

1 / 43

100

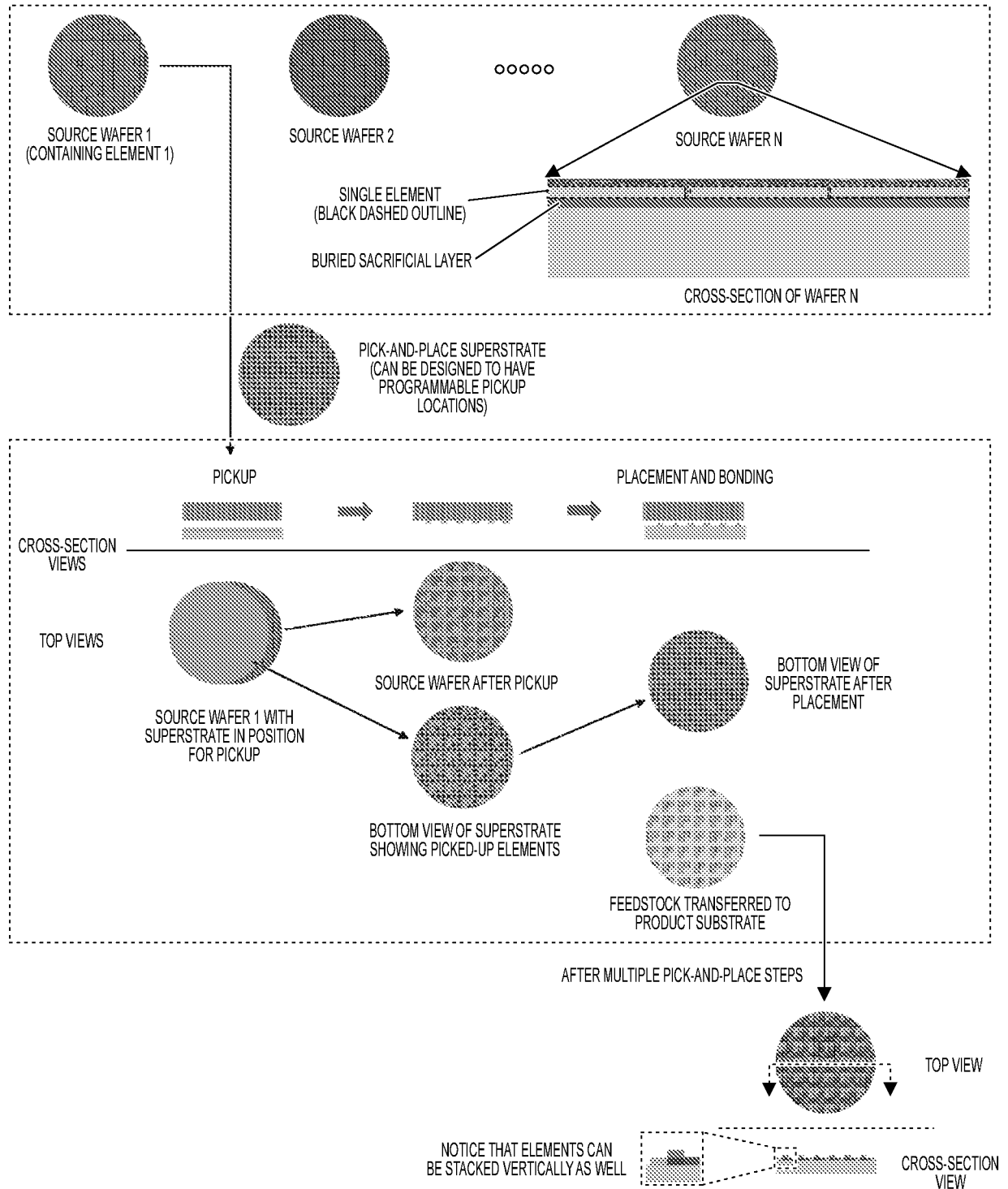
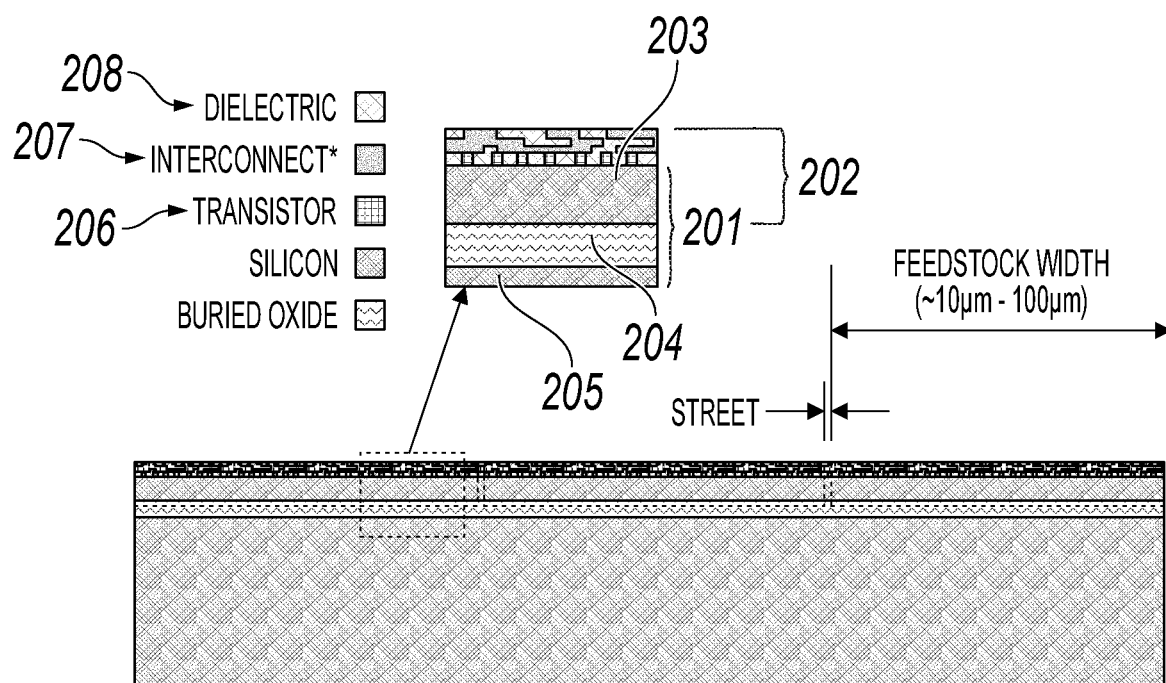


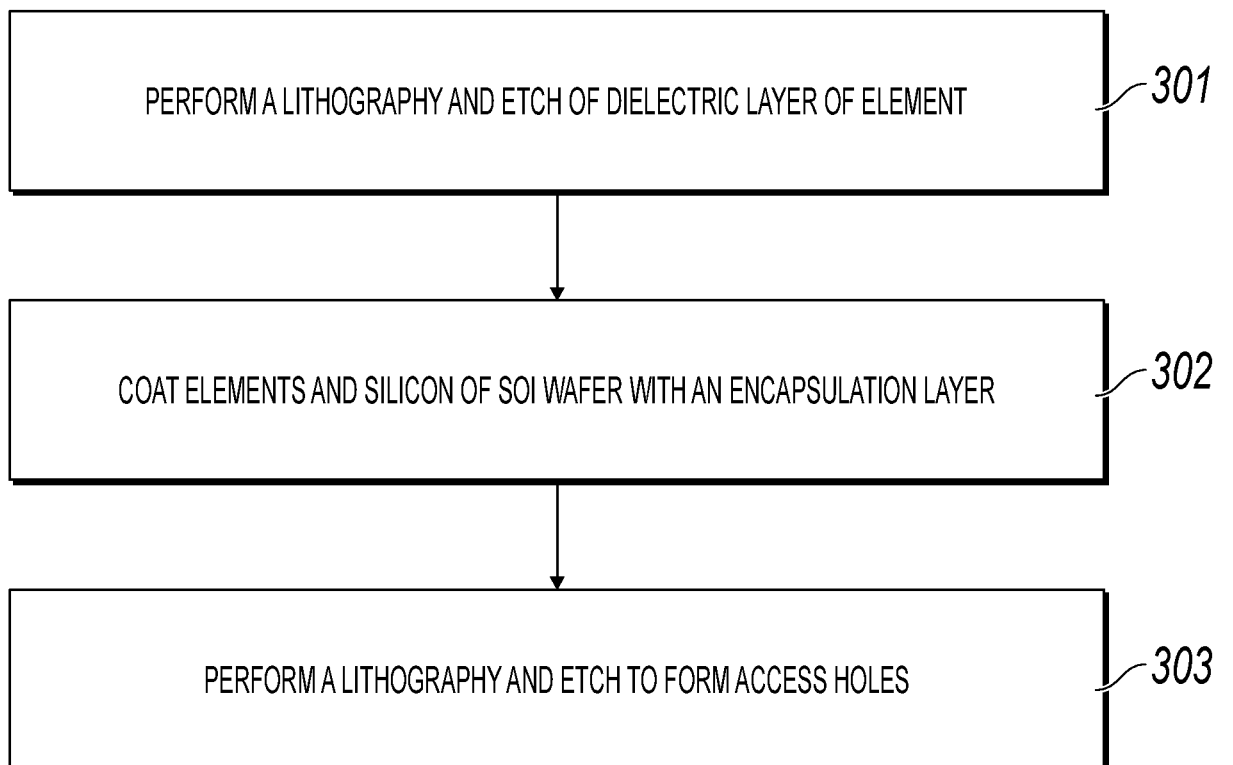
FIG. 1

2 / 43

**FIG. 2**

3 / 43

300

**FIG. 3**

4 / 43

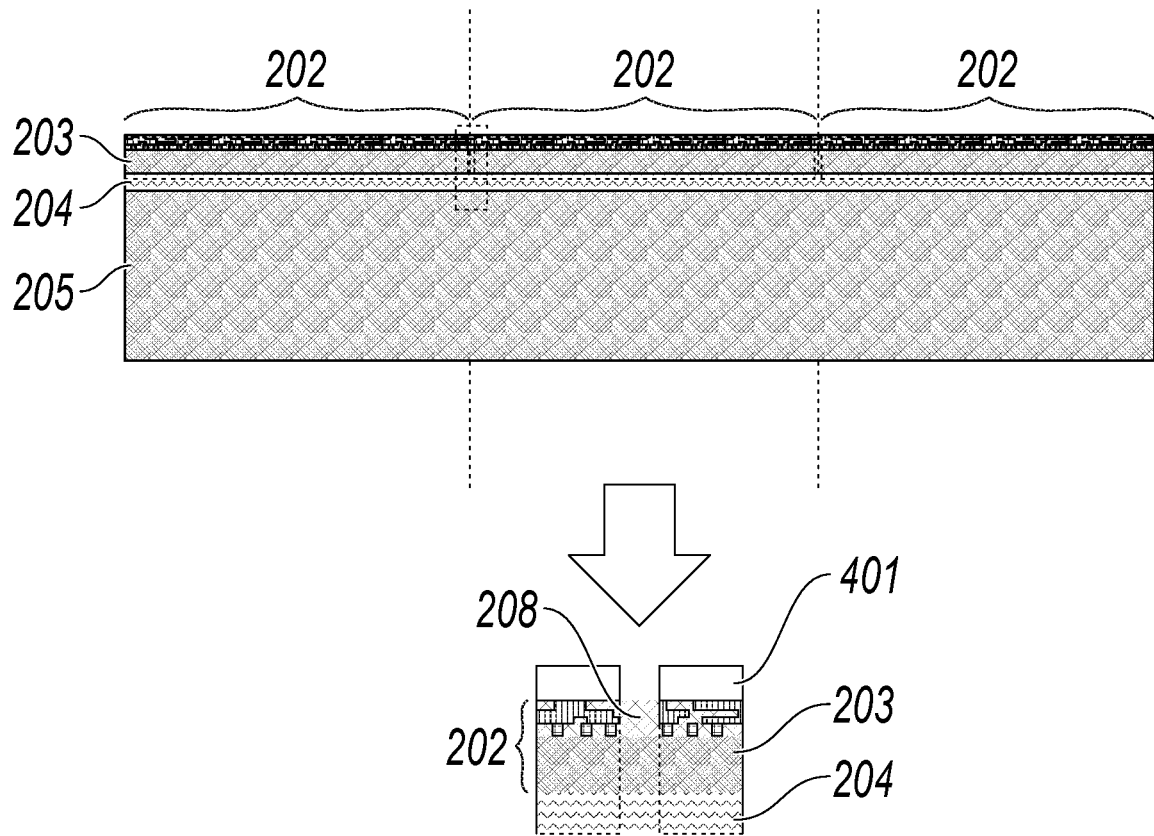
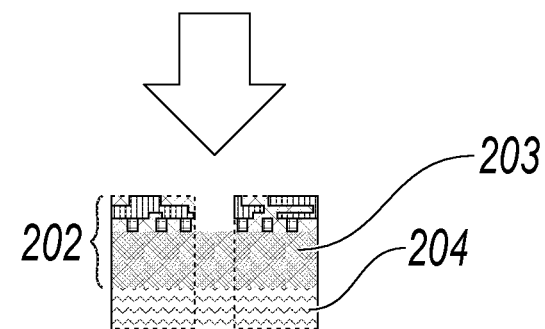


