A NOTATION FOR DESCRIBING MULTIPLE VIEWS OF VLSI CIRCUITS

Jean-Loup Baer, Meei-Chiueh Liem, Larry McMurchie,
Rudolf Nottrott, Lawrence Snyder, Wayne Winder
NW Laboratory for Integrated Systems
Department of Computer Science
University of Washington
Seattle, WA 98195

ABSTRACT:

A declarative hierarchical notation is introduced that allows the parametric representation of entire families of VLSI circuits. Layout, schematic diagrams and network structure are all accommodated by the notation in a way that emphasizes common elements. The notation is the basis of a structured environment for developing design generators as well as capturing design expertise.

1 Introduction

The application of full custom integrated circuit design to architectural problems requires a multitude of CAD techniques. Not only does the designer have to specify and simulate networks of components, but he also has to deal with the physical layout of those components. It seems apparent that the sheer volume of circuitry required for even the simplest architectures mandates an approach that utilizes previously captured circuit designs. One method of capturing expertise about entire families of circuit design is through the use of design generators.

We define a design generator as a program that produces instances from a family of circuit designs. The input is a problem-specific set of parameters; the most common output is the layout of the mask layers. Several systems have been developed for the construction of layout generators (e.g. [Bamji 85]).

Although the layout is the final means of specifying a circuit, it is simply too detailed and technology-dependent to efficiently capture design expertise. Other more abstract views or representations are necessary if one wishes to capture the design at a higher level and could likewise aid the user of the generator as he constructs a complex design from a number of generated modules. A behavioral model of the generated circuit, for example, could be used in a functional simulator. A network of devices could be used in a switch-level or analog simulator. A schematic diagram could be used for documentation purposes.

Our approach to developing a design generator environment is one that will support such multiple representations. In order to efficiently capture design expertise, the correspondence between such representations will be made evident.

[1]Funded by the Defense Advanced Projects Agency (Dod/DARPA) under Contract MDA 903-85-K-0072.

1.1 Design Generator Model

Given the need to support multiple representations, what might an ideal environment for developing generators look like? One such environment is shown in Figure 1. A frontend processes input parameters and performs a variety of manipulations on the input.

The second element of the environment is a database or "model" which guides the generation process. One element of the model is a list of the instance-specific input parameters provided by the frontend – the "catalog" file. The other two parts of the model contain information about the entire family of circuit designs. The declarative notation specifies how the various representations of the circuit are to be assembled. The leaf cells are the representation-specific primitives referenced in the notation. In the case of the layout, the leaf cells are simply the mask geometries. In the case of the network representation, the leaf cells may be behavioral models or subnetworks of devices.

The division of information between the declarative notation and the leaf cells is arbitrary. Most of the information may reside in a few leaves, as in the case of a network description of a PLA that references two behavioral models – one for the "and" and one for the "or" plane. Another network description for the same family of PLAs may reference all transistors in the design.

Finally there are the circuit generators themselves, which take the information in the model and produce the representations. These programs are independent of the particular circuit family, which is described entirely in the declarative notation and the leaf cells. The burden of circuit construction and analysis should be placed on the generator programs. The idea is to keep the notation simple and therefore free the

designer from as much detail as possible.

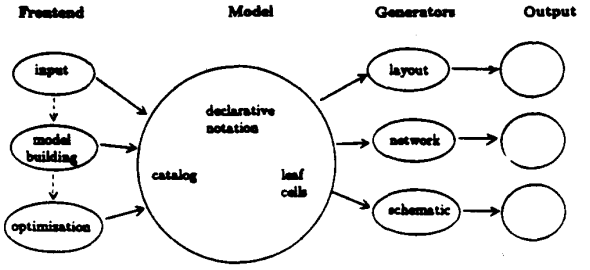

Figure 1: Circuit Design Generation

The key to the scheme we have proposed is the development of a declarative notation. In the remainder of this section we look at languages that describe multiple representations of circuits and outline our own goals. In Section 2, the salient features of the notation will be presented. In Section 3, the generator programs will be described in the context of an example.

1.2 The Multiple Representation Problem

The declarative description is an example of a hardware description language (HDL). HDL's describe to varying degrees functional behavior, network topology and geometrical structure.

One effort to represent behavior, topology and geometrical structure is *Zeus* [Lieberherr 83]. *Zeus* is a strongly typed procedural language that allows specification of both signal behavior as well as floorplanning information.

