using matrix method karnaughs graph crete combinotorial circuits like addition and subraction dmultiplication in logisim give table form procedures

Below is a **clear, step-by-step guide** on how to design **combinational circuits (Adder, Subtractor, Multiplier)** in **Logisim** using:

✔ **Matrix method** (truth table → Boolean matrix → simplification)
✔ **Karnaugh map (K-Map)**
✔ **Truth tables**
✔ **Final Logisim implementation steps**

I give everything in **table form** with **procedures**.

✅ 1. HALF ADDER – Matrix, K-Map, Circuit

Truth Table (Matrix Form)

A	B	Σ (Sum)	C (Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Boolean Matrix Representation

For **Sum**:

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

Matrix → XOR

For **Carry**:

$$C = AB$$

K-Map for SUM

A\B	0	1
0	0	1
1	1	0

Simplified:

$$\Sigma = A \oplus B$$

K-Map for CARRY

A\B	0	1
0	0	0
1	0	1

Simplified:

$$C = AB$$

Logisim Procedure

1. Open Logisim → Add "XOR" gate → inputs A, B → output → SUM
2. Add "AND" gate → inputs A, B → output → CARRY
3. Make inputs visible.

✅ 2. FULL ADDER – Matrix, K-Map, Circuit

Truth Table

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Matrix Boolean Forms

```
Sum = A ⊕ B ⊕ Cin
Cout = AB + BCin + ACin
```

Logisim Procedure

1. Use **two Half Adders**
2. Σ1 = A ⊕ B
3. Σ = Σ1 ⊕ Cin
4. Carry1 = A AND B
5. Carry2 = Σ1 AND Cin
6. Cout = Carry1 OR Carry2

✅ 3. BINARY SUBTRACTOR (A – B)

Truth Table (Half Subtractor)

A	B	D (Diff)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Matrix / Boolean Expression

```
Diff = A ⊕ B
Borrow = $\bar{A}$B
```

Logisim Procedure

Logisim Procedure

1. Place input A, B
2. XOR → Diff output
3. NOT A → AND with B → Borrow output

✅ 4. MULTIPLICATION (2-bit example)

Inputs:

- A1 A0
- B1 B0

Output bits:

- P3 P2 P1 P0

Truth Table (partial)

A1 A0	B1 B0	P3 P2 P1 P0
00	00	0000
01	01	0001
01	10	0010
10	01	0010
10	10	0100
11	11	1001

Boolean Derivation (Matrix)

```
P0 = A0B0
P1 = (A1B0) ⊕ (A0B1)
P2 = A1B1 ⊕ (A1B0 AND A0B1)
P3 = A1B1 AND (A1B0 AND A0B1)
```

Logisim Procedure

1. Use AND gates for partial products
    - A0B0
    - A1B0

- A1B0
- A0B1
- A1B1

2. Use Half Adders / Full Adders to combine them
3. Output P3–P0

📌 GENERAL DESIGN PROCEDURE (ALL CIRCUITS)

Step	Action
1	Write Truth Table
2	Convert to Boolean Matrix (minterms)
3	Draw K-Map for each output
4	Simplify expressions
5	Convert to logic gates
6	Draw in Logisim
7	Test using “Poke Tool”

✅ If you want, I can generate:

✔ **Matrix + K-Map + Logisim circuit** for

- 3-bit adder
- 4-bit subtractor
- 4×4 multiplier
- Comparator
- Decoder
- Priority encoder

✔ I can export **Logisim .circ file** also.

Tell me which circuit you want next.

i need for 8 bit

Nice — let's build 8-bit combinational circuits for addition, subtraction and multiplication in **Logisim**. I'll give you clear, copy-pasteable procedures, wiring diagrams (text form), per-bit equations, test vectors, and optimization options (ripple vs. CLA, array vs. Wallace). No waiting — everything you need right now.