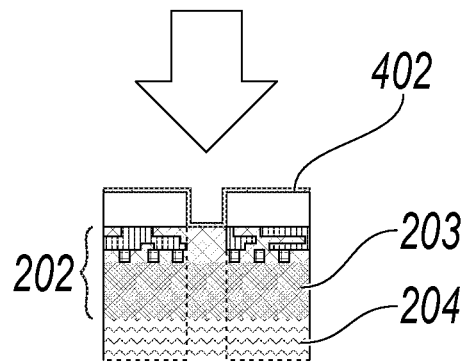
FIG. 4A



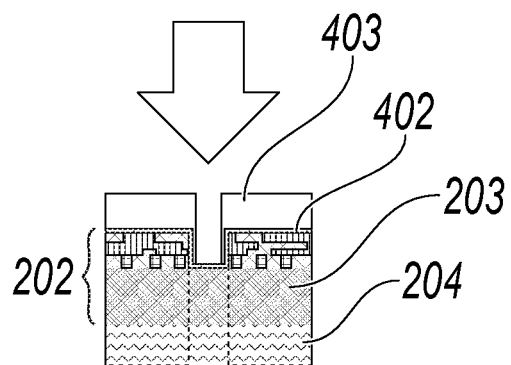
LITHO + ETCH OF DIELECTRIC LAYER

FIG. 4B

5 / 43



CONFORMALLY COATED ENCAPSULATION LAYER

FIG. 4C

LITHO + ETCH OF ACCESS HOLES

FIG. 4D

6 / 43

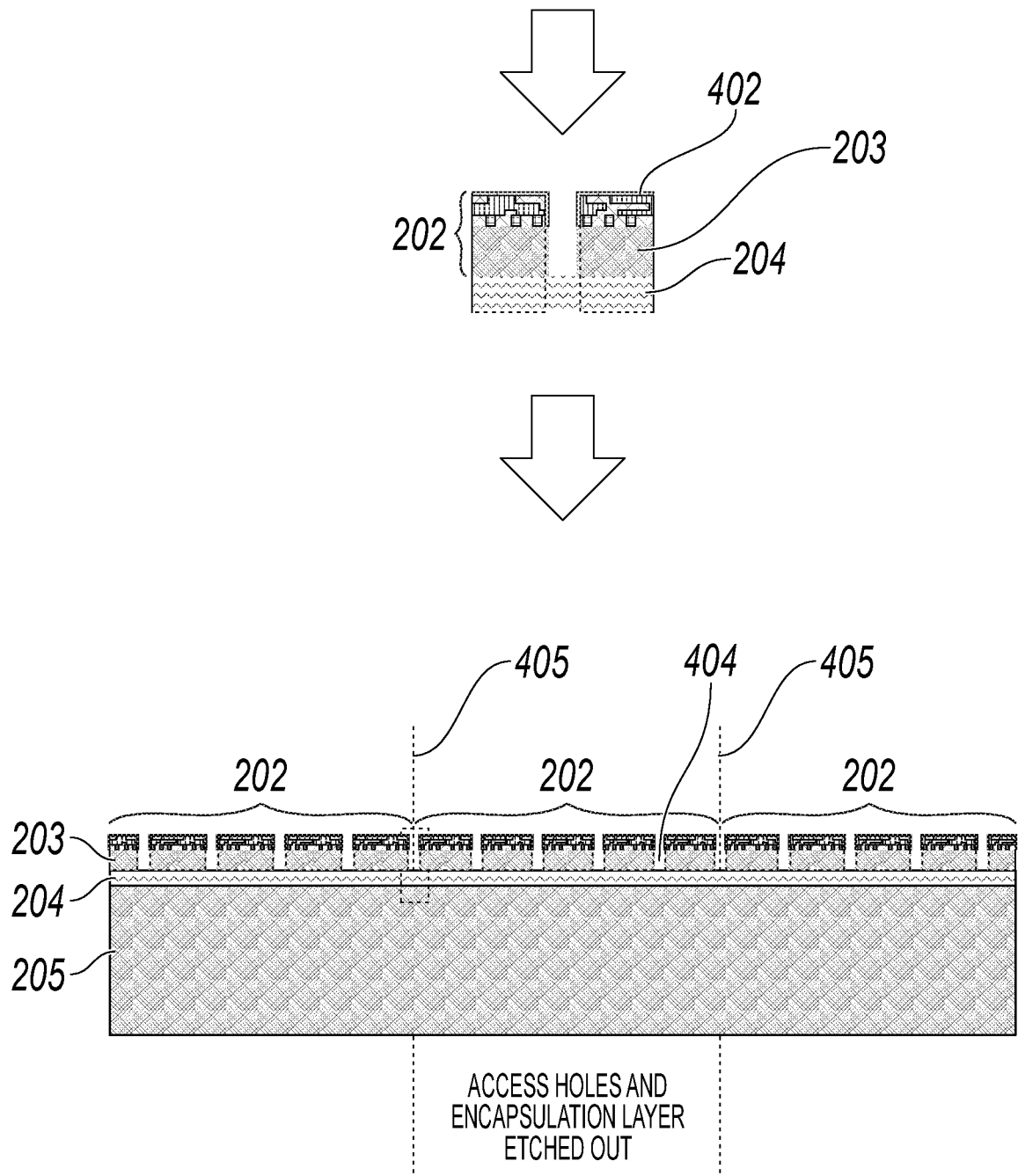


FIG. 4E

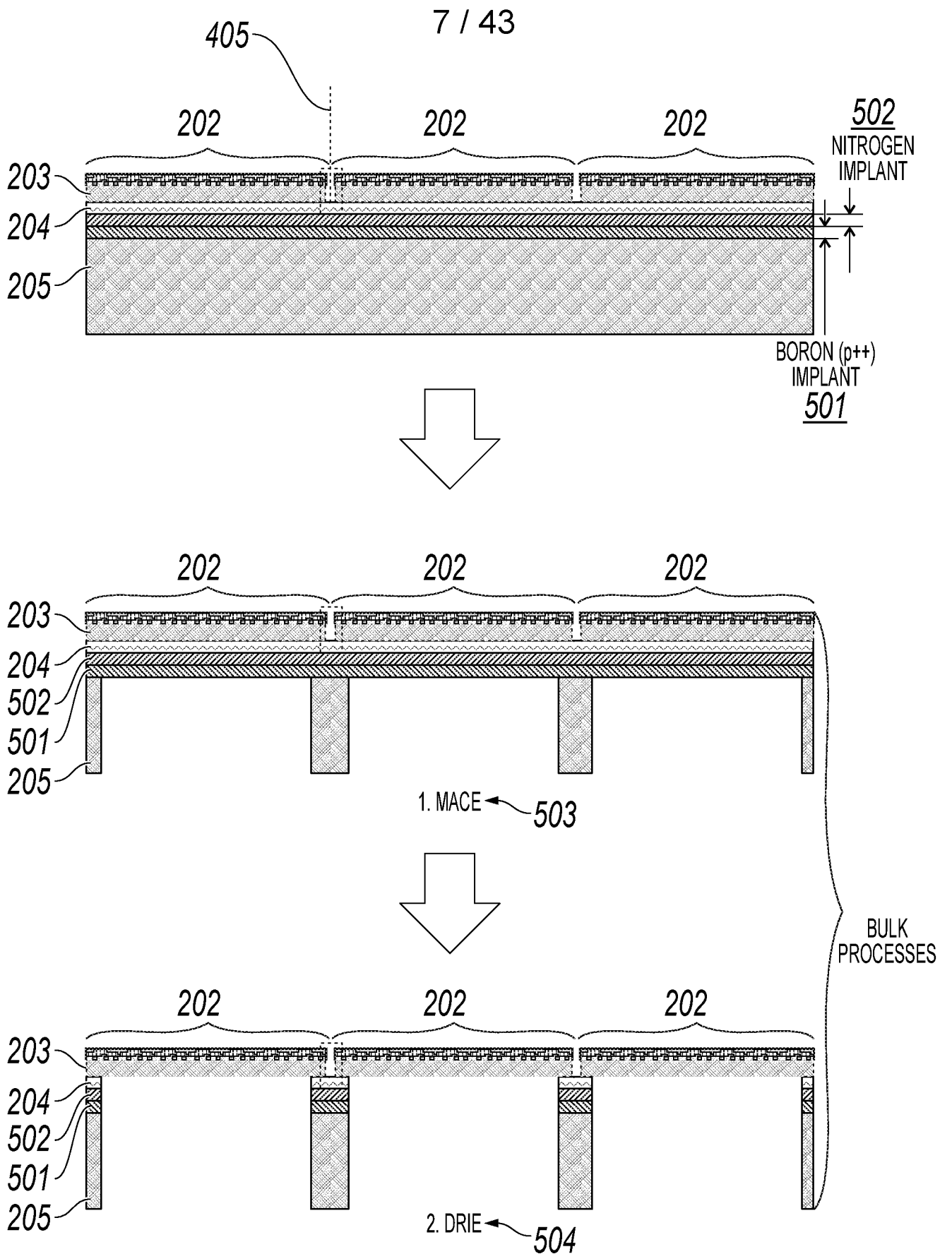
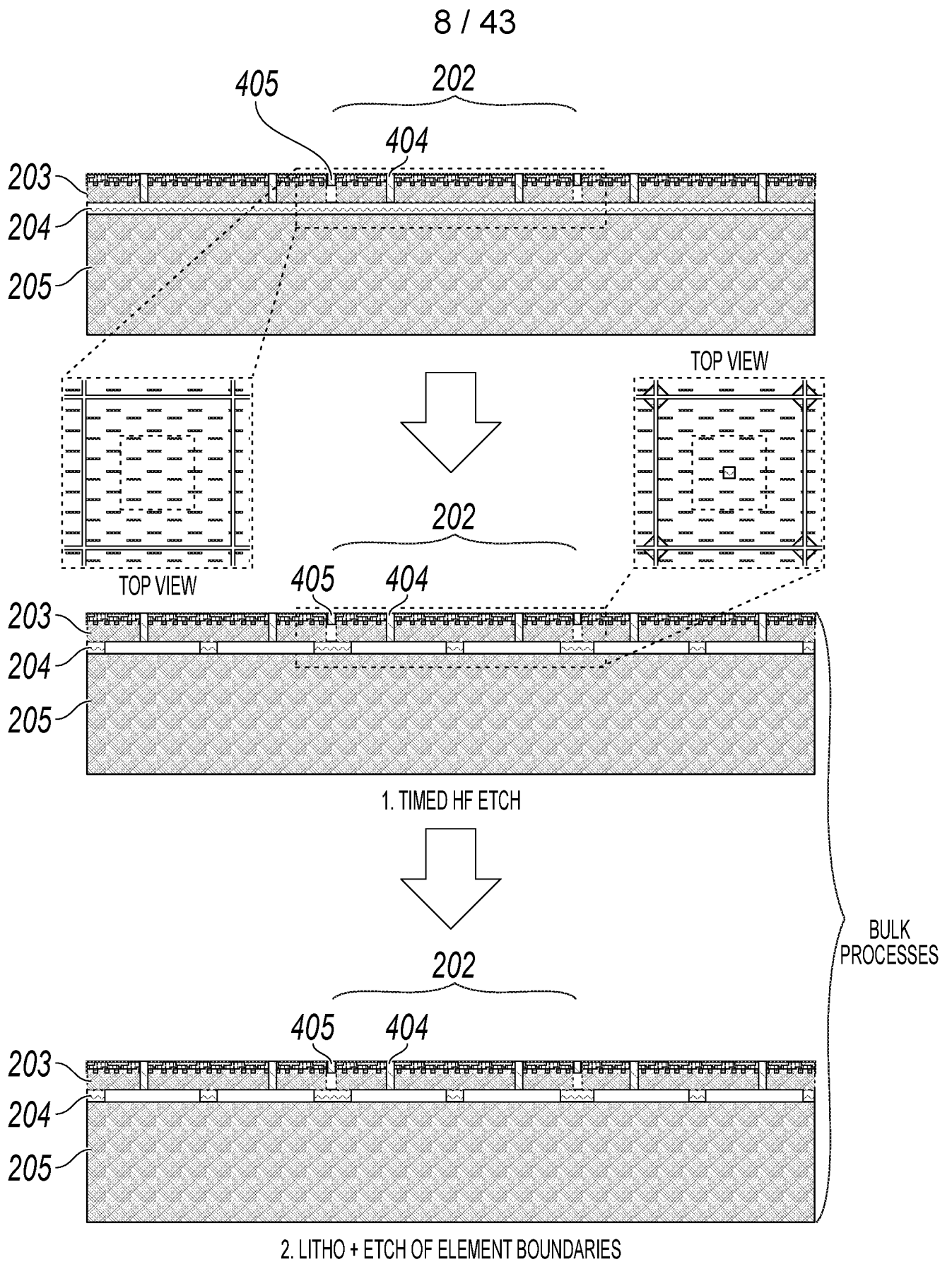
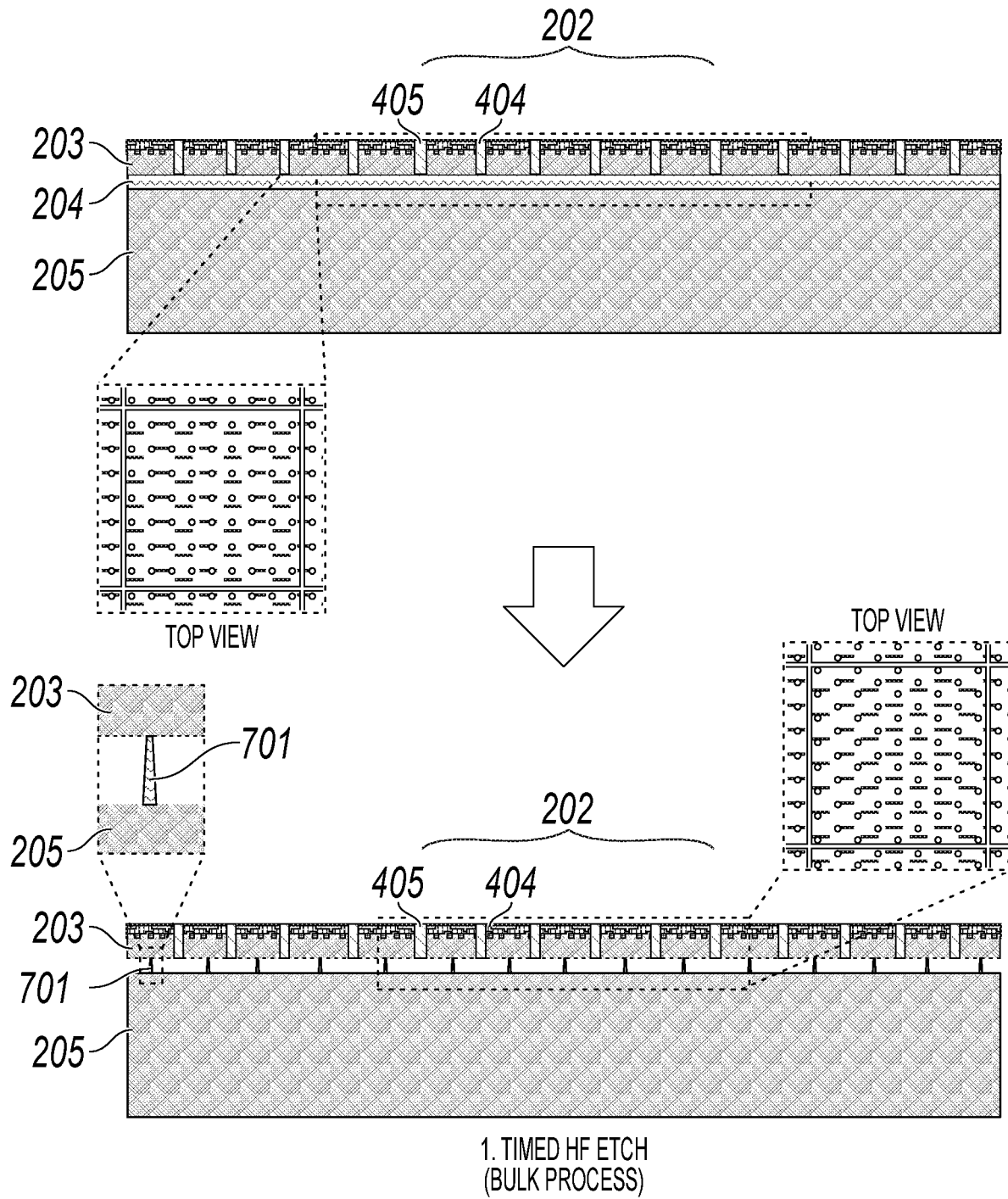


