Escher--A Geometrical Layout System
For Recursively Defined Circuits

Edmund Clarke [*], Yulin Feng [**]

* Department of Computer Science, Carnegie-Mellon University, Pittsburgh

** Department of Computer Science, University of Science and Technology of China, Hefei

ABSTRACT: An Escher circuit description is a hierarchical structure composed of cells, wires, connectors between wires, and pins that connect wires to cells. Cells may correspond to primitive circuit elements, or they may be defined in terms of lower level subcells. Unlike other geometrical layout systems, a subcell may be instance of the cell being defined. When such a recursive cell definition is instantiated, the recursion is unwound in a manner reminiscent of the procedure call copy rule in Algol-like programming languages. Cell specifications may have parameters that are used to control the unwinding of recursive cells and to provide for cell families with varying numbers of pins and other internal components. We illustrate how the Escher layout system can be used with several nontrivial examples, including a parallel sorting network and a FFT implementation. We also briefly describe the unwinding algorithm.

1. Introduction

Many circuits such as sorting networks, hardware multipliers, and FFT implementations can be described by recursive geometrical patterns. Some layout languages provide support for recursion ([3], [6], [7] and [10]); however, in all such systems familiar to us the circuit description is textual rather than geometrical. We believe that it is more natural to describe complicated circuits geometrically, rather than by giving a textual description and requiring that a program figure out the details of the layout. Some circuit editors have powerful iteration operators that can be viewed as implementing a form of tail recursion [4], but none allow full recursion. We have implemented a geometrical layout system (called the Escher System) in which recursive patterns can be specified directly and then instantiated to obtain layouts for complex circuits automatically. Figures 2-3 and 3-4 were generated by our system from recursive patterns.

An Escher circuit description is a hierarchical structure in which the basic building blocks are cells, wires, connectors between wires, and pins that connect wires to cells. Cells may correspond to primitive circuit elements such as NAND gates and latches, or they may be defined in terms of lower level subcells, which are defined in terms of even lower level subcells, etc. By using the Escher system, a number of primitive cells can be connected together in complex geometrical pattern to describe the layout for a large and intricate circuit. Designers do not need to worry about the absolute sizes and positions of various circuit components; only the topological relationships are important. Moreover, the system is completely interactive. Circuit diagrams are constructed using a pointing device ("mouse") and tablet.

Although many circuit editors provide a set of features similar to the ones that we have just listed, our system is unique in that a subcell may, in fact, be instance of the cell being defined. When a recursive cell definition is instantiated, the recursion is unwound in a manner reminiscent of the procedure call copy rule in Algol-like programming

languages. Cell specifications may have non-negative integer parameters that are used to control the unwinding of recursive cells and to provide for cell families with varying numbers of pins and other internal components. While the notion of parameterized cell specifications is quite common in textual hardware description languages, we believe that it has not been previously used with graphical circuit editors and, therefore, may be of independent interest.

2. Conventions for Specifying Recursive Circuit Diagrams

As an example of how the Escher system might be used, we consider the problem of laying out the Tally circuit described in [8] and also in [9]. This circuit has n inputs and n+1 outputs. The k-th output will be high and all other outputs low, if exactly k of the inputs are high. Figure 2-1 gives the Escher version of a recursive definition for the Tally circuit. The circuit for the base case, Tally(1), is shown in Figure 2-2. Both of these diagrams must be supplied by the user.
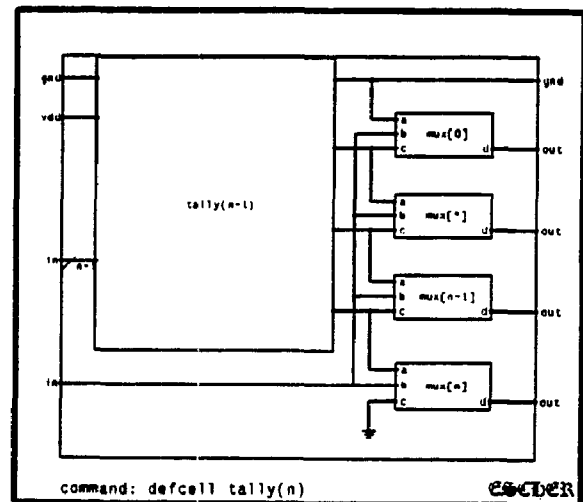

Figure 2-1: Recursive Pattern for Tally(n)

In these diagrams there are two kinds of cells: *Basic* cells that cannot be refined further (like the two input multiplexers), and *Composite* cells that contain other cells, wires, and connectors (like the recursive occurrence of Tally(n-1)). The cells that are directly contained within a composite cell are its *subcells*. Sometimes several subcells $S_1, S_2, \ldots, S_n$ are instances of the same cell C. In this case we say that C is the *source* of each of the S_i's.

