**1) Domain model (pure logic, no UI)**

**1.1 Bits & Boolean algebra**

* Bit (0|1), Nibble (4 bits).
* GP (generate/propagate pair).
* Gate operations: XOR, AND, OR (pure functions).
* Invariants: !(P\_i===1 && G\_i===1); truth-table assertions.

**1.2 Adder theory**

* PrefixCombine (right ∘ left for GP).
* CarryFormula (C\_i = G\_{i-1:0} ∨ (P\_{i-1:0} · C0)).
* SumFormula (S\_i = P\_i ⊕ C\_i).

**1.3 Adder strategies (pluggable)**

* AdderStrategy (interface).
* BrentKung4 (initial 4-bit CLA + one small combine).
* (Later) BrentKung8, Ripple, CarrySelect (optional).

**2) World, grid, and sectors**

**2.1 Coordinate systems**

* **World space** (authoritative, float).
* **Grid overlay** (rows×cols, cell size).
* Conversions world↔grid (and optional subgrid indices).

**2.2 Sectors**

* Sector with name, cells:Set<CellId>, tags, bbox, centroid.
* Registry: uniqueness, non-overlap checks, serialization.

**2.3 Geometry/obstacles (for future realtime mode)**

* Per-cell geometry: edge block mask, optional interior segments.
* Derived structures for pathfinding (subgrid occupancy or navmesh).

**3) Calculator substrate (the “board” memory)**

**3.1 Cells (single-writer discipline)**

* BitCell (A,B,P,G,C,S).
* PairCell (group GP such as [1:0], [3:0]).
* States: EMPTY → RESERVED → READY.

**3.2 Gate sites (optional)**

* Logical sites mapping to sectors: XOR site, AND site, Combine site.
* Protocol: input availability, atomic write, clear/reset behavior.

**3.3 Resource pools**

* GrainPool (finite count), checked out/in by actions.
* “Out-of-memory” behavior for demonstrations.

**4) Actions and plans (no global tick)**

**4.1 Actions (Command objects)**

* ComputeP(A,B→P)
* ComputeG(A,B→G)
* MaterializeGP(G,P→GP[i])
* CombineGP(RightGP, LeftGP → GP[span])
* CarryFromPrefix(PrefixGP, C0 → C\_i)
* SumFromPC(P\_i, C\_i → S\_i)
* Reserve/Write/Clear primitives (for cell state transitions).

**4.2 Plan graph (DAG)**

* PlanNode with id, dependsOn[], actions[].
* Plan contains nodes; validates acyclicity.

**4.3 Strategy → Plan mapping**

* Brent–Kung 4-bit:
  + Leaves: PG0..PG3 (and GP0..GP3).
  + Level-1: [1:0], [3:2].
  + Level-2: [3:0].
  + Small combine: [2:0].
  + Carries: C1..C4 (independent once prefixes exist).
  + Sums: S0..S3.

**5) Execution engines**

**5.1 VirtualExecutor (MVP)**

* Zero travel time; executes any node once deps are satisfied.
* Enforces single-writer, state transitions, and invariants.
* Emits a deterministic **trace** (event log) for later UI playback.

**5.2 RealtimeExecutor (phase 2)**

* Ant agents with positions, velocities, and steering.
* Pathing via subgrid or navmesh; walls/obstacles respected.
* Contention (queues), tool-use durations, travel timings.
* Uses the same Actions; only the scheduling/timing differs.

**6) Ant agents (for realtime; stub in MVP)**

**6.1 Ant model**

* Ant with id, worldPos, state (Idle/Move/Act), role (Worker/Scout/GC).
* Blackboard or task queue receiving PlanNode work items.

**6.2 Policies**

* Reservation protocol for target cells.
* Retry/timeout rules; backoff when blocked.

*(In virtual mode we bypass movement but still produce the same action trace.)*

**7) Orchestration & API**

**7.1 Orchestrator**

* Accepts decimal inputs; encodes to A\_i, B\_i.
* Selects strategy (BrentKung4).
* Builds plan; hands off to executor; returns results and trace.

**7.2 Validation hooks**

* Local carry identity: C\_{i+1} = G\_i ∨ (P\_i · C\_i).
* Prefix identity for each i.
* Final numeric check: (C4 S3 S2 S1 S0)\_2 == A+B (mod 2^5).

**7.3 Serialization**

* Save/restore board state, plan, and trace for replay.

**8) Telemetry and debugging**

**8.1 Trace schema**

* Event types: Reserve, Compute, Write, Combine, Carry, Sum, Clear.
* Fields: timestamps, nodeId, actor (ant or “virtual”), sector, inputs, outputs.

**8.2 Metrics**

* Counts: actions, grains borrowed/returned, cells written.
* For realtime: distance traveled, wait time, utilization.

**9) Tests (Vitest)**

**9.1 Unit**

* Gates (XOR/AND/OR), combine operator.
* Carry/sum formulas.
* Invariants and cell state machine.

**9.2 Integration**

* Full 4-bit run: multiple input pairs, including edge cases (0+0, 15+15).
* Intentional “out-of-memory” to verify failure mode.

**9.3 Determinism**

* VirtualExecutor produces identical trace across runs.

**10) Configuration**

* tsconfig strict mode (already).
* App config: nBits=4, C0=0, grid size, cell size, grain count.
* Feature flags: virtual=true, logLevel, assertions.

**11) Deliverables / milestones**

**M1 – Core math ready**

* Domain functions + tests.

**M2 – Board substrate**

* Cells, state machine, resource pool + tests.

**M3 – Planning & Virtual execution**

* Brent–Kung4 plan; VirtualExecutor; full run from inputs to sums; deterministic trace.

**M4 – Grid & sectors (UI-agnostic)**

* Sector registry; mapping of logical components to sectors; validation.

**M5 – Developer UX**

* CLI or small script to run pairs and dump traces; golden-file tests.

**M6 – Optional realtime groundwork**

* World space types; ant scaffold; geometry placeholders.

**12) Naming & conventions**

* Indices: LSB = bit 0; “right = higher index” in combines.
* Prefix ids: PFX[hi:lo]; bit cells: A0,B0,P0,G0,C1,S0 etc.
* Sector tags: "grain", "pg-row", "prefix-l1", "carry-track", "sum-row".

This outline is all you need to start creating tickets and folders without writing code yet. When you say “go,” I will turn **M1–M3** into minimal class files and tests that compile and run in your current repo, keeping the structure above.