Balanced Ternary CMOS Roadmap

Visual guide for a pharmacy background (think: 'drug development pipeline' for hardware)

Phase 1 (Weeks 1-4)

Cell library (SPICE)

Pharma analogy:
Assay development

Phase 2 (Weeks 5-8)

RTL + synthesis flow

Pharma analogy:
Formulation protocol

Phase 3 (Weeks 9-14)

FPGA prototype

Pharma analogy:
Pilot batch

Phase 4 (Weeks 15-22)

CPU build (BTISA)

Pharma analogy:
Scale-up process

Phase 5 (Weeks 23-26)

Benchmarks + publication

Pharma analogy:
Clinical report

Phase 6 (Weeks 27+)

ASIC tapeout + fabrication

Pharma analogy:
Manufacturing

What we are building (plain-English)

- A three-level logic 'alphabet' (-1, 0, +1) that is natural for signed numbers and forces/charges.
- A verified library of ternary logic cells (like validated reagents): inverter variants, MIN/MAX, NAND/NOR, MUX, full adder.
- A SystemVerilog model + tests (fast functional checks) that matches the SPICE cell behavior.
- An FPGA prototype (binary-encoded ternary) to prove algorithms and measure speed/power early.
- A small balanced-ternary CPU (BTISA) + benchmarks + a path to ASIC tapeout (OpenLane/Caravel).

Pharmacy translation

SPICE simulation
in-vitro assay measurements
Monte Carlo corners
batch-to-batch variability testing
Cell characterization (.lib)
QC spec sheet (curves + limits)
FPGA prototype
pilot plant / small-scale manufacturing
ASIC tapeout
final manufacturing release

A) The basic unit: 1 trit

A trit is a ternary digit. Balanced ternary uses:
-1 (negative) 0 (neutral) +1 (positive)

Pharmacy-style intuition:
- antagonist / no-effect / agonist
- negative / baseline / positive signal

-1	0	+1
T_NEG_ONE	T_ZERO	T_POS_ONE

B) Physical encoding on a wire

We implement 3 stable voltage regions (example 1.8 V process):
LOW = 0 V
MID = VDD/2 (approx 0.9 V)
HIGH = VDD (approx 1.8 V)
This is what the SPICE cell library must reliably produce.

C) FPGA encoding

On FPGA we store trits as 2 bits:

00	-> 0
01	-> +1
10	-> -1
11	-> INVALID / error flag

Note: some docs use a different 2-bit ordering. Any mapping is OK if consistent.

Key QA idea: ternary has 6 transitions (more than binary)

When we measure timing/power for cells, we must cover all of these:

0->1 **1->2** **2->1** **1->0** **0->2** **2->0**

- Binary only needs 0->1 and 1->0 timing arcs.
- Ternary needs 6 arcs because the middle level behaves differently.
- This is like checking stability at 3 storage temperatures instead of 2.

Phase 1 - Build the SPICE cell library (Weeks 1-4)

Like assay development: design cells, validate them, then publish a 'spec sheet' (.lib)

Cells to implement (priority order)

- Inverters: STI, PTI, NTI
- Logic gates: TMIN, TMAX, TNAND, TNOR
- Arithmetic: TSUM, TMUX3, BTHA, BTFA
- Later: TDFF, TLATCH (sequential)

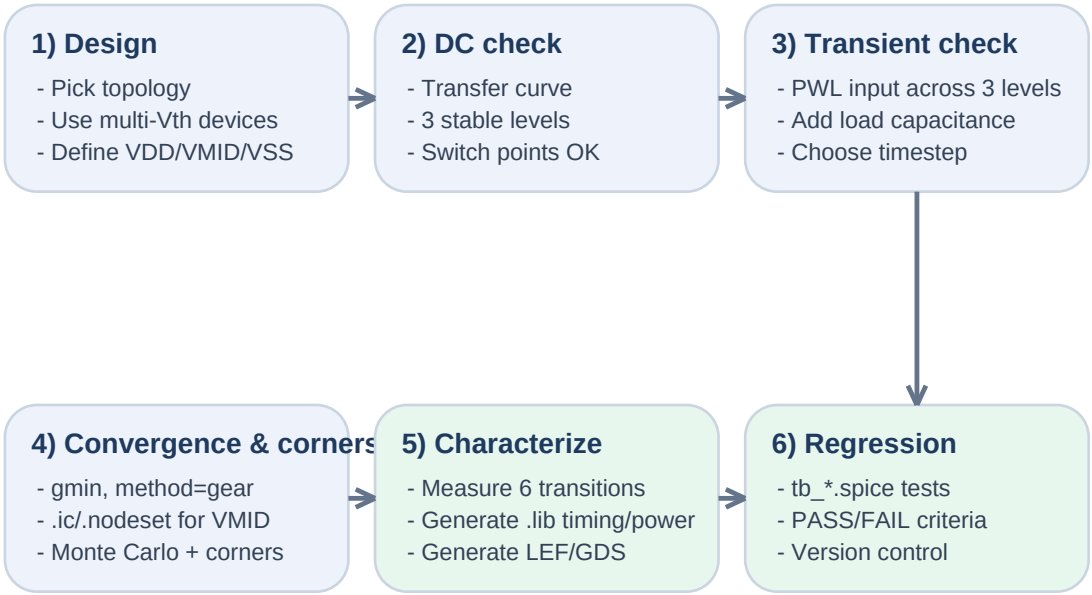
Target transistor counts (from roadmap):

TMIN/TMAX: 10 each TNAND/TNOR: 16 each

TSUM: 20 TMUX3: 24

BTHA: 30 BTFA: 42

Cell-lab workflow (SPICE -> validated cell)



SPICE setup reminders (common failure points)

- Use PWL inputs for all 3 levels + explicit load capacitance (1-50 fF).
- Keep timestep small (<= 1/10 of the smallest transition time).
- If convergence fails: try gmin=1e-12, method=gear, and .ic/.nodeset at VMID nodes.
- Run Monte Carlo + corners to check threshold mismatch tolerance.