Since the specification is parameterized by n, some abbreviations are needed to represent groups of lines and subcells that depend on n. When a definite value is provided for n, each such abbreviation in the specification may be evaluated.

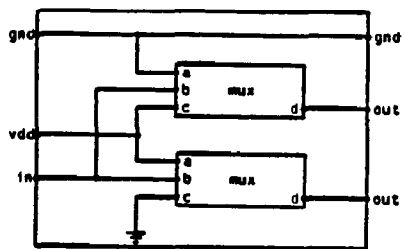**23rd Design Automation Conference**

Figure 2-2: TALLY(1), base case for the TALLY circuit

Groups in Escher are somewhat like one dimensional arrays in programming languages. A *group* is a horizontal or vertical array of identical cells with the appropriate interconnecting wires. The subcells of a group may be either basic cells or composite cells. They are distinguished from one another by an integer index, which increases from left to right in the case of a horizontal array or from top to bottom in the case of a vertical array. The initial and final values of the index may depend on a parameter of the cell containing the group; however, the increment must be a fixed positive integer. A group whose length depends on an undetermined parameter is represented by three subcells, one for the first subcell, one for the last subcell, and one in the middle with index "*" to represent all of the remaining subcells. Thus, the "*" serves exactly the same function in our formal specification that the ellipsis "..." serves in an informal specification. A number appearing after the "*" represents an index increment; when the "*" appears alone, the default value for the increment is 1. In the Tally example (Figure 2-1) there are a total of (n+1) multiplexers, the MUX[n] and MUX's with indices from 0 to n-1 in a group. When a group of subcells is specified, it is only necessary to give the the position of the first and the last subcells of the group with respect to some other part of the circuit. When the containing cell is instantiated and all of the parameters of the group are fixed, this information is sufficient to determine the position of each of the subcells of the group.

Finally, Escher uses a short diagonal mark on a wire to represent a group of wires. An expression associated with the mark indicates how many lines are in the group. We call such groups of wires, *buses*, and the associated expression: the *bus width*. In Figure 2-1 there are two buses, and each represents n-1 wires. We also use the convention that a wire connected to a subcell with index "*" actually represents the same number of wires as the number of omitted subcells.

Examination of the recursive specification for the TALLY circuit immediately shows how it works. Each multiplexer has three inputs labeled a, b, c and one output labeled d. If b is high, the output d selects the value c; otherwise, it selects the value a. It is easy to see that the base case is correct. We assume that TALLY(n-1) is correct and that k of the first n-1 inputs are high. By the induction hypothesis, the k-th output of TALLY(n-1) is high. If the nth input is also high, then all of the selector inputs of the multiplexers will be high, so each of the MUX's with index in the range from 0 to n-1 will select as its output the value of its c input, while the output of MUX[n] will be low. Thus, the $(k+1)^{th}$ output (counting from bottom to top) of TALLY(n) will be high and the other outputs will be low. A similar discussion can be used for the case in which the nth input of TALLY(n) is low.

After we instantiate the TALLY circuit with a given value, for example, n=6, the Escher system will automatically unwind the recursive specification into the circuit diagram shown in Figure 2-3. A final phase (that has not been completed) will compact the circuit diagram produced by the Escher system in accordance with a set of design rules appropriate to the transistor technology used to fabricate the chip.
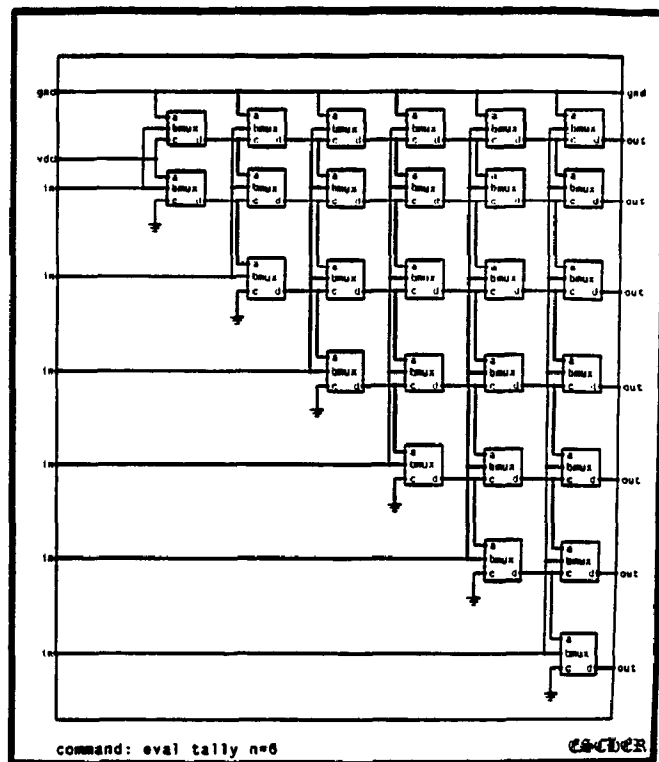

