Layout Verification by Extraction for Micro Total Analysis Systems

Bikram Baidya (bbaidya@ece.cmu.edu) and Tamal Mukherjee (tamal@ece.cmu.edu)
Carnegie Mellon University, Pittsburgh, PA 15213, USA

ABSTRACT

The increasing complexity of Micro Total Analysis Systems (μTAS) is leading to a growing need for layout verification. Currently, verification involves numerical simulation, which is slow, memory consuming and impractical for large designs. An alternative is to couple layout extraction with emerging schematic-based simulation for verification of complex designs. This paper reports on a prototype μTAS layout extractor which automatically constructs a schematic from a given layout. The extractor presented is capable of handling non-Manhattan and curved geometry and includes recognition heuristics specific to μTAS designs. The utility of the extractor is demonstrated using selected designs.

Keywords: μTAS verification, extraction, non-Manhattan, canonization, polygonization, element recognition

INTRODUCTION

Advances in microfabrication are enabling increasingly complex Micro Total Analysis System (μTAS) designs. Currently the only means to verify μTAS layouts is to mesh the design and then numerical analyze it. This approach is amenable for small designs, but can become prohibitively slow for large designs. Schematic-based simulation [1][2][3][4] have been proposed for efficient characterization of complex designs. Adopting this capability for layout verification currently requires manual translation of the layout into a schematic form. This paper introduces a prototype μTAS layout extractor capable of automatically reconstructing an extracted schematic from any given μTAS layout for subsequent verification via schematic-based simulation.

Layout extraction is widely used in VLSI and is increasingly used in suspended MEMS design [5][6][7]. μTAS layout extraction starts with the geometric data from the layout representation. Unlike suspended MEMS, non-Manhattan and curved geometry dominate μTAS layouts. Extraction uses curve detection followed by geometry processing to create an unique representation (which simplifies the element recognition heuristics). Finally the connectivity between the recognized components is extracted and used to generate a netlist which can be simulated using schematic models.

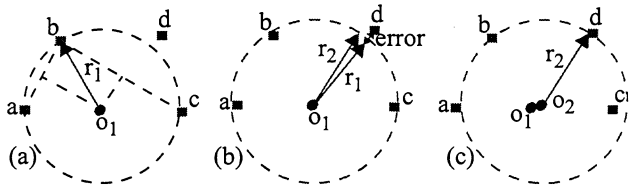

Fig. 1: Curve detection & fitting algorithm: (a) initial guess of center (o_1) and radius (r_1) using perpendicular bisectors of two chords, (b) update radius to r_2, (c) update center to o_2.

CURVE DETECTION

Turns, which are widely used in μTAS layouts tend to be stored as polygons by standard microfabrication layout representations (CIF, GDS). This makes detection and extraction of curved geometry essential. A survey of recently published μTAS designs indicate that almost all the curves are formed by a combination of circular arcs. This work assumes that all curves are formed by arcs of a circle that are represented as polygons using at least four non-trivial vertices. Geometrically three points are sufficient to fit a circular arc. The extra point is used by the curve detection algorithm to identify the *turn angle* (the difference in angle) between successive edges of a polygon. Pairs of adjacent edges for which the *turn angle* is less than the user specified *curve_angle*, and have the same *turn angle* sign are considered to be part of the same arc.

Accurate curve fitting algorithms use recursion and have large time complexity. Faster approximate algorithms can exploit the fact that the polygon vertices are generated from an actual curve in the layout. We have adapted the bounding sphere algorithm from [8] (see Fig. 1). First three spatially distributed points are selected to initialize the center and radius of the arc. Next, a pass through the vertices uses the error at each vertex to update the radius ($r_i = r_{i-1} + error/2$) and then update the center coordinates such that the new arc passes through the point on one side and has the older arc on the other side. The recognized arc is finally represented by an imaginary edge (referred to as *pseudoedge*) connecting the two end points of the arc.

