easy stuff and dropping to syscalls for performance?



VOLT SAYS

Exactly. Same idea. Same tradeoffs.

Clock trees and power management

An MCU has an internal clock tree that distributes the system clock to the CPU, the buses (AHB, APB1, APB2), and the peripherals. Each peripheral has its own prescaler. Configuring the clock tree is the first step of any STM32 project — a mistake here means your UART runs at the wrong baud rate. Low-power modes (Sleep, Stop, Standby) gate the clock to idle peripherals to save power. Battery-powered devices spend most of their time in Stop mode, waking up on an interrupt.

HANDS-ON PROJECT — Bench: STM32 GPIO Toggle at 1 Hz and 1 MHz

Goal: feel the difference between HAL and direct register access.

Tasks: (1) Toggle a GPIO at 1 Hz using HAL_GPIO_TogglePin + HAL_Delay. (2) Toggle the same GPIO at maximum speed using direct register writes (GPIOx->BSRR). Scope the output. Measure toggle frequency.

Verify: with HAL you get maybe 100 kHz. With direct writes on STM32F446 you can hit 84 MHz (half the system clock). That's 1000× faster.

Lesson: the HAL is convenient but hides performance. For tight loops, drop to register level.

CHECKPOINT — Can you answer these?

- What is the difference between Arduino and STM32 from a developer's perspective?
- Sketch an MCU clock tree: HSE PLL AHB APB1/APB2 peripherals.
- How do you configure an STM32 GPIO as output push-pull at 50 MHz?
- What is DMA? Why is it faster than interrupt-driven I/O?
- Name three low-power modes on an STM32. When would you use each?

Module 4.7 — Interrupts, Timers, ADC, DAC, PWM

Duration: 8–10 h · **Difficulty:** Intermediate

These are the five peripheral categories you will use in every MCU project. **Interrupts** let the CPU respond to events without polling. **Timers** count clock cycles and generate periodic events or PWM. **ADCs** sample analog voltages. **DACs** generate analog voltages. **PWM** encodes an analog value (the duty cycle) in a digital waveform — the basis of every switch-mode converter and motor driver.

Interrupts and priorities

An interrupt vector table maps interrupt sources to handler functions. The CPU saves its state (PC, registers), jumps to the handler, and restores state on return. The ARM Cortex-M NVIC (Nested Vectored Interrupt Controller) supports 16 priority levels — higher-priority interrupts can preempt lower ones. The rule: handlers must be short (microseconds, not milliseconds). Long handlers block other interrupts and break real-time behavior. For long work, set a flag in the handler and process it in the main loop.

Timers — the workhorse peripheral

A timer is a counter driven by a clock. It can: count up, count down, count up-down (center-aligned PWM). It generates interrupts on overflow, on compare match, on capture (an external edge latches the counter into a register — useful for measuring frequency or pulse width). PWM is just a timer running in compare mode where the output pin toggles on compare match — the duty cycle is compare_value / period.

$$f_{PWM} = f_{timer} / (\text{prescaler} \cdot \text{ARR}) \quad | \quad \text{Duty} = \text{CCR} / \text{ARR} \quad | \quad \text{Resolution} = \log_2(\text{ARR}) \text{ bits}$$

ADCs — sampling the analog world

Most MCU ADCs are SAR (Successive Approximation Register) converters: n-bit resolution requires n clock cycles per sample. STM32 ADCs typically run at 1–5 MSPS with 12-bit resolution. Sampling analog signals correctly requires an anti-alias filter (low-pass at f_sample/2) — without it, high-frequency noise folds back into the band of interest. The ADC has a sample-and-hold capacitor that must charge through the source impedance; if the source is high-impedance, you need a longer sample time or an op-amp buffer.



VOLT WARNS

An ADC without an anti-alias filter is broken by design. Sample rate alone does not stop aliasing. You have to low-pass the signal before sampling. If you don't, high-frequency noise folds back into your band. It becomes indistinguishable from signal.



AMP ASKS

Like the wagon-wheel effect in movies? Where wheels look like they're spinning backwards?



VOLT SAYS

Yes. Same thing. Same math. Aliasing in any sampled system.

HANDS-ON PROJECT — Bench: STM32 PWM Motor Speed Control

Goal: control a small DC motor speed with PWM from an STM32.

Hardware: STM32 Nucleo, L298N motor driver, 6 V DC motor, 12 V supply.

PWM: 10 kHz, 8-bit resolution (0–255). Timer 1 channel 1.

Drive: CCR controls duty (0=stop, 255=full speed).

Measure: sweep CCR from 0 to 255 in steps of 16. Measure motor RPM with a reflective IR sensor and a scope. Plot RPM vs duty cycle. Note the dead zone at low duty (motor needs ~30% duty to overcome friction).

Extension: close the loop — add a PI controller (more on this in Phase 6) to hold a target RPM.

CHECKPOINT — Can you answer these?

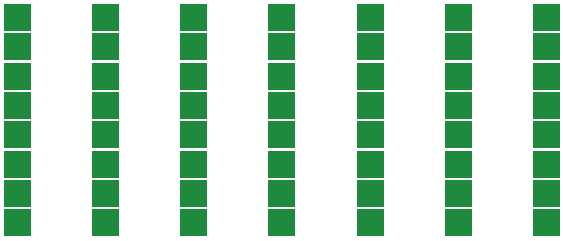
- What is the NVIC? How do priorities work in ARM Cortex-M?
- Compute the PWM frequency for a 84 MHz timer with prescaler=83 and ARR=999.
- What is sample-and-hold? Why does the ADC need it?
- What is aliasing? How do you prevent it when sampling an audio signal?
- Why should interrupt handlers be short? How do you handle long work safely?

Module 4.8 — Communication: UART, SPI, I2C, CAN

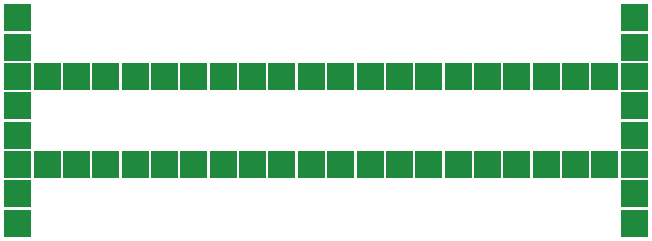
Duration: 6–8 h · **Difficulty:** Intermediate

CS BRIDGE

Serial protocols network protocols. UART is a point-to-point byte stream with start/stop framing — like a TCP socket without retries. SPI is a shared bus with explicit chip-select lines — like a USB hub. I2C is a shared bus with addressing — like Ethernet with MAC addressing. CAN is a multi-master bus with priority arbitration and error detection — like a contention-based LAN. The same design trade-offs (latency vs throughput, complexity vs robustness) appear in every protocol you already know.



SPI bus — shared SCK/MOSI/MISO plus a dedicated CS per slave. Full-duplex, fast, but pin-hungry.



I2C bus — 2 wires (SDA, SCL), 7-bit addressing, multi-master arbitration. Slow but beautifully pin-frugal.

Protocol	Wires	Topology	Speed	Typical use
UART	2 (TX, RX)	Point-to-point	9600–3 Mbaud	Debug serial, GPS, BLE modules
SPI	4 (MOSI, MISO, SCK, CS)	Bus, CS per slave	1–50 MHz	SD cards, displays, fast ADCs
I2C	2 (SDA, SCL) + gnd	Multi-drop, addressed	100/400 kHz, 1/3.4 MHz	Sensors, EEPROMs, small displays
CAN	2 (CANH, CANL) + gnd	Differential bus	up to 1 Mbit/s	Automotive, industrial
USB	2 (D+, D−) + power	Tiered star	1.5/12/480/5000 Mbit/s	PC peripherals
Ethernet	4 (2 pairs) or 8	Star via switch	10/100/1000/10000 Mbit/s	Networking

I2C — addressing and arbitration

I2C uses 7-bit (or 10-bit) addresses: every device on the bus has a unique address. The master generates the clock (SCL) and initiates transactions; slaves respond. Multiple masters are allowed — collision detection is arbitration: the master writing a 0 wins over one writing a 1 (wired-AND). Pull-up resistors on SDA and SCL (typically 4.7 k) set the bus idle state. I2C is slow but cheap (2 wires), which is why it dominates sensor connectivity.

HANDS-ON PROJECT — Bench: STM32 + BME280 over I2C

Goal: read temperature, humidity, and pressure from a BME280 sensor over I2C.

Wire: BME280 VCC 3.3V, GND GND, SDA PB7, SCL PB6 (I2C1 on Nucleo-F446RE).

Address: 0x76 or 0x77 depending on the SDO pin.

Code: use STM32 HAL I2C: HAL_I2C_Mem_Read to read the calibration registers and the data registers. Apply Bosch's compensation formulas (in the datasheet) to convert raw ADC values to physical units.

Verify: compare temperature reading to a thermometer, pressure to weather report, humidity to a hygrometer. All within ±5%.

Extension: log to a PC over UART at 1 Hz for 24 hours. Plot in Python.

CHECKPOINT — Can you answer these?

- Compare UART, SPI, I2C: wires, topology, speed, addressing.
- Why does I2C need pull-up resistors? What value, and why?
- How does I2C arbitration work? Can SPI do the same?
- Why is CAN used in cars instead of I2C?

Module 4.9 — ESP32 and Wireless Connectivity

Duration: 6–8 h · **Difficulty:** Intermediate

The ESP32 (Espressif, ~$5) is a dual-core Xtensa LX6 MCU at 240 MHz with built-in WiFi and Bluetooth. It is the de-facto standard for cheap IoT. Programmed via Arduino IDE (easy) or ESP-IDF (Espressif's FreeRTOS-based SDK, more powerful) or PlatformIO. The ESP32 has plenty of peripherals: ADCs, DACs, I2C, SPI, UART, PWM, capacitive touch, and a ULP (Ultra-Low-Power) coprocessor that runs while the main cores sleep.

WiFi on ESP32

The ESP-IDF WiFi stack is a FreeRTOS task. You configure it as a station (connect to an AP), an AP (other devices connect to it), or both. Common patterns: HTTP client (fetch data), HTTP server (serve a web page), MQTT (publish/subscribe to a broker), WebSocket (full-duplex). For Arduino users, the WiFi, HTTPClient, and PubSubClient libraries cover 90% of use cases.

Over-the-air updates (OTA)

Production IoT devices need remote firmware updates. The ESP32 supports OTA: download a new firmware image over HTTPS, write it to the unused OTA partition, verify the signature, swap the boot partition, and reboot. The dual-bank flash layout makes this safe — if the new image is corrupt, the bootloader falls back to the previous one. This is the same pattern as Android A/B updates.

HANDS-ON PROJECT — Bench: ESP32 WiFi Telemetry Dashboard

Goal: publish BME280 sensor data to a cloud dashboard via MQTT.

Hardware: ESP32 devkit + BME280 (I2C).

Broker: use a free HiveMQ or Mosquitto instance (or run mosquitto on a Pi).

Code: every 30 s: read sensor, publish JSON to topic sensors/esp32-01. Connect a web dashboard (e.g., Node-RED or Grafana) that subscribes and plots.

Power: measure current draw: deep sleep between readings, wake on timer. Target: 100 A average. Compute battery life on 2× AA.

Extension: add OTA — push a firmware update over MQTT and verify the device updates and reboots.

CHECKPOINT — Can you answer these?

- What is ESP-IDF? How does it differ from the Arduino-ESP32 core?
- How does ESP32 OTA work? Why is dual-bank flash important?
- Compute the battery life of an ESP32 that wakes every 60 s for 200 ms (200 mA active) and sleeps at 10 A otherwise, on a 2000 mAh Li-ion cell.
- When would you choose ESP32 over STM32? When the reverse?

Module 4.10 — Phase 4 Capstone: Digital FSM + MCU Driver

Duration: 10–14 h · **Difficulty:** Intermediate

HANDS-ON PROJECT — Phase 4 Capstone: FPGA FSM + STM32 Sensor Driver

Goal: build a system where an FPGA FSM drives an STM32 over SPI to read sensors.

Architecture: STM32 (master) reads BME280 over I2C, formats the data, sends it over SPI to the FPGA. The FPGA FSM receives the data, displays temperature on 4 seven-segment displays, and triggers alarms on LEDs if temperature/humidity are out of range.

STM32 side: HAL SPI master at 1 MHz. Use DMA for the SPI transfer. Pack temp/humidity/pressure into a 12-byte packet.

FPGA side: Verilog SPI slave (synchronous to FPGA clock, with synchronizer for SPI clock). FSM: IDLE RECEIVING UPDATE_DISPLAY CHECK_ALARMS IDLE.

Integration: wire the two boards together. Verify end-to-end: breathe on the BME280 temperature rises display updates within 100 ms.

Document: full block diagram, Verilog source, STM32 source, integration notes, photos, video. Save the project — you will reuse parts of it in Phase 10 capstones.

CHECKPOINT — Can you answer these?

- How would you debug an SPI link that works at 100 kHz but fails at 10 MHz?
- Why do you need a synchronizer on the SPI clock inside the FPGA?
- How does DMA improve SPI throughput? When does it not help?
- What is the difference between a Moore and Mealy FSM in Verilog implementation?

> "A flip-flop is a register. A shift register is a queue. A counter is an accumulator. A memory is an array. The hardware you are about to design is the data structures you already know — made of silicon instead of software."
>
> — Volt, on digital primitives



VOLT NARRATES

Phase 4 is done. You can write Verilog. Target an FPGA. Drive peripherals over SPI and I2C. Ship firmware on an STM32. Publish telemetry over WiFi. The capstone put all of it into one system. That integration is what separates a working engineer from a student. Phase 5 takes you into the frequency domain. Signals become spectra. Convolution becomes multiplication.



Phase 5 — Signals, Systems & DSP

Duration: 7 weeks · **Modules:** 10 · **Goal:** master the frequency-domain machinery — Fourier, Laplace, Z, DFT, FFT — and build a real-time audio spectrum analyzer and EQ on the ESP32.

Signals and systems is the math that ties together circuits, control, communications, and DSP. Once you internalize the Fourier transform, half of EE becomes a single idea seen from different angles. This phase is the most math-heavy of the curriculum — but as a CS engineer, you already know the core algorithm (FFT) and the core data structure (a discrete sequence). The work is to connect them to the underlying linear algebra and differential equations.



VOLT NARRATES

Phase 5 has the biggest idea in this book. Every signal is a sum of sine waves. Once you accept that, half of EE collapses into one fact seen from different angles. Filters. Modulation. Sampling. Control. All of it is the Fourier transform wearing different costumes.



AMP ASKS

Like how any function can be written as a Taylor series, but with sine waves instead of polynomials?



VOLT SAYS

Yes. Same idea. Different basis. Sines instead of powers of x.

Module 5.1 — Continuous-Time Signals and Systems

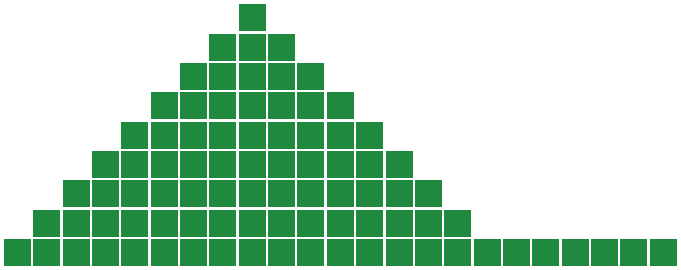
Duration: 5–7 h · **Difficulty:** Foundation

A continuous-time signal x(t) is a function from to (or). A continuous-time system maps an input signal to an output signal: y(t) = T{x(t)}. Three properties define the useful class of systems: **linearity** (T{a·x + b·x } = a·T{x } + b·T{x }), **time-invariance** (if x(t) y(t), then x(t−) y(t−)), and **causality** (y(t) depends only on x( ) for t). A system with all three is an LTI causal system — the kind we can analyze with convolution and Fourier transforms.

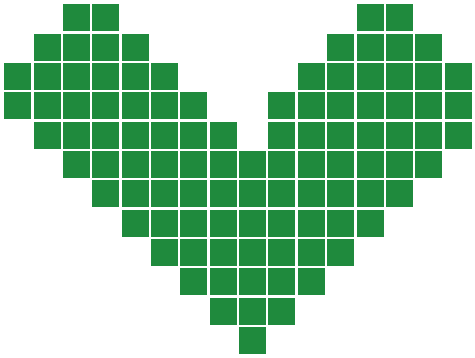
Impulse response and convolution

For an LTI system, the response to a unit impulse (t) completely characterizes the system. Call that response h(t). Then by linearity and time-invariance, the response to any input x(t) is the convolution: y(t) = (x * h)(t) = x( )·h(t−)d . This is the central theorem of LTI theory.

$$y(t) = (x \quad h)(t) = \int_{-}^{+} x(\ )\cdot h(t - \ )\, d$$



Impulse response h(t) of an LTI system. Convolve it with any input x(t) and you get the output y(t). The system's complete fingerprint.



Convolution demo — two signals sliding across each other. The overlap area at each shift is one output sample.

CS BRIDGE

Convolution 1D image filter. A 3×3 image convolution kernel and a 1D FIR filter are the same operation. Image: y[i,j] = h[k,l]·x[i−k, j−l]. Signal: y[n] = h[k]·x[n−k]. The only difference is dimensionality. A blur kernel is a 2D moving-average filter; an edge-detection kernel is a 2D derivative filter; a Gaussian blur is a 2D low-pass. Once you see this, every image-processing library you have ever used becomes a collection of 2D filters you already understand.

HANDS-ON PROJECT — Python: Convolution Sandbox

Goal: feel what convolution does, numerically.

Tasks: build h(t) = [1, 1, 1]/3 (moving average). Convolve with various inputs: (a) a step, (b) a sine, (c) white noise. Plot input and output side by side.

Try different h: a delta [1,0,0,...] — output = input. A difference h=[1,−1] — a derivative. A Gaussian — a smoother.

Insight: convolution with h is linear filtering; the shape of h determines what frequencies pass and what gets rejected.

CHECKPOINT — Can you answer these?

- Define linearity, time-invariance, causality. Give an example of a system that violates each.
- What is an impulse response? Why does it characterize an LTI system?
- Compute the convolution of x = [1, 2, 3] and h = [1, 1]. (Should be [1, 3, 5, 3].)
- Why is convolution the natural operation for LTI systems?

Module 5.2 — LTI Systems: Convolution and Eigenfunctions

Duration: 5–7 h · **Difficulty:** Foundation

The key insight that makes frequency-domain analysis possible: complex exponentials $e^{j\ t}$ are eigenfunctions of any LTI system. Feed $e^{j\ t}$ into a system with impulse response h(t), and the output is H(j)·$e^{j\ t}$ — the same function, scaled by a complex number H(j). That complex number is the **frequency response** of the system. This is why sinusoids are special — they pass through LTI systems unchanged in shape, only scaled and shifted in phase.

$$H(j\ ) = \int_{-}^{+} h(t)\cdot e^{-j\ t}\,dt \quad \text{(Fourier transform of h)}$$

Once you accept that complex exponentials are eigenfunctions, every signal can be decomposed into a sum of them (Fourier transform), each component scaled by H(j) independently, and the output reconstructed by the inverse transform. Convolution in time = multiplication in frequency. This is the single most important identity in signal processing: **y(t) = (x h)(t) Y(j) = X(j)·H(j)**.

CHECKPOINT — Can you answer these?

- Show that e^(j t) is an eigenfunction of any LTI system. What is the eigenvalue?
- State the convolution theorem. Why is it useful?
- Define H(j). What is its magnitude? Its phase?
- How do you find H(j) for an RC low-pass from its differential equation?

Module 5.3 — Fourier Series and CTFT

Duration: 8–10 h · **Difficulty:** Intermediate

The Fourier series represents a periodic signal as a sum of harmonically related sinusoids: x(t) = $\sum_{k}$ c_k·$e^{jk\ t}$, where = 2 /T is the fundamental. The coefficients c_k = (1/T)· $\int_{0}^{T}$ x(t)·$e^{-jk\ t}$dt. The Fourier transform generalizes this to aperiodic signals: X(j) = $\int$ x(t)·$e^{-j\ t}$dt, x(t) = (1/2)· $\int$ X(j)·$e^{j\ t}$d .

Fourier transform pairs you must know

Time domain x(t)	Frequency domain X(j)	What
(t) (impulse)	1 (flat)	An impulse contains all frequencies equally.
1 (constant)	2 · ( )	A DC signal has only =0.

Time domain x(t)	Frequency domain X(j)	What
cos(t)	·[(−) + (+)]	A cosine = two spectral lines.
sin(t)	/j·[(−) − (+)]	A sine = two spectral lines, odd.
rect(t/)	·sinc(/2)	A pulse = sinc spectrum.
$e^{-at}u(t)$	1/(a + j)	Exponential decay = low-pass.
$e^{-a\|t\|}$	2a/(a² + ²)	Two-sided decay.
(t − nT)	(2 /T)· (− k·2 /T)	Impulse train impulse train.