In the functional programming language μFP [Sheeran 83] the behavior specification implies a floorplan and routing. Because of the algebraic properties of μFP, transformations can be made that retain identical functional behavior and allow floorplans and routing to be modified. Another effort in this direction is CADIC [Becker 87], a system that employs an algebraic notation for combining behavioral and routing primitives with geometrical operators. μFP and CADIC are both elegant synthesis languages with interesting mathematical properties. Our assumption, however, is that the structure of the circuit family is already characterized: we desire an easy-to-write notation that captures this structure.

1.3 Goals in the Development of the Notation

The declarative notation has four major goals:

1. Network Topology and Geometric Structure
   The notation must allow a hierarchical specification of both network topology and geometrical structure in a way that emphasizes the correlation between the two representations. In order to make this correlation more apparent we avoided explicit representation of behavioral information, leaving it to be specified in leaf cells. In contrast, languages such as *Zeus* include behavior explicitly.
2. Representation of Entire Circuit Families
   As part of the environment in Figure 1, the notation must allow sufficient parametrization to describe entire circuit families. At the minimum this requires loops and conditionals.
3. Simplicity and Naturalness
   It is important that designers be able to capture circuit designs compactly and expressively. A declarative (as opposed to procedural) notation is the obvious choice.
4. Technology Independence
   One of the major problems with many powerful layout systems is the technology dependence that can creep in. Although elements of technology can be introduced into the notation (e.g. in a network of MOS devices), a mechanism should be present for hiding such details in the leaf cells. In this way the circuit features common across technologies can be emphasized.

2 Features of the Notation

2.1 Overview

The declarative notation consists of two parts: (1) a declaration, which includes the name of the circuit, the type of the representation, a list of parameters, and the names of leaf cells, and (2) a collection of statements used to describe an entire circuit family. A description can be regarded as a set of objects (leaf cells or abstract objects) and a set of relations among these objects.

The syntax of this high level description is designed to have expressions similar to those of the "C" programming language. The Extended Backus Naur Formalism (EBNF) definition for the declarative description can be found in [Liem 86].

2.2 Declaration

A simple declaration has the form:

```
NAME <circuit_name>;
TYPE <rep_type>;
PARAMETER <parameter_list>;
LEAF CELLS <cell_list>;
```

`<rep_type>` is a combination of LAYOUT, SCHEMATIC, or NETWORK. `<parameter_list>` is a list of inputs to the description. The values of these parameters are obtained from the catalog file and serve to make the description instance-specific. `<cell_list>` names all the leaf cells that are used in

the description. The leaf cell is the lowest level object in the hierarchy of a description.

2.3 Objects

Leaf cells are the primitive objects on which the operators are applied and out of which abstract objects are built. For example, a leaf cell can be the drawing of a "nand" gate used for the schematic description of a decoder or it can be the physical layout of a half-adder used for the layout description of a multiplier.

Abstract objects are created by combining leaf cells. An alias of a leaf cell, an array of leaf cells or a composition of heterogeneous leaf cells are each abstract objects. Moreover, an array of abstract objects, a group of heterogeneous abstract objects or a composition of these two are also abstract objects. An abstract object can also be defined recursively.

The highest level of the hierarchy is a single abstract object – the circuit that the designer intends to describe. At the lowest level, the circuit is a collection of leaf cells. The description of a circuit is thus hierarchical in nature: each abstract object is specified as a collection of lower level objects. Since an abstract object may be defined after it is used, the description of an object promotes "top-down" design or "stepwise refinement".

2.4 Operators

The operators which are used in our declarative descriptions can be arranged in the following groups: (1) connection, (2) arithmetic, (3) relational, (4) logical, and (5) assignment ("="). The connection operators take objects as arguments and produce objects as results. These operators, "--", "--^", "|", and "|^", are used to combine objects into more complicated abstract objects.

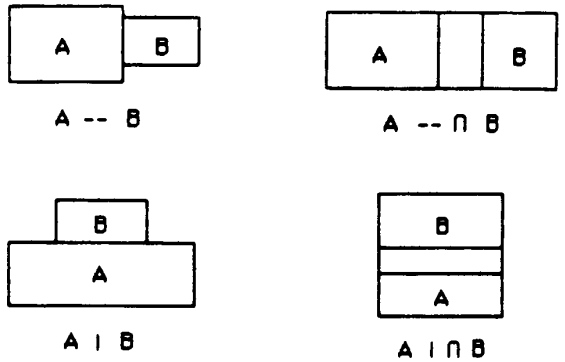

Figure 2: Geometric operators for combining objects.