CANONIZATION

In order to simplify the recognition heuristics used during extraction, the layout geometry needs to be represented uniquely. Edges of the geometry are classified into *external* and *internal edges* (Fig. 2) and can be defined to have two *faces*, one on either side of the edge. A *face* of an edge is *visible* if there exists a point, internal to the layout geometry, from which a line perpendicular to the edge can be drawn without intersecting any other edge. *External edges* form the boundary of the geometry and can have only one *visible face* (the side which faces the inside of the geometry). In contrast, *internal edges* have visibility on both their faces. Two edges are said to

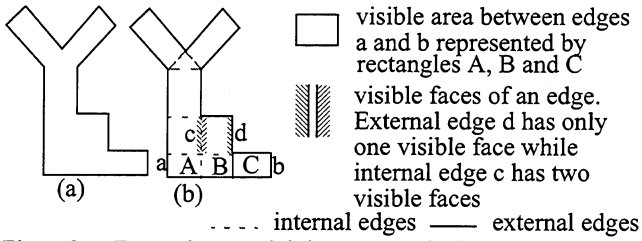

Fig. 2: Example explaining canonical representation. (a) initial layout geometry, (b) final canonized representation

be *mutually visible* if they are parallel and if a line perpendicular to both the edges can be drawn between *visible faces* of the edges without intersecting any other edge. The area internal to the geometry that is visible to both of the *mutually visible edges* is called the *visible area* for the pair of edges. We use a fractured representation of the layout geometry which we refer to as the *canonical representation* [7]. This representation uses the *minimum number of rectangles to cover layout area between mutually visible parallel edges of the geometry such that each polygon has at most one neighbor per edge and each edge is either fully covered by a neighbor or not covered at all*. Fig. 2 explains the idea of the canonical representation and shows the various types of edges in the final canonical representation.

The canonization algorithm works in two phases. The *split phase* creates the edges for the canonized geometry and is followed by the *polygonization phase,* where polygons used in the final representation are created.

Split Phase

The *split phase* starts by enumerating the various angles in the geometry. The angles corresponding to the *pseudoedges* introduced to represent curves are not considered in this enumeration. A pair of scanline sweeps is done for each angle in the geometry starting with the *split scan* which finds interedge visibility of edges that are perpendicular to the direction of scan. The edges that are perpendicular to the current direction of scan are marked as *active edges* while the others are marked as *inactive edges*. *Pseudoedges* are always represented as *inactive edges*. The components of the *inactive edges*, on the scanline, are considered, as the scanline moves, so that their contribution in *shadowing* the *active edges* is accounted. For any two *mutually visible active edges*, the lines orthogonal to the *active edges* at the boundary of the *visible area* are entered as *connecting edges* and the pair of *active edges* are split so that a junction is formed at their intersection with the *connecting edges*. The insertion of such *connecting edges* creates many overlapping edges in the edge database. A second *clean scan* in a direction orthogonal to the *split scan* to removes any overlap between the added *connecting edges* and the existing edges, if any, in that direction.

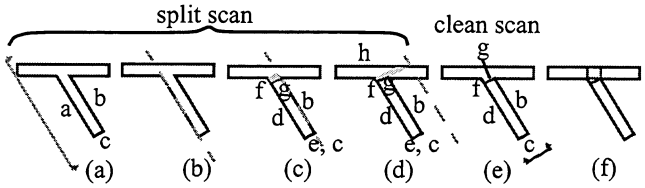

Fig. 3: example demonstrating the partitioning algorithm for fluidic geometries. (a)-(d) Different stages of a split scan, (e) result of a clean scan in the orthogonal direction, (f) final partitioned geometry

The application of splitting algorithms is shown in Fig. 3. When the scanline moves reaches *active edge b* (Fig. 3(c)), it creates two *connecting edges g* and *e*. It also splits edge *a* into edges *d* and *f*. While edge *d* ends due to its complete overlap with edge *b*, edge *f* is dragged along with the scanline. An *active edge* or parts of an *active edge* are removed from the scanline only if they overlap with an *inactive edge* or if they overlap with an *active edge* which is also an *external edge*. Finally the edge *f* gets *shadowed* by the *inactive edge h* (Fig. 3(d)) and hence no other *connecting edges* are formed. A *clean scan* is then performed, in a direction perpendicular to the *split scan*, to detect the overlap between edge *e* and *c* and removes edge *e* (Fig. 3(e)). While removing edges, an *external edge* is kept in preference to a *connecting edge*. Fig. 3(f) shows the final partitioned structure after scans in all the directions are completed.