Properties that make Fourier useful

- **Linearity:** a·x + b·x a·X + b·X .
- **Time shift:** x(t−) e^(−j)·X(j) — phase shift only, magnitude unchanged.
- **Frequency shift:** e^(j t)·x(t) X(j(−)) — basis of AM modulation.
- **Time scaling:** x(a·t) (1/|a|)·X(j /a) — compress in time, expand in frequency. This is why a short pulse has wide spectrum.
- **Convolution:** x*h X·H — convolution in time = multiplication in frequency.
- **Multiplication:** x·y (1/2)·X*Y — multiplication in time = convolution in frequency. Basis of modulation.
- **Parseval:** $\int |x(t)|^2 dt = (1/2\ ) \cdot \int |X(j\ )|^2 d$ — energy conserved.

HANDS-ON PROJECT — Python: Fourier Series of a Square Wave

Goal: see how a square wave is built from sinusoids.

Tasks: synthesize a 1 kHz square wave by summing the first N odd harmonics (1, 3, 5, 7, ...). Plot for N = 1, 3, 7, 21, 101. Watch the square emerge.

Insight: a square wave = (4/)· (1/(2k+1))·sin((2k+1)· t). Each additional harmonic reduces the ripple.

Extension: do the same for a triangle, sawtooth, and a half-wave rectified sine. Compare the spectra. Notice that smoother time-domain shapes have faster spectral rolloff.

CHECKPOINT — Can you answer these?

- State the Fourier transform and its inverse.
- What is the Fourier transform of a rectangle pulse? Of a delta function?
- State the convolution theorem and the multiplication theorem.
- Why does a shorter pulse have a wider spectrum?
- Compute the Fourier series of a square wave. How fast do the coefficients decay?

Module 5.4 — Laplace Transform and the s-Domain

Duration: 8–10 h · **Difficulty:** Intermediate

The Laplace transform generalizes the Fourier transform by allowing complex frequencies s = + j . X(s) = x(t)·e^{-st}dt. The Laplace transform exists for signals that don't have a Fourier transform (e.g., growing exponentials) and provides information about stability (poles in the left half s-plane stable). The Fourier transform is the Laplace transform evaluated along the j axis (= 0).

Why Laplace? Differential equations become algebra

Differentiation in time = multiplication by s. Integration in time = division by s. The Laplace transform converts a linear ODE with constant coefficients into an algebraic equation in s. Solve the algebra, invert the transform, done. This is how every circuit is analyzed in the frequency domain — replace C by 1/(sC), L by sL, R stays R, and the circuit becomes a resistor network in s. The same idea extends to mechanical systems (mass inductor, spring capacitor, damper resistor) — that is why the same math models circuits, vibrations, and control systems.

Poles, zeros, and stability

A transfer function H(s) is a ratio of polynomials in s. The roots of the numerator are **zeros** (H 0 at those s values). The roots of the denominator are **poles** (H). Poles in the left half plane (real part negative) correspond to decaying exponentials in time — stable. Poles on the j axis correspond to sustained oscillations. Poles in the right half plane correspond to growing exponentials — unstable. This is the entire stability story of linear systems in one picture.

$$H(s) = N(s) / D(s) = K \cdot (s - z_i) / (s - p_k)$$

Laplace pairs you must know

x(t)	X(s)	ROC
(t)	1	all s
u(t) (unit step)	1/s	Re(s) > 0
e^(−at)·u(t)	1/(s+a)	Re(s) > −a
t·e^(−at)·u(t)	1/(s+a)²	Re(s) > −a
sin(t)·u(t)	/(s² + ²)	Re(s) > 0
cos(t)·u(t)	s/(s² + ²)	Re(s) > 0
e^(−at)·sin(t)·u(t)	/((s+a)² + ²)	Re(s) > −a

HANDS-ON PROJECT — Python: Solve an RLC Transient with Laplace

Goal: verify that the Laplace method gives the same answer as direct ODE solution.

Circuit: series RLC, R=10 , L=1mH, C=10 F, step input V_s=5V at t=0.

Laplace method: write H(s) = (1/LC) / (s² + (R/L)s + 1/(LC)). Find poles: s = −5000 ± j·8660 (underdamped). Inverse Laplace: v_C(t) = 5·[1 − e^(−5000t)·(cos 8660t + 0.577·sin 8660t)].

Verify: compare to scipy.integrate.odeint on the original ODE. The two should overlay exactly.

LTspice: run transient sim. Overlay the Python curves.

CHECKPOINT — Can you answer these?

- Why does the Laplace transform use s = + j instead of just j ?
- How do you determine stability from pole locations?
- Compute the Laplace transform of e^(−2t)·sin(3t)·u(t).
- Find the poles of H(s) = 100/(s² + 10s + 100). Classify the response.
- Why is differentiation in time = multiplication by s in Laplace domain?

Module 5.5 — Sampling Theorem and Discrete-Time Signals

Duration: 6–8 h · **Difficulty:** Intermediate

Sampling converts a continuous signal x(t) into a discrete sequence x[n] = x(n·T), where T = 1/f_s is the sample period. The **Sampling Theorem** (Nyquist–Shannon): a band-limited signal with maximum frequency f_max can be perfectly reconstructed from samples taken at f_s 2·f_max. The minimum rate 2·f_max is the **Nyquist rate**. Below it, aliasing occurs — high-frequency content folds back into the baseband and is irrecoverably mixed with the signal.

$$f_s \quad 2 \cdot f_{max} \quad \text{(Nyquist criterion)}$$

CS BRIDGE

Aliasing texture mapping at wrong mip level. In graphics, sampling a high-frequency texture without mip-mapping produces moiré patterns — aliasing. The Nyquist theorem is the DSP version: you must low-pass before you sample, or high frequencies will alias. The same math underlies anti-aliasing in graphics (MSAA, SSAA) and anti-aliasing filters in ADCs. An ADC without an anti-alias filter is broken by construction.

Anti-alias filtering

Before sampling, you must low-pass the signal at f_s/2 (the Nyquist frequency). In practice, the filter is not a brick wall, so you sample at 2.5–10× the highest frequency of interest and use the extra margin for the filter roll-off. Audio CDs sample at 44.1 kHz to cover 20 kHz audio with a 2.1 kHz transition band. Oversampling sigma-delta ADCs sample at 64× or 128× the Nyquist rate and use digital filtering to decimate — much easier than building a sharp analog filter.

Reconstruction — the DAC and the zero-order hold

Reconstruction converts x[n] back to x(t). The ideal interpolator is the sinc function: x(t) = x[n]·sinc((t − nT)/T). Real DACs use a zero-order hold (ZOH) — each sample is held constant until the next. ZOH introduces a sinc-shaped frequency response with nulls at multiples of f_s. A reconstruction filter (analog low-pass at f_s/2) smooths the staircase.

HANDS-ON PROJECT — Python: Aliasing Demo

Goal: see aliasing numerically.

Tasks: sample a 10 kHz sine at 8 kHz. The samples should look like a 2 kHz sine (10 − 8 = 2). Verify by FFT.

Now: sample at 25 kHz. No aliasing; the FFT shows a clean 10 kHz peak.

Anti-alias: add a 4 kHz low-pass filter before the 8 kHz sampler. The 10 kHz sine is filtered out; the samples are near zero. No alias.

Insight: the anti-alias filter is non-negotiable. Sample rate alone is not enough — you must also filter.

CHECKPOINT — Can you answer these?

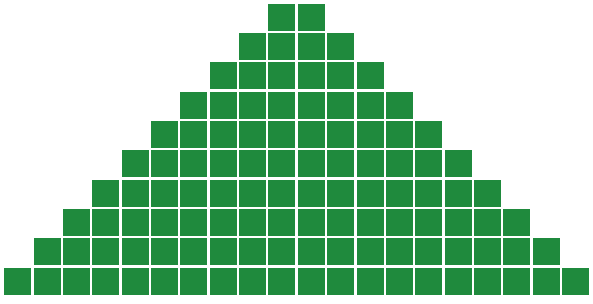
- State the Nyquist–Shannon sampling theorem.
- What is aliasing? How do you prevent it?
- A signal has energy up to 50 kHz. What is the minimum sample rate?
- Why does a CD use 44.1 kHz sampling for 20 kHz audio?
- What is a zero-order hold? What is its frequency response?

Module 5.6 — Z-Transform and Discrete-Time Systems

Duration: 7–9 h · **Difficulty:** Intermediate

The Z-transform is the discrete-time analog of the Laplace transform: $X(z) = \sum_n x[n]\cdot z^{-n}$. The complex variable z is related to s by $z = e^{sT}$. The j axis in the s-plane maps to the unit circle in the z-plane. Left-half s-plane (stable) maps to inside the unit circle. Poles inside $|z|=1$ are stable. Poles outside are unstable. Poles on the unit circle are marginally stable (oscillatory).

$$z = e^{sT} \quad | \quad X(z) = \sum_{n=-\infty}^{+\infty} x[n]\cdot z^{-n}$$



Digital band-pass response — poles near the unit circle at angle ± produce this shape.

Difference equations and transfer functions

A discrete LTI system is described by a linear constant-coefficient difference equation: y[n] = − $\sum_{k=1}^{N}$ a_k·y[n−k] + $\sum_{k=0}^{M}$ b_k·x[n−k]. Taking the Z-transform (zero initial conditions): Y(z)·(1 + $\sum$ a_k·z^{-k}) = X(z)· $\sum$ b_k·z^{-k}. So H(z) = Y(z)/X(z) = ($\sum$ b_k·z^{-k})/(1 + $\sum$ a_k·z^{-k}). If all a_k = 0: FIR (finite impulse response) — the filter is always stable. If some a_k 0: IIR (infinite impulse response) — stability depends on pole locations.

HANDS-ON PROJECT — Python: Filter a Signal with IIR

Goal: design and apply a 1st-order IIR low-pass.

Filter: y[n] = (1−)·y[n−1] + ·x[n], where = 1 − e^(−2 ·f_c/f_s). This is a one-pole low-pass at f_c.

Apply: filter a signal that is 50 Hz + 1 kHz at f_s = 8 kHz with f_c = 200 Hz. The 1 kHz should be attenuated, the 50 Hz should pass.

Compare to FIR: design an FIR with the same cutoff (scipy.signal.firwin) and apply. Compare the outputs and the computational cost.

Insight: IIR is cheaper (fewer coefficients) but has phase nonlinearity. FIR is exactly linear phase but needs more taps for the same selectivity.

CHECKPOINT — Can you answer these?

- What is the relationship between s-plane and z-plane?
- How do you test stability of an IIR filter?
- Why are FIR filters always stable? What is the cost?
- Convert the analog H(s) = 1/(s+1) to a digital H(z) using the bilinear transform with f_s = 1000 Hz. Where is the pole?

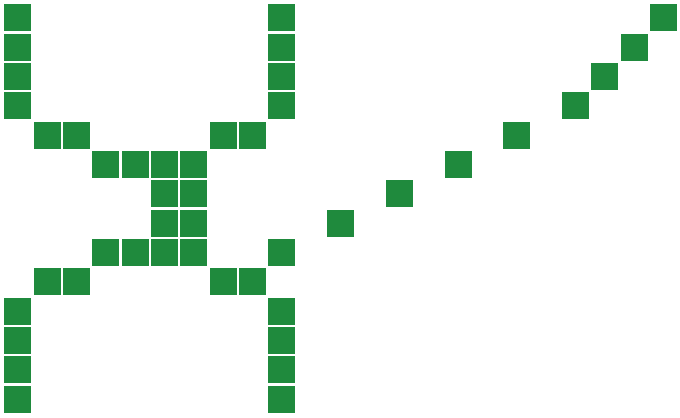
Module 5.7 — DTFT, DFT, and FFT

Duration: 8–10 h · **Difficulty:** Intermediate

> "Every signal is a sum of sinusoids. This is the single most important sentence in electrical engineering — every other topic in the second half of the curriculum is a corollary."
>
> — Volt, on Fourier

CS BRIDGE

FFT divide-and-conquer sorting. The FFT is the divide-and-conquer algorithm for the DFT, exactly as mergesort is for sorting. The DFT is O(N^2); the FFT is O(N log N). The radix-2 FFT splits an N-point DFT into two N/2-point DFTs (even and odd samples), recursively. Cooley and Tukey rediscovered this in 1965; Gauss knew it in 1805. The same butterfly pattern appears in fast Walsh-Hadamard transforms, in JPEG's DCT, and in number-theoretic transforms for cryptographic multiplication.



FFT butterfly — the radix-2 Cooley-Tukey diagram. O(N log N) instead of O(N^2); the most important algorithm in DSP.

DFT definition

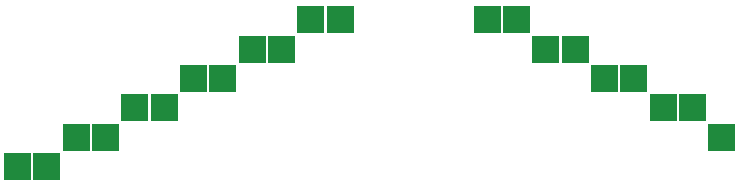
The DFT of an N-point sequence x[n] is $X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi kn/N}$, for k = 0, ..., N−1. The inverse is $x[n] = (1/N) \cdot \sum X[k] \cdot e^{j2\pi kn/N}$. The DFT treats x[n] as one period of a periodic signal. Bin k corresponds to frequency $f_k = k \cdot f_s/N$. The DFT is sampled in frequency — between bins, energy leaks into adjacent bins (spectral leakage). Windowing (Hann, Hamming, Blackman) reduces leakage at the cost of widening the main lobe.

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi kn/N} \quad | \quad f_k = k \cdot f_s / N$$



VOLT INSIGHT

The FFT is the one algorithm most responsible for the digital age. Without it, MP3 audio would be impossible. JPEG images would be huge. MRI scans would take minutes instead of seconds. 5G would not exist. Cooley and Tukey rediscovered it in 1965. Gauss knew it in 1805. It's mergesort applied to complex exponentials.



A pure sinusoid in time a single peak in the FFT. The eigenfunction of every LTI system.



A square wave decomposes into odd harmonics (Fourier series): f, 3f, 5f, 7f, ... at amplitudes 1, 1/3, 1/5, ...

FFT — the algorithm

The radix-2 Cooley-Tukey FFT requires N to be a power of 2. It splits the DFT into even and odd samples: X[k] = E[k] + W_N^k·O[k], where E is the N/2-point DFT of even samples, O is the N/2-point DFT of odd samples, and $W_N = e^{-j2 /N}$ is the twiddle factor. Recurse down to 1-point DFTs (trivial). The recursion has log (N) levels, each doing O(N) work — total O(N log N).

HANDS-ON PROJECT — Python: Real-Time Audio Spectrum Analyzer

Goal: build a real-time spectrum analyzer on your laptop.

Tasks: (1) capture audio from microphone using sounddevice. (2) window each 1024-sample block with Hann. (3) FFT. (4) plot magnitude in dB vs frequency. (5) repeat 30 times per second.

Verify: whistle, clap, play a tone — confirm peaks appear at the right frequencies. Confirm a square wave has odd harmonics.

Extension: add a waterfall display (time on x, frequency on y, magnitude as color).

CHECKPOINT — Can you answer these?

- What is the difference between DTFT, DFT, and FFT?
- What is spectral leakage? How do you reduce it?
- Compute the DFT of x = [1, 1, 0, 0]. (Hint: it is real.)
- Why does FFT require N to be a power of 2? What if it isn't?
- How many complex multiplications does an N-point FFT do? Compare to direct DFT.

Module 5.8 — FIR and IIR Filter Design

Duration: 8–10 h · **Difficulty:** Advanced

FIR design — the windowed sinc method

An ideal low-pass filter has a brick-wall frequency response, which corresponds to an infinite sinc in time. Truncate the sinc to N taps and window it (Hann, Hamming, Blackman) to reduce sidelobes. The resulting filter has linear phase (symmetric impulse response), which is critical for audio and data applications. Filter length N sets the transition bandwidth: f 4·f_s/N for a Hann window.

IIR design — bilinear transform of analog prototypes

Design an analog filter (Butterworth, Chebyshev, elliptic) using classic tables or scipy.signal.butter. Transform to digital via the bilinear transform: s $\approx$ (2/T)·(1 − z^{-1})/(1 + z^{-1}). This preserves the magnitude response but warps the frequency axis — pre-warp the cutoff before designing. IIR filters are much shorter than FIR for the same selectivity, but they have nonlinear phase and can be unstable.

Choosing FIR vs IIR

Property	FIR	IIR
Stability	Always	Depends on poles
Phase	Exactly linear (if symmetric)	Generally nonlinear
Order for same selectivity	High (100+ taps)	Low (5–10 coefficients)
Computational cost	Higher	Lower
Design ease	Easy (windowed sinc, Parks-McClellan)	Easy (analog + bilinear)
Feedback	No	Yes — can oscillate if quantized
Typical use	Audio, data, image	Control, comms, real-time

HANDS-ON PROJECT — Python + ESP32: Real-Time Audio EQ

Goal: build a 3-band (bass, mid, treble) audio equalizer.

Frontend: ESP32 with I2S MEMS mic (INMP441) sampling at 16 kHz. Output via I2S DAC (MAX98357A) to a speaker.

Filter: three parallel IIR filters (low-pass at 200 Hz, band-pass at 1 kHz, high-pass at 5 kHz), each with a gain control. Sum the outputs.

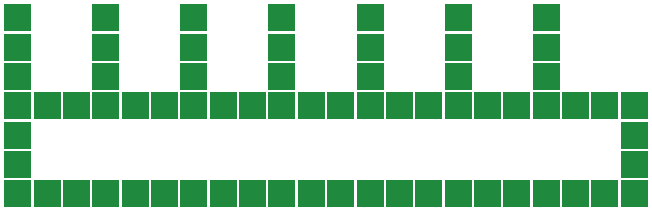
Implementation: ESP32 with ESP-IDF. Use I2S DMA. Apply the filters sample-by-sample in the I2S callback. ~0.5 s per sample per filter.

Verify: feed a sweep from 20 Hz to 8 kHz. Adjust the bass knob — the low-frequency output should change. Same for mid and treble.

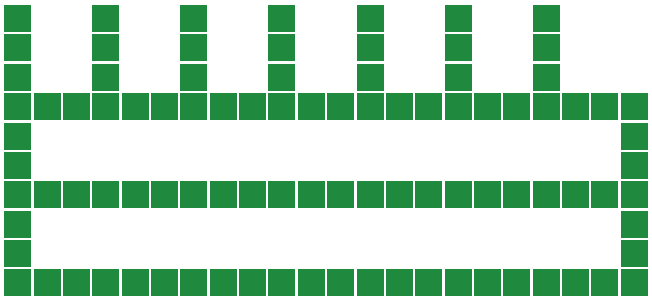
Extension: add a small OLED display showing the FFT spectrum in real time.

CHECKPOINT — Can you answer these?

- Design a 64-tap FIR low-pass at 2 kHz with f_s = 16 kHz using a Hann window.
- Convert a 2nd-order Butterworth analog low-pass at 1 rad/s to digital at f_s = 1 kHz via bilinear.
- Why is linear phase important for audio? Why can we tolerate nonlinear phase in control?
- Why do IIR filters risk instability when implemented with fixed-point arithmetic?



FIR filter — tapped delay line, weighted sum. Always stable; exactly linear phase if symmetric.



IIR filter — like FIR but with feedback. Fewer coefficients for same selectivity, but check pole stability.

Module 5.9 — Adaptive Filters (Intro)

Duration: 5–7 h · **Difficulty:** Advanced

An adaptive filter updates its coefficients in real time to minimize an error signal. The classic algorithm is LMS (Least Mean Squares): w[n+1] = w[n] + ·e[n]·x[n], where w is the coefficient vector, is the step size, e[n] = d[n] − w^T·x[n] is the error, d is the desired signal, and x is the input. Applications: noise cancellation (headphones), echo cancellation (speakerphones), channel equalization (modems), system identification.

$$w[n+1] = w[n] + \quad \cdot e[n] \cdot x[n] \quad | \quad e[n] = d[n] - w^T \cdot x[n]$$

CS BRIDGE

LMS stochastic gradient descent. LMS is literally SGD on the mean-square error cost J = E[|e|²]. The gradient is −2·e[n]·x[n]; the update w += ·e·x is SGD with learning rate . The same algorithm powers every neural network you have ever trained. The convergence condition (small enough, input whitened) is the same as for SGD.

HANDS-ON PROJECT — Python: Noise Cancellation Demo

Goal: cancel a tonal noise from a signal.

Setup: signal = 1 kHz tone + white noise. Reference input = the 1 kHz tone (think: a microphone captures ambient noise separately).

LMS: 32-tap FIR filter, = 0.01. Filter the reference; subtract from signal. The 1 kHz tone should cancel; the white noise remains.

Verify: plot the spectrum before and after. The 1 kHz peak should drop by 40+ dB.

CHECKPOINT — Can you answer these?

- What is an adaptive filter? Name three applications.
- Write the LMS update rule. Why does it work?
- What is the convergence condition on ?
- How is LMS related to SGD?