FIG. 5

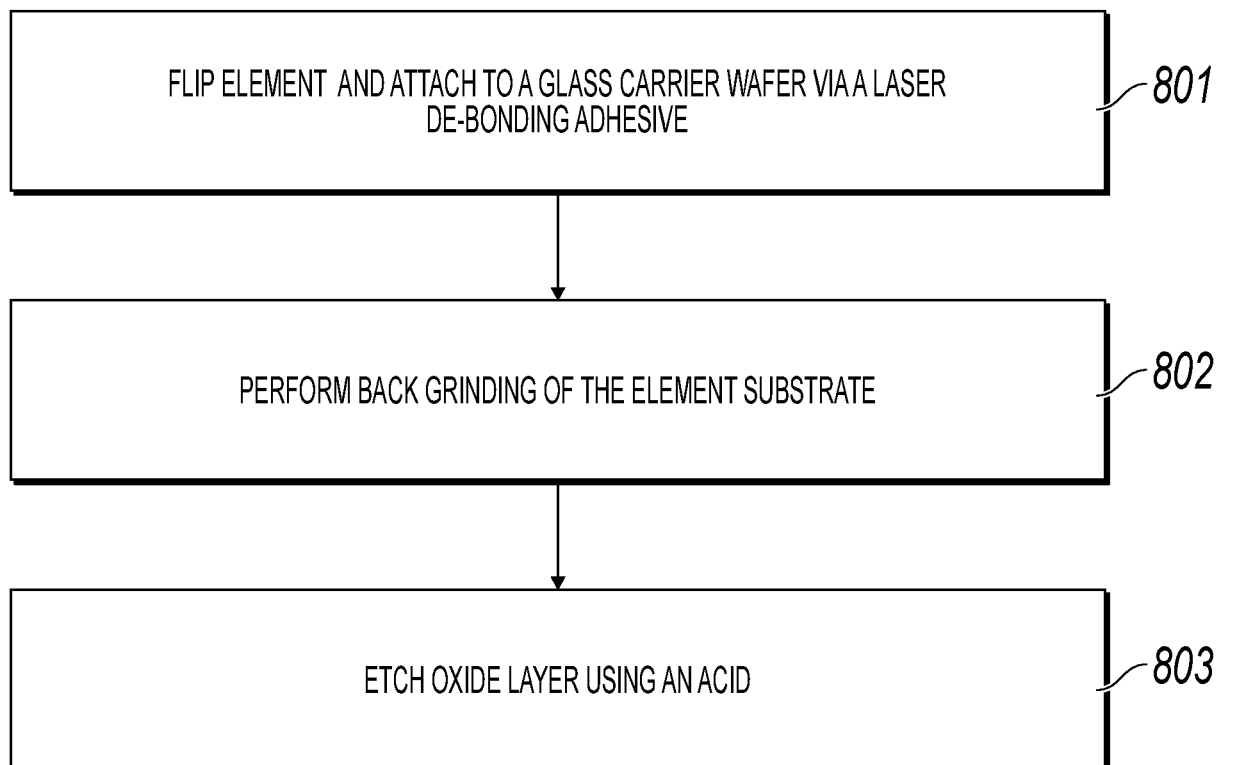
**FIG. 6**

9 / 43

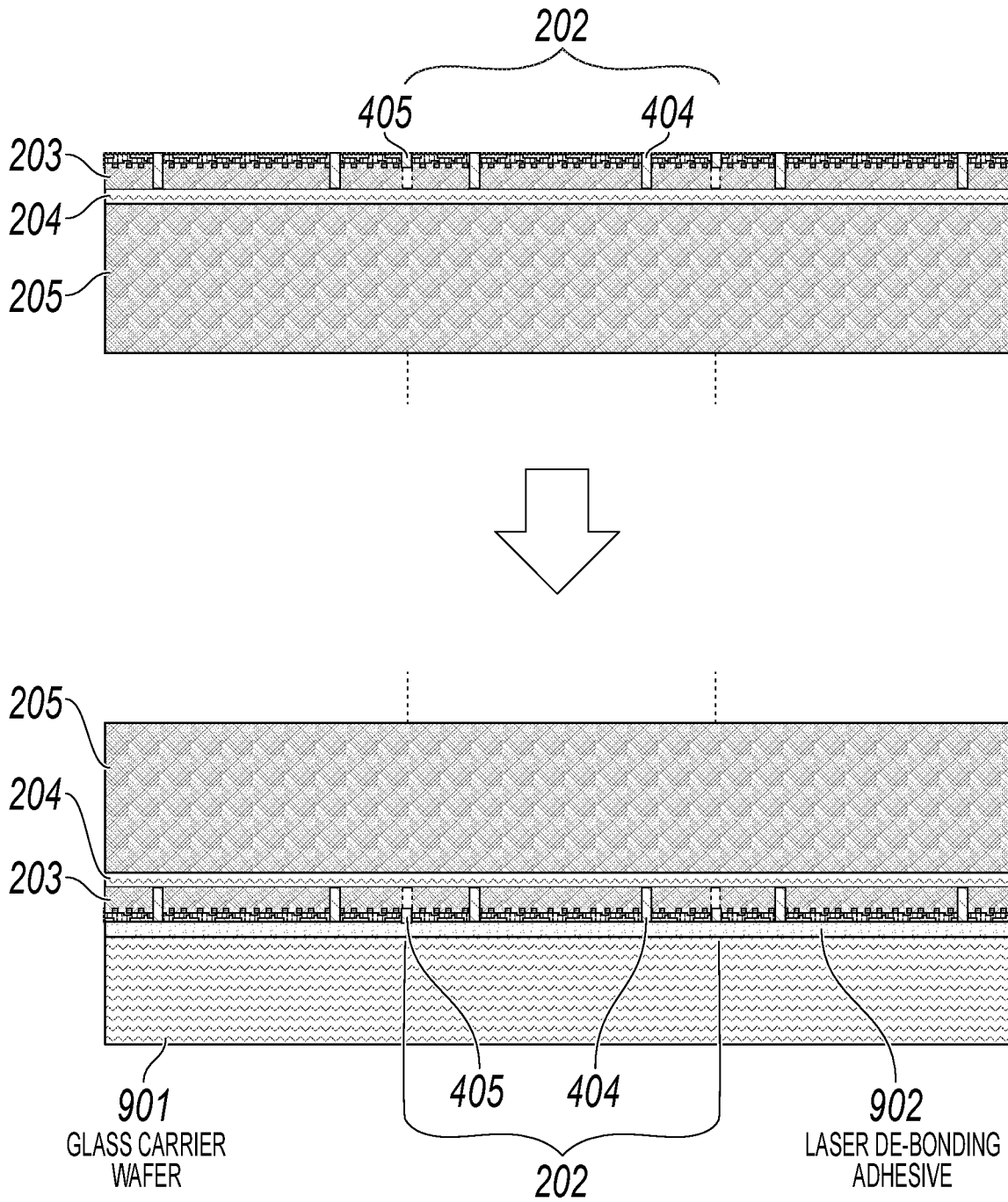
**FIG. 7**

10 / 43

800

**FIG. 8**

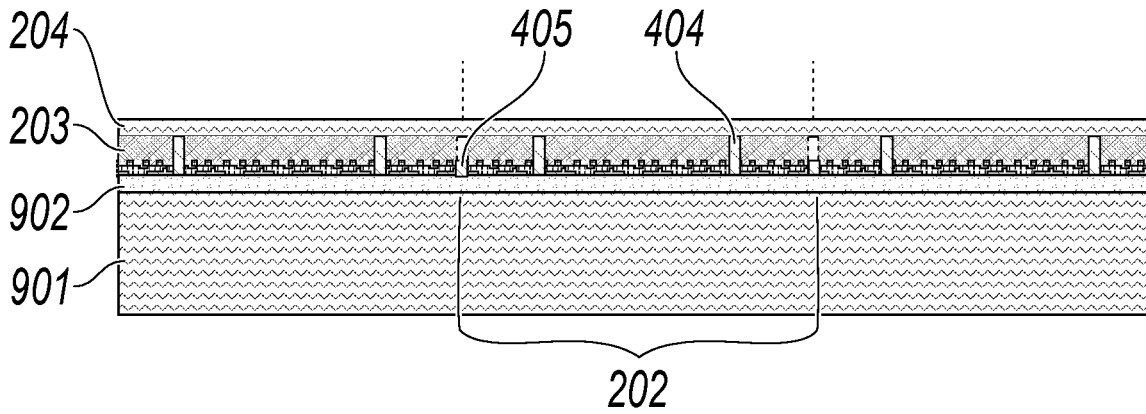
11 / 43



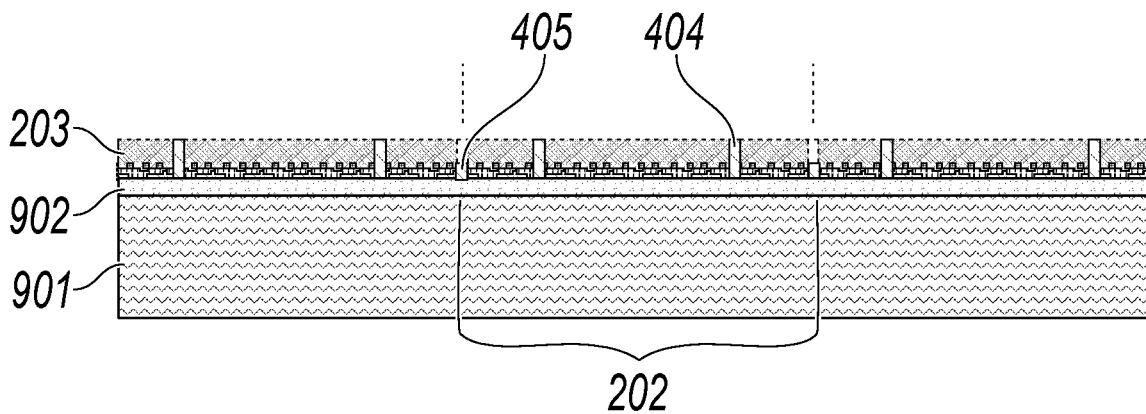
1. FLIP AND ATTACH TO GLASS CARRIER WAFER

FIG. 9A

12 / 43



2. BACKGRINDING

FIG. 9B

3. HF ETCH

FIG. 9C

14 / 43

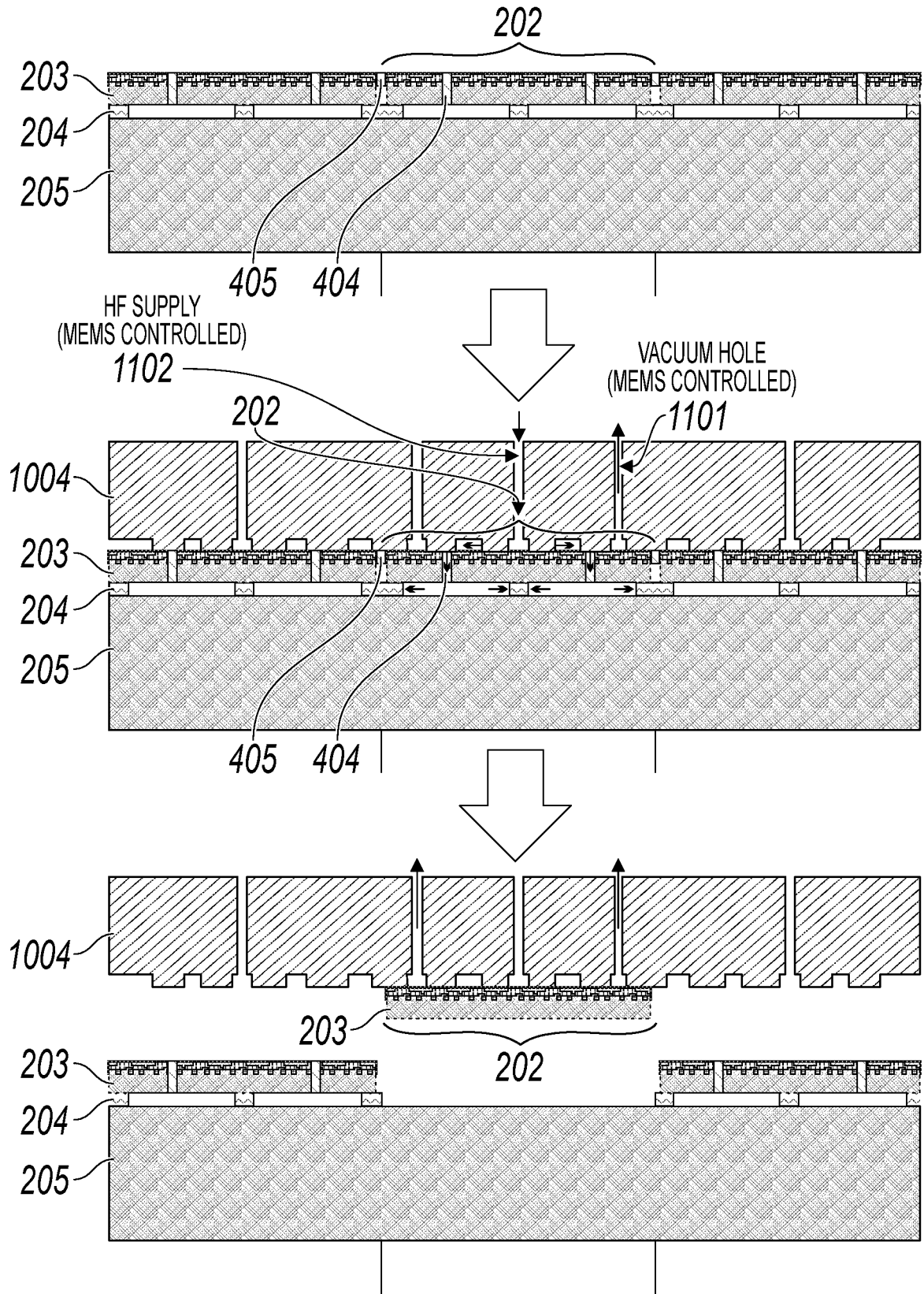


FIG. 11

15 / 43

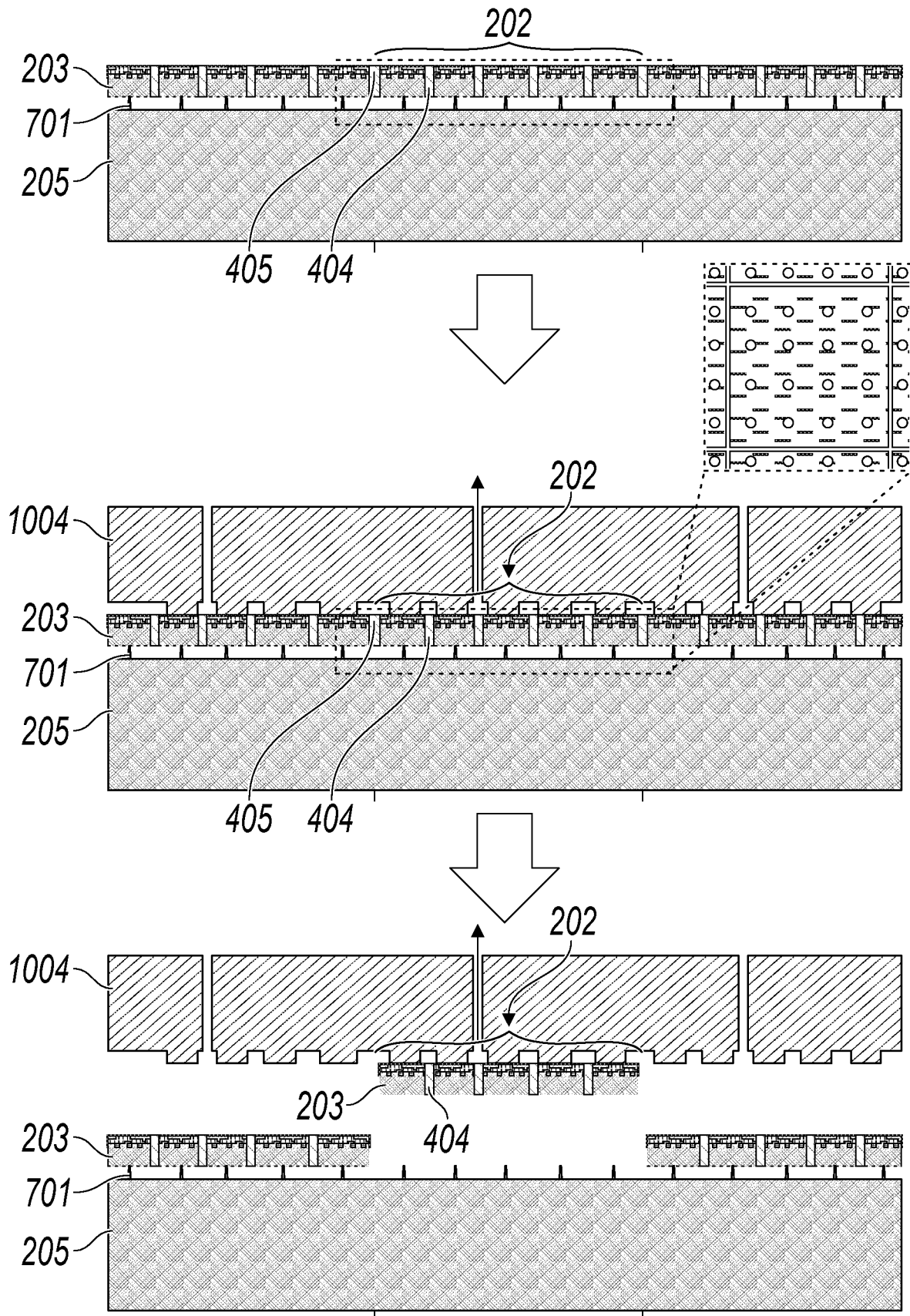
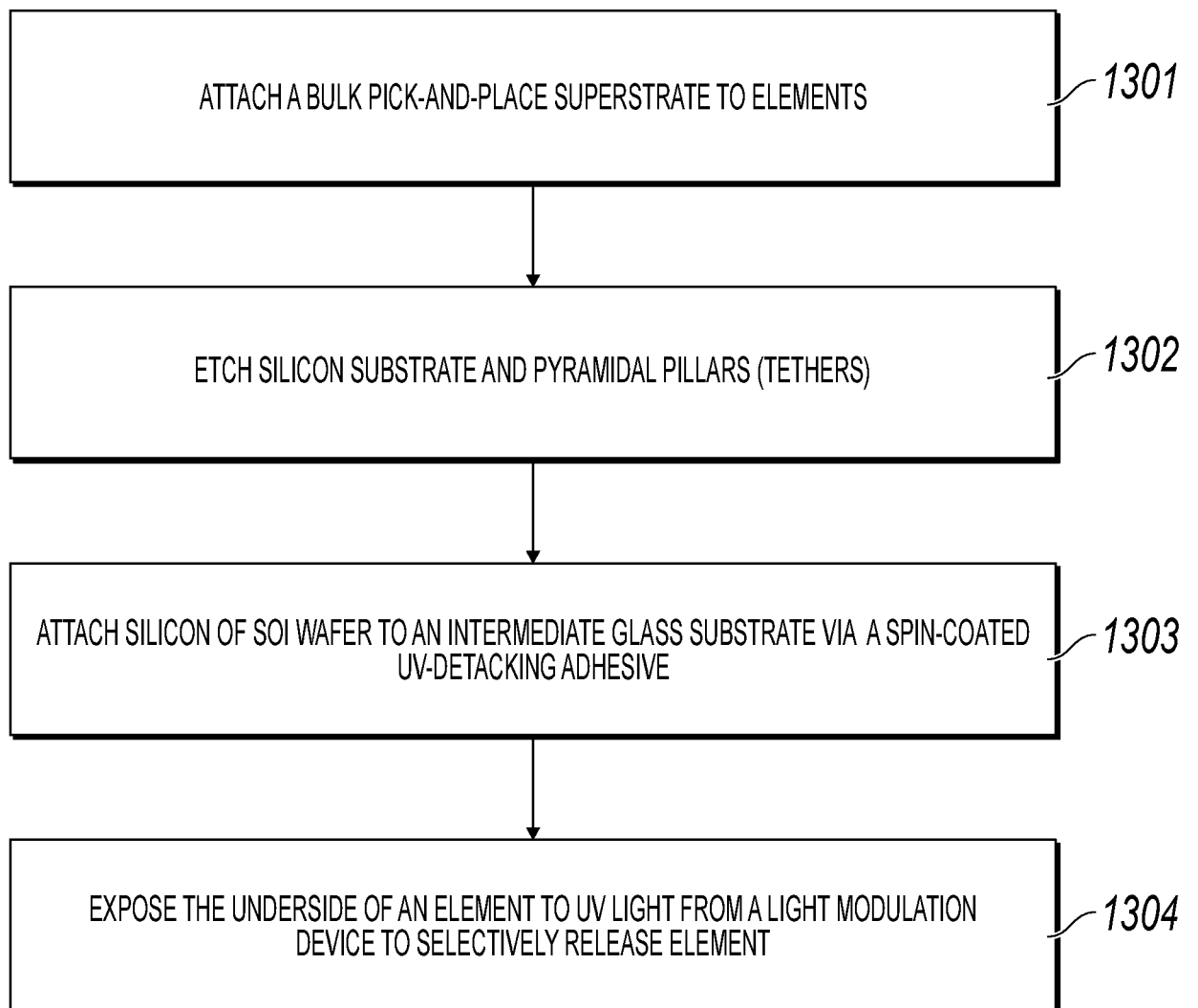