Polygonization phase

A modified version of tree-link analysis [9] is used to polygonize the edge database. First, the edges of the canonized geometry (Fig. 4(a)) are represented as branches of a graph with the vertices corresponding to the nodes of the graph (Fig. 4(b)). Any consecutive trivial branches of the graph are merged together for simplicity (for e.g. edges g, l, m in Fig. 4(a) are merged to form branch g in Fig. 4(b)). Next, the incidence matrix of the graph is manipulated to obtain the cutset-loop matrix the columns of which give fundamental loops in the graph (Tables 1-3). Since the graph is not a

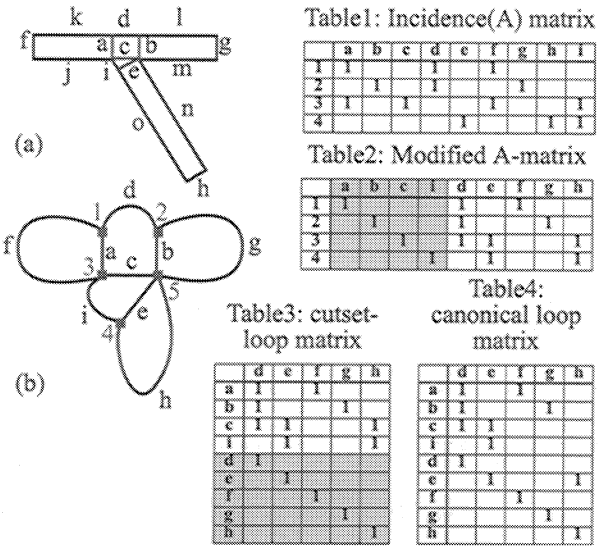

Fig. 4: Example explaining polygonization algorithm. (a) edge database after split phase, (b) the reduced graph. The tables 1-4 show the steps from A-matrix to the final canonical matrix

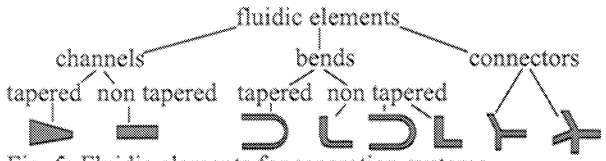

Fig. 5: Fluidic elements for separation systems

directed graph, the only operations used in the manipulation of incidence matrix are modified to be XORs of rows and exchanging of order if columns. Finally columns of the cutset-loop matrix are checked against each other to see if XOR of one column with another produces a loop with a smaller area. In the example column h of Table 3 is XOR'ed with column e to get the new column h in Table 4 because the operation reduces the area of the polygon formed by column h. The columns of the resulting canonical-loop matrix give the canonical loops, i.e. loops that are not contained by other loops, and hence the polygons of the canonical representation. A sparse matrix implementation is used for the matrices and the algorithms to reduce storage and time complexity.

ELEMENT RECOGNITION

Various approaches for fluid transport in microchannel-based systems have been explored. The prototype implementation presented here focusses on electrokinetically driven fluidic separation systems because of its wide use in μTAS. The elements constituting such systems can be classified into *channels*, *bends* and *connectors* (Fig. 5). The recognition algorithm uses the neighbor and geometry information of each of the polygons, generated from the canonization routine, to detect the elements. The polygons are first sorted using the number of neighbors. *Channels* and *bends* have only two neighbors while *connectors* may have more than two neighbors (three neighbors for *Y-connector* and four neighbors for *cross-connector*). The two neighbor polygons are then classified based on the direction of turn from one neighbor to the other. *Channels* produce no change in the flow direction while *bends* introduce change in flow direction. The *channels* and *bends* may be either *tapered* (width at two ends not equal) or *non tapered* (width same along the axis). The *channels* and *two port connectors* can be further classified as *wide* or *narrow* depending on their width. The *turns* are classified as *tapered* or *non tapered* based on the number of curves used in the polygon and by checking the center coordinates of the curved edges. A *non tapered turn* will have only two curved edges which will be concentric.

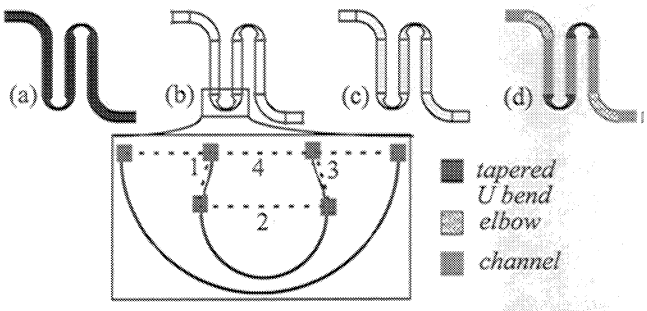

Fig. 6: Example demonstrating the extraction steps. (a) input layout, (b) edge database after *split phase* showing the *pseudoedges* for a single *turn*, (c) after *polygonization*, (d) final recognized geometry.