Module 5.10 — Phase 5 Capstone: Software-Defined Audio

Duration: 12–16 h · **Difficulty:** Advanced

HANDS-ON PROJECT — Phase 5 Capstone: Real-Time Audio Spectrum Analyzer + EQ

Goal: combine FFT analysis with real-time filtering on the ESP32.

Architecture: I2S mic 50% overlap buffer 256-point Hann window FFT OLED display magnitude in dB. Simultaneously, 5-band IIR EQ filters the audio; output via I2S DAC.

UI: rotary encoder adjusts the selected band; button cycles bands. OLED shows: spectrum bar graph (top half), EQ band gains (bottom half).

Performance: FFT every 16 ms (60 Hz display refresh). EQ filters run sample-by-sample. CPU load should be under 50% on ESP32 dual-core.

Document: schematic, PCB if you make one, source code, measurement plots (frequency response of each EQ band), video demo. Save everything.

CHECKPOINT — Can you answer these?

- Sketch the signal flow from microphone to speaker.
- How do you ensure the FFT does not underrun the I2S output?
- What is the latency through the system? How would you reduce it?
- Why 50% overlap on the FFT windows?



VOLT NARRATES

Phase 5 is done. You can break any signal into its spectrum. You can design filters. You can build a real-time audio analyzer. You can ship DSP code on an ESP32. Phase 6 takes these frequency-domain tools and applies them to the hardest problem in engineering. Making a physical system behave the way you want. Despite noise. Despite disturbance. Using feedback.



Phase 6 — Control Systems & Robotics

Duration: 6 weeks · **Modules:** 10 · **Goal:** master classical and modern control — root locus, Bode/Nyquist, PID, state-space — and apply it to a closed-loop motor control and a line-following robot.

Control theory is the engineering of feedback. A plant (the thing being controlled) has dynamics; a sensor measures the output; a controller computes the input that drives the output to a target. The math is Laplace transforms (Phase 5), stability via pole locations, and frequency-response tools (Bode, Nyquist). Robotics adds kinematics — the geometric relationship between joint angles and end-effector position. As a CS engineer, you have already seen feedback in error-correcting codes, in TCP congestion control, and in iterative optimization; control theory is the same idea applied to physical systems.



VOLT NARRATES

Feedback is everywhere. TCP congestion control is a feedback loop. Hash table resizing is a feedback loop. Iterative compilers are feedback loops. Control theory gives the math a proper name. It lets you prove a loop will be stable. Not just hope it converges.



AMP ASKS

So control theory is the formal version of what I do intuitively in distributed systems?



VOLT SAYS

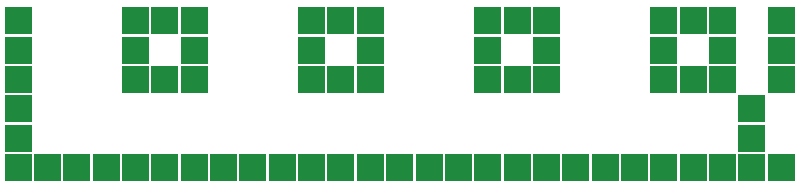
Yes. Same loops. With proofs.

Module 6.1 — Feedback, Modeling, Transfer Functions

Duration: 5–7 h · **Difficulty:** Intermediate

A feedback system has three blocks: plant P(s), sensor H(s), controller C(s). The closed-loop transfer function is T(s) = C·P / (1 + C·P·H). The denominator 1 + C·P·H is the **characteristic equation**; its roots are the closed-loop poles. Stability requires all closed-loop poles to be in the left half plane.

$$T(s) = C(s)\cdot P(s) / (1 + C(s)\cdot P(s)\cdot H(s)) \quad | \quad L(s) = C(s)\cdot P(s)\cdot H(s) \text{ (loop gain)}$$



Negative feedback loop — setpoint error controller plant output. Sensor closes the loop. Stability depends on the loop gain L(s).

Modeling physical systems

Every physical system has a model. Mechanical: mass-damper-spring is m· + b· + k·x = F. Electrical: RLC is L·q¨+ R·q + (1/C)·q = V. Thermal: C·d /dt = (_env −)/R. Fluid: tank level dynamics. They are all second-order linear ODEs — the same math. The art of modeling is choosing what to include and what to ignore. A model that is too simple will not capture the behavior you care about; a model that is too complex is unusable for design.

CHECKPOINT — Can you answer these?

- Sketch a standard negative-feedback block diagram. Label plant, sensor, controller.
- Derive T(s) = CP / (1 + CPH). What is the characteristic equation?
- Write the ODE for a mass-spring-damper. What is the transfer function X(s)/F(s)?
- Why is modeling called an ‘art’?

Module 6.2 — Stability: Routh-Hurwitz and Pole Locations

Duration: 5–7 h · **Difficulty:** Intermediate

A linear system is stable if and only if all poles of its transfer function are in the open left half plane (real part strictly negative). For low-order systems, you can find the poles directly. For higher-order systems, the Routh-Hurwitz criterion tells you how many poles are in the right half plane without solving for them — you build a Routh array from the polynomial coefficients and count sign changes in the first column.

Routh-Hurwitz procedure

For a polynomial a_n·s^{n} + a_{n−1}·s$^{n−1}$ + … + a_0, build the Routh array: first two rows are the coefficients in alternation. Each subsequent row is computed from the two rows above. The number of sign changes in the first column equals the number of right-half-plane poles. Zero sign changes means stable.

HANDS-ON PROJECT — Python: Verify Routh-Hurwitz Numerically

Goal: practice Routh-Hurwitz on 5 characteristic equations.

Tasks: for each polynomial, build the Routh array by hand and predict the number of RHP poles. Then verify with numpy.roots.

Examples: s^3+3s^2+3s+1 (stable), s^3+s^2+s+3 (unstable, 2 RHP), $s^5+2s^4+3s^3+4s^2+5s+5$ (unstable), etc.

Pass: all 5 hand-computed answers match numpy.

CHECKPOINT — Can you answer these?

- State the Routh-Hurwitz stability criterion.
- Build the Routh array for $s^3 + 4s^2 + 5s + 2$. How many RHP poles?
- Why is a single pole at s=0 considered marginally stable, not stable?
- Why is checking pole locations sufficient for linear stability?

Module 6.3 — Time-Domain Response and Steady-State Error

Duration: 5–7 h · **Difficulty:** Intermediate

The step response of a second-order system T(s) = _n²/(s² + 2 _n·s + _n²) has four key metrics: rise time t_r (10% to 90%), peak time t_p = / _d, percent overshoot %OS = e^(− / (1− ²))·100%, settling time t_s (2% criterion) 4/(_n). These four numbers describe the transient; they are what you tune the controller to meet.

$$t_r \quad 1.8 / \ _n \quad | \quad \%OS = e^{- \ / \ (1- \ ^2)} \cdot 100\% \quad | \quad t_s \quad 4 / (\ \cdot \ _n)$$

Steady-state error and system type

The steady-state error to a step input is e_ss = 1/(1 + K_p), where K_p is the position error constant. For a ramp input, e_ss = 1/K_v (velocity error constant). For a parabola, e_ss = 1/K_a. A system is **type N** if it has N poles at the origin (integrators in the loop). Type 0: finite step error, infinite ramp error. Type 1: zero step error, finite ramp error. Type 2: zero step and ramp error, finite parabola error. Adding an integrator increases system type — and reduces stability margin. The trade-off is fundamental.

HANDS-ON PROJECT — Python: Step Response Comparison

Goal: visualize how and _n affect step response.

Tasks: plot step responses for = 0.1, 0.3, 0.5, 0.7, 1.0, 1.5 at _n = 1. Mark rise time, overshoot, settling time on each.

Verify: confirm the formulas t_r 1.8/ _n, %OS = e^(− / (1− ²))·100, t_s 4/(_n) match the simulations.

Extension: design a 2nd-order system to meet spec: t_s < 1 s, %OS < 10%. Find and _n. (Answer: > 0.6, _n > 6.7 rad/s.)

CHECKPOINT — Can you answer these?

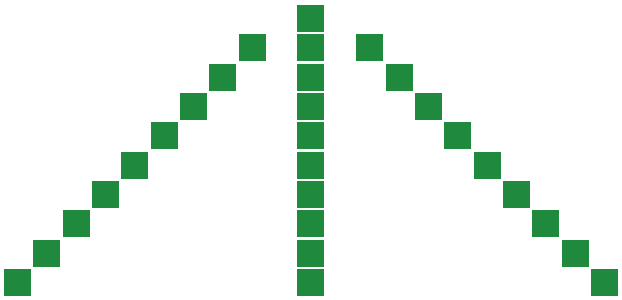
- Define rise time, peak time, overshoot, settling time.
- Compute %OS for = 0.5. Compute t_s for _n = 2.
- Define system type. What is the steady-state error of a Type 1 system to a step? To a ramp?
- Why does adding an integrator increase system type but reduce stability?

Module 6.4 — Root Locus Design

Duration: 7–9 h · **Difficulty:** Intermediate

Root locus is a graphical method for seeing how the closed-loop poles move as a single parameter (typically a gain K) varies from 0 to . The locus starts at the open-loop poles (K=0) and ends at the open-loop zeros (K=). The rules for sketching by hand: real-axis segments to the left of an odd number of OL poles/zeros; asymptotes at angles (2k+1) /(n−m) centered at the centroid (p_i − z_i)/(n−m); breakaway points where dK/ds = 0.

$$1 + K \cdot G(s)\cdot H(s) = 0 \quad | \quad \text{asymptote angle} = (2k+1) \ / (n_p - n_z)$$



Root locus — closed-loop poles trace curves in the s-plane as K varies from 0 to . Cross into the right-half-plane and the system goes unstable.

HANDS-ON PROJECT — Python: Root Locus Plotter

Goal: use Python control library to plot root locus.

Tasks: use control.root_locus on a few example plants: (a) 1/(s(s+1)(s+2)) — third-order, find K for stable operation. (b) 1/(s²(s+5)) — what compensation is needed for stability? (c) (s+1)/(s(s+2)(s+3)) — with zero, find K for = 0.7.

Tool: python-control library (slycot optional).

Verify: overlay the unit-step response at the chosen K; confirm transient meets spec.

CHECKPOINT — Can you answer these?

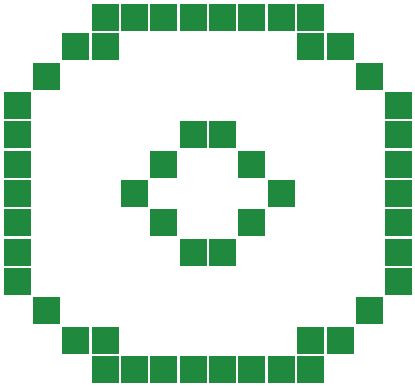
- State the rules for sketching a root locus by hand.
- Where does the root locus start? Where does it end?
- Why do we add zeros (lead compensation)? Poles (lag)?
- Sketch the root locus for G(s) = K/(s(s+1)(s+2)). For what K is the system marginally stable?

Module 6.5 — Frequency Response: Bode, Nyquist, Margins

Duration: 7–9 h · **Difficulty:** Intermediate

Frequency-response methods analyze the open-loop transfer L(j) instead of the closed-loop poles. Two complementary plots: **Bode** (separate magnitude and phase vs frequency) and **Nyquist** (polar plot of L(j)). The **gain margin** GM is the factor by which you can multiply L before the system goes unstable; the **phase margin** PM is the additional phase lag you can add at the gain-crossover frequency before instability. PM is the more useful number — 30–60° is typical for well-designed loops.

GM = 1 / |L(j $_{pc}$)| | PM = 180° + L(j $_{gc}$) ($_{pc}$: phase crossover, $_{gc}$: gain crossover)



Nyquist plot — polar plot of L(j). Encirclements of −1 tell you how many unstable closed-loop poles you have.

Rules of thumb: PM 30° ~20% overshoot, PM 60° ~8% overshoot, PM 90° no overshoot. The bandwidth (frequency where |T| drops by 3 dB) approximates the rise time: t_r 1.8/ _bw.

HANDS-ON PROJECT — Python: Bode and Nyquist Plots

Goal: compute and plot Bode and Nyquist for an arbitrary L(s).

Tasks: for L(s) = K/(s(s+1)(s+5)): plot Bode for K = 1, 5, 20, 50. Find GM and PM at each. Verify with the python-control margin() function.

Verify: predict closed-loop stability from Nyquist (encirclements of −1). Match to Routh-Hurwitz.

Design: find K that gives PM = 45°. Confirm by simulating step response.

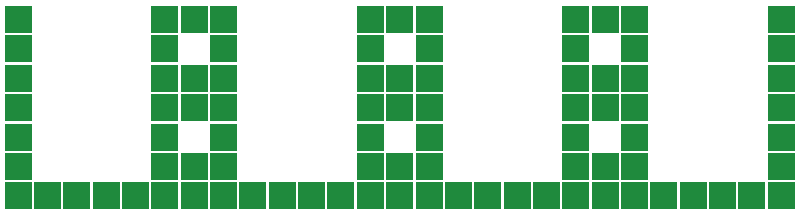
CHECKPOINT — Can you answer these?

- Define gain margin and phase margin. Which is more useful in practice?
- What PM gives ~10% overshoot? ~no overshoot?
- State the Nyquist stability criterion. How do you count encirclements?
- How is bandwidth related to rise time?

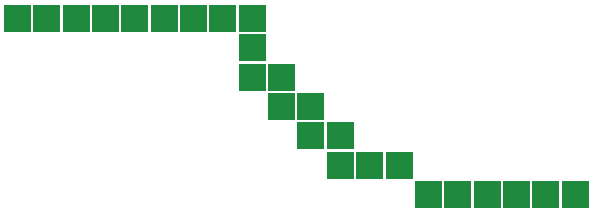
Module 6.6 — PID Design and Tuning

Duration: 7–9 h · **Difficulty:** Intermediate

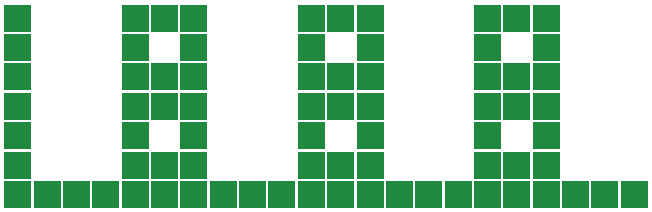
The PID controller C(s) = K_p·(1 + 1/(T_i·s) + T_d·s) is the workhorse of industrial control. P (proportional) gives fast response but steady-state error. I (integral) removes steady-state error but slows response and reduces stability. D (derivative) damps oscillation but amplifies noise. The art is tuning the three gains.



PID feedback loop — setpoint error PID plant output. Sensor feeds back to the summing junction.



Open-loop Bode magnitude — used to read gain margin (GM) and phase margin (PM) for stability.



PID controller block — three parallel paths (P, I, D) summed. The most-deployed controller in industrial automation.

Ziegler-Nichols tuning

Two classic methods. **Closed-loop**: increase K_p until the system oscillates sustainably (at gain K_u, period T_u). Set K_p = 0.6·K_u, T_i = 0.5·T_u, T_d = 0.125·T_u (the 'PID' ZN setting). **Open-loop**: step the input, measure the process reaction curve (delay L and slope R), set K_p = 1.2/(R·L), T_i = 2·L, T_d = 0.5·L. ZN gives aggressive tuning; detune by 50% for production. Tyreus-Luyben and Cohen-Coon are refinements.

HANDS-ON PROJECT — Bench: PID Temperature Control with Arduino

Goal: control the temperature of a small heater (power resistor) using PID.

Hardware: Arduino Uno, 10 5 W power resistor, DS18B20 temp sensor, MOSFET to switch the resistor, 12 V supply.

Implementation: read temp every 1 s. Compute PID. Output PWM to MOSFET. Use Arduino PID library.

Tune: use ZN closed-loop method. Find K_u and T_u. Apply PID settings.

Verify: step setpoint from 30°C to 50°C. Measure overshoot, settling time. Compare to a simulated PID in Python with the same plant model.

Detune: if overshoot > 10%, reduce K_p by 30% and increase T_i by 50%.

CHECKPOINT — Can you answer these?

- What does each of P, I, D do? What are the trade-offs?
- Apply ZN closed-loop tuning for K_u = 5, T_u = 0.4 s. What are K_p, T_i, T_d?
- Why is derivative often filtered? What is the time constant?
- What is integral windup? How do you prevent it?

Module 6.7 — State-Space: Controllability and Observability

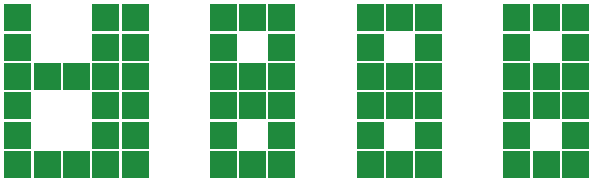
Duration: 7–9 h · **Difficulty:** Advanced

CS BRIDGE

State-space linear dynamical system in ML. The state-space model = A·x + B·u, y = C·x + D is exactly the form of a linear RNN. The matrices A, B, C, D are the weights. Controllability (can you reach any state from any other?) is the reachability of the underlying graph. Observability (can you infer the state from outputs?) is dual. The LQR controller and the Kalman filter are dual — LQR is optimal control, Kalman is optimal estimation, and they share the same Riccati equation.

State-space is the modern (post-1960) way to handle MIMO (multi-input multi-output) systems. Instead of a single transfer function, you have a vector state x, vector input u, vector output y. The dynamics are first-order matrix ODEs. Stability: A is stable iff all eigenvalues have negative real part. Controllability: the controllability matrix [B, AB, A²B, ..., A^(n−1)B] has full rank. Observability: the observability matrix [C; CA; CA²; ...; CA^(n−1)] has full rank.

$$= A\cdot x + B\cdot u \quad | \quad y = C\cdot x + D \quad | \quad \text{Controllable iff } \operatorname{rank}([B, AB, \ldots, A^{n-1}B]) = n$$



State-space block diagram — A matrix is the dynamics, B is the input coupling, C is the output mapping. Eigenvalues of A are the system poles.

LQR — Linear Quadratic Regulator

LQR finds the optimal state-feedback u = −K·x that minimizes the cost J = ∫(x^T Q·x + u^T R·u)dt. The gain K is found by solving the algebraic Riccati equation. LQR has guaranteed stability margins (∞ 60° phase margin, ∞ 6 dB gain margin) and is widely used in aerospace, robotics, and process control. Choose Q to penalize state errors, R to penalize control effort.

HANDS-ON PROJECT — Python: Inverted Pendulum on a Cart

Goal: stabilize an inverted pendulum using LQR.

Model: cart with position x and pendulum angle θ. State: [x, ẋ, θ, θ̇]. Linearize around θ=0. A and B from standard inverted-pendulum equations.

Design: Q = diag([1, 0, 10, 0]), R = 1. Solve DARE with scipy.linalg.solve_discrete_are.

Simulate: start with θ=0.1 rad. Confirm LQR stabilizes in < 2 s. Confirm cart moves to keep pendulum upright.

Visualize: animate the cart-pendulum system with matplotlib.

CHECKPOINT — Can you answer these?

- Define controllability and observability. How do you test each?
- What is the dual relationship between LQR and Kalman filter?
- Set up state-space for a mass-spring-damper. What is A? B? C?
- Why does LQR have guaranteed stability margins?

Module 6.8 — Digital Control: Sampled-Data Systems

Duration: 5–7 h · **Difficulty:** Advanced

Digital control implements the controller in software: sample the output, compute the control law, output via DAC, repeat every T seconds. The discrete-time equivalent of the plant is x[k+1] = A_d·x[k] + B_d·u[k], where A_d = e^(A·T) and B_d = ∫_0^T e^A ·B dτ. The zero-order hold (ZOH) on the DAC adds a (1 − e^(−sT))/s factor to the plant.

$$A_d = e^{A \cdot T} \quad | \quad B_d = \int_0^T e^{A} \cdot B \, d\tau$$

Sample rate matters: too slow and you lose information and stability margin; too fast and quantization noise dominates. Rule of thumb: sample at 10× the closed-loop bandwidth. Below 5× you start losing phase margin noticeably.

CHECKPOINT — Can you answer these?

- How do you discretize a continuous-time state-space model?
- What is a zero-order hold? What does it do to the frequency response?
- What sample rate would you choose for a 10 Hz bandwidth control loop?
- Why does reducing sample rate reduce phase margin?

Module 6.9 — Robotics Kinematics

Duration: 6–8 h · **Difficulty:** Advanced

CS BRIDGE

Forward kinematics function composition. Each joint applies a rotation and translation to the next link. Forward kinematics is composing these transforms: T_world_to_end = T_0·T_1·T_2·…·T_n. Inverse kinematics is inverting the composition — generally harder (nonlinear, multiple solutions, singularities). Denavit-Hartenberg (DH) convention is a standard way to parameterize joint transforms.