Figure 2-3: Tally(6) Instantiation

3. Divide and Conquer Circuits

The simplest recursive circuits have only a single recursive subcircuit. This case is somewhat like tail-recursion in programming languages and is relatively easy to implement. The Tally circuit in Figure 2-1 is an example of such a recursion. Unfortunately, not all recursive circuits have such a simple structure. Many interesting circuits are based on a *divide and conquer* strategy in which a complicated task is realized by a number of subcircuits each of which is a recursive instance of the circuit being defined. Adders, multipliers, sorters, FFT circuits, etc., can all be structured in this manner. Figuring out by hand an appropriate layout for an instance of such a circuit can be quite tricky. Once the recursive structure of the circuit has been determined, the Escher system may be used to unwind a particular instance of the circuit.

To illustrate these ideas we show how the Escher system can be used to obtain a layout for a simple *parallel sorting network* [5]. If n is a power of 2, this network will sort a sequence of n k bit numbers into increasing order. The standard divide and conquer approach is to sort the first half and the second half in parallel and then merge the two sorted sequences. The Escher specification for such a circuit is shown in Figure 3-1. Note that every bus width number here means the number of k-bit wires.

The Merge cell can also be defined recursively. To merge two sequences "a" and "b", we merge the even-indexed elements of "a" with the odd-indexed elements of "b", and the odd-indexed elements of "a" with the even-indexed elements of "b". The outputs of the two half-size merging circuits are sent through an array of comparators. Each comparator "CMP" sorts two k-bit numbers in order. Figure 3-2 gives the recursive definition of Merge(n). Pass(n), shown in Figure 3-3, contains only wires and is used to separate the even-indexed inputs and the odd-indexed inputs.
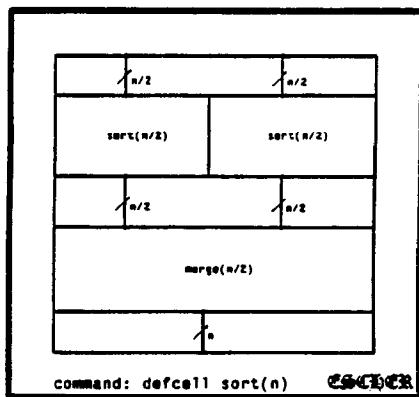
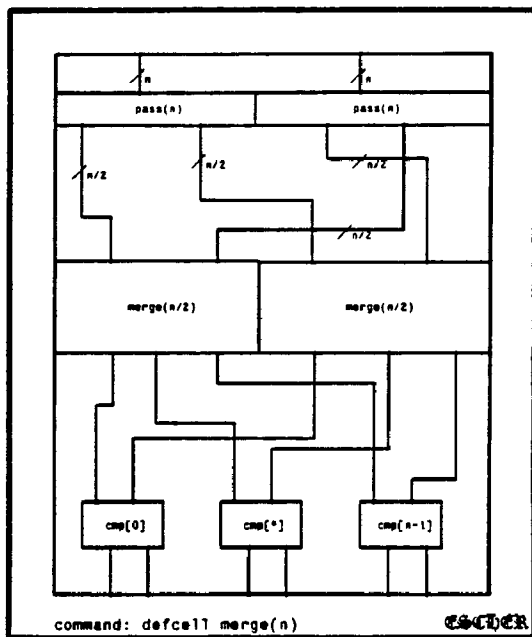Figure 3-1: Recursive pattern for Sort(n)

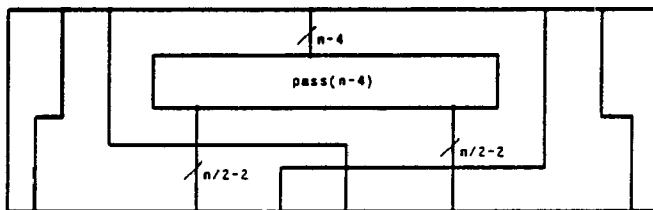Figure 3-2: Recursive pattern for Merge(n)