FIG. 12

16 / 43

1300
**FIG. 13**

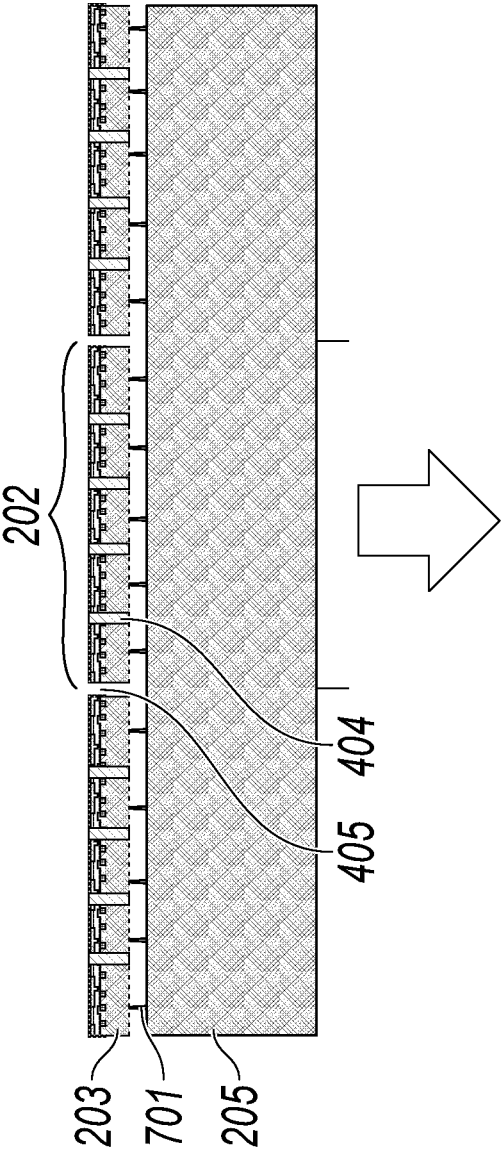


FIG. 14A

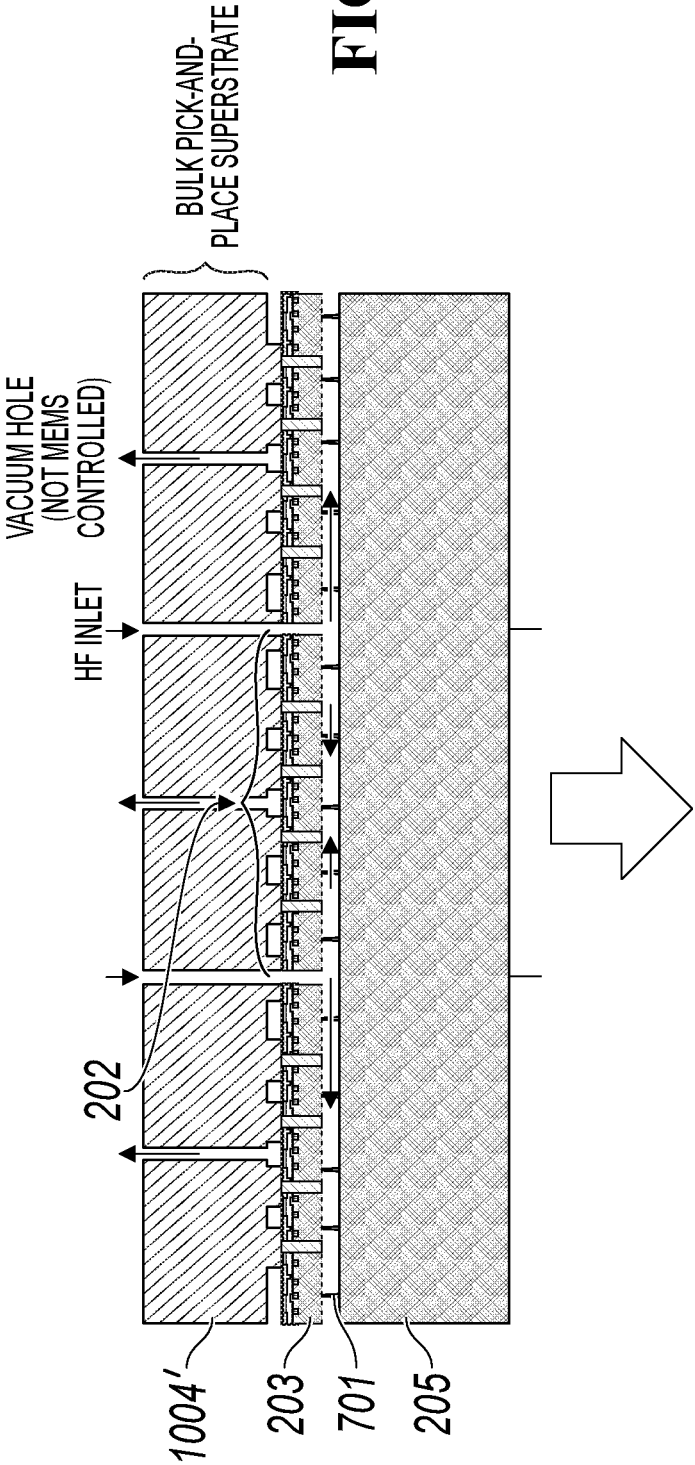


FIG. 14B

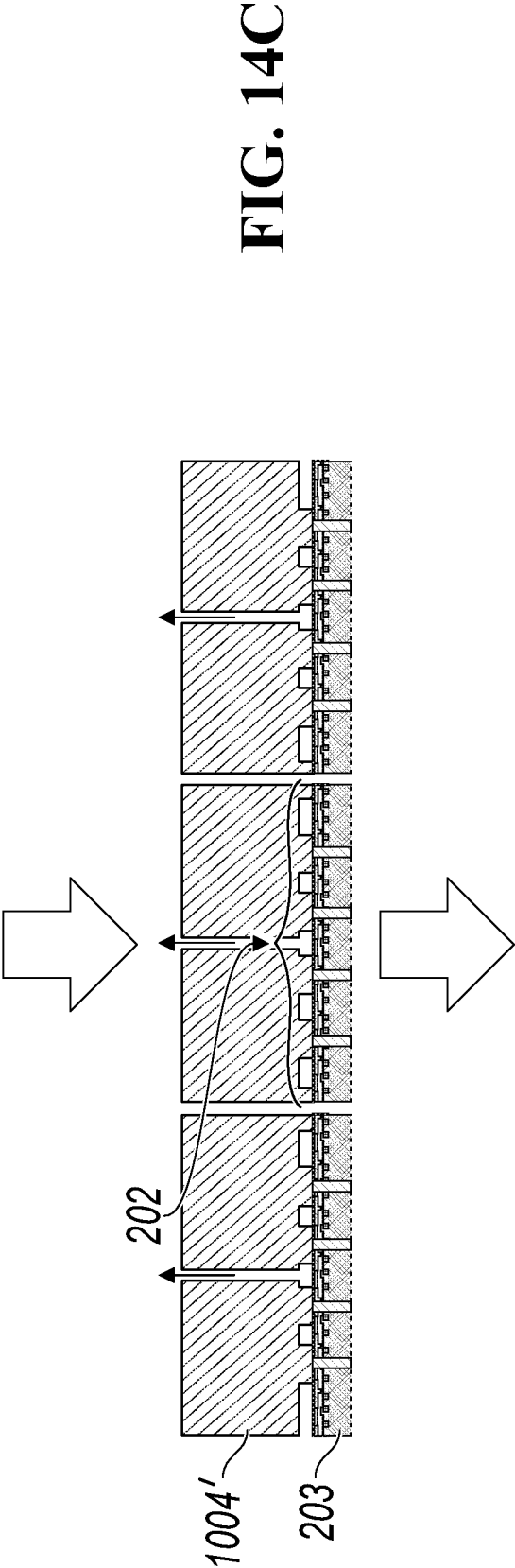


FIG. 14C

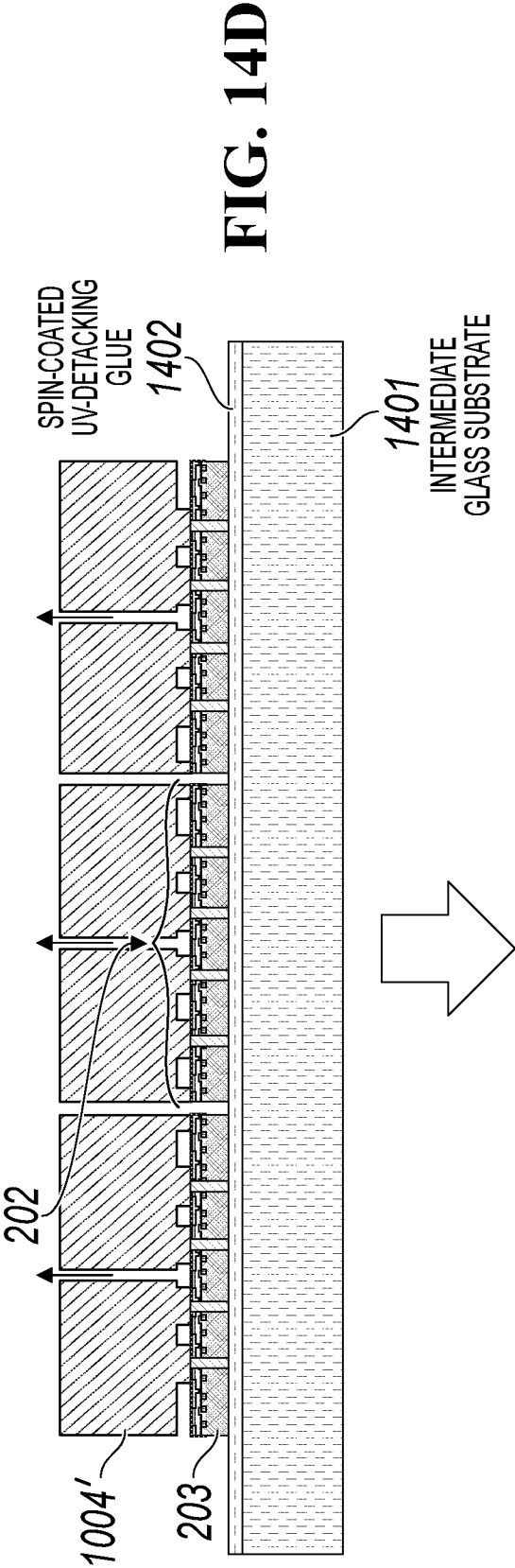


FIG. 14D

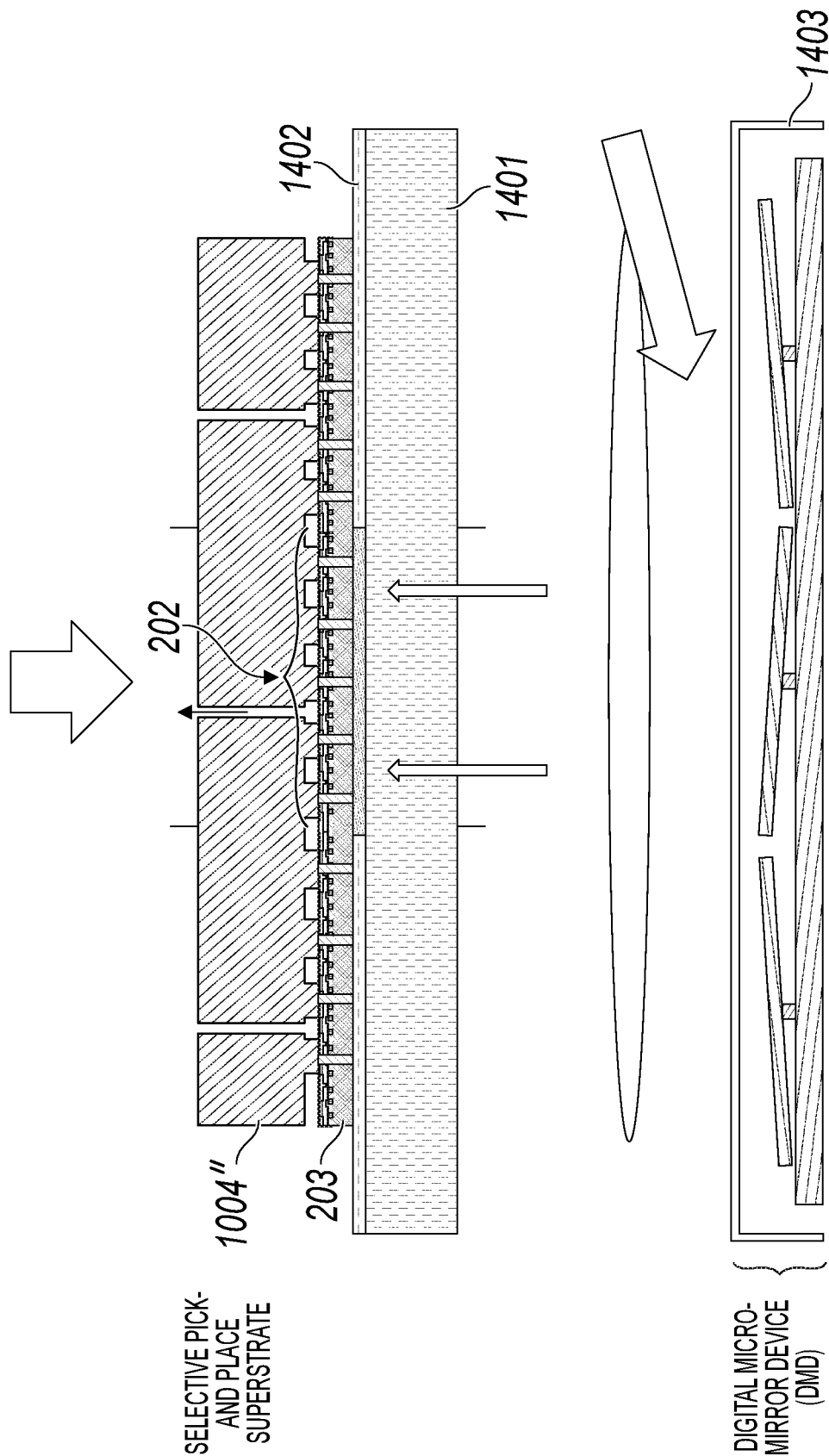
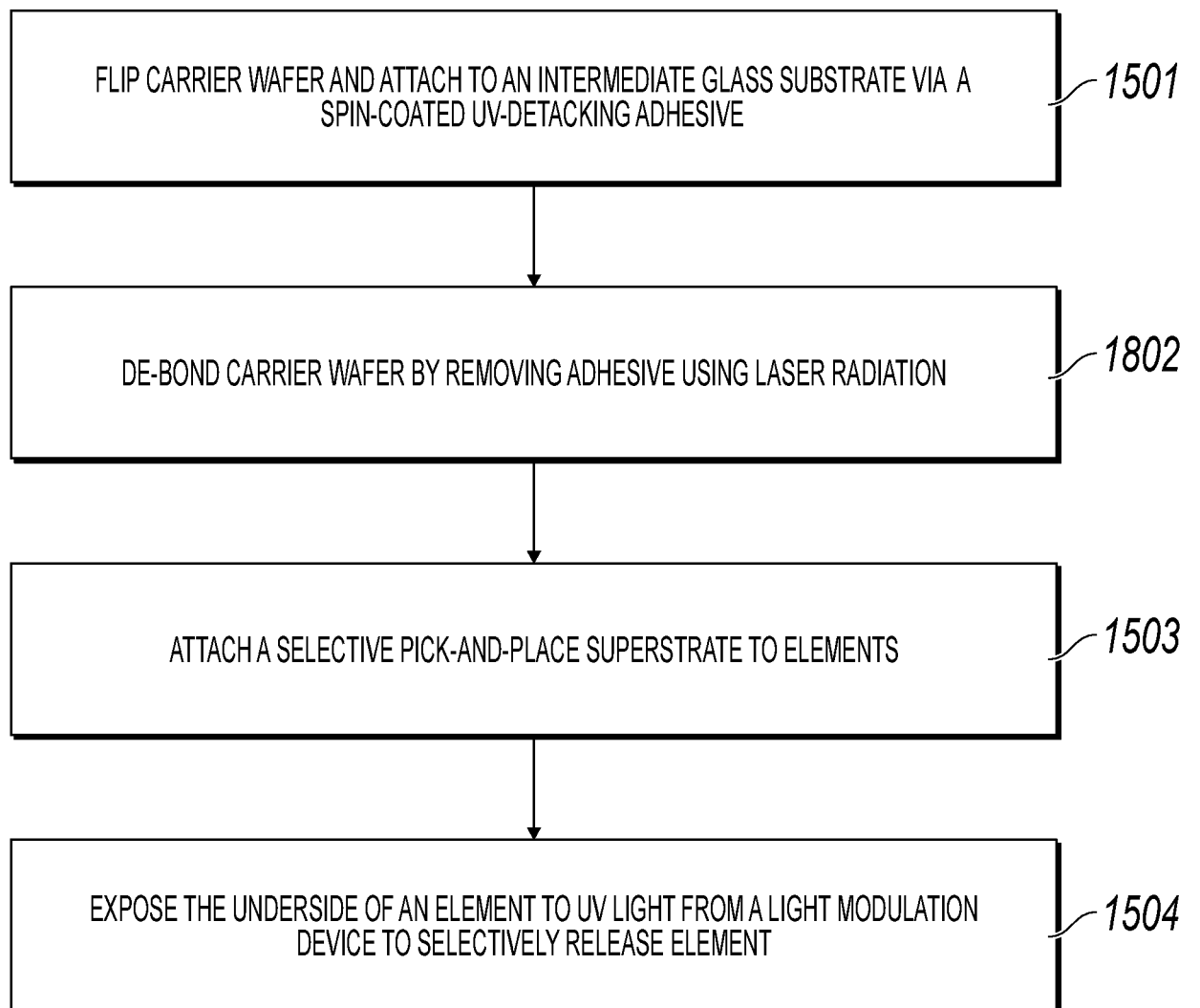