Forward kinematics with DH parameters

Assign a frame to each link following DH rules. Each joint contributes a homogeneous transform T_i = Rot_z(_i)·Trans_z(d_i)·Trans_x(a_i)·Rot_x(_i). Multiply them: T_0_n = T_1·T_2·…·T_n. The end-effector position and orientation are in the last column and rotation block of T_0_n.

Inverse kinematics — the hard problem

Given a target end-effector pose, find joint angles. Three approaches: (1) **analytic** — solve the equations explicitly (only possible for simple geometries, e.g., 2-link planar arms). (2) **Jacobian-based** — iterate = J^{+}· x where J is the Jacobian (partial derivatives of end-effector pose w.r.t. joint angles) and J^{+} is its pseudoinverse. (3) **Optimization** — minimize ||f( ) – target||² with gradient descent or Levenberg-Marquardt.

HANDS-ON PROJECT — Python: 2-DOF Planar Arm Inverse Kinematics

Goal: compute IK for a 2-link arm and animate it.

Setup: arm with link lengths L1 = L2 = 1. End-effector target (x, y).

Analytic IK: _2 = acos((x² + y² − L1² − L2²)/(2·L1·L2)), _1 = atan2(y, x) − atan2(L2·sin _2, L1 + L2·cos _2).

Animate: pick a path (circle, figure-8). Sample target points. Solve IK at each. Animate the arm following the path.

Extension: add a 3rd link (3-DOF). Now analytic IK has multiple solutions — elbow up and elbow down. Use Jacobian-based IK.

CHECKPOINT — Can you answer these?

- What is forward kinematics? Inverse kinematics?
- State the DH convention. What are the four parameters per joint?
- Solve analytic IK for a 2-link planar arm reaching (0.5, 0.5) with L1=L2=1.
- What is a Jacobian? How do you use it for IK?
- What is a kinematic singularity? Give an example.

Module 6.10 — Phase 6 Capstone: Line-Following Robot

Duration: 12–16 h · **Difficulty:** Advanced

HANDS-ON PROJECT — Phase 6 Capstone: Closed-Loop Line-Following Robot

Goal: build a robot that follows a black line on a white floor at 1 m/s.

Mechanical: differential-drive chassis with two DC motors and a caster. Wheel diameter 6 cm, wheelbase 15 cm.

Sensors: 5-element IR reflective sensor array (e.g., Pololu QTRX-MD-05A).

Compute: ESP32 reads sensors, computes control, drives motors via L298N.

Controller: PID on the line position error. K_p, K_i, K_d tuned on the bench. Sensor reading line position estimate (weighted average). Error PID differential motor command.

Kinematics: forward: v_L, v_R (v,). Differential: v = K·e. Use the differential-drive kinematic equations.

Test: start on a straight track. Tune K_p first (proportional only). Add K_d when oscillating. Add K_i if there is steady-state offset. Then test on curves of decreasing radius.

Document: schematic, code, sensor calibration, controller tuning log, video. Time on a standard track. Compare to the simulated kinematics.

CHECKPOINT — Can you answer these?

- Sketch the block diagram of the line follower. Where is the plant, the sensor, the controller?
- How does differential drive work? What is the relationship between v_L, v_R, v, ?
- How do you estimate line position from 5 binary sensor readings?
- Tune K_p, K_i, K_d in what order? What symptom does each address?

"Stability is not a property of the plant. It is a property of the loop. A well-designed controller stabilizes an unstable plant; a poorly-designed one destabilizes a stable plant."

— Volt, on feedback

"LQR and the Kalman filter are dual. Optimal control and optimal estimation are the same Riccati equation read forward and backward in time. That symmetry is one of the most beautiful results in engineering mathematics."

— Volt, on duality



VOLT NARRATES

Phase 6 is done. You can design a controller in the time domain with root locus. In the frequency domain with Bode and Nyquist. In the state domain with LQR. You can tune a PID. Model a physical system. Stabilize an inverted pendulum. The line-following robot capstone is the first time we ask you to ship something that moves. Phase 7 takes you into power electronics. Where the voltages and currents get large enough to burn.



Phase 7 — Power Electronics & Machines

Duration: 7 weeks · **Modules:** 10 · **Goal:** master power converters (buck/boost/inverter), electric machines (DC/induction/BLDC), and motor drives — and build a working FOC BLDC driver.

Power electronics is where the rubber meets the road for energy. Every modern energy system — from a 5 W USB charger to a 1 MW grid-scale inverter — is built from the same set of switching converter topologies. Electric machines convert between electrical and mechanical energy; motors consume 50% of the world's electricity, so even small efficiency improvements matter at scale. This phase is heavy on LTspice simulation and bench work — the voltages and currents here will kill components (and occasionally burn fingers) if you get it wrong.



VOLT WARNS

Phase 7 is where this book gets physical. The voltages and currents here can kill components. Burn fingers. Start fires. Pay attention to ratings. Derate by 30 percent. Use fuses. Never probe a live high-voltage circuit without thinking first. The math is the same as Phase 6. Feedback loops. Transfer functions. Bode plots. But the stakes are higher.



AMP ASKS

How high are we talking? Like, wall-outlet high?



VOLT WARNS

Sometimes higher. Buck converters step down 400V to 12V. That kind of high.

Module 7.1 — Power Semiconductor Devices

Duration: 5–7 h · **Difficulty:** Intermediate

Device	Symbol	On-state	Off-state	Speed	Typical use
Diode	—	V_F ~ 0.7 V (Si), 0.3 V (Schottky)	Blocks V_RRM	Fast	Rectifiers, freewheel
MOSFET	—	R_DS(on) m	Blocks V_DS	Very fast (< 100 ns)	DC-DC up to ~200 V
IGBT	—	V_CE(sat) ~ 1.5 V	Blocks V_CES	Medium (s)	Motor drives 600 V – 6.5 kV
SCR (thyristor)	—	V_T ~ 1.5 V	Blocks V_RRM until forward triggered	Slow (ms)	Phase control, HVDC

Device	Symbol	On-state	Off-state	Speed	Typical use
GaN FET	—	R_DS(on) m , no Q_rr	Blocks V_DS	Ultra fast (< 10 ns)	High-freq DC-DC
SiC FET	—	R_DS(on) m	Blocks V_DS to 1700 V+	Very fast	EVs, grid inverters

Why GaN and SiC are eating silicon

Wide-bandgap semiconductors (GaN, SiC) have higher breakdown field, higher thermal conductivity, and faster switching than silicon. This means smaller devices, smaller passives (because you can switch at MHz instead of kHz), and higher efficiency. Modern USB-C chargers switch at 100+ kHz in 5 cm³ because of GaN. EV inverters use SiC at 800 V to handle fast charging. The trade-off is cost — GaN and SiC are still 2–5× silicon per amp — but the system-level savings (smaller magnetics, smaller heatsinks) often win.

Switching loss and gate drive

Every switching event dissipates energy: P_sw = ½·V·I·(t_r + t_f)·f_sw. At 100 kHz and 50 V/10 A, even 50 ns transitions dissipate 1.25 W per switch — added to conduction loss I²·R_DS(on). Gate drivers provide the high peak current (1–8 A) needed to charge and discharge the gate capacitance quickly. Bootstrap gate drivers use a capacitor to float the high-side driver above the switch node — elegant but limits duty cycle and frequency.

CHECKPOINT — Can you answer these?

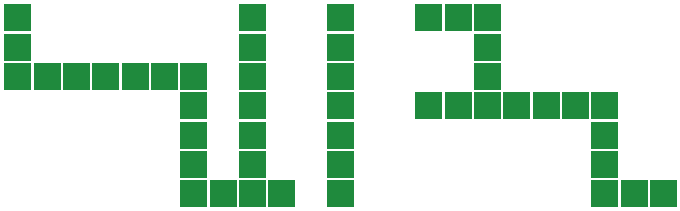
- Why are Schottky diodes preferred for low-voltage rectification?
- When do you choose IGBT over MOSFET? When SiC over silicon?
- Compute the switching loss for 50 V, 10 A, 100 kHz, 50 ns transitions.
- Why does a high-side N-channel MOSFET need a bootstrap gate driver?

Module 7.2 — Non-Isolated DC-DC Converters

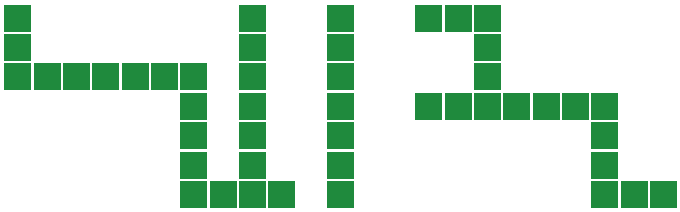
Duration: 8–10 h · **Difficulty:** Intermediate

Three topologies dominate non-isolated DC-DC: **buck** (step-down), **boost** (step-up), **buck-boost** (inverting, can step up or down). All three operate on the same principle: a switch chops the input into pulses, an inductor stores energy during on-time and releases it during off-time, a capacitor smooths the output. The duty cycle D sets the conversion ratio.

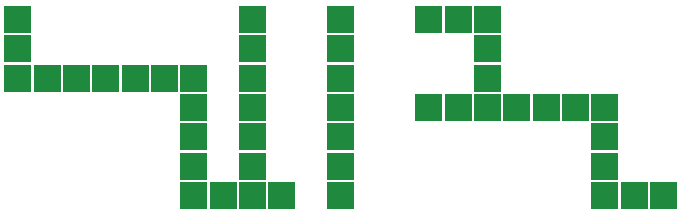
$$\text{Buck: } V_o = D \cdot V_{in} \quad | \quad \text{Boost: } V_o = V_{in} / (1 - D) \quad | \quad \text{Buck-boost: } V_o = -D/(1-D) \cdot V_{in}$$



Buck converter — switch + diode + inductor + capacitor. V_o = D · V_in (D = duty cycle).



Boost converter — V_o = V_in / (1−D). The output is higher than the input. Used in PFC stages, LED drivers, EV chargers.



Buck-boost converter — V_o = −D/(1−D) · V_in. Inverts polarity; can step up or down. Used in LED drivers and battery regulators.

Buck converter — the canonical design

Switch Q (MOSFET), diode D (or synchronous FET), inductor L, cap C, load R. Switch closes: inductor current ramps up at di/dt = (V_in − V_o)/L. Switch opens: inductor current ramps down at di/dt = −V_o/L (current freewheels through D). At equilibrium, the volt-second balance gives V_o = D·V_in. Inductor ripple: $I_L = (V_{in} - V_o)\cdot D/(L\cdot f_{sw})$. Output ripple: $V_o = I_L/(8\cdot C\cdot f_{sw})$.

Continuous vs discontinuous conduction

If the inductor current never reaches zero, the converter is in continuous conduction mode (CCM) — V_o/V_in is independent of load. If the current reaches zero before the next switching cycle, the converter is in discontinuous conduction mode (DCM) — V_o/V_in depends on load. DCM has lower switching loss but higher ripple. Most designs stay in CCM at full load and may drop into DCM at light load — your controller must handle both.

Synchronous rectification

Replace the diode with a second MOSFET. The diode drop (0.5 V Schottky) at 10 A dissipates 5 W; a 10 m FET dissipates 1 W. Worth it for any converter above a few amps. The trade-off is more complex gate drive (you must prevent cross-conduction) and a small efficiency penalty at very light load (where the diode would actually be more efficient because the FET gate charge is wasted).

HANDS-ON PROJECT — LTspice: 12V-to-5V Buck at 2 A, 200 kHz

Goal: design and simulate a buck converter.

Specs: V_in = 12 V, V_o = 5 V, I_o = 2 A, f_sw = 200 kHz, V_o < 50 mV.

Design: D = 5/12 = 0.417. L = (V_in − V_o)·D/(I_L · f_sw) = (7·0.417)/(0.4·200k) = 36 H. C = I_L/(8· V_o·f_sw) = 0.4/(8·0.05·200k) = 50 F.

Simulate: LTspice with pulse voltage source as gate driver, IRFZ44N MOSFET, 1N5822 Schottky, ideal L and C, 2.5 load.

Measure: verify V_o = 5 V within 1%, ripple < 50 mV. Sweep load from 0.5 A to 3 A; verify regulation. Sweep V_in from 10 V to 15 V; verify line regulation.

Bench: build with MC34063 or LM2596 module. Compare measurement to simulation.

CHECKPOINT — Can you answer these?

- Derive V_o = D·V_in for a buck converter from volt-second balance.
- Design a 24 V to 12 V buck at 5 A, 500 kHz, with 1% ripple. Choose L and C.
- When does a buck enter DCM? What changes in the conversion ratio?
- Why is synchronous rectification more efficient at high current?

Module 7.3 — Isolated Converters: Flyback, Forward, Bridge

Duration: 6–8 h · **Difficulty:** Intermediate

Galvanic isolation is required for safety in any converter that touches AC mains or that has a human-touchable output. The transformer provides isolation and (simultaneously) voltage step-up/down via the turns ratio. The simplest isolated topology is the **flyback**, where the transformer is actually used as a coupled inductor — energy is stored in the core during switch on-time and released to the output during switch off-time. Flyback is cheap (one switch, one diode, one cap on the secondary) but limited to ~100 W due to core loss and high ripple.

Three isolated topologies

Topology	Power	Complexity	Use case
Flyback	< 100 W	Low	Phone chargers, small appliances, multi-output
Forward (1-switch)	< 200 W	Medium	Industrial aux supplies, ATX standby
Push-pull / half-bridge	100 W – 500 W	Medium	Telecom DC-DC, ATX main
Full-bridge	500 W – 5 kW+	High	Server, EV, industrial

Topology	Power	Complexity	Use case
Resonant LLC	500 W – 5 kW+	High	Server, EV onboard chargers

Transformer design — the hard part

Isolated converter design is dominated by transformer design. You must choose core material (ferrite for < 1 MHz), core size (set by power and frequency), turns ratio (set by I/O voltage), wire gauge (set by RMS current, with skin effect at high frequency — use Litz wire above 100 kHz), and winding structure (affects leakage inductance and inter-winding capacitance). The classic reference is Colonel Wm. T. McLyman's Transformer and Inductor Design Handbook. For first designs, use a magnetics vendor's online tool (e.g., Wurth, Coilcraft, Magnetics Inc.).

HANDS-ON PROJECT — LTspice: 5V/2A Flyback from 170 V DC

Goal: simulate a flyback converter.

Specs: V_in = 170 V DC (rectified 120 VAC), V_o = 5 V, I_o = 2 A, f_sw = 100 kHz.

Design: N_p/N_s = 20 (turns ratio). L_p = 1 mH. D = V_o·N_p / (V_in·N_s + V_o·N_p) 0.37. Peak primary current I_p_pk = 2·I_o·N_s/(N_p·(1−D)) 0.6 A.

Simulate: use a coupled inductor model for the transformer. Verify V_o, I_p waveform (triangular), D. Measure the leakage inductance spike on the drain — that's what a snubber must clamp.

Snubber: add an RCD snubber across the primary. Tune R and C to keep V_drain below 80% of MOSFET rating.

CHECKPOINT — Can you answer these?

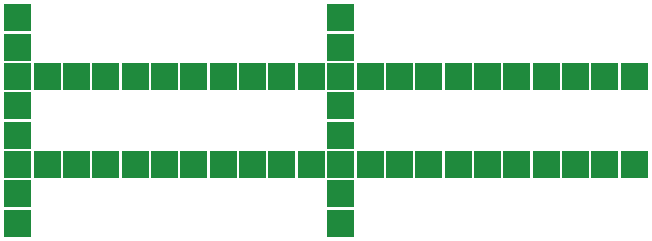
- Why is the flyback transformer actually a coupled inductor?
- What limits flyback to ~100 W?
- Why do high-frequency transformers need Litz wire?
- What is leakage inductance, and why must you snubber it?

Module 7.4 — Inverters: Square, PWM, SPWM, SVPWM

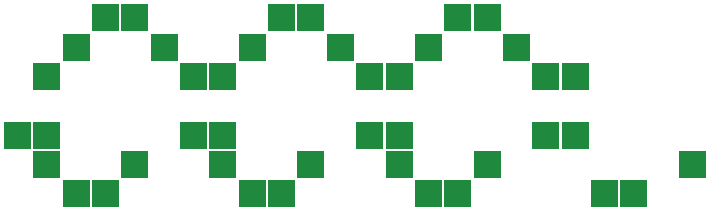
Duration: 7–9 h · **Difficulty:** Advanced

An inverter converts DC to AC. The single-phase full-bridge (H-bridge) has four switches: S1/S4 conduct for positive output, S2/S3 for negative. The output is a square wave if you just alternate. To get a sine-shaped output, modulate the duty cycle with a sine — **sinusoidal PWM (SPWM)**: compare a sine reference to a triangle carrier; the comparator output drives the switches. The output is a train of pulses whose average over a carrier period is a sine.

$$V_{o,avg} = m_a \cdot V_{dc} \cdot \sin(\ _m t) \quad (m_a = \text{modulation index}, m_a \ \ 1)$$



H-bridge — four switches arranged in an H. Drive S1/S4 for positive output, S2/S3 for negative. The heart of every DC-AC inverter.



SPWM — sine reference vs triangle carrier. Comparator output drives the H-bridge. Average output = scaled sine.

Space Vector PWM (SVPWM) — the three-level trick

For three-phase inverters, SVPWM is more efficient than SPWM. It treats the three-phase output as a single rotating voltage space vector. There are 8 switching states (6 active + 2 zero); any target vector is synthesized by time-averaging two adjacent active vectors and a zero vector. SVPWM gives ~15% more output voltage than SPWM for the same DC bus, which is why every commercial motor drive uses it.

HANDS-ON PROJECT — Python + LTspice: SPWM Inverter

Goal: generate SPWM, simulate an H-bridge inverter.

Python: generate 5 kHz triangle carrier and 50 Hz sine reference at m_a = 0.8. Output two complementary PWM signals (high/low side of each phase leg).

LTspice: build H-bridge with 4 MOSFETs, gate drivers, DC bus = 400 V. Drive with the PWM signals. Load with 10 + 10 mH.

Measure: output voltage before and after LC filter. Spectrum shows 50 Hz fundamental + harmonics at 5 kHz and sidebands.

Efficiency: compute efficiency at 1 kW output. Should be > 90%.

CHECKPOINT — Can you answer these?

- What is SPWM? How does m_a affect the output?
- Why is SVPWM more efficient than SPWM for three-phase inverters?
- What is dead time, and why is it needed in an H-bridge?
- Compute the THD of a square wave inverter output. (Hint: ~45%.)

Module 7.5 — Rectifiers and Power Factor Correction

Duration: 5–7 h · **Difficulty:** Intermediate

A simple diode-bridge + capacitor input draws current in short spikes at the peak of each AC cycle — power factor can be 0.5–0.7 even with a resistive load. Above 75 W, regulations (ENERGY STAR, IEC 61000-3-2) require active power factor correction (PFC). A PFC stage is a boost converter that forces the input current to follow the input voltage shape — making the load look resistive to the line.

Boost PFC

Standard topology: diode bridge boost converter DC bus. The boost controller has two control loops: a fast current loop that makes the inductor current follow |V_ac|·sin(t) scaled by an error signal, and a slow voltage loop that regulates the DC bus. The result: input current is sinusoidal, in phase with voltage, PF > 0.99. The PFC controller IC (e.g., NCP1654, UCC28180) implements both loops in analog.

CHECKPOINT — Can you answer these?

- Why do simple rectifier-capacitor inputs have poor power factor?
- How does a boost PFC work? What are its two control loops?
- Why is PFC mandatory above 75 W by regulation?
- Compute the input current of a 100 W PFC'd supply at 230 VAC. (Hint: 0.43 A RMS, sinusoidal.)

Module 7.6 — DC Machines

Duration: 4–6 h · **Difficulty:** Intermediate

DC motors are the simplest electric machine. The torque is T = K· ·I_a (field flux × armature current), the back-EMF is E = K· · . Combining: T = K²· ²·(V − E)/R_a. Speed control: vary V (armature voltage control — common), vary (field weakening — for high-speed operation), or vary R_a (rheostatic — wasteful). Modern DC motor drives use PWM on the armature voltage.

$$T = K\cdot \cdot I_a \quad | \quad E = K\cdot \cdot \quad | \quad = (V - I_a\cdot R_a) / (K\cdot)$$

Four DC motor types

Type	Field	Torque-speed	Use
Permanent magnet	PM	Linear, easy to drive	Toys, fans, EVs (small)
Shunt	Independent winding	Flat speed (self-regulating)	Industrial drives
Series	Series with armature	High starting torque	Traction (old trains, EVs)
Compound	Shunt + series	Compromise	Cranes, hoists

CHECKPOINT — Can you answer these?

- Why does a DC motor draw high current at startup? How is this limited?
- What is back-EMF and why is it important?
- Compare PMDC, shunt, series, compound motors.
- How does field weakening achieve speeds above the rated speed?