The recognized geometry together with all the neighbor information is then used to generate the netlist using schematic models.

RESULTS

This section presents some results highlighting the usefulness of the μTAS layout extractor.

Serpentine channel

Fig. 6 uses a serpentine channel using *tapered turns* [10] to highlight the various steps in extraction. The input layout (Fig. 6(a)) is first converted into a canonized edge database (Fig. 6(b)) in the split phase of the *canonization* routine. The inset of Fig. 6(b) shows the *pseudoedges* for a single *tapered turn*. The *pseudoedges* (represented by dotted lines) 1, 2 and 3 form the inner curve of the turn and the *pseudoedge* 4 represents the outer curve of the turn. Fig. 6(c) and (d) show the geometry after *polygonization* and recognition respectively. The final recognized polygons contain the element type, geometry information and the neighbor information and hence can be used to generate the extracted schematic.

Parallel mixer

Fig. 7(a) shows the layout of a parallel mixer identical to the one described in [1]. The sample and the buffer added to the respective reservoirs are electrokinetically driven so that they mix in the merged channels. Detectors are placed near the waste reservoir to observe the mixing ratio. The extracted schematic (Fig. 7(b) was simulated using the resistance-based models from ARCHITECT [2] to get the mixing ratio in the detection channels (Fig. 7(c)). The results (Fig. 7(c)) are compared with actual data in [1] and have very small differences which can be attributed to parasitics, similar to the ones for suspended MEMS described in [11], which will be investigated in future. This example demonstrates the usefulness of the μTAS layout extractor in capturing the behavior of a microfluidic system.

Non-Manhattan mixer

The usefulness of the extractor in verifying complex non-Manhattan μTAS designs is demonstrated using the layout (Fig. 8(a)) of a mixer similar to the D32 mixer in [3]. The extractor successfully recognized the different channels and connectors in the layout. The extracted schematic (Fig. 8(b)) was simulated using resistance-based models similar to [3]. Note that the *connectors* and the *wide channels* have been currently represented using a resistance network as suggested in

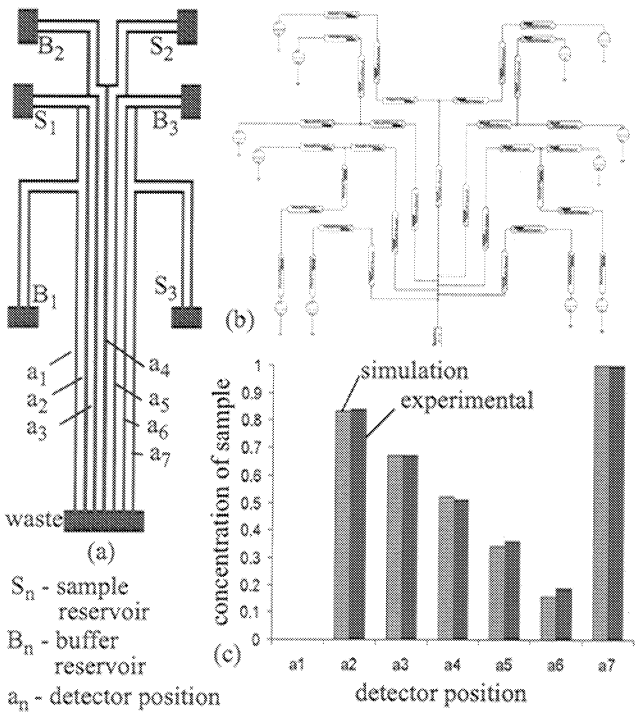

Fig. 7: (a) Parallel mixer [1] layout, (b) extracted schematic using ARCHITECT [2] models, (c) results from the simulation of extracted schematic compared with experimental data [1]

[3]. The sample to buffer ratio in each of the detector positions A1-4 were found to be 6.12, 12.5, 3.6 and 0.36 respectively.