1) 8-bit ADDER — (Result: 8-bit sum + carry out)

Concept (per bit)

For bit index `i = 0..7` (0 = LSB):

- `Sum_i = A_i ⊕ B_i ⊕ C_i`
- `C_{i+1} = A_iB_i + B_iC_i + A_iC_i` (equivalently `G_i + P_i·C_i`, where `G_i = A_i·B_i`, `P_i = A_i ⊕ B_i`)

Implementations

A — Ripple-Carry (simple, easy in Logisim)

- Use 8 **Full Adder** blocks chained: Cin (initial) → FA0 → FA1 … → FA7 → Cout.
- In Logisim: create a 1-bit FA using two XORs, two ANDs, one OR, or use built-in components if available.
- Use **splitters** for 8-bit buses:
  - Create inputs `A[7..0]`, `B[7..0]`.
  - Wire `A[i]`, `B[i]` to FA_i.
  - `Cin` for full 8-bit addition set to 0 (unless you want add with carry).
- Outputs: `S[7..0]`, final `Cout`.

Steps (Logisim)

1. Place two 8-bit input pins (or two 8-bit splitters) named `A` and `B`.
2. Place eight 1-bit full-adder circuits in a row.
3. Use tunnels or short wires: connect `A[i]`, `B[i]` from splitter to each FA.
4. Set initial `Cin = 0` (constant 0) or input `Cin`.
5. Chain carry lines from `FA_i` Cout to `FA_{i+1}` Cin.
6. Place 8-bit output splitter for `S[7..0]` and one pin for `Cout`.

B — Carry Lookahead Adder (faster, more gates)

- Compute per-bit `G i = A i·B i`, `P i = A i ⊕ B i`.

- Compute carries in blocks: for 8 bits you can implement two 4-bit CLA blocks or full 8-bit CLA.
- Group carry formula for 4-bit block (bits 0–3):

  `C1 = G0 + P0·C0`

  `C2 = G1 + P1·G0 + P1·P0·C0`

  `C3 = G2 + P2·G1 + P2·P1·G0 + P2·P1·P0·C0`

  `C4 = G3 + P3·G2 + P3·P2·G1 + P3·P2·P1·G0 + P3·P2·P1·P0·C0`
- Use these to get `C4` to feed next block. This reduces propagation delay in hardware; in Logisim it's a matter of more gates.

Test vectors (quick)

- `A = 0x7F (01111111), B = 0x01 (00000001) → S = 10000000 (0x80), Cout = 0`
- `A = 0xFF, B = 0x01 → S = 00000000, Cout = 1`

---

2) 8-bit SUBTRACTOR (A – B) — (Result: 8-bit difference + borrow/cout)

Method 1 — Two's complement adder (recommended)

`A − B = A + (two's-complement of B) = A + (~B + 1)`

Implementation (Logisim)

- Invert B: put 8 XOR gates each with `B_i` and control bit `INV` = 1 to get `B' = ~B` (or use NOT on 8-bit bus).
- Set `Cin = 1` to add the +1 for two's complement.
- Use the same 8-bit adder as above: `A + B' + 1`.
- Final `Cout` = indicates no borrow if `Cout = 1` (for unsigned subtraction) — interpret properly per signed/unsigned.

Steps

1. Place `B` splitter → 8 XORs with second input = `SUB` control (1 for subtract).
2. If `SUB=1`, XOR flips B; set adder `Cin = SUB`.
3. Use 8-bit adder to compute `D = A + B' + SUB`.
4. `D[7..0]` is result; `Cout` can be used as borrow flag (unsigned).

Method 2 — Borrow-ripple subtractor (per bit)

- Per bit: `Diff_i = A_i ⊕ B_i ⊕ Borrow_i`

  `Borrow_{i+1} = (~A_i & B_i) | (Borrow_i & (~A_i ⊕ B_i))`
- More complex; two's complement is simpler and standard.

Test vectors

- `A=00001010 (10)`, `B=00000101 (5)`, SUB=1 → Result `00000101 (5)`, Cout=1 (no borrow).
- `A=00000001`, `B=00000010` → Result `11111111` (if unsigned, Cout=0 indicates borrow).