Module 7.7 — Induction Machines

Duration: 6–8 h · **Difficulty:** Advanced

The three-phase induction motor is the workhorse of industry. Stator windings produce a rotating magnetic field at synchronous speed n_s = 120·f/p (rpm), where p is pole count. The rotor runs slightly below n_s — the **slip** s = (n_s − n)/n_s. Without slip, there is no relative motion between rotor and rotating field, no induced rotor current, no torque.

Equivalent circuit

The per-phase equivalent circuit has: stator resistance R_s, stator leakage L_s, magnetizing inductance L_m, rotor resistance R_r referred to stator, rotor leakage L_r referred to stator, and a slip-dependent resistor R_r·(1−s)/s representing the mechanical load. Torque: T = (3/ _s)·|I_r|²·R_r·(1−s)/s. The torque-speed curve has a starting torque, a breakdown torque, and a synchronous speed at no-load.

Starting methods

Direct-on-line (DOL) starting draws 6–7× rated current for ~1 s — acceptable for small motors but stresses the supply. Star-delta starting reduces starting current to ~2× by starting in wye (lower voltage per phase) then switching to delta. VFD (variable-frequency drive) starts at low frequency and low voltage, ramping up — the gentlest method. Auto-transformer and soft-starter are intermediate options.

CHECKPOINT — Can you answer these?

- Define slip. Why can't an induction motor run at synchronous speed?
- Sketch the torque-speed curve of an induction motor. Mark starting, breakdown, and synchronous torques.
- Compute the synchronous speed of a 4-pole motor at 50 Hz. At 60 Hz.
- Why does a VFD produce less starting stress than DOL?

Module 7.8 — Synchronous Machines and BLDC/PMSM

Duration: 7–9 h · **Difficulty:** Advanced

Synchronous motors run at exactly n_s = 120·f/p — no slip. The rotor field comes from permanent magnets (PMSM, BLDC) or from a DC excitation winding (large synchronous machines). PMSM (sinusoidal back-EMF) and BLDC (trapezoidal back-EMF) are essentially the same machine with different drive strategies — PMSM uses three sinusoidal currents, BLDC uses two-phase square-wave currents. Both are 90+% efficient and power everything from drones to EVs.

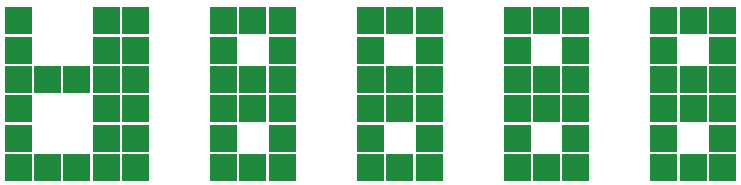
BLDC commutation

BLDC uses three Hall-effect sensors to detect rotor position in six sectors (60° each). Each sector corresponds to a specific pair of conducting phases. The drive switches phases every 60°. This is **trapezoidal (six-step) commutation**. Simple, robust, but produces torque ripple at each switching.

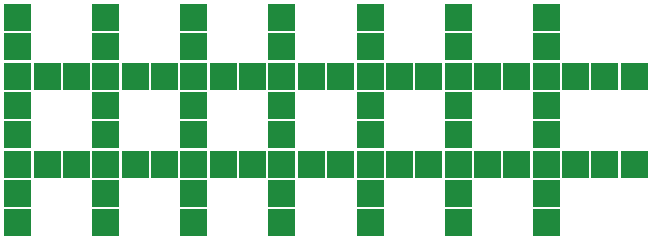
PMSM and Field-Oriented Control (FOC)

FOC (also called vector control) treats the PMSM like a DC motor via a mathematical transformation. The three-phase currents are transformed to a rotating reference frame (Clarke + Park transforms) aligned with the rotor flux. In this frame, the d-axis current controls flux (set to zero for maximum torque per amp) and the q-axis current controls torque — exactly like the field and armature currents of a DC motor. Two PI controllers regulate i_d and i_q; the outputs are transformed back to three-phase voltages for the inverter. FOC delivers smooth, ripple-free torque and is the standard for EVs, drones, and industrial servos.

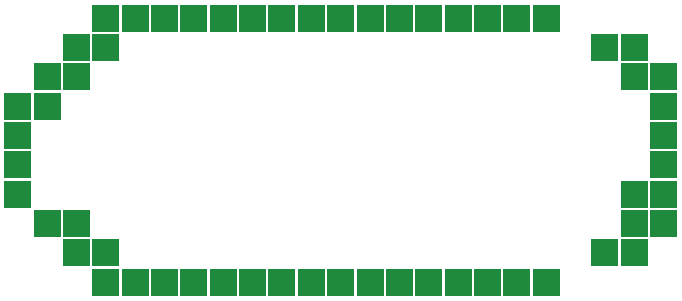
$$\text{Clarke: } i_\alpha, i_\beta = (2/3)\cdot[i_a - \tfrac{1}{2}i_b - \tfrac{1}{2}i_c,\ (\sqrt{3}/2)(i_b - i_c)] \quad \big| \quad \text{Park: } i_d, i_q = [i_\alpha\cdot\cos\theta + i_\beta\cdot\sin\theta,\ -i_\alpha\cdot\sin\theta + i_\beta\cdot\cos\theta]$$



FOC block diagram — Clarke+Park transform PI controllers (i_d, i_q) inverse Park+Clarke 3-phase inverter.



BLDC six-step commutation — Hall sensors detect rotor sector; two phases conduct at a time. Simple, robust, but with torque ripple.



Induction motor — three-phase stator windings produce a rotating field; cage rotor is dragged along at sub-synchronous speed (slip).

HANDS-ON PROJECT — Python: Simulate FOC of a PMSM

Goal: simulate FOC and verify smooth torque.

Plant: PMSM with R_s = 0.5 , L_d = L_q = 1 mH, K_t = 0.1 Nm/A, J = 0.001 kg·m².

Controller: two PI loops on i_d, i_q. i_d* = 0, i_q* = T*/K_t. Inverse Park, inverse Clarke, space-vector modulation. Sample at 10 kHz.

Simulate: step T* from 0 to 1 Nm at t = 10 ms. Confirm i_q tracks its reference, i_q settles in ~5 ms. Torque is smooth, no ripple.

Compare: run the same motor with six-step BLDC commutation. Note the torque ripple (~10%). FOC is fundamentally smoother.

CHECKPOINT — Can you answer these?

- What is the difference between BLDC and PMSM?
- Explain Clarke and Park transforms. What is the reference frame?
- Why does FOC give smoother torque than six-step commutation?
- What is the role of i_d and i_q in FOC? Why set i_d* = 0?

Module 7.9 — Motor Drives: V/f and FOC

Duration: 5–7 h · **Difficulty:** Advanced

V/f control for induction motors

The simplest induction motor drive: maintain V/f constant so the stator flux stays at rated value. At low frequency, boost the voltage to compensate for stator resistance. V/f is open-loop (no speed feedback) — fine for fans, pumps, and conveyors. Speed regulation is poor (~5%) but the drive is cheap and robust.

FOC for induction and PMSM

Closed-loop FOC requires a rotor position sensor (encoder, resolver, or sensorless estimator). For induction motors, the rotor flux angle must be estimated from the slip frequency and stator currents. For PMSM, the rotor angle is measured or estimated via back-EMF or high-frequency injection. FOC delivers 0.1% speed regulation and 4-quadrant operation (motoring and braking in both directions) — essential for servos and EVs.

CHECKPOINT — Can you answer these?

- Why does V/f control keep the flux constant?
- Why is V/f open-loop? What kind of load is it suitable for?
- What sensors does closed-loop FOC require?
- What is sensorless FOC? What are its limitations?

Module 7.10 — Phase 7 Capstone: BLDC FOC Driver

Duration: 14–20 h · **Difficulty:** Advanced

HANDS-ON PROJECT — Phase 7 Capstone: BLDC FOC Motor Controller

Goal: build a complete FOC driver for a small PMSM (drone motor).

Hardware: STM32G4 Nucleo (with motor-control SDK), 3-phase inverter board (B-G431B-ESC1 discovery kit or custom), 2204 drone motor with encoder, 12 V 5 A bench supply.

Software: STM32 MCSDK (Motor Control SDK) generates the FOC framework. Tune PI gains for current loop and speed loop.

Implementation: current loop at 16 kHz, speed loop at 1 kHz. ADC samples phase currents synchronously with PWM. Hall sensors or encoder for position.

Tests: (1) Open-loop ramp: spin the motor at fixed duty. (2) Tune current loop with step response. (3) Tune speed loop. (4) Measure torque vs i_q. (5) 4-quadrant: motor and brake. (6) Sensorless start.

Document: schematic, code, tuning procedure, measurement plots, video. This is the most substantial build so far — budget 3 weekends.

CHECKPOINT — Can you answer these?

- Sketch the FOC block diagram: from current sensor to gate driver.
- Why is ADC sampling synchronized to PWM center?
- How do you tune the current loop? The speed loop?
- What happens if i_d* is not zero? (Hint: field weakening.)

"FOC is the trick that turns a PMSM into a DC motor. Rotate the reference frame to align with the rotor flux, and the three AC currents collapse into two DC quantities: one for flux, one for torque. The math is two transforms; the payoff is the smoothest torque you can get from an electric motor."

— Volt, on field-oriented control



VOLT NARRATES

Phase 7 is done. You understand switching converters. You can simulate a buck and an H-bridge in SPICE. You can drive a BLDC motor with six-step or FOC. Phase 8 takes you up in frequency. Past the lumped-circuit approximation. Into the world where wires become transmission lines and signals radiate.



Phase 8 — EM Fields, RF & Communications

Duration: 6 weeks · **Modules:** 9 · **Goal:** cover Maxwell's equations in real engineering problems, transmission lines, antennas, RF front-ends, and analog + digital modulation.

EM and RF are where the lumped-circuit approximation breaks down. Once the physical dimensions of your circuit approach the wavelength of the signals it carries, you can no longer pretend wires are equipotential. A 1 GHz signal has a 30 cm wavelength — your PCB traces are transmission lines, not wires. This phase teaches you the tools to handle that: Smith charts, scattering parameters, antenna theory, and the analog signal chain that drives modern wireless.



VOLT NARRATES

Phase 8 is where circuits stop being circuits. When the wavelength gets close to the size of your board, wires become transmission lines. Components radiate. Ohm's law quietly stops working. The new vocabulary is the price of admission to RF. Smith chart. S-parameters. VSWR. Reflection coefficient. Pay it. The wireless world is worth it.



AMP ASKS

So a wire isn't just a wire anymore? It's a transmission line?



VOLT SAYS

Yes. Above 100 MHz or so. Below that, you can pretend. Above, you can't.

Module 8.1 — Vector Calculus for EM

Duration: 4–6 h · **Difficulty:** Foundation

CS BRIDGE

Vector calculus 3D gradient operations in graphics. Gradient ( ), divergence (·F), and curl (×F) are the same operators you use in fluid simulation, image processing (gradient = Sobel filter), and physics engines. The gradient of a scalar field gives the direction of steepest ascent. Divergence measures whether a point is a source (positive) or sink (negative). Curl measures rotation. Maxwell's four equations are written in this language; once you internalize the three operators, Maxwell becomes readable.

Three operators, three theorems

Operator	Symbol	Acts on	Returns	Theorem
Gradient		Scalar field	Vector field	Fundamental theorem (line integral)
Divergence	·F	Vector field	Scalar field	Divergence (Gauss) theorem
Curl	×F	Vector field	Vector field	Stokes' theorem

Three identities you must know: ×( ) = 0 (curl of gradient is zero — conservative fields), ·(×F) = 0 (divergence of curl is zero — solenoidal fields), and ×(×F) = (·F) – ²F (curl of curl identity, used to derive the wave equation from Maxwell).

CHECKPOINT — Can you answer these?

- Compute for = x²y + 3yz. Compute ·F for F = (x², y², z²).
- State the divergence theorem and Stokes' theorem.
- Derive the wave equation from Maxwell in free space.
- Why is the curl of a gradient always zero?

Module 8.2 — Electrostatics: Gauss, Boundary Conditions

Duration: 5–7 h · **Difficulty:** Intermediate

Gauss's law: **E**·d**A** = Q_enc/ . For symmetric charge distributions (point, line, plane, sphere), use a Gaussian surface that matches the symmetry and the integral becomes trivial. A point charge Q gives E = Q/(4 r²). An infinite plane of charge gives E = /(2) — independent of distance. A spherical shell gives zero field inside and E = Q/(4 r²) outside (same as point charge).

Capacitance and dielectrics

Two conductors at different potentials hold charges ±Q; the ratio C = Q/V is the capacitance. Parallel plate: C = ·A/d. Coaxial: C = 2 /ln(b/a). A dielectric material between the plates multiplies by the relative permittivity _r (4–8 for FR4, 10 for alumina, 80 for water). The energy stored is W = ½·C·V². The force between plates is F = ½·V²·dC/dx.

Boundary conditions

At an interface between two dielectrics: the tangential E is continuous, and the normal D = E has a discontinuity equal to the surface charge. At a conductor surface: tangential E = 0 (no field inside a conductor at equilibrium), normal E = / . These boundary conditions plus Gauss's law let you solve any static problem.

CHECKPOINT — Can you answer these?

- Use Gauss's law to find E for an infinite line charge .
- Compute the capacitance of 1 m of coax with a=1mm, b=5mm, _r=2.25.
- State the boundary conditions on E and D at a dielectric interface.
- Why does a dielectric increase capacitance?

Module 8.3 — Magnetostatics: Biot-Savart, Ampère

Duration: 5–7 h · **Difficulty:** Intermediate

Biot-Savart law: **B** = (/4)· I·d**l**×**r** /r². The magnetic field of a current loop. For an infinite straight wire: B = ·I/(2 ·r). For a solenoid (n turns per meter): B = ·n·I inside. Ampère's law: **B**·d**l** = ·I_enc — the dual of Gauss's law, useful for symmetric current distributions.

Inductance and magnetic energy

Inductance L = /I (flux linkage per amp). Solenoid: L = ·N²·A/l. Coax: L = (/2)·ln(b/a)·l. Energy stored: W = ½·L·I². The force between two current-carrying loops is F = ½·I²·dL/dx (electromechanical energy conversion — the basis of every motor and actuator).

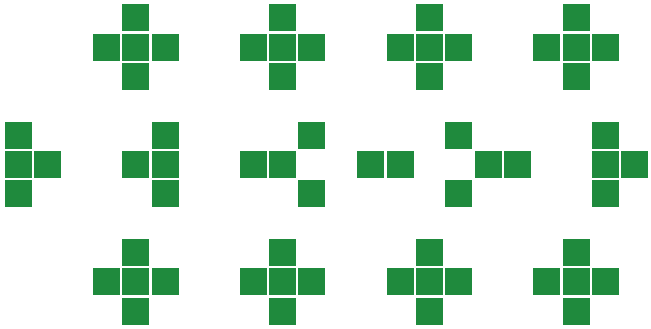
CHECKPOINT — Can you answer these?

- Use Biot-Savart to find B at the center of a circular loop of radius R carrying current I.
- Compute the inductance of a 10-turn, 5 cm-long, 1 cm-radius solenoid.
- State Ampère's law. Apply it to a coaxial cable.
- How does a ferromagnetic core increase inductance?

Module 8.4 — Maxwell's Equations and EM Waves

Duration: 6–8 h · **Difficulty:** Advanced

In free space (no charges, no currents), Maxwell's equations reduce to the wave equation: ²**E** = · ²**E**/ t², with wave speed v = 1/ ( ) = 3×10 m/s = c. Light is an EM wave. The wave has E and B perpendicular to each other and to the direction of propagation — a transverse wave. The ratio |E|/|B| = c. The Poynting vector **S** = (1/)·**E**×**B** gives the power flow per unit area.



EM wave — E and B perpendicular to each other and to the direction of propagation. Light is exactly this, at ~500 THz.

Plane waves and polarization

A plane wave E(z, t) = E ·cos(kz − t)·x has wavelength = 2 /k, frequency f = /(2), and propagates in +z. The wave impedance of free space = (/) 377 . Polarization: linear (E stays in one direction), circular (E rotates), elliptical (general). Antennas are sensitive to polarization — a vertical antenna receives vertically-polarized waves best.

The EM spectrum

Band	Frequency	Wavelength	Use
DC	0		Power
Audio	20 Hz–20 kHz	15,000 km–15 km	Sound (not EM, but same math)
RF	3 kHz–300 GHz	100 km–1 mm	Radio, WiFi, 5G, radar
Microwave	300 MHz–300 GHz	1 m–1 mm	Oven, radar, satcom
Infrared	300 GHz–430 THz	1 mm–700 nm	Heat, IR remote, LiDAR
Visible	430–790 THz	700–400 nm	Human vision
UV	750 THz–30 PHz	400–10 nm	Sterilization, fluorescence
X-ray	30 PHz–30 EHz	10 nm–10 pm	Imaging, crystallography
Gamma	> 30 EHz	< 10 pm	Nuclear medicine, astrophysics

CHECKPOINT — Can you answer these?

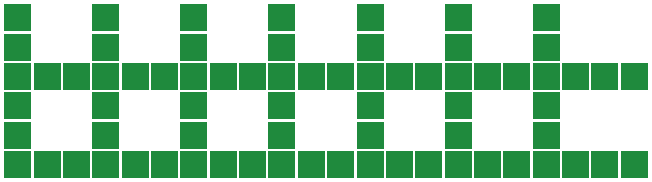
- Derive the wave equation from Maxwell in free space. What is v?
- What is the Poynting vector? What is its physical meaning?
- What is the wave impedance of free space?
- Compute the wavelength of a 2.4 GHz WiFi signal in air. In water (_r=80)?

Module 8.5 — Transmission Lines and the Smith Chart

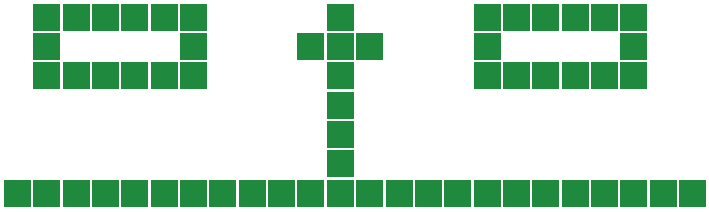
Duration: 7–9 h · **Difficulty:** Advanced

A transmission line is any pair of conductors where the physical length is comparable to the signal wavelength. Coax, microstrip (PCB trace over ground), stripline, twisted pair, waveguide. The line has distributed inductance L per meter and capacitance C per meter, giving characteristic impedance Z = (L/C) (typically 50 for RF, 75 for video, 100 for twisted-pair Ethernet). Signals propagate at v_p = 1/ (LC) 2c/3 for coax (due to dielectric).

$$Z_0 = \sqrt{L/C} \quad | \quad v_p = 1/\sqrt{LC} \quad | \quad = (Z_L - Z_0) / (Z_L + Z_0)$$



Transmission line — distributed L per meter and C per meter. Characteristic impedance Z = (L/C); 50 is the RF standard.



Matching network — series + shunt L/C arrangement. Goal: transform Z_load to Z so 0 and no power reflects.

Reflections and matching

If the load impedance Z_L Z_0, part of the wave reflects back. The reflection coefficient = (Z_L − Z_0)/(Z_L + Z_0). For Z_L = Z_0, = 0 — no reflection, perfect match. For Z_L = 0 (short), = −1 — full reflection with inversion. For Z_L = (open), = +1 — full reflection in phase. Standing waves form when forward and reflected waves interfere; the standing wave ratio SWR = (1+| |)/(1−| |). SWR = 1 is matched; SWR is fully reflected.

The Smith chart — a graphical calculator for RF

The Smith chart is a polar plot of overlaid with constant-resistance and constant-reactance circles. Any impedance Z is a point on the chart. Adding series L or C moves the point along constant-resistance circles. Adding shunt L or C moves along constant-conductance circles. Matching is the art of moving the point to the center (Z = Z_0). Once you internalize the chart, RF design becomes geometric — you can design matching networks by inspection.

HANDS-ON PROJECT — Python: Interactive Smith Chart

Goal: build a Smith chart tool in Python.

Tasks: plot the Smith chart (constant-R and constant-X circles). Plot a load impedance. Animate adding series L, series C, shunt L, shunt C and trace the path on the chart.

Use: design a matching network for Z_L = 100 − j50 to Z_0 = 50 at 1 GHz. Use L-match, -match, and stub matching. Verify each design with scikit-rf.

Bench (optional): build the matching network with surface-mount L and C on a PCB. Measure with a VNA (NanoVNA $50). Confirm the match.

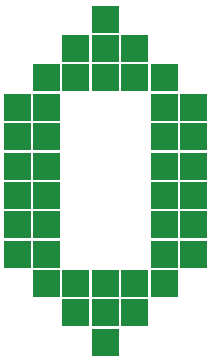
CHECKPOINT — Can you answer these?

- What is the characteristic impedance of a transmission line?
- Compute for Z_L = 25 , Z_0 = 50 . What is the SWR?
- What is the velocity factor of a coax with _r = 2.25?
- Why is 50 the standard RF impedance?
- Design an L-network to match 100 + j50 to 50 at 1 GHz.