For NETWORK descriptions, the connection operators are all identical and cause creation of an object made up of the argument objects connected according to the signal lists given with each object. For example, "`A[in, out] = B[in, mid] | C[mid, out]`" means that "A" is made of "B" and "C", with the first signal of "A" connected to the first signal of "B", the second signal of "A" connected to the second signal of "C", and the second signal of "B" connected to the first signal of "C".

For LAYOUT and SCHEMATIC descriptions, connection operators cause creation of an object made up of the argument objects positioned geometrically. These positions are shown in Figure 2. The amount of overlap for operators "--^" and "|^" is controlled by attributes of the leaf cells. These operators are implemented within the primitives of Coordinate Free LAP [Beckett 85].

The remaining connection operators, mirror and rotate, act as "identity" operators in NETWORK descriptions.

Arithmetic, relational and logical operators are used as in conventional programming languages.

2.5 Control Constructs

Two fundamental control constructs are provided to enhance the expressiveness of a description: IF (decision making) and looping. IF is used to specify conditions. Looping is expressed in either of two forms. The first form provides the number of times for repetition; for example, "`(-- (X) (m))`" represents m horizontally joined instances of X. The second form provides the upper bound, lower bound and step of the loop index; for example, "`(| (X[i]) (i=4..0,-2))`" represents 3 instances of X, with X[4] situated at the bottom, X[2] in the middle, and X[0] on the top.

3 Declarative Descriptions of a Decoder

This section provides examples of descriptions for three circuit representations of a family of precharged decoders with n inputs and 2^n outputs. The descriptions given are for the network, the layout and the schematic diagram. Two different descriptions are given for the network representation to illustrate the manner in which the notation facilitates the design approach of step-wise refinement.

3.1 Network Description

A network description specifies the connectivity between a number of hierarchically structured objects. At the lowest level, these objects are the leaf cells referred to by the description. Leaf cells for a network description contain Network C source code expressing the behavior of the object represented by the leaf cell. Network C [Beckett 86], or NC, is a language similar to C with additional features to specify behavior, time delays and connectivity.

3.1.1 High level

The first network description of the decoder is a very simple, high-level description (Figure 3).

```
NAME       decoder;                                  (1)
TYPE       NETWORK;                                  (2)
PARAMETER  n;          /* number of inputs */        (3)
LEAF CELLS dec_row;                                  (4)
INPUT      x, clock;                                 (5)
OUTPUT     y[2**n];                                  (6)
{                                                    (7)
decoder[x,                                           (8)
        (, (y[i]) (i=2**n-1..0)),                    (9)
        clock ]                                      (10)
    = (| (dec_row[row, x, clock, y[row]])            (11)
              (row=2**n-1..0));                      (12)
}                                                    (13)
```

Figure 3: High-level network description of an n-input decoder.

Since this first network description is at a very high level of abstraction, it uses only one leaf cell (line 4) and has two levels of hierarchy. The top of the hierarchy - "decoder" - consists of a set of 2^n interconnected "dec_row" objects (lines 11, 12).

In addition to the items in the declarative part of the layout and the schematic descriptions, network descriptions also have declarations of i/o signals (lines 5, 6) which may be either single signals or vectors The output signal "y" is declared implicitly as a vector. A vector is equivalent to a bus and may therefore have one dimension only. The number of bus signals is given in brackets. An automatic expansion of a vector takes place if necessary.

Sets of signals may be specified using a looping construct with a comma as the loop operator (line 9). For example, "(, (y[i]) (i=2**n-1..0))" is equivalent to the signal list "y[2**n-1]," ..., "y[0]".

The object "decoder" has the signals "x", "y[2**n-1]", ..., "y[0]" and "clock" in its argument list. (Note that "x" is the integer representation of a bus of binary signals.) The argument list of "dec_row" consists of the variable "row" and the signals "x", "clock", "y[row]". The value of the variable "row" depends on the particular instance of "dec_row".

```
dec_row()
{
    network trigger row_nr, in, clk;
    network         select;

    if (clk == 1 && row_nr == in) {
        select = 0;
    } else {
        select = 1
    };
}
```

Figure 4: The leaf cell dec_row, referenced in Figure 3.