FIG. 14E

20 / 43

1500
**FIG. 15**

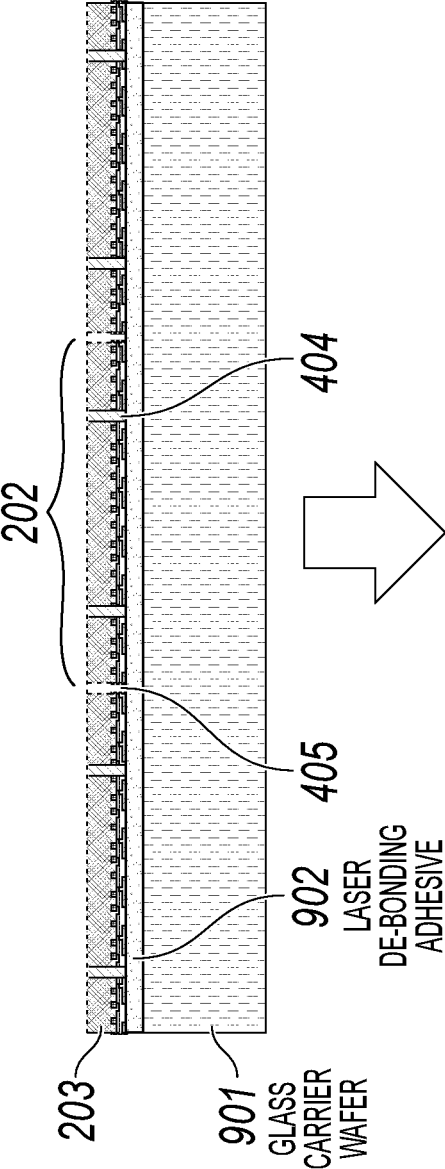


FIG. 16A

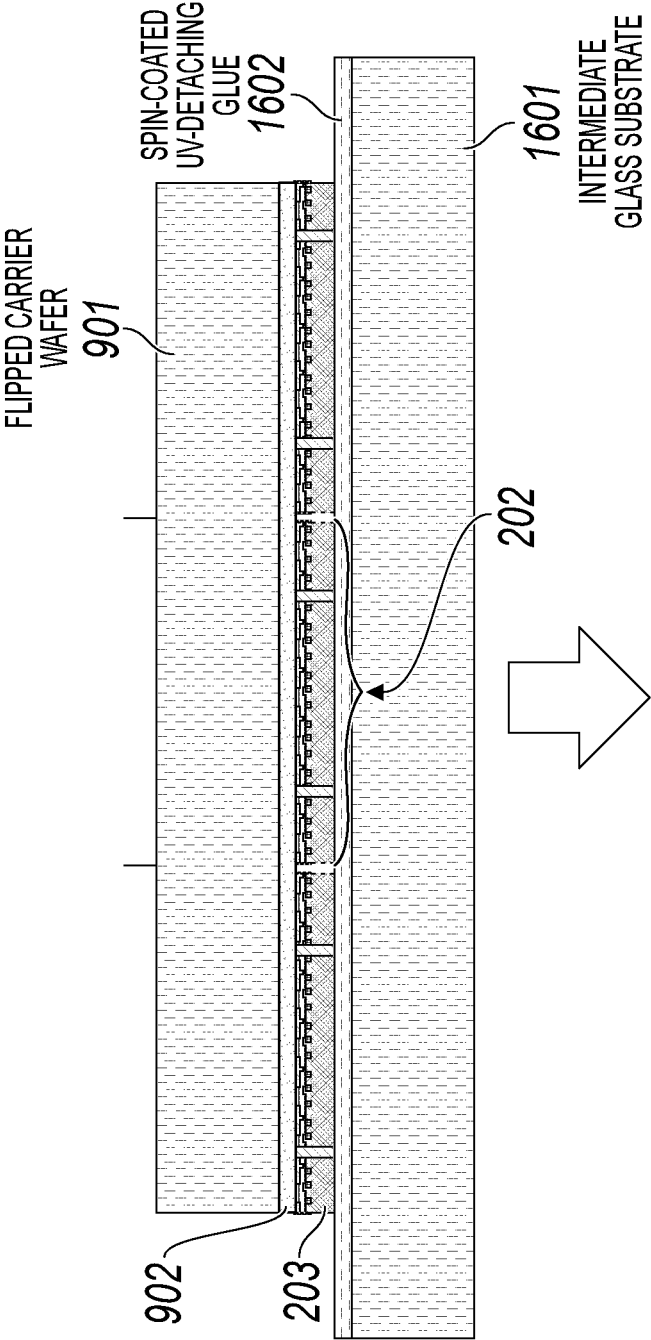


FIG. 16B

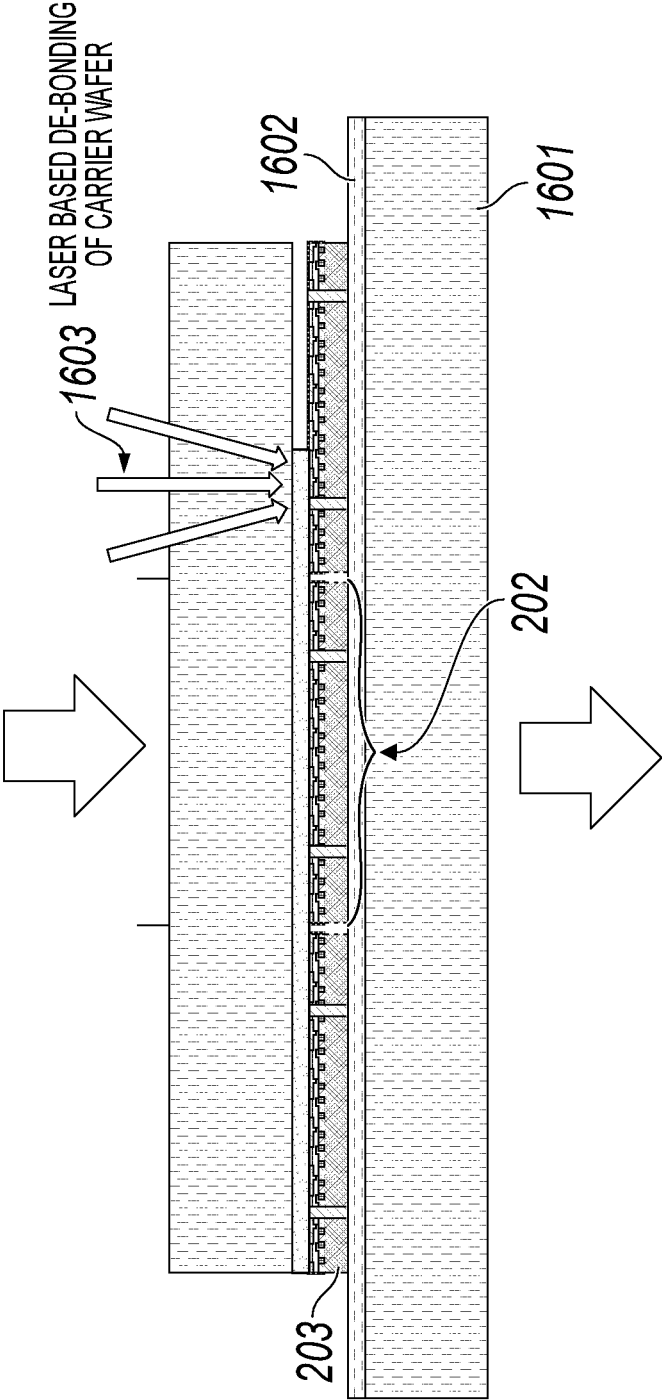


FIG. 16C

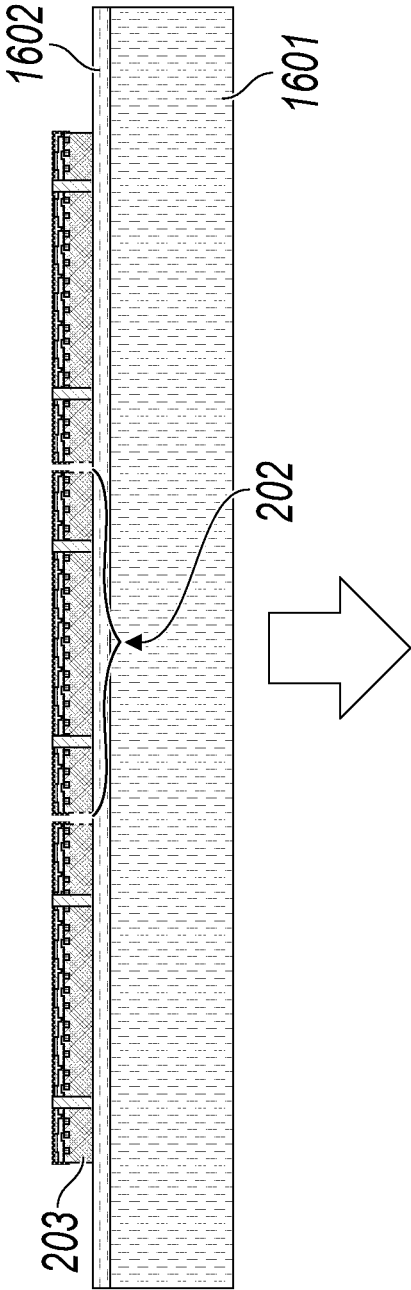


FIG. 16D

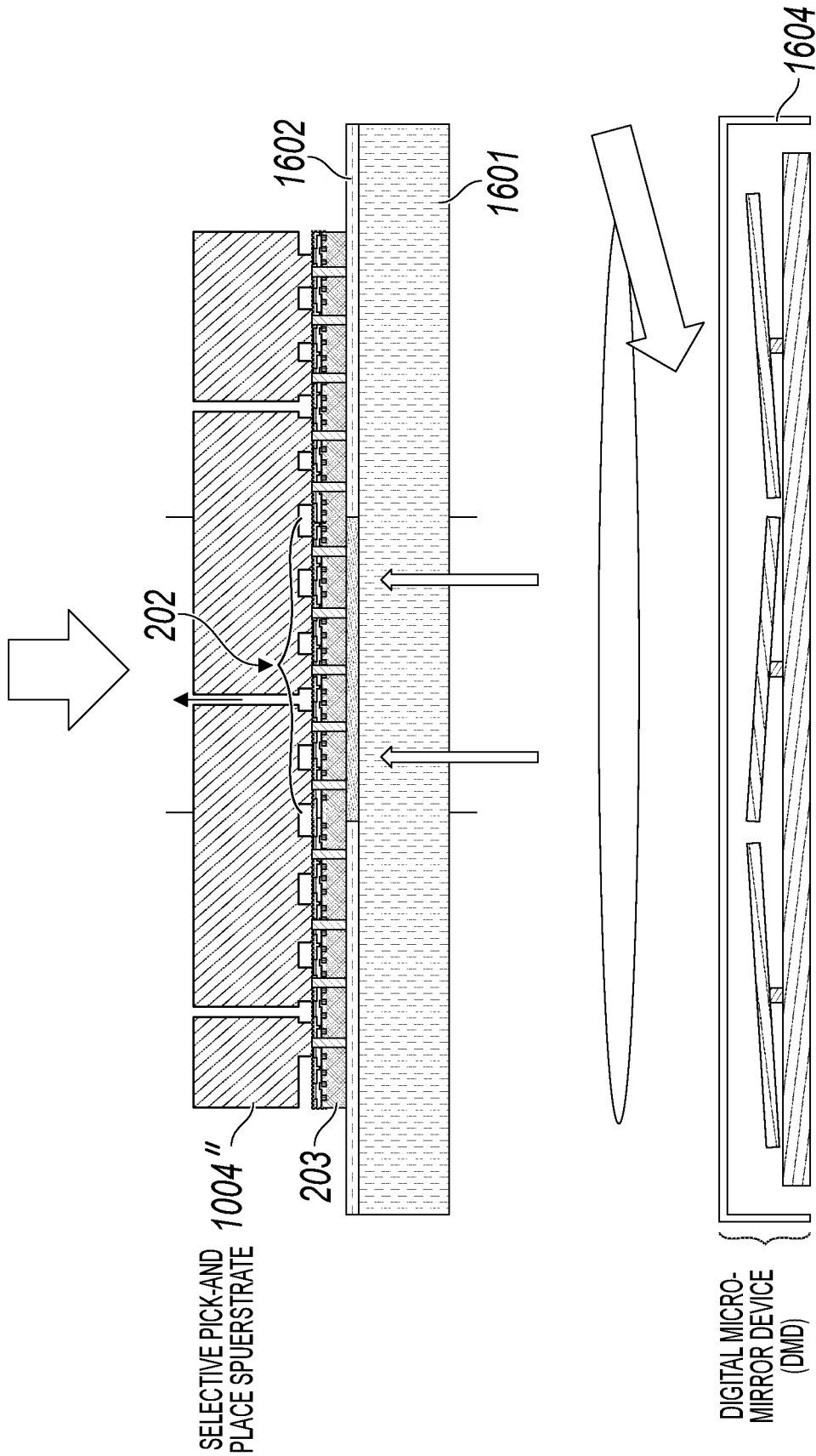
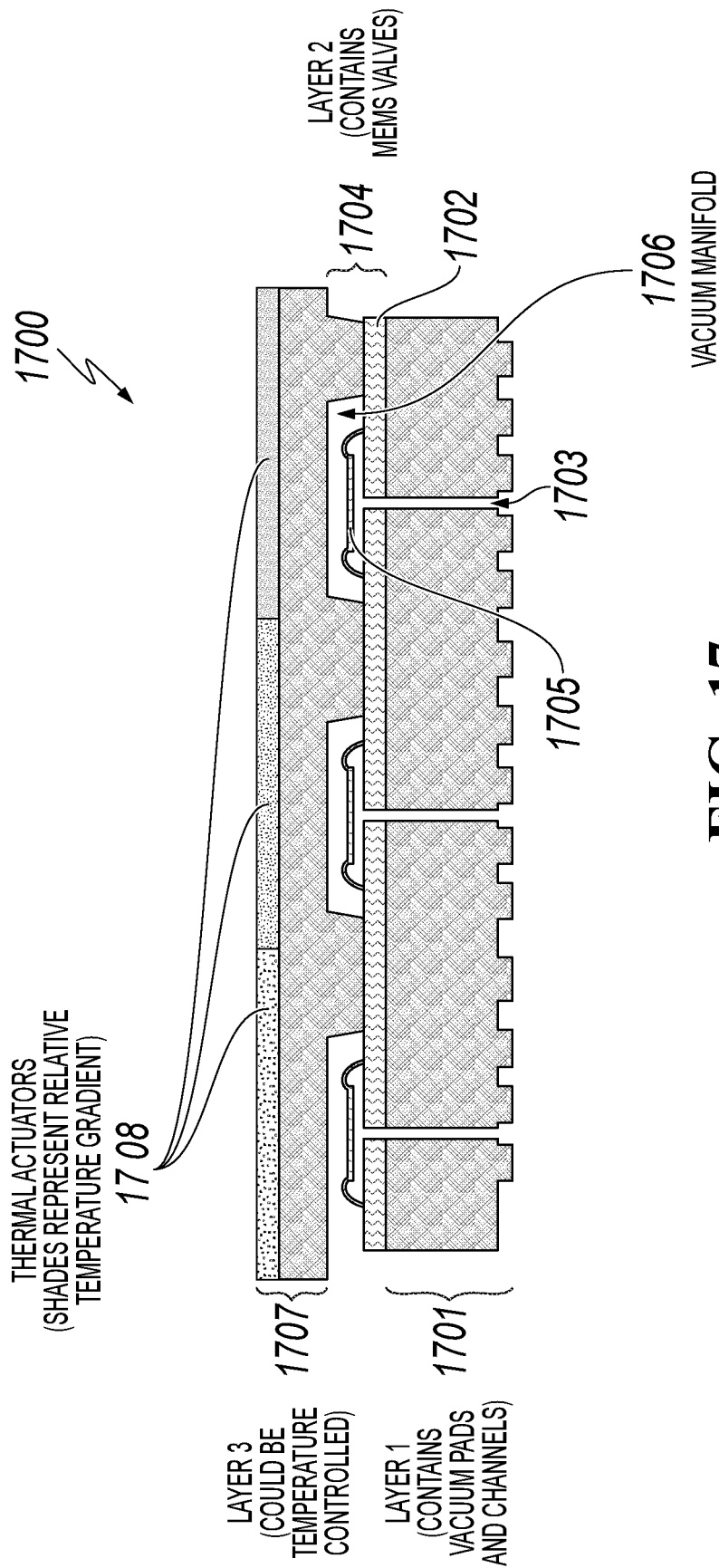