Phase 2 - RTL + synthesis flow (Weeks 5-8)

Like writing the protocol: fast functional models + mapping to physical cells

A) RTL 'language' for ternary

We define a SystemVerilog package (ternary_pkg.sv) that:
- Fixes a 2-bit encoding for trits: 00=0, 01=+1, 10=-1, 11=invalid (error flag).
- Provides functions for core ops: t_neg, t_min, t_max, add-with-carry, conversions

Common ternary operators (GT-LOGIC style)

- ~x : ternary inversion (STI)
- x & y: ternary MIN (AND-like)
- x | y: ternary MAX (OR-like)
- x ^ y: ternary XOR / (A+B) mod 3

B) Fast functional simulation (minutes)

Goal: check math & control logic without SPICE (quick regression).
Typical flow:
1) Write RTL: btfa.sv, ternary_adder.sv, ternary_alu.sv
2) Write testbench: tb_btfa.sv
3) Run script: iverilog -> vvp

```
cd hdl/sim
./run_sim.sh

# internally:
iverilog -g2012 -o sim.vvp ...
vvp sim.vvp
```

C) Mapping RTL -> real cells (binary EDA is the constraint)

Because most open-source synthesis is binary, we typically use one of these paths:

- Binary encoding path: represent each trit as 2 bits, synthesize with normal tools, then map to ternary cells after.
- Direct instantiation: instantiate ternary cells directly in RTL so synthesis bypasses ternary portions.
- MRCS (Mixed Radix Circuit Synthesizer): ternary-aware netlist generation (HSPICE/Verilog) for tapeout flows.

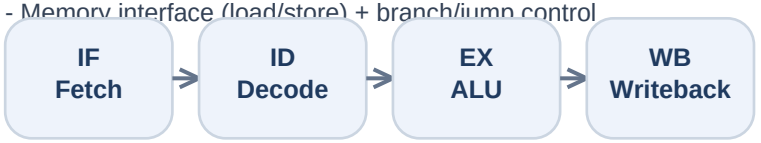
A) FPGA ALU prototype (binary I/O, ternary inside)

Top module: ternary_alu_top.sv
- Inputs/outputs are binary (2 bits per trit).
- Internally converts to trit_t arrays, runs ternary_alu, converts back.

Constraints: ternary_alu.xdc (e.g., 100 MHz clock).
Build: Vivado TCL script -> bitstream.

B) CPU build (BTISA v0.1 style)

Key pieces:
- ISA spec doc (btisa_v01.md) and encoder/decoder RTL
- Register file: 9 registers x 27 trits (ternary_regfile.sv)
- 4-stage pipeline (IF/ID/EX/WB) feeding the ternary ALU
- Memory interface (load/store) + branch/jump control



What to measure on FPGA (like pilot batch release tests)

- Correctness: known-vector tests for ALU ops and BTFA carry behavior.
- Timing: Fmax and slack under constraints; note encoding overhead.
- Area: LUT/FF/BRAM/DSP usage; compare with binary baseline.
- Power: board-level or tool-estimated power; report energy/op.
- Error handling: verify '11' invalid state is caught or sanitized.

Phases 5-6 - Benchmarks, docs, and ASIC tapeout

Turn results into a publishable artifact + a fabrication-ready package

A) Benchmarks + publication package (Weeks 23-26)

Create benchmark_results.md including:
- cell transistor counts and 6-transition timing table
- FPGA utilization, Fmax, throughput, power
- comparison vs binary + other ternary work

Write paper outline + clean README + open-source release.

B) ASIC preparation (Weeks 27+)

OpenLane setup + config.json points to:
- VERILOG_FILES (design)
- LIB_* (your ternary .lib variants)
- EXTRA_LEFS / EXTRA_GDS (your ternary cells)

Integrate with Efabless Caravel template + run precheck.

Definition of Done (DoD) - the 'release criteria' checklist

- All Phase 1 cells pass DC + transient + Monte Carlo + corners.
- Liberty has 6 transition arcs for each cell; LEF/GDS exist for layout.
- RTL regression passes (no invalid '11' leaks unless intended).
- FPGA build is scripted; bitstream reproducible; metrics recorded.
- CPU runs at least 1-2 demo programs (e.g., Fibonacci) end-to-end.
- Benchmark report includes binary baseline and literature comparison.
- OpenLane run completes with no DRC/LVS showstoppers for the demo block.
- Caravel precheck passes; tapeout package is complete.

Fabrication options: Efabless Open MPW (free, open-source) or paid MPW (ChipIgnite).