Figure 3-3: Recursive pattern for connections Pass(n)

If we instantiate the recursive specification shown in Figure 3-1 with n = 16, our system automatically generates the pattern shown in Figure 3-4.
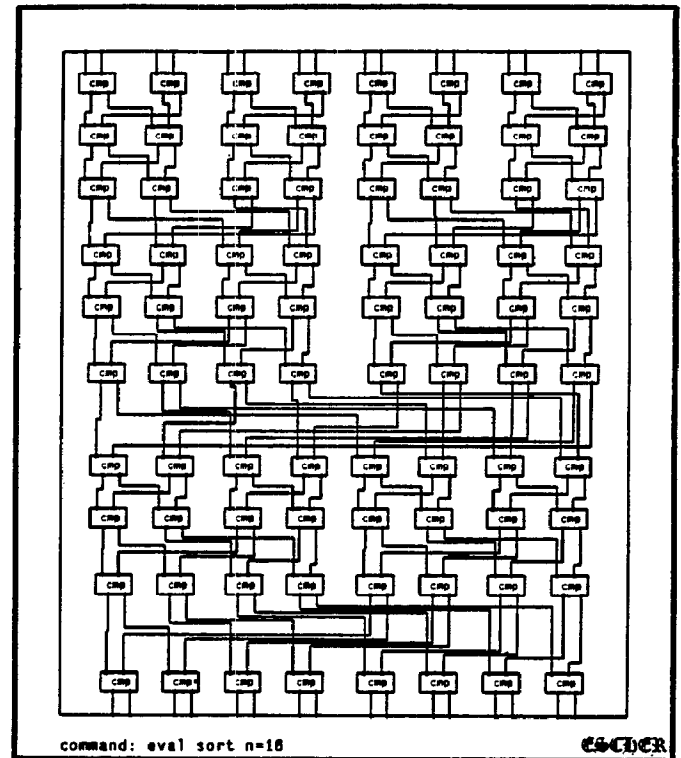

Figure 3-4: SORT(16) Instantiation

In the full version of this paper [1] we also show how Escher can be used to obtain layouts for a recursive discrete Fourier transform and for the *parallel prefix circuit* described by Fischer and Ladner in [2].

4. An Overview of the Unwinding Algorithm

A cell is represented in the Escher system by a record structure consisting of three fields, the AttributeList, the PointNet, and the SubCellList. The AttributeList contains the name of the cell, its parameter list, and its position (TopY, BottomY, RightX, LeftX) with respect to a fixed coordinate system. The PointNet is used to keep track of the different kinds of points (pins, bends, connectors, vias, transistors, etc.) and their locations. Each point is represented by a record structure that specifies its type, its coordinates PosX and PosY, and how it is connected to the other components of the cell. All of the points in a cell are linked together in an undirected graph structure called the PointNet. From each point in the cell it is possible to find the next connected point in a vertical or horizontal direction by following the appropriate link in the PointNet. The SubCellList contains a descriptor for each component subcell. A subcell descriptor has a pointer to the source of the subcell, an assignment of symbolic expressions for any parameters of the source cell, and information on the position and orientation of the subcell (i.e., whether it has been flipped or rotated). Subcells in a group are linked together in a circular list. Some information in the AttributeList of the source cell, like the cell name, is also duplicated to prevent unnecessary searching.

A recursive circuit specification is unwound into a tree structure in which nodes correspond to cells, and one node is a son of another if the cell corresponding to the first node is a subcell of the cell corresponding to the second. Thus, a cell will appear at level i in the tree if it is a subcell of a cell that appears at level i-1. A layout is generated from the tree in a *bottom-up* fashion in which layouts are determined for all of the sons of a node before laying out the node itself. To accomplish this task it may be necessary to move various circuit components in order to make room for components generated at lower levels. The algorithms

that Escher uses for this purpose are discussed in detail in the full version of this paper [1].

When we unwind a recursive Escher specification, we must be careful not to duplicate steps if we encounter the same cell more than once as we traverse the tree. For example, if we use the naive algorithm to unwind SORT(4), we have to uuwind SORT(2) twice and MERGE(2) once; when we unwind MERGE(2), we must unwind PASS(2) twice, MERGE(1) twice, and CMP twice. In fact, with the naive algorithm it is possible to create examples in which the number of duplicated steps will be exponential in the size of the original Escher specification.