---

3) 8×8 MULTIPLIER — (Result: 16-bit product)

Two common approaches

A — Combinational array (shift-add / schoolbook) — simple to implement in Logisim

- Generate 8 partial products: `P_row[i] = B[i] & A[7..0]` (each partial product is an 8-bit vector ANDed with B_i).
- Shift each partial product left by i bits.
- Sum all partial products (8 values) — use a tree of adders (pairwise additions) or cascade ripple adders.
- Output is 16 bits `Product[15..0]`.

Wiring pattern (text diagram)

Partial products (example):

```
pp0 = A[7..0] & B0  -> place from Product[7..0]
pp1 = A[7..0] & B1  -> shifted left 1 -> add to Product bits [8..1]
pp2 = A[7..0] & B2  -> shifted left 2 -> add to Product bits [9..2]
...
pp7 = A[7..0] & B7  -> shifted left 7 -> add to Product bits [15..7]
```

- Use 8 AND arrays (each A bit AND with B_i).
- Add with several 8/16-bit adders (you'll need intermediate widths; best to use 16-bit adders for partial sums).

Implementation in Logisim (practical)

1. Create A[7..0] and B[7..0] inputs.
2. For i=0..7: create eight 8-bit AND blocks (use bitwise AND by wiring A bus to one

input and single bit B_i to the other via a splitter).

3. Use splitters to shift (connect pp_i to bits [i..i+7] of a 16-bit bus).
4. Add them pairwise using 16-bit adders (start with pp0 as 16-bit value with zeros in high bits).
5. You can do a tree (pp0+pp1) -> s01, (pp2+pp3) -> s23, then s01 + s23 -> s0123, etc., finishing with 3 or 4 stages.

Resource note: this is gate-heavy but straightforward.

B — Wallace tree / carry-save / multiplier optimizations

- Faster in logic depth, but more complex to implement manually in Logisim (requires carry-save adders). Only attempt if you want a performance-oriented design.

Test vectors

- `A = 0xFF (255), B = 0xFF (255)` → `Product = 65025 (0xFE01)` → binary `11111110 00000001`.
- `A = 0x01, B = 0x80` → `Product = 0x80`.

---

4) LOGISIM BUILDING BLOCKS & PRACTICAL TIPS

Useful components

- **Splitters**: use to split 8-bit buses into single bits and to assemble outputs.
  - Example: splitter with bit width 8, outputs 8.
- **Tunnels**: keep diagram tidy.
- **Subcircuits**: build 1-bit FA, then make subcircuit and place it 8 times.
- **Bitwise gates**: Logisim supports n-bit AND/XOR which simplifies partial product generation.
- **Constants**: for Cin (0 or 1), SUB control signal.

Naming conventions

- Use labels: `A[7]..A[0]`, `B[7]..B[0]`, `S[7]..S[0]`, `P[15]..P[0]`.
- Color wires or group with tunnels to prevent confusion.

Testing

- Use Poke tool and try:
  - edge cases: all zeros, all ones.
  - random vectors: compare with expected decimal/hex result.

- For multiplier, test extremes: `0x00 * anything`, `0xFF * 0xFF`, `0x80 * 0x02`, etc.

---

5) PERFORMANCE & GATE ESTIMATES (rough)

- 8-bit ripple adder: 8 full adders → each FA ≈ 2 XOR, 2 AND, 1 OR → roughly 8×(5 gates) = 40 gates (logic count varies by model).
- 8×8 array multiplier: 64 AND cells + many adders. Expect several hundred gates.