FIG. 16E



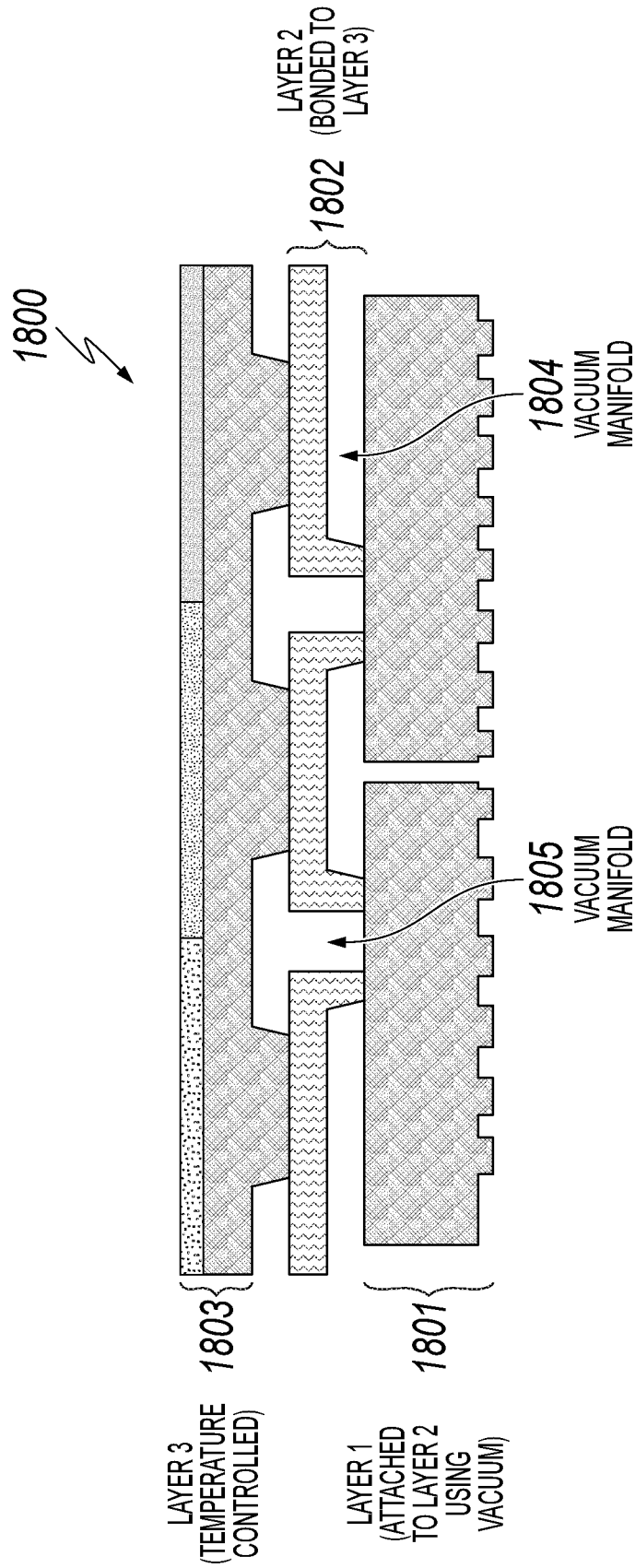


FIG. 18

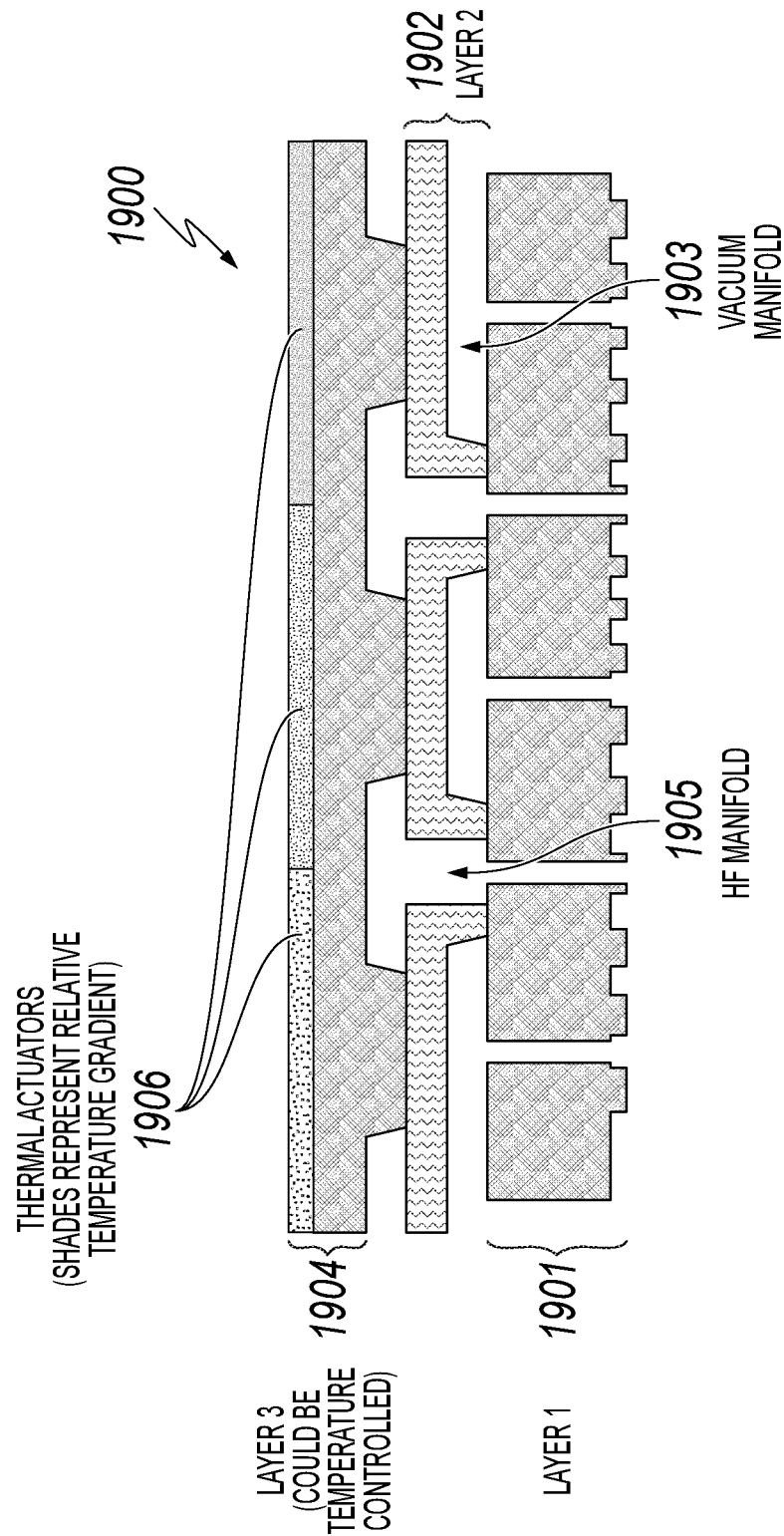
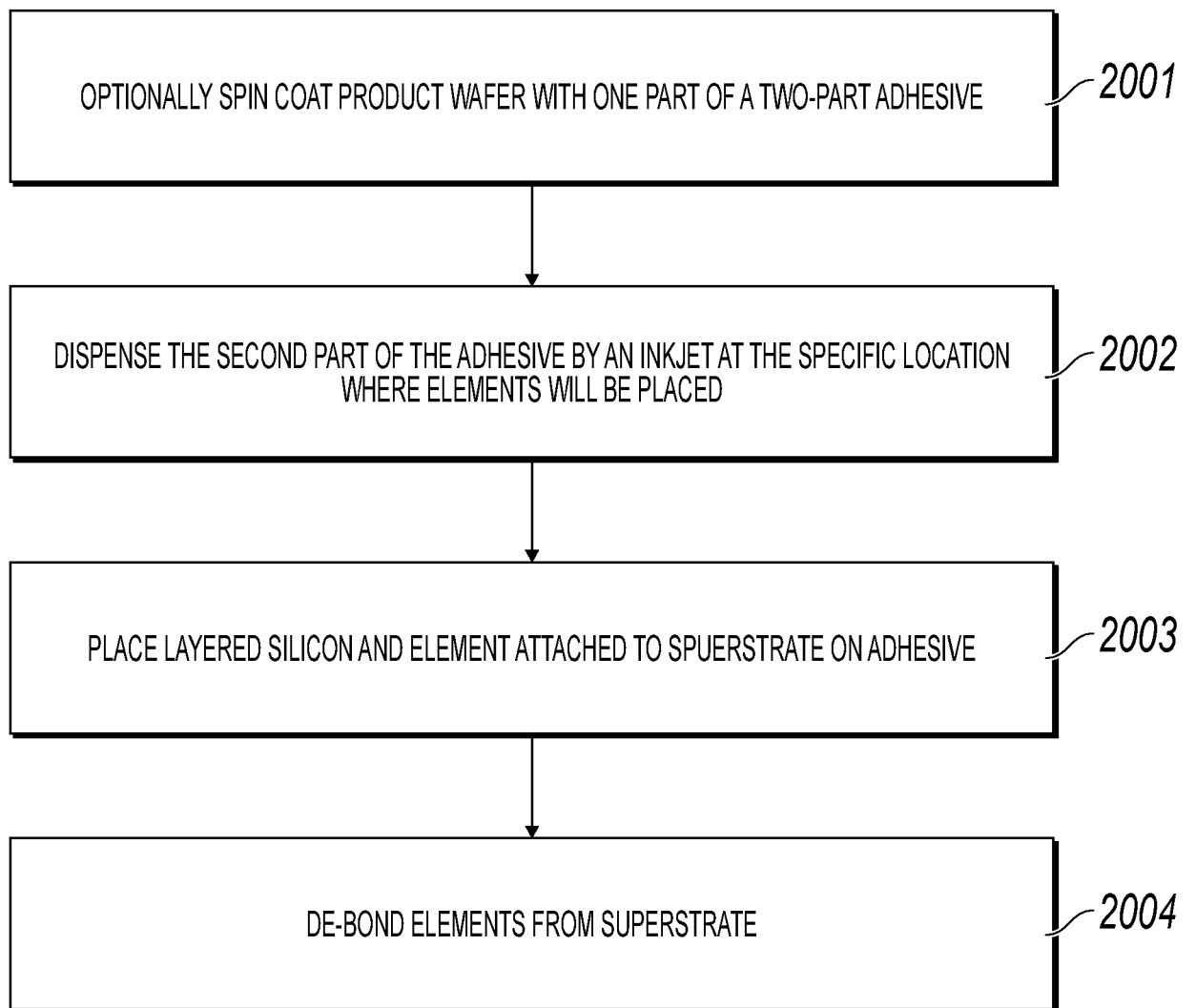
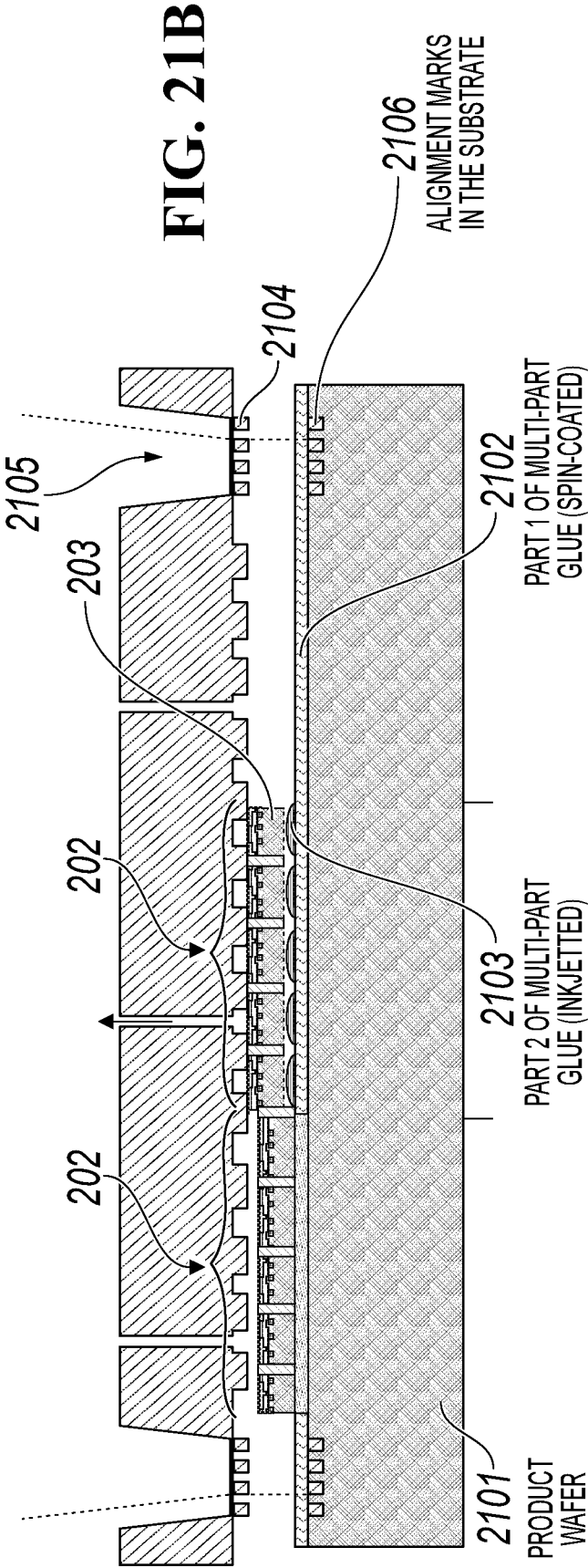
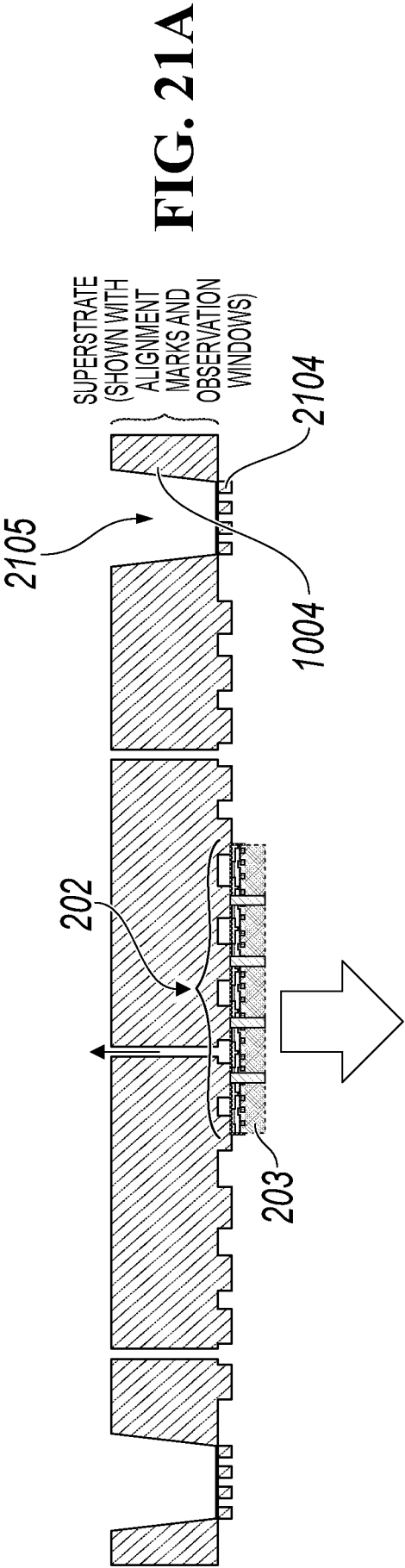