Module 8.6 — Antennas: Dipole, Patch, Gain, Pattern

Duration: 5–7 h · **Difficulty:** Intermediate

An antenna converts a guided wave (in a transmission line) to a free-space wave (radiated). The half-wave dipole is the reference: length /2, fed at the center, radiates broadside with a doughnut-shaped pattern. Gain ~2.15 dBi. Input impedance ~73 (close to 75 coax). The patch antenna is a flat rectangle of metal /4 above a ground plane — used in PCBs, GPS, WiFi arrays. Compact but narrowband (~1% bandwidth).



Half-wave dipole — vertical conductor (/2) with doughnut-shaped radiation pattern (broadside max).

Antenna parameters

- **Gain:** dBi = dB relative to isotropic. Dipole = 2.15 dBi. Patch = 6–9 dBi. Horn = 15–25 dBi.
- **Radiation pattern:** polar plot of radiated power. Main lobe, sidelobes, nulls.
- **Beamwidth:** angular width of main lobe at −3 dB. Narrow beam = high gain.

- **Bandwidth:** frequency range over which SWR < 2 (or other criterion).
- **Polarization:** direction of E field. Vertical, horizontal, circular.
- **Efficiency:** radiated power / input power. Losses come from conductor, dielectric, mismatch.

The Friis transmission equation

Power received at distance R from a transmitter: P_r = P_t·G_t·G_r·(/(4 R))². The (/(4 R))² term is the free-space path loss. At 2.4 GHz over 100 m, that's −40 dB. With 2 dBi antennas at each end, P_r is P_t − 36 dB. A WiFi router at 100 mW (20 dBm) delivers −16 dBm at the receiver — well above the −90 dBm sensitivity of a typical WiFi chip. This is why WiFi works across a house.

$$P_r = P_t \cdot G_t \cdot G_r \cdot (\ / (4\ R))^2 \quad \text{(Friis equation)}$$

HANDS-ON PROJECT — Bench: 2.4 GHz Dipole + RTL-SDR

Goal: build a simple 2.4 GHz receiver and see WiFi signals.

Antenna: half-wave dipole from two 31 mm pieces of wire on an SMA connector.

Receiver: RTL-SDR ($25) tuned to 2.4 GHz. SDR# or GNU Radio as the spectrum analyzer.

Observe: turn on a WiFi router nearby. See the spectrum light up at 2.412 GHz (channel 1). Walk around — signal strength changes with orientation and distance.

Map: walk the perimeter of a room with the SDR. Plot signal strength vs position. You are doing RF site survey.

CHECKPOINT — Can you answer these?

- What is antenna gain in dBi? What is the gain of a half-wave dipole?
- Compute the free-space path loss at 2.4 GHz over 10 m.
- Why is a patch antenna narrowband? How can you increase its bandwidth?
- What is the polarization of a vertical dipole? What happens if the receiver is horizontal?

Module 8.7 — RF Front-End: LNA, Mixer, Oscillator, PLL

Duration: 6–8 h · **Difficulty:** Advanced

Every radio receiver (superheterodyne) has the same architecture: antenna LNA mixer IF filter demodulator. The **LNA** (low-noise amplifier) boosts the weak signal with minimal added noise. The **mixer** multiplies the RF signal by a local oscillator (LO) to shift it to a lower intermediate frequency (IF) where filtering is easier. The **PLL** (phase-locked loop) generates the LO with crystal-stable frequency.

$$\text{Noise figure: } F = (SNR)_{in} / (SNR)_{out} \quad | \quad \text{Noise temp: } T_e = T_0 \cdot (F - 1)$$

The Friis cascade — first stage dominates

Total noise figure of a cascade: F = F_1 + (F_2 − 1)/G_1 + (F_3 − 1)/(G_1·G_2) + … The first stage dominates — its noise is amplified by everything downstream, while later-stage noise is divided by the preceding gain. This is why receivers have a low-noise first stage (LNA) close to the antenna.

Mixers — frequency translation

An ideal mixer outputs the product of two inputs: v_out = v_RF·v_LO. By trigonometric identity, this produces signals at f_RF + f_LO and |f_RF − f_LO|. Filter to keep one and reject the other. Image rejection is a key design parameter — the image frequency (2·f_LO − f_RF) appears at the same IF as the desired signal. Solutions: image-reject filter before the mixer, or I/Q (quadrature) mixing that cancels the image mathematically.

CHECKPOINT — Can you answer these?

- Sketch a superheterodyne receiver. What is the IF?
- Why does the LNA dominate the noise figure?
- What is the image frequency? How do you reject it?
- What is a PLL? What does it lock to?

Module 8.8 — Modulation: AM, FM, PM, ASK, FSK, PSK, QAM

Duration: 7–9 h · **Difficulty:** Advanced

CS BRIDGE

Modulation encoding scheme. Modulation is the encoding of bits onto a carrier. ASK (amplitude shift keying) is 1-bit-per-symbol NRZ on the amplitude. FSK is two tones (like a Bell modem). PSK is phase shifts. QAM combines amplitude and phase — 16-QAM carries 4 bits per symbol, 64-QAM carries 6, 256-QAM carries 8. The Shannon limit C = B·log (1 + SNR) tells you the maximum bits/s for a given bandwidth and SNR. Modern WiFi (802.11ax) uses 1024-QAM — 10 bits per symbol — and pushes close to the Shannon limit.

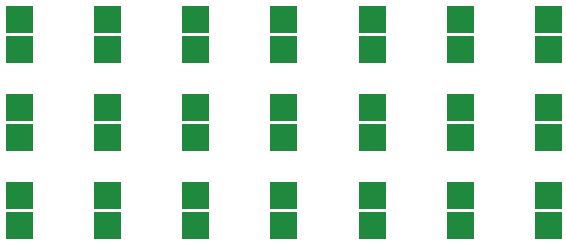
Analog modulation — AM and FM

AM: m(t) = (A_c + m·cos _m t)·cos _c t. The envelope carries the signal. Bandwidth = 2·f_m. Simple to modulate and demodulate (envelope detector), but vulnerable to noise. FM: phase of the carrier varies with m(t). Bandwidth = 2·(+1)·f_m (Carson's rule), where = f/f_m. Wider bandwidth, but much better noise immunity than AM — this is why FM radio sounds better.

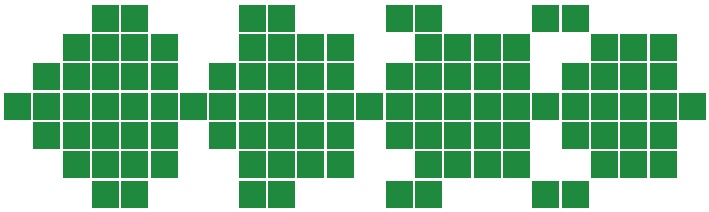
Digital modulation — constellation diagrams

Digital modulation maps bits to points in the I-Q plane. BPSK: 2 points on the real axis. QPSK: 4 points on the unit circle. 16-QAM: 16 points in a 4×4 grid. The constellation diagram tells you

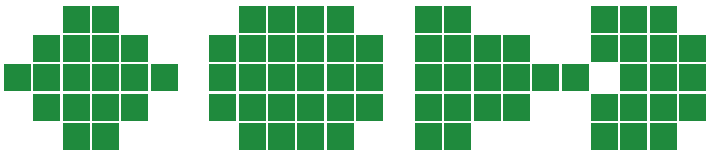
everything: minimum distance between points (sets BER), peak-to-average power ratio (sets amplifier backoff), and spectral efficiency (bits/s/Hz).



16-QAM constellation — 16 points in a 4×4 I-Q grid. 4 bits per symbol. Spectral efficiency = 4 bits/s/Hz.



AM modulation — carrier × envelope. Bandwidth = 2·f_m. Simple to modulate and demodulate but vulnerable to noise.



FM modulation — constant amplitude, varying frequency. Wider bandwidth than AM but much better noise immunity.

VOLT INSIGHT

Every WiFi packet. Every LTE subframe. Every Bluetooth ping. They all use some variant of PSK or QAM. The Shannon limit C = B·log (1+SNR) tells you the max bits per second for a given bandwidth and SNR. Modern systems push close to that limit. The rest is coding, modulation, and signal processing.

AMP ASKS

So Shannon basically defined the speed limit for all wireless?

VOLT SAYS

Yes. In 1948. We've been chasing his limit ever since.

Shannon: $C = B \cdot \log_2(1 + SNR)$ | Spectral efficiency: $= \log_2(M)$ [bits/symbol for M-QAM]

HANDS-ON PROJECT — Python + GNU Radio: BPSK Modulator

Goal: transmit and receive a BPSK signal.

Tasks: (1) Generate a random bit stream. (2) Map to ±1 (BPSK). (3) Upsample and pulse-shape with root-raised cosine. (4) Multiply by carrier. (5) Add noise. (6) Downconvert, matched filter, sample. (7) Compute BER vs SNR.

Verify: BER should match the theoretical curve BER = Q((2·SNR)).

Extension: do the same for QPSK and 16-QAM. Plot the constellation at each SNR. Watch the points spread as noise increases.

CHECKPOINT — Can you answer these?

- Compute the Shannon limit for a 1 MHz channel at 20 dB SNR.
- Sketch the constellation of QPSK, 16-QAM, 64-QAM. How many bits per symbol?
- Why does FM have better noise immunity than AM?
- What is the relationship between bit rate, symbol rate, and bits per symbol?

Module 8.9 — Phase 8 Capstone: FM Transmitter + SDR Receiver

Duration: 10–14 h · **Difficulty:** Advanced

HANDS-ON PROJECT — Phase 8 Capstone: FM Transmitter + SDR Receiver

Goal: build an FM transmitter and receive your own signal with an SDR.

Transmitter: use an oscillator (e.g., MAX2606 VCO or a discrete Colpitts) modulated by audio from your phone. Tune to an unused FM band (88–108 MHz). Keep power under 100 W to stay legal.

Receiver: RTL-SDR + GNU Radio. Build an FM demodulator: bandpass filter at your transmit frequency limit frequency discriminator de-emphasis audio sink.

Tests: (1) Transmit a tone. Receive and confirm. (2) Transmit music. Listen. (3) Measure SNR. (4) Walk away — measure range.

Extension: modulate with digital data (FSK). Send a UART byte stream. Demodulate in GNU Radio. Compute BER.

Note: transmit power is regulated. In most countries, < 100 W on FM is unlicensed. Above that you need a license. Check your local rules.

> "The grid is the largest machine ever built by humans. You are about to understand how it works."
>
> — Volt, on power systems

CHECKPOINT — Can you answer these?

- Sketch the FM transmitter and SDR receiver block diagrams.
- How does the SDR demodulate FM? What is the role of the discriminator?
- What is the legal limit for unlicensed FM transmission in your country?
- How would you extend this to send digital data?



VOLT NARRATES

Phase 8 is done. You have Maxwell's four equations. You understand transmission lines and impedance matching. You can build an antenna. You can decode FM with an SDR. Phase 9 takes you to the other end of the scale. The nanometer. Where one chip holds a billion transistors and the physics is fabrication.



Phase 9 — VLSI & IC Design

Duration: 5 weeks · **Modules:** 8 · **Goal:** understand the CMOS transistor, layout, fabrication flow, and digital IC design — and prototype a pipelined adder on FPGA plus a custom inverter layout.

VLSI (Very Large Scale Integration) is the engineering of putting a billion transistors on a chip of silicon. As a CS engineer you have written software for these chips; this phase teaches you how the chips themselves are designed. The work splits into **front-end** (RTL design and verification — very close to CS) and **back-end** (synthesis, place-and-route, layout — more EE-specific). Both are essential to understanding why chips cost what they cost and why they fail when they fail.



VOLT NARRATES

Phase 9 is the chip-designer's view. The front-end is Verilog. Almost the same as the FPGA work in Phase 4. The back-end is new. Synthesis. Place-and-route. Timing sign-off. Layout. The economics are brutal. A leading-edge fab costs 15 billion dollars. A single mask set costs 5 million. That's why only three companies on Earth can build a 5 nm chip.



AMP ASKS

Only three companies? That doesn't sound like a healthy market.



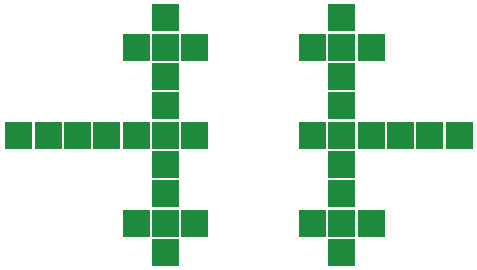
VOLT INSIGHT

It isn't. But that's the cost of leading-edge physics. Welcome to modern semiconductors.

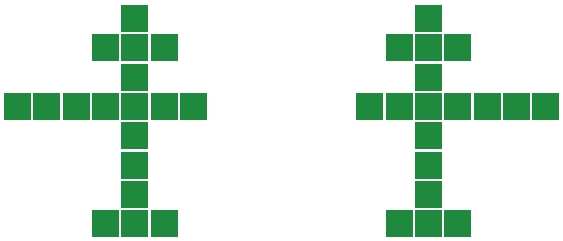
Module 9.1 — CMOS Inverter: Static and Dynamic Behavior

Duration: 6–8 h · **Difficulty:** Intermediate

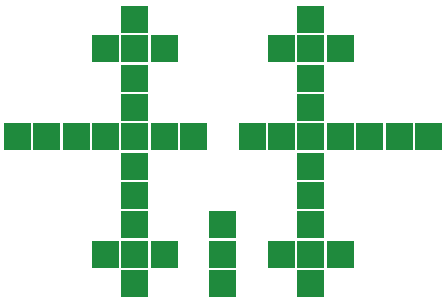
The CMOS inverter is the simplest logic gate and the foundation of all CMOS logic. It has one PMOS (pull-up) and one NMOS (pull-down). When input is low, PMOS conducts, NMOS is off, output is V_DD. When input is high, NMOS conducts, PMOS is off, output is 0. No DC current flows in either steady state — this is why CMOS dominates: static power is essentially zero.



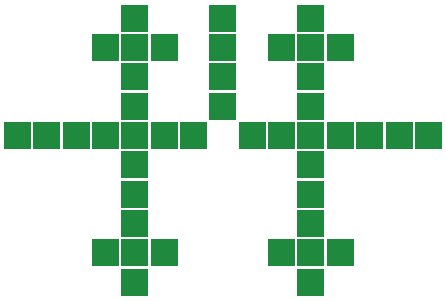
CMOS inverter — PMOS pull-up (top), NMOS pull-down (bottom), input left, output at midpoint.



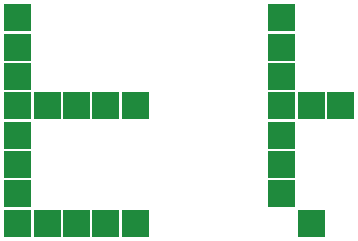
Current mirror — two matched transistors. Reference current on left is mirrored to the load on the right.



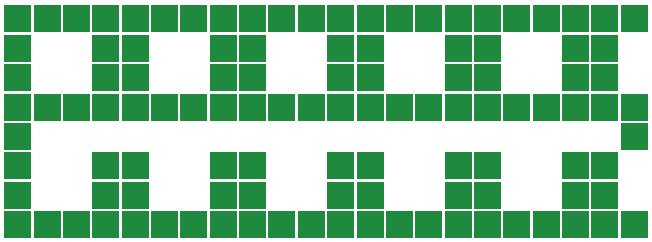
CMOS NAND — 2 PMOS in parallel (pull-up), 2 NMOS in series (pull-down). The natural CMOS gate; smallest area.



CMOS NOR — 2 PMOS in series (pull-up), 2 NMOS in parallel (pull-down). Slower than NAND due to weaker PMOS.



CMOS D flip-flop — master-slave with transmission gates. The register element of every digital chip.



Inverter layout — top-down mask view. PMOS in n-well (top), NMOS in p-substrate (bottom), poly gate crosses both.

Voltage transfer characteristic (VTC)

The VTC plots V_out vs V_in. It has five regions: NMOS off, NMOS saturation / PMOS linear, both saturation, NMOS linear / PMOS saturation, PMOS off. The switching threshold V_M is where V_out = V_in; for symmetric transistors (_N = _P) it is at V_DD/2. Noise margin NM_H = V_OH − V_IH, NM_L = V_IL − V_OL. Larger noise margins = more robust logic.

Dynamic behavior — propagation delay and power

Propagation delay t_p = (t_pHL + t_pLH)/2 (C_L·V_DD)/(2·I_on). For a 1 fF load and 100 A on-current, t_p 5 ps. Real gates have C_L ~ 1–10 fF and t_p ~ 10–100 ps. Dynamic power: P = ·C·V²·f, where is the activity factor (fraction of cycles that switch). At 1 GHz, 1 fF, 1 V, =0.1: P = 0.1 W per gate — a billion gates is 100 W. This is why modern CPUs spend so much effort on clock gating, voltage scaling, and dark silicon.

$$t_p \quad C_L \cdot V_{DD} / (2 \cdot I_{on}) \quad | \quad P_{dyn} = \cdot C \cdot V^2 \cdot f \quad | \quad P_{static} = V_{DD} \cdot I_{leak}$$

CHECKPOINT — Can you answer these?

- Sketch a CMOS inverter. Which transistor is on for each input state?
- Define V_M, V_IL, V_IH, V_OL, V_OH. Compute noise margins.
- Compute the dynamic power of a 1 fF gate at 1 GHz, 1 V, =0.2.
- Why does static power matter in modern CMOS? What is leakage?

Module 9.2 — CMOS Layout and Fabrication Flow

Duration: 6–8 h · **Difficulty:** Intermediate

Layout is the geometric description of every layer on the chip. Layers include n-well, p-substrate, polysilicon (gate), n+ and p+ diffusion (source/drain), contacts, metal1, via1, metal2, etc. Design rules (minimum width, spacing, enclosure) come from the foundry and ensure the chip can be manufactured. Magic VLSI is the open-source layout editor; commercial tools are Cadence Virtuoso and Synopsys Custom Compiler.

The CMOS fabrication flow

- **Substrate:** start with a silicon wafer (p-type for n-well process).
- **Wafer prep:** grow oxide, deposit nitride, apply photoresist.
- **Photolithography:** expose through a mask, develop. Pattern is now in resist.
- **Etch:** remove material where resist is open. Strip resist.
- **Implant:** ion implantation to dope the silicon (n+ or p+).
- **Deposition:** grow or deposit new layers (oxide, polysilicon, metal).
- **Planarization:** chemical-mechanical polishing (CMP) flattens the surface.
- **Metal stack:** repeat deposition/litho/etch for each metal layer (modern processes have 12–15 metals).
- **Passivation:** final oxide/nitride layer with openings for bond pads.

A modern 5 nm process has ~80 mask layers and 600–800 processing steps. The fab costs $15–20 billion to build. This is why only TSMC, Samsung, and Intel can build leading-edge chips — and why the geopolitical stakes around Taiwan (where TSMC is) are so high.

HANDS-ON PROJECT — Magic VLSI: Layout an Inverter

Goal: draw the layout of a CMOS inverter and pass DRC.

Tool: Magic VLSI with a 0.5 m SCMOS design rule set (free).

Tasks: draw the n-well, p-diffusion for PMOS, n-diffusion for NMOS, polysilicon gate crossing both, contacts, metal1 routing. Connect V_DD, GND, V_in, V_out.

DRC: run design rule check. Iterate until clean. Common errors: minimum width, spacing, enclosure violations.

Extract: extract a SPICE netlist from the layout. Simulate in LTspice or ngspice. Verify the VTC matches a schematic-only version — proves the layout is electrically correct.

CHECKPOINT — Can you answer these?

- List the layers in a typical CMOS process (n-well through metal1).
- What is the difference between n+ and p+ diffusion?
- Why does the polysilicon gate need to overlap the diffusion?
- What is DRC? What is LVS? Why are both needed?

Module 9.3 — Combinational CMOS Design and Gate Sizing

Duration: 5–7 h · **Difficulty:** Intermediate

CMOS logic is built from pull-up networks of PMOS and pull-down networks of NMOS. The networks are duals: series in one parallel in the other. A NAND gate has two PMOS in parallel (pull-up) and two NMOS in series (pull-down). A NOR has PMOS in series and NMOS in parallel. Complex gates (AOI, OAI) extend this to multi-level functions in one stage.

Gate sizing and logical effort

Bigger transistors switch faster (more current) but have more input capacitance (slowing the previous gate). **Logical effort** (Sutherland-Sproull) is a method for sizing gates to minimize delay through a chain. Each gate type has a logical effort g (1 for inverter, 4/3 for NAND2, 5/3 for NOR2). The total delay through a chain is d = (g_i·h_i + p_i), where h_i is the electrical effort (C_load/C_in) and p_i is the parasitic. Optimal sizing equalizes the stage effort g·h across all stages.

CHECKPOINT — Can you answer these?

- Draw the schematic of a 3-input NAND gate. How many transistors?
- What is the duality between PMOS pull-up and NMOS pull-down networks?
- Define logical effort. What is g for an inverter? A NAND2?
- Size a chain of 4 inverters driving a 100 fF load with input capacitance 1 fF.