---

6) Example: Step-by-step Logisim walk-through to make an 8-bit ripple adder (short)

1. New project. Place two `Input` components; set bit width = 8 (right-click → data bits = 8). Label them `A`, `B`.
2. Place constant or input `Cin` (1-bit).
3. Build a `1-bit FullAdder` subcircuit:
   - Three inputs `A`, `B`, `Cin`; outputs `Sum`, `Cout`.
   - Internals: XOR1 = A ⊕ B, Sum = XOR1 ⊕ Cin; Carry1 = A·B; Carry2 = XOR1·Cin; Cout = Carry1 + Carry2.
4. Back in main circuit, place 8 instances of `FullAdder`.
5. Use an 8-bit splitter to route `A` and `B` bits to each FA input.
6. Connect carries: Cin → FA0.Cin, FA0.Cout → FA1.Cin, …, FA7.Cout → `Cout`.
7. Gather Sum outputs into an 8-bit output splitter labeled `S`.
8. Simulate and test.

---

7) Want me to generate…

- an explicit Logisim `.circ` file (I can create the file contents for you), or
- a downloadable step-by-step diagram (canvas/code),
- or the Verilog/VHDL for the 8-bit adder/subtractor/multiplier?

Tell me which format you prefer and I'll produce it **right now** (circuit file / Verilog / step-by-step Logisim XML).

VokoscreenNG

VokoscreenNG is a **free, open-source screen recorder** for Linux and Windows.
Below is a **simple, complete guide** to installing and using it.

✅ 1. What is VokoscreenNG?

It is a **screen recording software** used for:

- Recording full screen
- Recording a specific window or area
- Recording webcam + screen
- Audio capture (mic + system)
- Saving in MKV/MP4/WEBM formats
- Recording cursor, timer, countdown, etc.

Perfect for tutorials, coding videos, gameplay, online classes, etc.

✅ 2. Install VokoscreenNG

Linux (Ubuntu / Debian / Mint)

Open Terminal:

```
sudo apt update
sudo apt install vokoscreen-ng
```

Fedora

```
sudo dnf install vokoscreenNG
```

Arch / Manjaro

```
sudo pacman -S vokoscreenNG
```

Windows

Download from official site:
https://linuxecke.volkoh.de/vokoscreen/vokoscreen.html
(Click **Windows Installer**)

✅ 3. How to Use VokoscreenNG

Start

Open:

Applications → Multimedia → VokoscreenNG

🎥 4. Main Tabs Explained

A) Screen Tab (Video Settings)

Choose recording mode:

- **Full Screen**
- **Window**
- **Area Selection** (drag selection)
- **Magnification window**
- **Follow cursor** mode

Resolution adjustment:

- FPS = 25/30/60
- Format = mkv, mp4, webm
- Video Codec = h264, h265, VP8, VP9

B) Audio Tab

Enable audio:

- Microphone
- System sound ("Monitor of ...")
- Mix two audio sources if needed.

C) Camera Tab

Overlay webcam video on screen:

- Enable webcam
- Size adjustment

- Position selection

D) Effects Tab

Options:

- Show mouse cursor
- Mouse click highlight
- Countdown before recording (3–10 seconds)
- Follow-cursor zoom

▶️ 5. Start / Stop Recording

Start recording:

Click the **blue Start button**.

Stop recording:

Press the **red Stop button**
or use keyboard shortcut:

```
Ctrl + Shift + F12
```

The video is saved automatically in:

```
~/Videos/vokoscreenNG/
```

🎬 6. Output Settings

Choose:

- **File format:** mp4, mkv, webm
- **Frame rate:** 24–60 FPS
- **Codec:** H.264 recommended for best compatibility
- **Quality:** set % (50–100%)

💡 Tips for best recording

✔ Use MP4 + H.264 for YouTube and easy sharing
✔ Use 30 FPS for normal screen, 60 FPS for gameplay
✔ Use Area-recording for tutorials (smaller file size)
✔ Make sure audio **Monitor of Built-in Audio** is selected for system sound

---

📌 If you want, I can provide:

✔ Best settings for **coding videos**
✔ Best settings for **gaming**
✔ Best settings for **low-end laptop**
✔ Complete tutorial with screenshots
✔ How to record camera + screen in VokoscreenNG
✔ How to fix no-sound issue

Just tell me what you want!