CONCLUSION

This paper reports a prototype implementation of a μTAS layout extractor. Computationally efficient algorithms for representation and recognition of microfluidic elements from the layout geometry have been detailed. The results presented highlight the usefulness of the extraction tool in accurately constructing a schematic from a given layout. The verification methodology, using extraction, presented here will help reduce the verification cycle of μTAS designs.

ACKNOWLEDGEMENTS

This research effort is sponsored by the Defense Advanced Research Projects Agency under the Air Force Research Laboratory, Air Force Material Command, USAF, under grant number F30602-01-2-0587 and in part by National Science Foundation (NSF) Award CCR-9901171.

REFERENCES

[1] S.C. Jacobson, T.E. McKnight, J.M. Ramsey, "Microfluidic Devices for Electrokinetically Driven Parallel and Serial Mixing," *Analytical Chemistry*, 1999, vol. 71, pp. 4455-9.

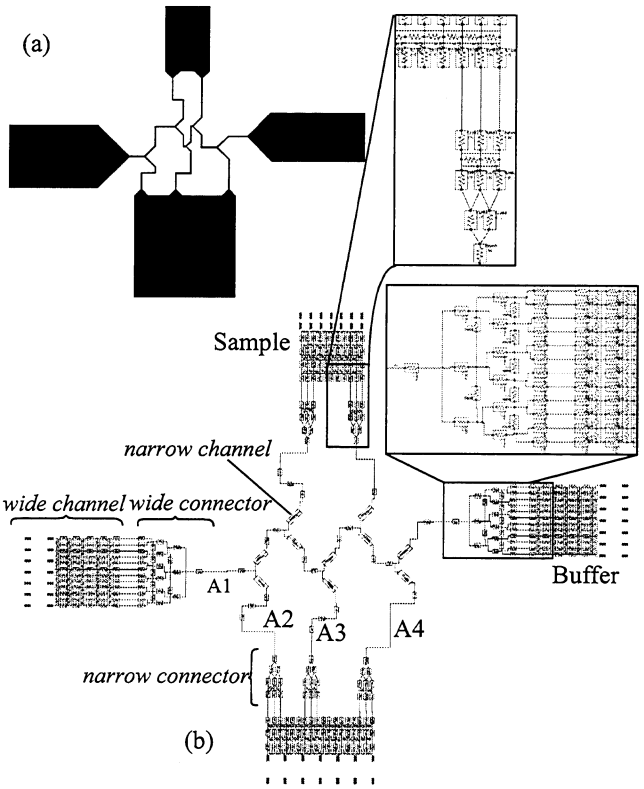

Fig. 8: Example of a non-Manhattan layout extraction. (a) Layout of a mixer similar to D32 mixer in [3], (b) extracted schematic using resistance-based models similar to [3].

[2] J.C. Harley, R.F. Day, J.R. Gilbert, M. Deshpande J.M. Ramsey, S.C. Jacobson, "System Design of Two Dimensional Microchip Separation Devices," *μTAS 2001*, pp. 63-5.

[3] S.B. Cheng, C.D. Skinner, W. Allegretto, D.J. Harrison, E. luid Mixing Design: Comparison of Simple versus Complex Modeling Methods," *μTAS 2001*, pp. 617-8.

[4] R. Qiao, N.R. Aluru, "A Compact Model for Flowrate and Pressure Computation in Micro-fluidic Devices," *MSM '02*, pp. 58-61.

[5] M.A. Maher and H.J. Lee, "MEMS Systems Design and Verification Tools," *Proc. SPIE Smart Structures and Materials '98*, pp. 40-48.

[6] N.R. Swart, "A Design Flow for Micromachined Electromechanical systems," *IEEE Design and Test of Computers*, vol. 16, No. 19, 1999, pp. 39-47.

[7] B. Baidya, S.K. Gupta, T. Mukherjee, "An Extraction based Verification Methodology for MEMS," *J. MEMS*, vol. 11, no. 1, pp. 2-11

[8] A.S. Glassner, *Graphics Gems*, Academic Press, 1990.

[9] L.T. Pillage, R.A. Rohrer, C. Visweswariah, *Electronic Circuit and System Simulation Methods*, McGraw-Hill.

[10] J.I. Molho, et. al. "A low dispersion turn for miniaturized electrophoresis," *Sensor & Actuator Workshop 2000*, pp. 132-7.

[11] B. Baidya and T. Mukherjee, "Layout Extraction for Integrated Electronics and MEMS Devices," *Transducers '01*, pp. 280-3.