Instead, Escher uses a directed acyclic graph structure to represent the nesting of subcells. We call this data structure the *Subcell Nesting Graph or SNG*. Since each subcell corresponds to at most one node in the SNG, it is only necessary to unwind a given subcell once. The graph for SORT(4) is shown in Figure 4-1. Note that each of the subcells SORT(4), SORT(2), CMP, MERGE(2), MERGE(1), and PASS(2) is represented uniquely this time.

The unwinding algorithm consists of two phases. In the first phase we evaluate all of those expressions that depend on the parameters of the cell and create the SNG. Expressions may appear in the specifications of groups and buses, and they may be used as parameters of lower level subcells. After we have figured out the exact number of subcells in a subcell group, we may have to enlarge the cell to obtain enough space for the omitted subcells in the group. An algorithm for this purpose is described in detail in the full paper. Next, we copy the subcells into the cell. After a cell has been evaluated it will be linked to its source cell in the SNG. The SNG for cell CL(V) will not be complete until all of its descendant subcells have been processed in this manner.

The second phase in the unwinding process is a depth first traversal of the SNG. When all of the subcells of a cell in the SNG have been unwound, we replace each subcell with its source body and mark the cell as unwound. It may be necessary to enlarge a cell to obtain enough space for filling in the subcell bodies. An algorithm for this step is given in the full paper. The last step in this phase is to eliminate jogs in wires that result from these substitutions. A technique for doing this is also given in the full paper. Finally, some simple compaction algorithms are used to shorten wires and move subcells closer together.
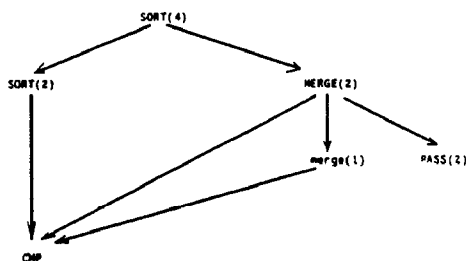

Figure 4-1: Directed Acyclic Graph for SORT(4)

5. Conclusion and Directions for Future Research

We list below some of the problems with the current system that we hope to address in a future version:

- **Multiple parameters.** As currently implemented, the Escher system only permits cell specifications with a single recursive parameter. A number of interesting examples can be specified most naturally by using multiple recursive parameters. It should be fairly easy to modify the current implementation so that multiple parameters are permitted.

- **Compaction and optimization.** The layouts produced by our system frequently contain long wires and have area that grows more rapidly with the recursion depth than necessary. Although we have implemented some simple compaction algorithms, we believe that this problem requires much more thought. It may be possible to design compaction algorithms that take advantage of the hierarchical structure of Escher specifications. However, the simple algorithms that have already been implemented do not make use of this information.

- **Combined textual and geometric description.** For certain applications like simulation a textual circuit description may be quite useful. We envision a VLSI design system with multiple *windows* which would permit *both* textual and geometric descriptions of circuit components. One window would contain a geometrical representation of the circuit like the one described in this paper. Another window would contain a representation of the circuit in an appropriate (textual) hardware description language. The textual description could be used directly for simulation, verification, etc. A change in the geometrical description would be automatically reflected by a corresponding change in the HDL representation. The dual representation would provide access to the best features of both types of design systems.

REFERENCES

1. E. Clarke and Y. Feng . Escher-- A Geometrical Layout System for Recursively Defined Circuits. CMU-CS-85-150, Department of Computer Science, Carnegie Mellon University, July, 1985.

2. M. Fischer and R. Ladner. "Paralllel prefix computation". *Journal of the ACM 27*, 4 (1980).

3. S.M.German, K.J.Lieberherr. "Zeus: a language for expressing algorithms in hardware". *Computers* (1985).

4. J.Ousterhout. "Caesar: An interactive layout editor for VLSI design". *VLSI design* (Fourth Quarter 1981), 34-38.

5. D.E.Knuth. *The art of computer programming.* Volume : *Sorting and searching.* Addison-Wesley, 1973.

6. R.J.Lipton, S.C.North, R.Sedgewick et tal. ALI: a procedural language to describe VLSI layouts. 19th design automation conference, IEEE, 1982, pp. 467-474.

7. W.K.Luk, J.E.Vuillemin. Recursive implementation of optimal time VLSI integer multipliers. VLSI design of digital systems, ed. F.Anceau & E.J.Aas, 1983, pp. 155-168.

8. C.A..Mead, L.A.Conway. *Introduction to VLSI systems.* Addison-Wesley, 1980.

9. Mary Sheeran. muFP-- An algebraic VLSI design language. PRG-39, Oxford University Computing Lab., November, 1984.

10. P.Henderson. Functional geometry. Symposium on LISP and functional programming, ACM, 1982, pp. 179-187.