Module 9.4 — Sequential CMOS: Latches and Flip-Flops

Duration: 4–6 h · **Difficulty:** Intermediate

A latch is transparent when its clock is high (or low). A flip-flop is edge-triggered — it samples the input only on the clock edge. The classic positive-edge D flip-flop is built from two latches in a master-slave configuration: the master is transparent during the low phase, the slave during the high phase, so the output changes only on the rising edge.

Setup, hold, and clock-to-Q

Setup time t_su: the input must be stable for t_su before the clock edge. Hold time t_h: must remain stable for t_h after. Clock-to-Q t_cq: delay from clock edge to output update. The minimum clock period is T_clk t_cq + t_pd,logic + t_su — the sum of register delay, combinational delay, and setup. Hold is checked independently: t_cq + t_cd,logic t_h.

CHECKPOINT — Can you answer these?

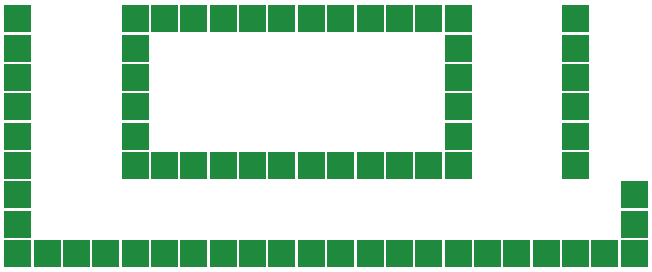
- Draw the master-slave D flip-flop. How many latches?
- Define t_su, t_h, t_cq. What is the minimum clock period?
- What is a scan flip-flop? Why is scan used in testing?
- Why is clock skew a problem for hold-time checks?

Module 9.5 — Static Timing Analysis

Duration: 5–7 h · **Difficulty:** Advanced

CS BRIDGE

STA longest-path in a DAG. Setup analysis finds the longest path (critical path) from any launch flop to any capture flop. Hold analysis finds the shortest path. Both are graph problems on the same DAG (nodes = gates, edges = wires). STA tools (PrimeTime, Tempus) do these graph traversals with delay models that account for wire capacitance, input slew, and coupling.



STA paths — launch flop combinational cloud capture flop. Setup = longest path; hold = shortest path. Slack 0 is mandatory.

STA is exhaustive — it checks every path in the design, not just the ones you thought to simulate. It is the standard sign-off for digital tape-out. Setup slack = T_clk − (t_cq + t_pd,logic + t_su) — must be 0. Hold slack = (t_cq + t_cd,logic) − t_h − clock_skew — must be 0. Negative slack = the design will not work at the target frequency.

Timing corners

Process, voltage, and temperature (PVT) vary. Each combination is a ‘corner’. Standard corners: SS (slow-slow) at 0.9 V, 125°C — worst setup. FF (fast-fast) at 1.1 V, −40°C — worst hold. STA must sign off at all corners — a chip that works at one corner can fail at another.

HANDS-ON PROJECT — Vivado: STA on a Pipelined Multiplier

Goal: see STA in action on a small design.

Tasks: implement a 16×16 pipelined multiplier in Verilog. Synthesize for Artix-7. Open the timing report.

Find: the critical path. What is the maximum clock frequency? Which resources (DSP, LUT, carry chain) does the path use?

Fix: if the critical path is too long, add pipeline registers. Re-run STA. Confirm Fmax improves.

Verify: check hold time at all corners (SS, TT, FF). Fix any violations.

CHECKPOINT — Can you answer these?

- Define setup and hold slack. What does negative slack mean?
- Why must STA check multiple PVT corners?
- What is the difference between setup and hold analysis?
- How does pipelining improve Fmax?

Module 9.6 — Low-Power Design

Duration: 4–6 h · **Difficulty:** Advanced

Power dissipation has three components: dynamic (switching), short-circuit (during the brief overlap of pull-up and pull-down), and leakage (subthreshold and gate). At advanced nodes (28 nm and below), leakage becomes dominant. Low-power techniques: **clock gating** (disable clock to idle blocks — cuts dynamic power 30–60%), **power gating** (cut V_DD to idle blocks — cuts leakage 90%+), **voltage scaling** (DVFS — run slower at lower V), **substrate biasing** (reverse body bias increases V_th, cuts leakage).

CHECKPOINT — Can you answer these?

- Name the three components of CMOS power dissipation.
- How does clock gating save power? Power gating?
- What is DVFS? What is the trade-off?
- Why is leakage a problem at advanced nodes but not at older ones?

Module 9.7 — ASIC vs FPGA Design Flow

Duration: 4–6 h · **Difficulty:** Intermediate

Stage	ASIC	FPGA
Design entry	Verilog/VHDL	Verilog/VHDL
Functional sim	iverilog/VCS	iverilog/Vivado sim
Synthesis	DC Ultra / Genus	Vivado synthesis
Place & route	Innovus / ICC2	Vivado implementation
Timing sign-off	PrimeTime / Tempus	Vivado STA
Physical sign-off	DRC, LVS, ERC, antenna	Bitstream generation
Manufacturing	Tape-out to fab (8–24 weeks)	Download bitstream (instant)
Unit cost (high vol)	Low	High
NRE	$1M–$50M+	Free (tool license)
Performance	Best	~50–70% of ASIC
Power	Best	2–5× ASIC

Rule of thumb: prototype on FPGA, ship in ASIC. For volumes below 100k units, FPGA wins. Above 1M units, ASIC wins. In between is a judgment call based on performance, power, and

time-to-market.

CHECKPOINT — Can you answer these?

- List the stages of an ASIC flow. How does it differ from FPGA?
- When does ASIC beat FPGA on cost? On performance?
- What is NRE? Why does it dominate ASIC economics?
- What is tape-out? Why does it take 8–24 weeks?

Module 9.8 — Phase 9 Capstone: Pipelined Adder + Custom Inverter

Duration: 12–16 h · **Difficulty:** Advanced

HANDS-ON PROJECT — Phase 9 Capstone: FPGA Pipelined Adder + Magic Inverter

Part A — FPGA: implement a 32-bit carry-lookahead adder in Verilog. Pipeline to 4 stages. Synthesize to Artix-7. STA: target 200 MHz Fmax. Onboard: drive from a slide switch + counter, display result on 7-seg. Verify functional + timing.

Part B — Magic: layout a 2-input NAND gate in Magic. DRC clean. Extract to SPICE. Simulate the VTC and propagation delay. Compare to schematic-only simulation. Document layout area (in ²).

Document: schematic, layout, simulation waveforms, STA report, synthesis resource usage, photos/video of the FPGA running.

CHECKPOINT — Can you answer these?

- What was your Fmax? Which path was critical? How did pipelining help?
- What was your NAND layout area? What is the unit (²)?
- Did the extracted simulation match the schematic simulation? Any discrepancies?
- What would you do differently if you were targeting an ASIC tape-out?



VOLT NARRATES

Phase 9 is done. You can write RTL. Synthesize it. Place-and-route it. Sign off on timing. Lay out a gate by hand. You understand why chips cost what they cost. You understand why Moore's law has been a MOSFET story for fifty years. Phase 10 takes you to the largest EE system ever built. The power grid. And asks you to ship one of six capstone projects.



Phase 10 — Power Systems, Energy & Capstones

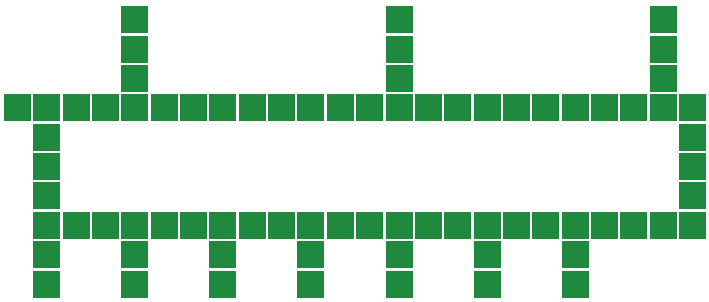
Duration: 8 weeks · **Modules:** 10 + 6 capstones · **Goal:** understand the grid end-to-end and ship one of six capstone projects.

Phase 10 has two halves. The first half (modules 10.1–10.9) covers the bulk power system: generation, transmission, distribution, protection, stability, smart grids, storage, renewables, EV charging. The second half is six capstone projects — pick one or two, budget 4–6 weeks each, and ship a working hardware demo. By the time you finish, you will have a portfolio that demonstrates end-to-end EE fluency.



VOLT NARRATES

Phase 10 is the closing arc. Nine modules on the power grid. The largest machine humans have ever built. Then six capstones. Each one integrates everything you've learned into a shipping hardware project. Pick the capstone that scares you most. That's the one that will teach you the most.



Grid topology — generation transmission distribution load. The largest machine ever built by humans.

Module 10.1 — Power Generation

Duration: 4–6 h · **Difficulty:** Intermediate

Generation technologies convert primary energy (chemical, nuclear, solar, kinetic) to electrical energy. Thermal plants (coal, gas, nuclear) boil water to drive a steam turbine connected to a synchronous generator. Hydro plants use falling water. Gas turbines use combustion gases directly. Wind turbines use aerodynamic lift. Solar PV uses the photovoltaic effect. Each has different economics, dispatchability, and environmental impact.

Source	Capacity factor	Capital $/W	Marginal $/MWh	CO
Coal	50–60%	$2–3	$20–40	High
Gas CCGT	50%	$1	$30–50	Medium
Nuclear	90%	$6–9	$10	Zero

Source	Capacity factor	Capital $/W	Marginal $/MWh	CO
Hydro	30–50%	$2	$0	Zero
Wind (onshore)	35%	$1.5	$0	Zero
Solar PV	20%	$1	$0	Zero
Battery storage	—	$1.5/Wh	—	Zero

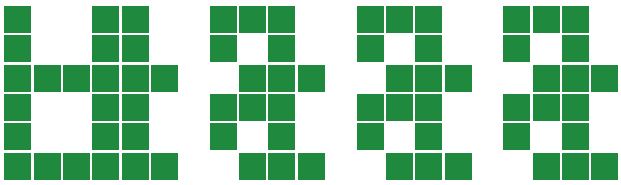
CHECKPOINT — Can you answer these?

- Define capacity factor. Why is nuclear's so high?
- Why are solar and wind called 'intermittent'?
- What is dispatchability? Which sources are dispatchable?
- Compute the LCOE (levelized cost of energy) for a 1 MW solar farm.

Module 10.2 — Transmission and Distribution

Duration: 4–6 h · **Difficulty:** Intermediate

The grid moves power from generators to loads via a hierarchy of voltage levels. Transmission (200–1000 kV) moves bulk power long distances with low loss (I^2R loss is minimized at high V). Subtransmission (50–100 kV) feeds substations. Primary distribution (10–30 kV) feeds neighborhoods. Secondary distribution (120/240 V or 230/400 V) feeds homes. Each transition is a transformer substation.



Transformer substation — HV bus step-down transformer LV bus with breakers. The grid's interconnection points.

Why high voltage for transmission?

Power loss in a line: $P_{loss} = I^2 \cdot R$. For a fixed power $P = V \cdot I$, current scales as $1/V$. So P_{loss} scales as $1/V^2$. Doubling the voltage reduces losses by 4×. This is why long-distance transmission is at 200–1000 kV. The trade-off: higher voltage requires bigger insulators, larger towers, more clearance, and expensive transformers at each end.

HVDC — for very long distances

Above ~600 km, HVDC (high-voltage DC) beats AC: no skin effect, no reactive loss, no stability limit. But HVDC requires expensive converter stations at each end (LCC or VSC). Offshore wind farms and underwater interconnects use HVDC. The longest HVDC link in

operation is in China at 3000+ km, 1.1 MV.

CHECKPOINT — Can you answer these?

- Why is bulk power transmitted at high voltage?
- Compute the loss reduction when going from 200 kV to 500 kV for the same power.
- When does HVDC beat AC? Why?
- What are the voltage levels in a typical grid from generator to home?

Module 10.3 — Power System Protection

Duration: 5–7 h · **Difficulty:** Intermediate

Faults (short circuits) happen. The protection system detects them and isolates the faulted section before it damages equipment or starts a fire. Three components: instrument transformers (CT and VT) step down current and voltage for measurement; protective relays (numerical, microprocessor-based) decide whether to trip; circuit breakers interrupt the fault current.

Protection principles

- **Selectivity:** only the breaker closest to the fault should trip. Adjacent breakers are backups.
- **Speed:** faults must be cleared in 3–5 cycles (60–100 ms) to limit damage.
- **Reliability:** protection must operate when needed, and not operate when not needed.
- **Sensitivity:** detect faults even at low fault current (high-impedance faults).

Common protection schemes

- **Overcurrent (50/51):** trip if current exceeds threshold. Simple, used on radial feeders.
- **Differential (87):** trip if current in current out. Used on transformers, generators, buses.
- **Distance (21):** trip if apparent impedance (V/I) is below threshold — fault is within a distance. Used on transmission lines.
- **Under/over-frequency (81):** trip on frequency excursions. Used for load shedding.

CHECKPOINT — Can you answer these?

- What is the difference between a CT and a VT? What do they do?
- What is selectivity? Why is it important?
- How does differential protection work? Where is it used?
- Why must faults be cleared in 3–5 cycles?

Module 10.4 — Power Quality and Grid Stability

Duration: 4–6 h · **Difficulty:** Intermediate

Power quality is the set of deviations from ideal 50/60 Hz, nominal voltage, pure sine wave. Problems: voltage sags and swells, harmonics, flicker, transients, frequency deviations. Sources: large motor starts (sags), nonlinear loads like rectifiers (harmonics), arc furnaces (flicker), lightning (transients), sudden load/generation imbalance (frequency deviations).

Grid stability — three timescales

- **Rotor angle stability (seconds):** generators must stay in synchronism after a disturbance. Loss of synchronism = blackout.
- **Frequency stability (seconds to minutes):** load and generation must balance. Imbalance causes frequency drift. Primary control (governor), secondary (AGC), tertiary (dispatch).
- **Voltage stability (minutes):** reactive power balance. Insufficient reactive power voltage collapse.



VOLT WARNS

A blackout cascade is one of the most terrifying things in engineering. A fault trips a line. Load shifts to parallel lines. They overload and trip. Generators lose synchronism. Frequency plummets. More lines trip. In minutes, an entire region is dark. The 2003 Northeast blackout took 8 minutes. The protection system's job is to stop the cascade before it starts.



AMP ASKS

Like a distributed system cascade failure? One node dies, the load moves, more nodes die?



VOLT SAYS

Same exact pattern. Different physics. Same math.

CHECKPOINT — Can you answer these?

- Name three power-quality problems and their sources.
- What causes grid frequency to drift? How is it corrected?
- What is reactive power and why does voltage stability depend on it?
- What is a blackout cascade? How does protection try to prevent it?

Module 10.5 — Smart Grid, Microgrids, DER Integration

Duration: 4–6 h · **Difficulty:** Intermediate

A smart grid adds bidirectional communication, distributed sensing, and automated control to the traditional grid. Benefits: faster fault location, demand response, integration of distributed energy resources (DERs — rooftop solar, batteries, EVs), and better asset utilization. A

microgrid is a local grid that can connect to the main grid or island itself — used for hospitals, military bases, and campuses.

DER integration challenges

- **Variability:** solar output drops when a cloud passes. Need fast-ramping backup or storage.
- **Voltage regulation:** distributed PV can raise local voltage above limits.
- **Protection:** bidirectional fault currents break traditional radial protection schemes.
- **Inverter coordination:** thousands of inverters must ride through faults and not all disconnect at once.

CHECKPOINT — Can you answer these?

- What is a smart grid? Name three benefits.
- What is a microgrid? What is islanding?
- What are the challenges of integrating high-penetration solar?
- What is demand response? Give an example.

Module 10.6 — Energy Storage and Battery Management

Duration: 5–7 h · **Difficulty:** Intermediate

Storage is the missing piece of the renewable energy puzzle. Lithium-ion dominates for < 4-hour storage; pumped hydro for daily/weekly; flow batteries and compressed air for utility-scale long-duration. A BMS (battery management system) is required for any lithium-ion pack — it monitors cell voltages, temperature, and current; balances cells; and protects against over-charge, over-discharge, and over-temperature.

SOC and SOH estimation

State of charge (SOC) is the remaining capacity as a percentage. Coulomb counting (integrating current) is simple but drifts. Model-based methods (extended Kalman filter on an equivalent-circuit battery model) are accurate but computationally heavy. State of health (SOH) tracks capacity fade and internal resistance growth over the battery's life.

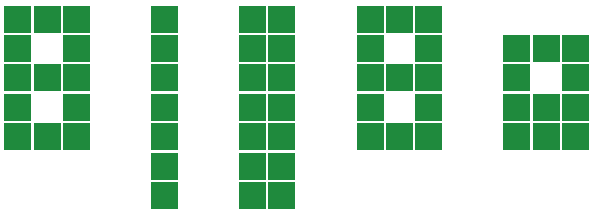
CHECKPOINT — Can you answer these?

- Name three storage technologies and their best use case.
- What does a BMS do? Why is it mandatory for lithium-ion?
- What is SOC? What methods are used to estimate it?
- What is SOH? How is it different from SOC?

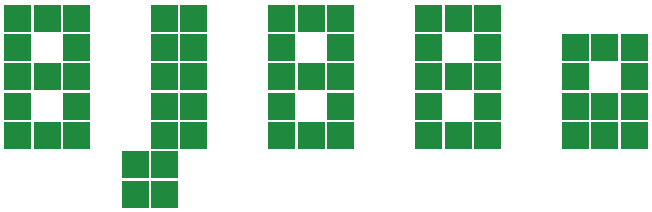
Module 10.7 — Solar PV Systems and MPPT

Duration: 5–7 h · **Difficulty:** Intermediate

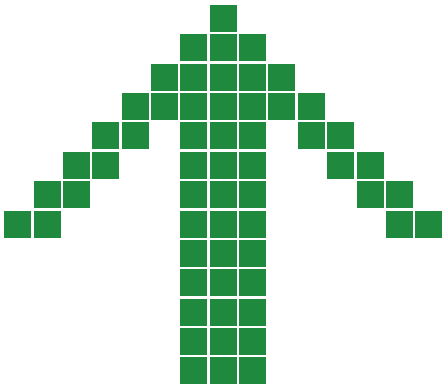
A solar cell is a diode illuminated by light. Photons create electron-hole pairs; the diode's built-in field separates them, producing current. The I-V curve has a knee at the maximum power point (MPP). The MPP voltage depends on irradiance and temperature — it moves continuously as the sun and weather change. **MPPT** (maximum power point tracking) is the algorithm that adjusts the converter duty cycle to keep the panel at its MPP.



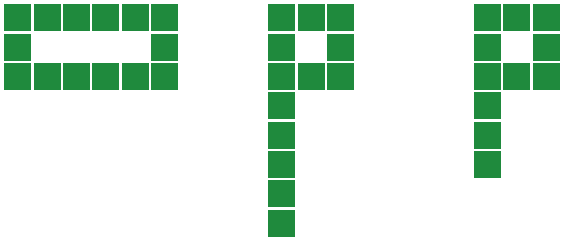
Solar PV + MPPT — panel on left, MPPT controller in middle, DC-DC converter, battery/load on right.



PV system — solar panel MPPT charge controller battery inverter AC load. The standard off-grid topology.



Wind turbine — rotor blades capture kinetic energy; nacelle houses the generator. Power = ½· ·A·v³·C_p (Betz limit 0.593).



EV charger — AC input rectifier (PFC) DC-DC converter battery. Level 2 (7-22 kW) or DCFC (50-350 kW).

CS BRIDGE

MPPT hill climbing. The most common MPPT algorithm, perturb & observe (P&O;), is literally hill climbing on the P-V curve. Perturb the voltage, observe the power: if power increased, keep perturbing in the same direction; if it decreased, reverse. Other algorithms: incremental conductance (gradient sign of dP/dV), fractional open-circuit (heuristic), and model-based. All are optimization at heart — same math you used for gradient descent.

PV system topologies

- **String inverter:** one inverter for a string of 10–20 panels in series. Cheap, single point of failure.
- **Microinverter:** one inverter per panel. More expensive, no single point of failure, panel-level MPPT.
- **Power optimizer + string inverter:** DC-DC per panel for MPPT, central inverter for grid tie.
- **Off-grid:** charge controller + battery + inverter. No grid; for remote sites.

HANDS-ON PROJECT — Python: Simulate a PV Panel + MPPT

Goal: simulate a 250 W panel and run P&O; MPPT.

Model: single-diode model with V_oc = 38 V, I_sc = 9 A, V_mpp = 31 V, I_mpp = 8 A.

Irradiance profile: simulate a cloudy day with rapid changes.

MPPT: P&O; with 0.5 V step, 100 ms update. Track the MPP.

Compare: compute energy harvested with and without MPPT. The MPP tracker should deliver 25–35% more energy than a fixed-voltage operation.

