Perfect — you're now at **step 8 of `parse_spd`**, where the parser figures out whether each via is on the **top cavity (1)** or **bottom cavity (0)**. Let's document `_fill_via_locs`.

📘 Documentation

`_fill_via_locs(brd, node_info)`

Purpose

Determine whether each **IC via** and **decap via** is connected to the **top** or **bottom** cavity of the board, based on its layer number.

Outputs:

- `brd.ic_via_loc` → array of `1` (top) or `0` (bottom) for IC vias.
- `brd.decap_via_loc` → same for decap vias.

Rules

- **Top cavity (1):** node is on the **minimum layer** in the board.
- **Bottom cavity (0):** node is on the **maximum layer**.
- **Neither top nor bottom:** default to **top (1)** (matches main.py behavior).
- If `node_info` is missing: default **all vias to top (1)**.

How it works

1. **Empty checks**

```
pythonCopy codeif not getattr(brd, "ic_node_names", None):
    brd.ic_via_loc = np.array([], dtype=int)
if not getattr(brd, "decap_node_names", None):
    brd.decap_via_loc = np.array([], dtype=int)
```

- Ensures we don't crash if there are no IC or decap vias.

---

2. **Fallback if node_info missing**

```
pythonCopy codeif not node_info:
    brd.ic_via_loc = np.ones(len(brd.ic_via_xy), dtype=int)
    brd.decap_via_loc = np.ones(len(brd.decap_via_xy), dtype=int)
    return
```

- If node metadata wasn’t parsed → mark everything as top.

---

3. **Find global top/bottom layers**

```
pythonCopy codelayers = [v["layer"] for v in node_info.values()]
top_layer = min(layers)
bot_layer = max(layers)
```

- Determines smallest and largest layer indices from all nodes.

---

4. **Assign IC via locs**

```
pythonCopy codeic_locs = []
for n in brd.ic_node_names:
    lyr = node_info.get(n, {}).get("layer", top_layer)
    ic_locs.append(1 if lyr == top_layer else (0 if lyr == bot_layer
else 1))
brd.ic_via_loc = np.array(ic_locs, dtype=int)
```

- Checks each IC via’s node layer:
    - If layer = top_layer → 1.
    - If layer = bot_layer → 0.
    - Else → 1 (treated as top).

---

5. **Assign decap via locs**

```
pythonCopy codedec_locs = []
for n in brd.decap_node_names:
    lyr = node_info.get(n, {}).get("layer", top_layer)
    dec_locs.append(1 if lyr == top_layer else (0 if lyr == bot_layer
else 1))
brd.decap_via_loc = np.array(dec_locs, dtype=int)
```

Example

SPD nodes:

```
makefileCopy codeNode10::GND  Layer=Signal1   (top layer)
Node11::GND  Layer=Signal5   (bottom layer)
Node12::PWR  Layer=Signal3   (middle)
```

IC connections:

```
bashCopy code.Connect ic_port1
1 $Package.Node12
2 $Package.Node10
.EndC
```

Decap connections:

```
bashCopy code.Connect decap_port1
1 $Package.Node11
2 $Package.Node10
.EndC
```

Parsed results:

```
pythonCopy codebrd.ic_via_loc    = [1, 1]   # Node12 (middle → treated
as top), Node10 (top)
brd.decap_via_loc = [0, 1]   # Node11 (bottom), Node10 (top)
```

Why it matters

- These arrays tell the solver whether each via connects into the **top cavity (1)** or **bottom cavity (0)**.
- This distinction is used in the **node-voltage method** to assemble the impedance matrix correctly, since IC ports are often modeled as entering from the top while decaps may connect from bottom.

 Summary:
`_fill_via_locs` sets the “location flag” for each IC and decap via: top (1) or bottom (0), with middle layers defaulting to top. This ensures the PDN solver knows whether a via belongs to the top or bottom cavity of the board.

 Do you also want me to extend the logging here (similar to what we did for buried vias) so that it prints both **IC locs and decap locs** clearly, instead of just IC?