FIG. 19

27 / 43

2000
⚡**FIG. 20**



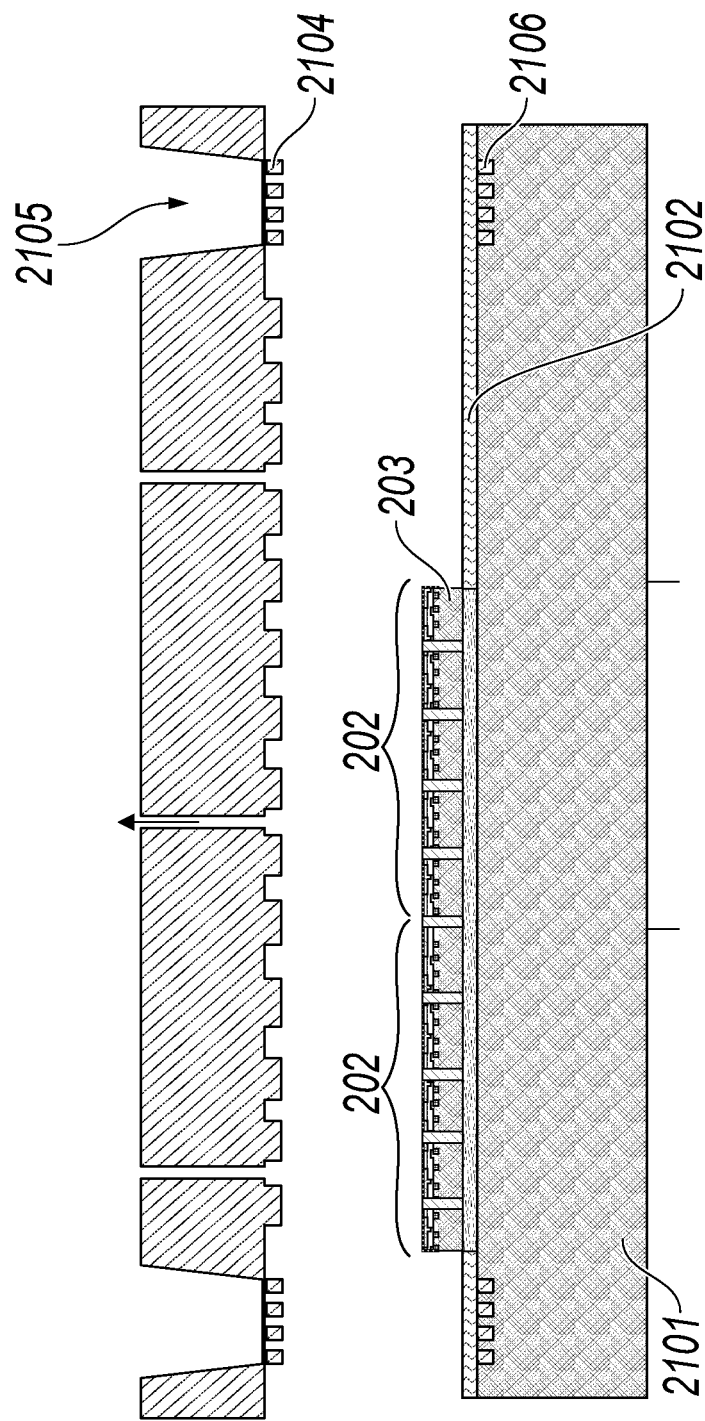


FIG. 21C

30 / 43

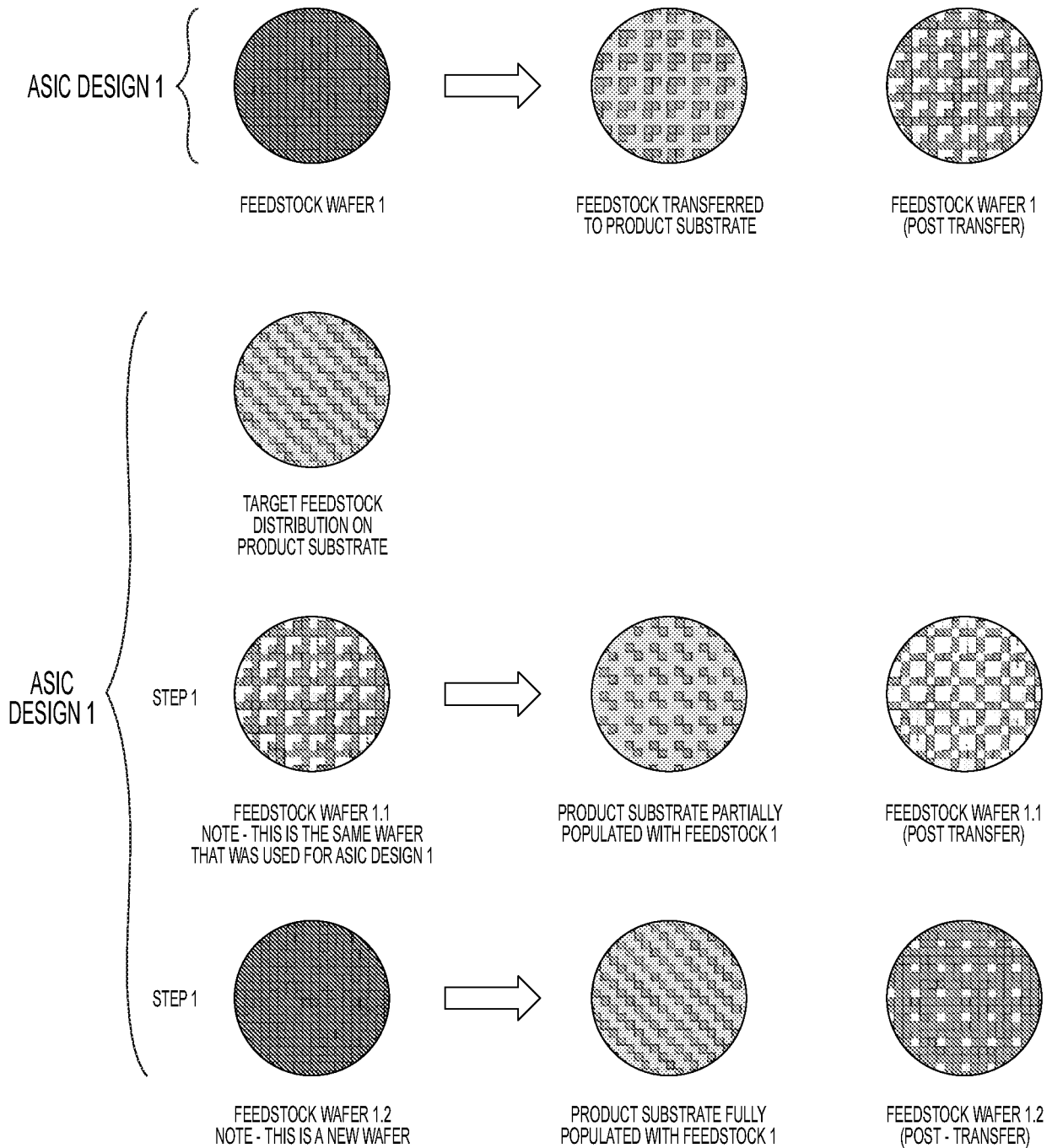


FIG. 22

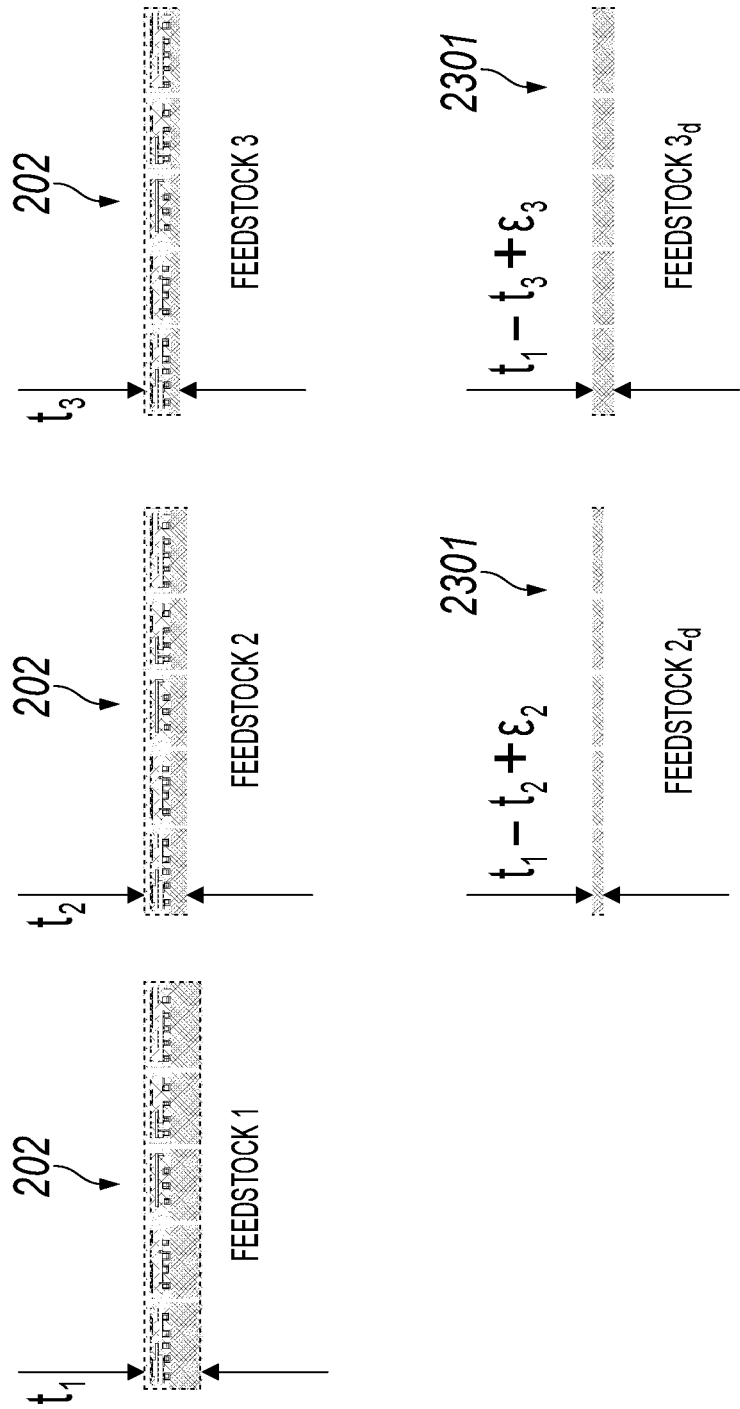


FIG. 23

1700

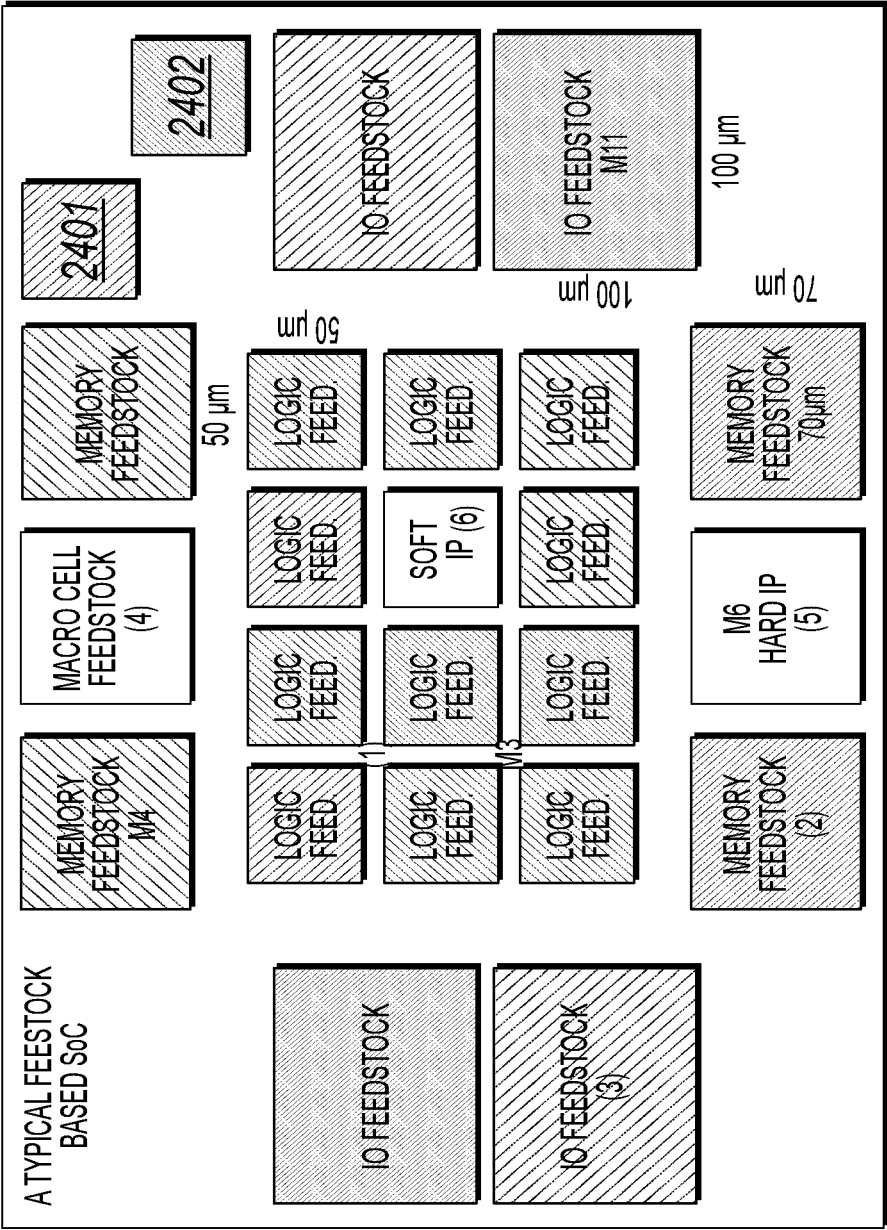
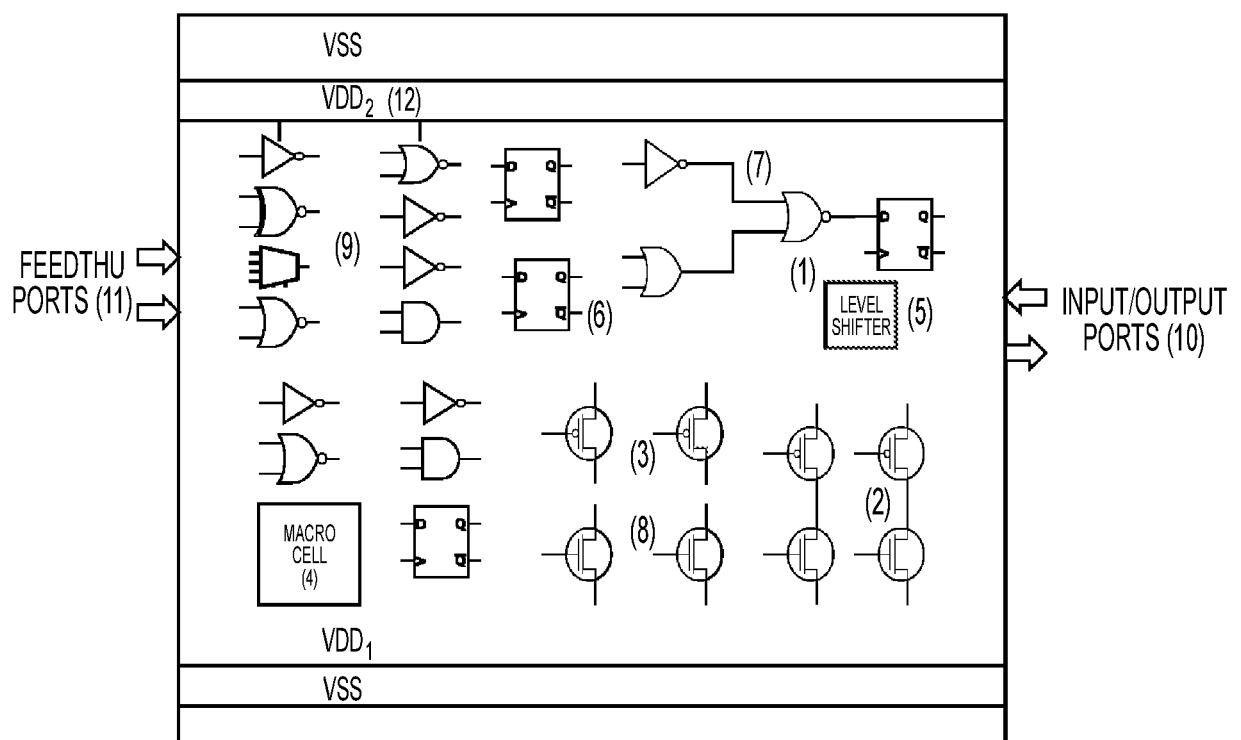