CHECKPOINT — Can you answer these?

- Sketch the I-V and P-V curves of a solar cell. Mark V_oc, I_sc, V_mpp, P_max.
- Explain perturb & observe MPPT. What are its limitations?
- Compare string, microinverter, and optimizer PV topologies.
- Why does shading one panel in a string hurt the whole string?

Module 10.8 — Wind Energy Systems

Duration: 3–5 h · **Difficulty:** Intermediate

A wind turbine extracts kinetic energy from moving air. Power = ½· ·A·v³·C_p, where is air density, A is swept area, v is wind speed, and C_p is the power coefficient (Betz limit: 0.593). Modern turbines operate at C_p 0.45. The v³ dependence means wind power is extremely sensitive to wind speed — a 10% speed increase gives 33% more power.

Three wind turbine types

- **Fixed-speed induction:** simple, robust, but poor energy capture and no reactive control. Legacy.
- **Variable-speed DFIG:** doubly-fed induction generator with a partial-power converter on the rotor. ~30% of turbines.
- **Variable-speed PMSG with full converter:** permanent-magnet generator with full-rated converter. Best energy capture, full reactive control. ~70% of new turbines.

CHECKPOINT — Can you answer these?

- Compute the power in a 50 m diameter stream of air at 8 m/s with C_p = 0.45.
- What is the Betz limit? Why can't a turbine extract more?
- Why are modern turbines variable-speed?
- What is a DFIG? How does it differ from a PMSG?

Module 10.9 — EV Charging Infrastructure

Duration: 3–5 h · **Difficulty:** Intermediate

EV charging has three levels: Level 1 (120/230 V AC, ~1.4 kW — overnight home charging), Level 2 (240/400 V AC, 7–22 kW — home and workplace fast charging), DC fast charging (50–350 kW DC, 80% charge in 20–40 minutes). The onboard charger handles AC levels (rectifies and charges the battery); DC fast chargers bypass the onboard charger and feed the battery directly.

Charging standards

- **SAE J1772:** Level 1/2 connector. Common in US/EU for AC charging.
- **CCS (Combined Charging System):** J1772 + two DC pins. Standard for fast charging in EU/US.
- **CHAdeMO:** Japanese DC fast charging standard. Being phased out.
- **Tesla/NACS:** Tesla's connector, now becoming the US standard (SAE J3400).
- **MCS (Megawatt Charging System):** for trucks and buses, up to 3.75 MW.
- **GB/T:** Chinese standard, AC and DC.

Grid impact of fast charging

A 350 kW fast charger pulls as much power as a small neighborhood. A station with 10 of them is 3.5 MW — comparable to a small industrial plant. Local grid upgrades, on-site battery buffers (to smooth demand peaks), and PV canopies all play a role. Vehicle-to-grid (V2G) — using EV batteries as grid storage — is an emerging opportunity.

CHECKPOINT — Can you answer these?

- What are the three EV charging levels? What is the typical power and use case?
- Compare CCS, NACS, and CHAdeMO.
- Compute the time to charge a 75 kWh EV at 7 kW (Level 2). At 350 kW (DCFC).
- What is V2G? What is the value to the grid?

Module 10.10 — The Six Capstones

Duration: 4–6 weeks each (pick one or two)

Pick one or two of the six capstones below. Budget 4–6 weeks per capstone. Each integrates material from multiple phases and produces a working hardware demo. Document the design, build, test, and lessons learned. By the end you will have a portfolio piece that proves you can ship a real EE project.

HANDS-ON PROJECT — Capstone 1: FOC Motor Controller

Build on Phase 7 Module 10. extend to a full closed-loop motor controller with position, speed, and current loops.

Hardware: STM32G4 + 3-phase inverter + 5010 BLDC + AS5048 encoder + 12 V supply.

Software: STM32 MCSDK FOC framework. Tune current loop (16 kHz), speed loop (1 kHz), position loop (100 Hz).

Features: 4-quadrant operation, sensorless startup, field weakening, regen braking, CAN bus interface.

Tests: step response, disturbance rejection, max speed, max torque, efficiency map.

Deliverable: PCB, firmware, tuning log, demo video, design doc.

HANDS-ON PROJECT — Capstone 2: Solar MPPT Inverter

Goal: build a 100 W off-grid solar inverter with MPPT.

Hardware: 50 W solar panel, MPPT boost converter (12 V 24 V), battery, DC-AC inverter (modified sine or pure sine), ESP32 for control.

MPPT: P&O; algorithm in ESP32. Track MPP under varying irradiance.

Inverter: SPWM H-bridge with LC filter. 230 V / 50 Hz output.

Protection: over-current, over-voltage, under-voltage, over-temperature.

Tests: MPPT efficiency vs irradiance, inverter THD, battery charge profile, run-time with typical load (LED lamp + phone charger).

Deliverable: schematic, PCB, firmware, test report, demo video.

HANDS-ON PROJECT — Capstone 3: Custom PCB Product

Goal: take a product from concept to PCB manufacture and assembly.

Product: pick a small, useful device — examples: USB-C PD trigger board, ESP32 sensor board with battery + solar input, LED driver, logic analyzer, signal generator.

Flow: spec schematic (KiCad) BOM PCB layout DRC order from JLCPCB ($10 for 5 boards) order parts from DigiKey assemble test firmware enclosure (3D print).

Documentation: full design rationale, schematic, layout, BOM with part numbers, assembly photos, test results, firmware, enclosure design.

Deliverable: 5 working boards, one in enclosure, ready to use.

HANDS-ON PROJECT — Capstone 4: Robotics Platform

Goal: build a differential-drive robot with closed-loop control and SLAM-lite.

Hardware: Raspberry Pi 4 (or ESP32-S3) + STM32 motor driver + 2 DC motors with encoders + IMU (MPU6050) + RPLIDAR A1 (or webcam for visual odometry).

Software: ROS 2. Differential-drive kinematics. PID speed control on STM32. IMU + encoder fusion for odometry. occupancy grid mapping (Gmapping or Cartographer). Autonomous navigation (Nav2).

Tests: closed-loop step response, odometry accuracy over 10 m, mapping of a room, autonomous navigation between waypoints.

Deliverable: robot, firmware, ROS workspace, maps, navigation demo video.

HANDS-ON PROJECT — Capstone 5: Smart Grid Demo

Goal: build a tabletop microgrid with PV + battery + load + IoT telemetry.

Components: small PV panel, MPPT charge controller, Li-ion pack with BMS, AC load (LED lamp + fan), inverter, ESP32 for telemetry, Raspberry Pi running Mosquitto + InfluxDB + Grafana.

Features: real-time power flow monitoring, automatic load shedding on low battery, grid/island switching, remote control via MQTT.

Tests: charge profile from PV, discharge profile under load, autonomy time, load shedding behavior, dashboard responsiveness.

Deliverable: working microgrid, dashboard screenshots, test report, video.

HANDS-ON PROJECT — Capstone 6: SDR / RF Receiver

Goal: build a software-defined radio receiver front-end.

Hardware: antenna (dipole or patch for chosen band), LNA (e.g., SPF5189), RTL-SDR or AD9361, Raspberry Pi for processing.

Software: GNU Radio flowgraph: source band-pass filter AGC demodulator (FM/AM/FSK) audio sink or data sink.

Project: receive and decode (a) FM broadcast, (b) ADS-B aircraft transponders (1090 MHz), (c) NOAA weather satellite images (137 MHz), (d) AIS ship tracking (162 MHz). Pick one or more.

Tests: sensitivity measurement, selectivity, dynamic range, decoding success rate.

Deliverable: working receiver, decoded data (audio, image, or position feed), design doc, video.

Closing the Curriculum

If you have worked through all ten phases, including at least one capstone, you have done what a four-year EE degree does — minus the general-education requirements and the electives, plus a level of practical, project-based fluency that most graduates lack. You can read a datasheet, design a circuit, lay out a PCB, program an MCU, debug a control loop, simulate a converter, and ship a hardware product. The next step is to keep building — pick a domain that interested you most (motors, RF, power, VLSI, control) and go deeper. The curriculum is the foundation; the work that follows is the career.

> "You started as a computer scientist curious about circuits. You are finishing as an engineer who can ship hardware. That is not a small thing — it is the rarest skill combination in the technology industry."
>
> — Volt, closing the curriculum



VOLT NARRATES

The curriculum is over. The work begins. Pick something you found fascinating. A motor drive. A radio front-end. A solar inverter. A custom chip. Go deep. The ten phases gave you the foundation. The next ten thousand hours will give you the expertise. I'll see you on the bench.

Appendix A — Toolchain Setup Guide

Detailed installation and hello-world instructions for every tool used in the curriculum. Commands shown for Linux/macOS; Windows users should use WSL2 where possible.

A.1 LTspice

Install: download from analog.com (free, Windows native; macOS via Wine or run in a Windows VM). **Hello world:** draw a 1 k resistor in series with a 1 F cap from a 5 V step source. Transient sim 0–10 ms with 1 s step. Verify the cap voltage rises with = 1 ms.

A.2 KiCad 8

Install: kicad.org downloads for Windows/macOS/Linux. **Hello world:** schematic editor place a resistor and cap wire them in series annotate run ERC (electrical rules check) open PCB editor import netlist place footprints route run DRC.

A.3 Arduino IDE

Install: arduino.cc. **Hello world:** File Examples 01.Basics Blink. Upload to Arduino Uno. LED blinks at 1 Hz.

A.4 ESP32 (PlatformIO)

Install: VS Code + PlatformIO extension. **Hello world:** new project with ESP32 DevKitC board. Sketch: WiFi.begin(ssid, password); loop: print WiFi.localIP(). Open serial monitor at 115200 baud.

A.5 STM32CubeIDE

Install: st.com (free, Eclipse-based). **Hello world:** new project with Nucleo-F446RE. In .ioc: enable GPIOA pin 5 as output (LD2). Generate code. In main loop: HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); HAL_Delay(500). Build, flash, LED blinks at 1 Hz.

A.6 Python (NumPy/SciPy/matplotlib/control)

Install: python.org then pip install numpy scipy matplotlib control jupyter sounddevice. **Hello world:** import numpy as np, matplotlib.pyplot as plt; t = np.linspace(0, 1, 1000); plt.plot(t, np.sin(2*np.pi*5*t)); plt.show()

A.7 Octave

Install: gnu.org/software/octave. **Hello world:** A = [2 1; 1 3]; b = [1; 2]; x = A \ b; disp(x)

A.8 Vivado WebPACK

Install: amd.com (Xilinx) — Vivado WebPACK is free, supports Artix-7. **Hello world:** new project Verilog counter (Module 4.4) synthesize run implementation generate bitstream program the FPGA.

A.9 Magic VLSI

Install: ubuntu: sudo apt install magic. **Hello world:** open Magic, draw a single PMOS, run DRC, fix any errors. See Module 9.2.

A.10 GNU Radio

Install: gnuradio.org — prebuilt binaries for Windows/macOS; apt install gnuradio on Ubuntu. **Hello world:** GNU Radio Companion source (null source) low-pass filter QT GUI sink. Run. See the spectrum display.

Appendix B — Hardware Bill of Materials

Two tiers: Starter (~$150) covers Phases 0–3. Full (~$600) adds what you need for Phases 4–10 and the capstones.

B.1 Starter Kit (~$150)

Item	Qty	Est $	Notes
Breadboard (830-point)	1	8	Full-size, jumper-wire friendly
DMM (digital multimeter)	1	25	Any cheap DMM with V/A/ /continuity
Bench PSU (0–30 V, 0–3 A)	1	40	Or repurpose an old ATX PSU
Resistor kit (1/4 W, 30 values)	1	15	10 to 1 M , 20 each
Capacitor kit (ceramic + electrolytic)	1	15	10 pF to 1000 F
Inductor kit	1	10	1 H to 1 mH
Diode kit (1N4148, 1N4007, 1N5819, Zeners)	1 set	8	All common types
Transistor kit (2N3904/3906, 2N7000, IRFZ44N)	1 set	8	BJT NPN/PNP + MOSFETs
Op-amp kit (LM358, TL072, NE5532, LM741)	5 ea	6	DIP-8 packages
555 timer	5	3	NE555 or LM555
Regulators (7805, 7812, LM317)	5 ea	5	TO-220
LEDs (assorted colors)	20	3	With 330 resistors
Jumper wires (M-M, M-F, F-F)	1 set	6	Dupont style
Arduino Uno (or clone)	1	15	Genuine or clone

B.2 Full Kit (~$600)

Item	Qty	Est $	Notes
Oscilloscope (2-channel, 100 MHz)	1	200	Rigol DS1102Z or Siglent SDS804X
Logic analyzer (8-channel)	1	15	Saleae clone or Sysclk
ESP32 devkit	2	10	ESP32-DevKitC or NodeMCU-32S
STM32 Nucleo-F446RE	1	25	Or Nucleo-G431 for motor control
B-G431B-ESC1 (motor control kit)	1	40	For Phase 7 + Capstone 1
BLDC motor (2204 drone) + prop	1	15	With AS5048 encoder for FOC
L298N motor driver	1	5	Or BTS7960 for higher current
Small DC motor with encoder	2	20	For Phase 6 robotics

Item	Qty	Est $	Notes
FPGA board (Nexys A7 or Basys 3)	1	150	For Phase 4 + 9
RTL-SDR + antenna kit	1	40	For Phase 8 + Capstone 6
Sensors (BME280, MPU6050, GPS, etc.)	1 set	30	I2C and SPI breakouts
Li-ion cells + holder	4	15	18650 cells with BMS for robotics
Bench load (DC electronic load)	1	60	For power converter testing
Soldering iron + solder + flux	1 set	40	Hakko FX-888D or similar

Sourcing tips: DigiKey and Mouser for US/EU; LCSC for Asia (cheap, fast shipping, no minimum); AliExpress for generic modules (slower but cheapest); Amazon for last-minute. For PCB manufacture: JLCPCB (~$10 for 5 boards, 3–7 day turnaround), PCBWay, Elecrow.

Appendix C — CS-to-EE Concept Map

The full mapping from CS to EE, for reference. Each row is a bridge you can walk across to learn the EE concept faster.

CS concept	EE concept	Connection
FSM	Sequential circuit	Same graph structure, same state-register pattern.
Recurrence	LTI difference equation	y[n] = a·y[n−1] + b·u[n] is a first-order IIR filter.
Markov chain	Discrete-time LTI system	State transition matrix A; steady-state = eigenvector of eigenvalue 1.
Convolution (image filter)	Convolution (LTI system)	y[n] = h[k]·x[n−k]. Identical code.
DCT (JPEG)	DFT / FFT	Both are fast orthogonal transforms over sinusoid bases.
DAG / topo sort	Signal-flow graph / Mason's rule	SFG analysis is a topological traversal.
Graph Laplacian	Circuit incidence matrix	Same connectivity encoding. MNA uses it.
Hash function	Modulation / encoding	Both map data to a signal space robustly.
ECC (Hamming, LDPC)	Channel coding	Same linear algebra over GF(2).
Pipelined CPU	Pipelined digital design	Register between stages; same trade-off.
Critical path (STA)	Longest path in DAG	Setup-time analysis is longest-path computation.
MDP	LQR optimal control	Both minimize cost-to-go over a dynamical system.
Hill climbing	MPPT (solar)	P&O; is hill climbing on the P-V curve.
PID controller	Gradient descent (sort of)	Both use error feedback; both are heuristics.
PageRank	Markov chain steady-state	Both find the dominant eigenvector.
Backprop	Adjoint sensitivity in circuits	Chain rule through a graph = adjoint method.
Memory hierarchy	Cache RAM disk	Voltage source cap inductor analog for energy storage.
Coroutine / async	Event-driven interrupt handler	Both run on demand, not by polling.
Race condition	Setup/hold violation	Both arise from uncontrolled timing.
Load balancer	Three-phase load balancing	Both distribute work to keep utilization even.
Backpressure	Current limiting	Both throttle input to protect a downstream system.
Compaction (LSM tree)	Memory folding in DSP	Both merge adjacent levels to bound size.

Appendix D — Math Reference and Formula Sheets

D.1 Trigonometric identities

- sin² + cos² = 1
- sin(A ± B) = sin A cos B ± cos A sin B
- cos(A ± B) = cos A cos B sin A sin B
- sin 2A = 2 sin A cos A; cos 2A = cos²A − sin²A
- e^{j} = cos + j sin (Euler)
- cos = (e^{j} + e^{-j})/2; sin = (e^{j} − e^{-j})/(2j)

D.2 Derivatives and integrals used in EE

f(t)	df/dt	f(t) dt
t^{n}	n·t^{n-1}	t^{n+1}/(n+1)
e^{at}	a·e^{at}	e^{at}/a
sin(t)	·cos(t)	−cos(t)/
cos(t)	− ·sin(t)	sin(t)/
u(t) (unit step)	(t) (impulse)	t·u(t)
(t)	'(t)	u(t)

D.3 Laplace transform pairs

f(t)	F(s)
(t)	1
u(t)	1/s
t·u(t)	1/s²
e^{-at}·u(t)	1/(s+a)
sin(t)·u(t)	/(s² + ²)
cos(t)·u(t)	s/(s² + ²)
e^{-at}·sin(t)·u(t)	/((s+a)² + ²)

D.4 Z-transform pairs

x[n]	X(z)
[n]	1
u[n]	z/(z − 1) (|z| > 1)
$a^n \cdot u[n]$	z/(z − a) (|z| > |a|)
$n \cdot a^n \cdot u[n]$	$a \cdot z/(z - a)^2$
cos(n)·u[n]	z(z − cos)/(z^2 − 2z cos + 1)
sin(n)·u[n]	z sin /(z^2 − 2z cos + 1)

D.5 Vector calculus operators

- **Gradient:** = (/ x, / y, / z). Returns vector pointing in direction of steepest ascent.
- **Divergence:** ·F = F_x/ x + F_y/ y + F_z/ z. Returns scalar; positive at sources, negative at sinks.
- **Curl:** ×F = vector measuring rotation.
- **Laplacian:** ² = ·( ). Appears in Poisson's and wave equations.
- **Divergence theorem:** ·F dV = F·dA.
- **Stokes' theorem:** (×F)·dA = F·dl.

Appendix E — 12-Month Pacing Calendar

A week-by-week schedule mapping every module to a specific week. Assumes 12 hours/week (about 1.5–2 hours/day). Adjust to your own pace; the order matters more than the exact timing.

Week	Phase	Module	Topic
1	0	0.1	Math bootcamp — calculus, ODEs, linear algebra
2	0	0.2	Physics — electrostatics, magnetostatics, Maxwell
3	0	0.3	CS-to-EE bridge
4	0	0.4	Toolchain setup
5–6	1	1.1–1.3	DC fundamentals, KCL/KVL, nodal/mesh
7	1	1.4–1.5	Thevenin/Norton, superposition
8–9	1	1.6–1.7	Capacitors/inductors, transients, capstone
10–11	2	2.1–2.2	Phasors, impedance
12	2	2.3	AC power
13	2	2.4	Resonance, Bode
14	2	2.5–2.6	Three-phase, transformers, capstone
15–16	3	3.1–3.2	Diodes, BJTs
17	3	3.3	MOSFETs
18	3	3.4	Op-amps
19	3	3.5	Active filters
20	3	3.6	Oscillators, 555
21	3	3.7	Linear regulators
22	3	3.8	Audio amp capstone
23–24	4	4.1–4.3	Boolean, combinational, sequential
25	4	4.4	Verilog
26	4	4.5	FPGA
27	4	4.6–4.7	MCU, peripherals
28	4	4.8	UART/SPI/I2C
29	4	4.9–4.10	ESP32, capstone
30–31	5	5.1–5.3	CT signals, Fourier
32	5	5.4	Laplace
33	5	5.5	Sampling theorem
34	5	5.6	Z-transform

Week	Phase	Module	Topic
35	5	5.7	DFT, FFT
36	5	5.8	FIR/IIR filters
37	5	5.9–5.10	Adaptive, capstone
38–39	6	6.1–6.3	Feedback, stability, time response
40	6	6.4	Root locus
41	6	6.5	Bode/Nyquist
42	6	6.6	PID
43	6	6.7	State-space
44	6	6.8–6.10	Digital control, kinematics, capstone
45–46	7	7.1–7.4	Power devices, DC-DC, isolated, inverters
47	7	7.5–7.7	Rectifiers/PFC, DC machines, induction
48–49	7	7.8–7.10	BLDC/FOC, drives, capstone
50	8	8.1–8.3	Vector calc, electrostatics, magnetostatics
51	8	8.4–8.6	Maxwell, transmission lines, antennas
52	8	8.7–8.9	RF front-end, modulation, capstone
+4 wk	9	9.1–9.8	VLSI + capstone (extend if needed)
+8 wk	10	10.1–10.10	Power systems + capstone

This calendar fits the core curriculum (Phases 0–8) into 52 weeks at 12 h/week. Phases 9 and 10 extend past week 52 — plan for an extra 12 weeks. If you have more time per week, compress proportionally. If you have less, extend. The order matters more than the timing — do not skip ahead.