FIG. 24

33 / 43

**FIG. 25**

34 / 43

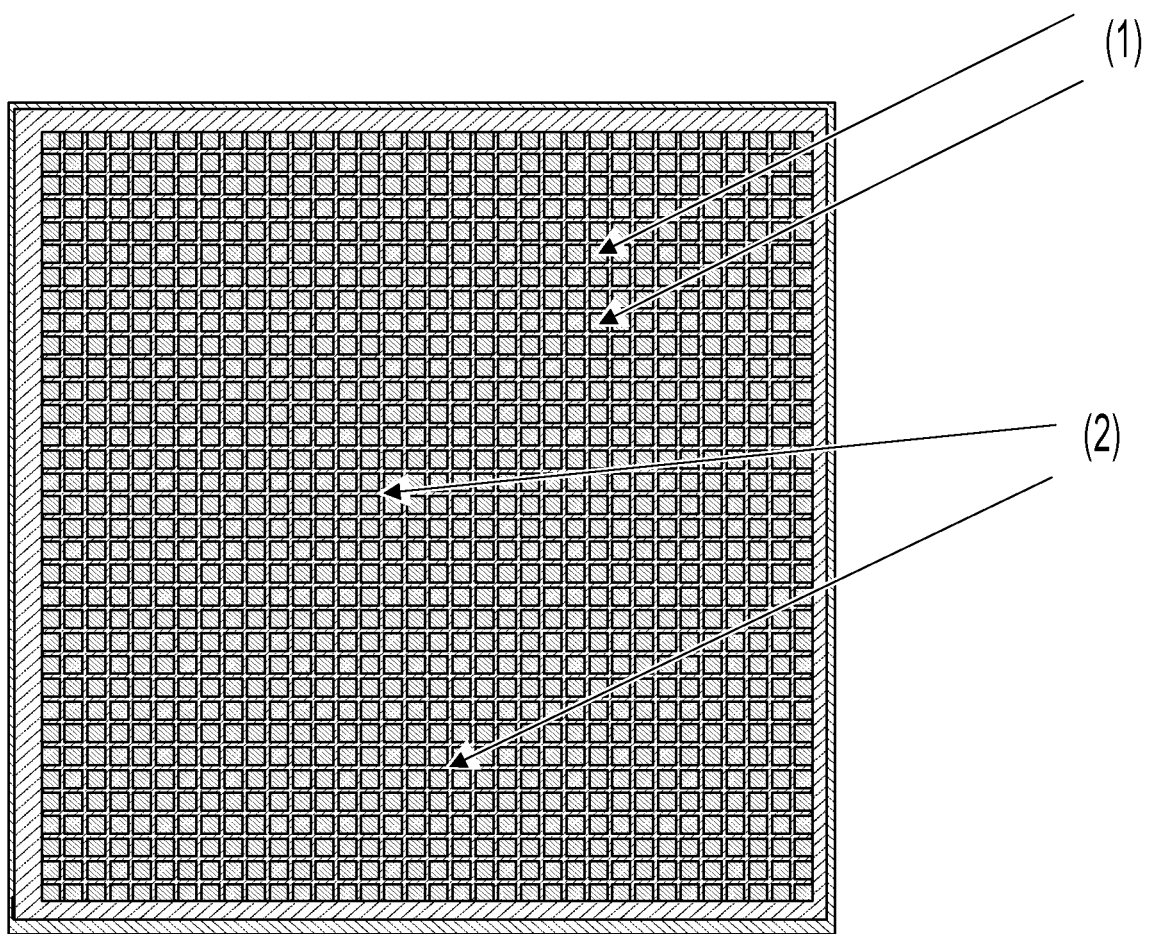


FIG. 26

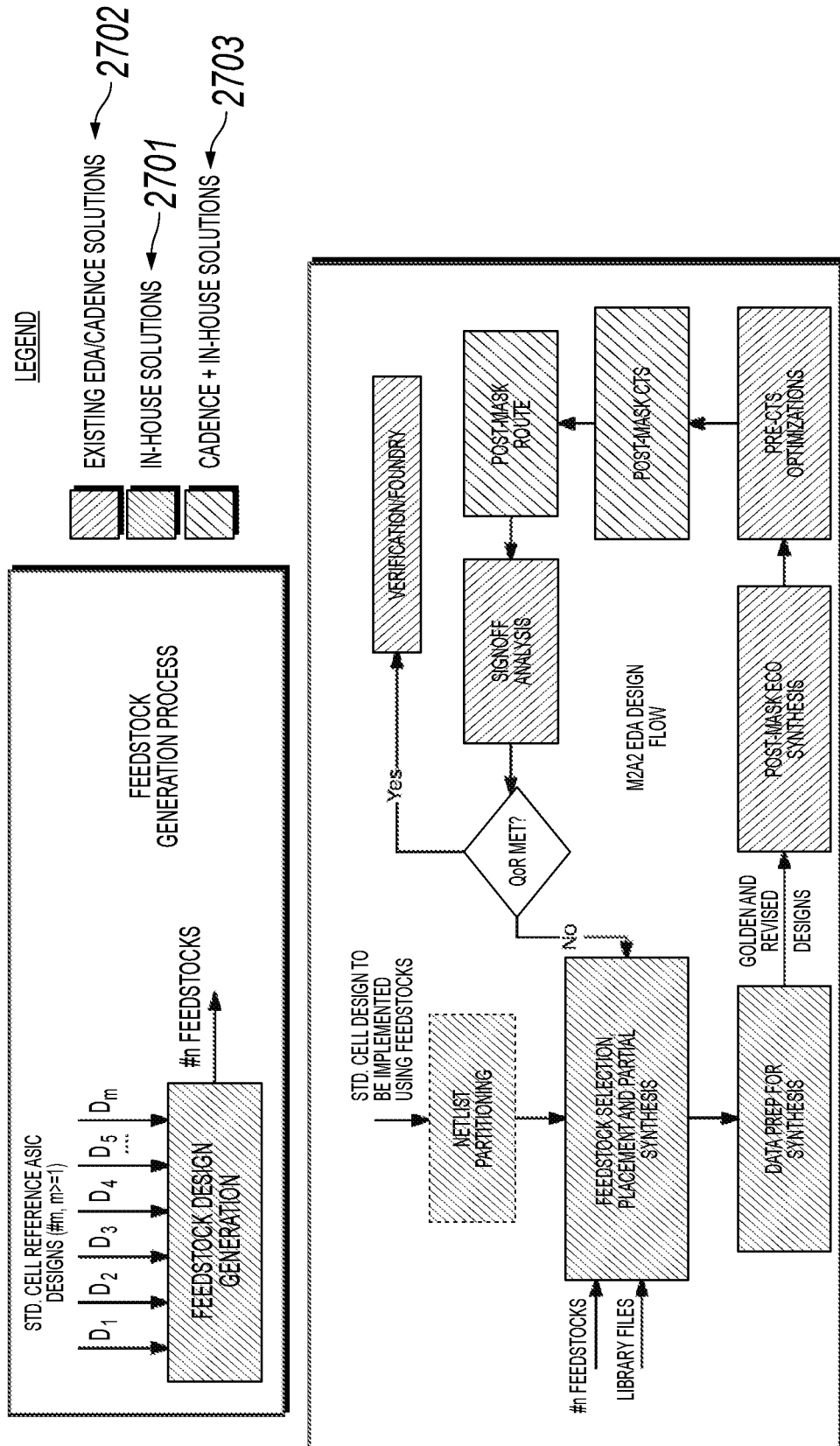


FIG. 27

36 / 43

Algorithm 1 Greedy approach based FDG Algorithm

```

1: processInputs(ASIC designs data);
2: scanFloorplansAndgenerateWindows();
3: analyzeWindows(standard cells placement data);
4: while (matched_windows  $\neq$   $\alpha * total\_windows$ ) do
5:   getReferenceWindow();  $\triangleright$  returns max. utilization unmatched window
6:   for ( $1 \dots unmatched\_windows$ ) do
7:     similarity = computeSimilarityMetric(window,reference_window);
8:     if (similarity  $>$  threshold_value) then
9:       groupSimilarWindows(window,reference_window);
10:    else
11:      generateNewWindows(window);  $\triangleright$  slide and generate window
12:    end if
13:  end for
14: end while
15: improveFeedstockDesign();  $\triangleright$  Greedy approach based cell addition/deletion
    wrt. matching windows
16: if (funcell_mapping  $=====$  1) then
17:   for ( $1 \dots num\_feedstocks$ ) do
18:     assignDriveStrengthVTCCellMapping(feedstock);
19:   end for
20: end if
21: for ( $1 \dots num\_feedstocks$ ) do
22:   legalizeFeedstockCellsPlacement(feedstock);
23: end for
24: generateReports();  $\triangleright$  generate output feedstock reports

```

FIG. 28

37 / 43

Algorithm 2 Clustering and Graph matching techniques based FDG Algorithm

```

1: processInputs(ASIC designs data);
2: scanFloorplansAndgenerateOverlappingWindows();
3: analyzeOverlappingWindows(standard cells placement data);
4: generateCriticalCellFactor(timing_report); ▷ generate timing aware weights
5: while (required_num_feedstocks ≠ formed_feedstocks) do
6:   findReferenceWindow(); ▷ max. critical factor unmapped window
7:   for (1..unmatched_windows) do
8:     similarity = findSimilarityIndex(referene.window,window); ▷ applies
min-cost Bi-partite graph matching, logic restructuring, etc.
9:     threshold.value = generateDynamicSimilarityThreshold();
10:    if (similarity>threshold.value) then
11:      groupSimilarWindows(window,feedstock);
12:    end if
13:  end for
14:  threshold_grouping.value = generateSimilarityGroupingThreshold();
15:  if (total_grouped_windows>threshold_grouping.value) then
16:    formInitialFeedstock(groupedWindows);
17:    removeUnmatchedWindows(groupedWindows);
18:    formed_feedstocks ++;
19:  end if
20: end while
21: mapUnmappedWindows(unmappedWindows, all_feedstocks);
22: updateInitialFeedstocks(); ▷ serves as starting points for k-Means algo
23: while (kmeans_iteration_improvement>threshold_improvement.value)
do
24:   cost = performKMeansClustering(windows,all_feedstocks); ▷ applies
min-cost bipartite matching, k-Means clustering, logic restructuring, etc.
25:   updateFeedstocks(); ▷ update feedstock based on mapped windows,
legalize cell placement using median and Abacus algorithm, etc.
26:   kmeans_iteration_improvement = cost(n-1) - cost(n);
27: end while
28: generateReports(); ▷ generate output feedstock reports

```

FIG. 29

38 / 43

Algorithm 3 Greedy approach based FPS Algorithm

```

1: processInputs(ASIC design data);
2: scanFloorplanAndgeneratePartitions();
3: assignCellsToPartitions();
4: findCriticalNonCriticalPartitions();
5: for (1..critical_partitions) do
6:   feedstockAssignment(all_feedstocks,partition); ▷ greedy approach based
   feedstock mapping
7: end for
8: for (1..noncritical_partitions) do
9:   feedstockAssignment(all_feedstocks,partition); ▷ greedy approach based
   feedstock mapping
10: end for
11: groupFormationAndMapping();           ▷ group nearby windows and map
   unmatched cells to spare cells placed in group
12: unmatchedCellsMapping();             ▷ Map unmatched cells with spare cells in
   design
13: for (1..placed_feedstocks) do
14:   legalizeFeedstockPlacement(feedstock);
15: end for
16: generateReports();                   ▷ generate output feedstock placement reports

```

FIG. 30

Algorithm 4 Graph matching techniques based FPS Algorithm

```

1: processInputs(ASIC design data);
2: scanFloorplanAndgeneratePartitions();
3: assignCellsToPartitions();
4: generateCriticalCellsFactor(timing_report);    ▷ timing aware cell/window
   critical factor
5: for (1...all_partitions) do
6:   mapFeedstock(all_feedstocks,partition);    ▷ applies min-cost Bi-partite
   graph matching, timing aware net weighting, logic restructuring, etc.
7: end for
8: mapUnmappedCellsToSpareCells(all_feedstocks,unmapped_cells);    ▷
   applies min-cost Bi-partite graph matching, timing aware net weighting,
   logic restructuring, etc.
9: partialSynthesis(mapped_cells);
10: for (1...placed_feedstocks) do
11:   legalizeFeedstockPlacement(feedstock);
12: end for
13: generateReports();    ▷ generate output feedstock placement reports
  
```

FIG. 31

Algorithm 5 Post-Mask Clock Tree Synthesis Algorithm

```

1: deleteCells(spare.CTS.cells);
2: createDesiredRegionsStripes();
3: setupCTSEnvironment();
4: performCTS();           ▷ performs CTS in Cadence Innovus tool
5: processInputs(CTS.added.cells,spare.CTS.cells);
6: for (1..CTS.added.cells) do
7:   mapCTSCellToSpareCell();   ▷ applies either greedy approach based
   mapping or min-cost bipartite graph matching techniques, etc.
8:   if (cell_map == 1) then
9:     swap.cell ← CTS.added.cell;
10:    delete(CTS.added.cell);
11:    swapNetlistPinAndCellConnections();
12:   end if
13: end for
14: for (1..existing.CTS.added.cells) do
15:   delete(CTS.added.cell);
16:   preserveNetlistLogic();
17:   changeNetlistConnections();
18: end for
19: generateTclCommandFile();   ▷ contains cell/pin add/del/swap commands
20: source < TCL_FILE >       ▷ source it in Innovus session loaded with CTS
   database to perform postMask CTS

```

FIG. 32

41 / 43

Algorithm 6 Post-Mask Post-CTS/Post-Route Buffer Insertion Algorithm

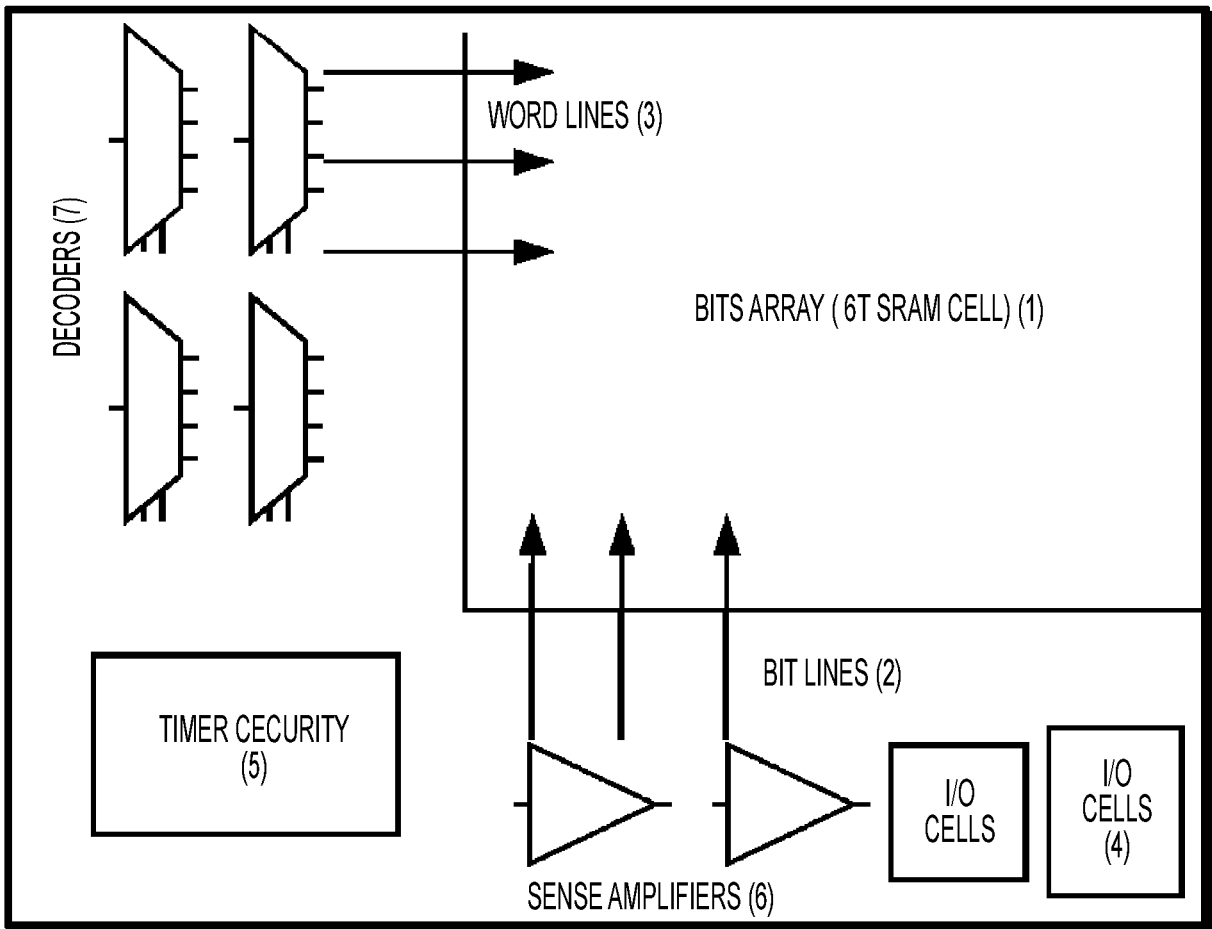
```

1: analyzeTimingReports();
2: sortDataPathsSlacks();
3: sortClockPathsInsDelaySkew();
4: if (dataPathOptimizations == 1) then
5:   findViolatingPathCells();
6:   findPositiveSlackPathCells();
7:   fixDataPathViolations(spare_buffers,positive_slack_path_buffers);    ▷
   applies either greedy approach based mapping or min-cost bi-partite graph
   matching techniques, etc.
8: end if
9: if (clockPathOptimizations == 1) then
10:  findSkewInsDelayPathCells();
11:  findBalancedClockTreeBranchCells();
12:  fixClockPathViolations(spare_CTS_buffers,balanced_branch_CTS_cells);
   ▷ applies either greedy approach based mapping or min-cost bi-partite
   graph matching techniques, etc.
13: end if
14: generateTclCommandFile();
15: source < TCL_FILE >    ▷ source in Innovus session to resolve violations

```

FIG. 33

42 / 43

3400
⚡**FIG. 34**

3500

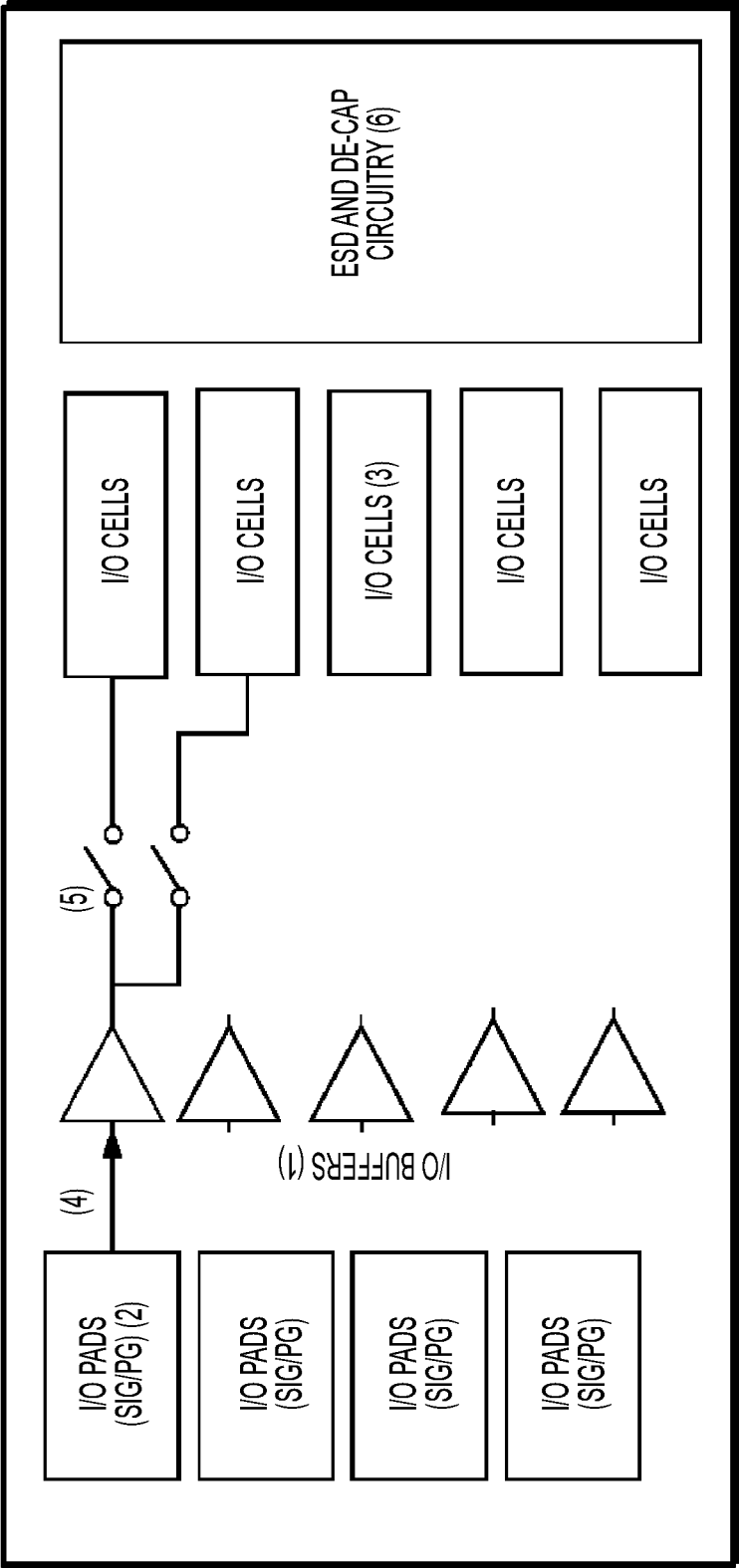


FIG. 35