The leaf cell "dec_row" is very similar to an ordinary C function (Figure 4). NC functions like this one model the behavior of some part of a circuit. The variable type "network" is a special NC data type through which signals are referenced. (Ordinary variables in the notation, such as "row", take on specific values when an instance of an object is generated. They are also passed to models as network variables.) Network variables are similar to formal parameters in a "C" function definition. The model, however, is invoked only if a variable declared as "network trigger" changes.

When the network generator operates upon the description in Figure 3, a network of 2^n instances of "dec_row" is generated. During the NC simulation of this network, the "network trigger" parameters of the model ("row_nr", "in", and "clk") receive their values from the leaf cell instances The model uses these values to compute the new value for "select".

3.1.2 Low level

The second network description of the decoder (Figure 5) is considerably more detailed than the first one. We have particularized the high level description to a precharged "nand" style decoder. The description uses three leaf cells and has four levels of hierarchy. It illustrates in more detail some of the features of the notation for network descriptions discussed in the previous example.

```
NAME       decoder;
TYPE       NETWORK;
PARAMETER  n;
LEAF CELLS inv[in, out],
           eval[clock, out],
           pre[clock, out],
           nfet[gate, source, drain];
INPUT      x[n], clock;
OUTPUT     y[2**n];
VECTOR     x_b[n], node[n];
{
decoder[(, (x[i]) (i=n-1..0)),
        (, (y[i]) (i=2**n-1..0)), clock]
        =   in_row[x, x_b]
            | (| (dec_row[row, x, x_b, clock, y[row]])
              (row=2**n-1..0));

in_row[x, x_b]
        =   (| (inv[x[col], x_b[col]]) (col=n-1..0));

dec_row[row, x, x_b, clock, sel]
        =   eval[clock, node[n-1]]
            | (| (select[row, col, x[col], x_b[col],
                 node[col],node[col-1]]) (col=n-1..1))
            | select[row,0,x[0], x_b[0], node[0], sel]
            | pre[clock, sel];

select[row, col, in, in_bar, s, d]
        =   nfet[ in, s, d],    IF (2**col) & row != 0
        =   nfet[in_bar, s, d];

}
```

Figure 5: Low-level network description of the decoder.

The description specifies a network of nfet and pfet devices in which the behavior specification is re-

duced to that of individual devices, except for the leaf cell "in". (A level of description intermediate between this and the high-level description discussed previously would be the definition of each row as a precharged "nand" function with appropriate input bits "x[col]" and "x_b[col]".)

The scope of signal names is local to an object definition. Signals within an object definition are connected by name. For example, the "sel" argument of "dec_row" is logically connected to all other occurrences of "sel" in the definition of "dec_row".

3.2 Layout Description

The layout representation in Figure 6 directs the construction of a decoder whose floorplan is shown in Figure 7. The leaf cells of the description are *Magic* files [Ousterhout 85]. The layout generator produces a hierarchy of *Magic* cells with the decoder being the root cell. These cells may be used like any other *Magic* cells, i.e. they may be viewed or edited with *Magic* or may be incorporated into a larger design.

```
NAME            decoder;
TYPE            LAYOUT, SCHEMATIC;
PARAMETER       n;
LEAF CELLS      route_l, inv[in], route_r, eval[clock],
                pre[select, clock], one, zero;
INPUT           x[n], clock;
OUTPUT          y[2**n];
{
decoder     =   in_row
                | (| (dec_row[row]) (row = 2**n-1..0));

in_row      =   route_l
                -- (-- (inv[x[col]]) (col = n-1..0))
                -- route_r;
dec_row[row]
            =   eval[clock]
                -- (-- (select[row, col]) (col=n-1..0))
                -- pre[y[row], clock],     IF row == 0
            =   eval[]
                -- (-- (select[row, col]) (col=n-1..0))
                -- pre[y[row], ];

select[row, col]
            =   one,        IF ((2**col) & row) != 0
            =   zero;
}
```

Figure 6: Layout description of the decoder.

An important feature of the layout description is the application of labels. In Figure 6 the formal parameters of the leaf cells (e.g. "in" in "inv") refer to labeled points in the cells. When "inv" is instantiated, a label is created from "x[col]" and applied to the point labeled "in". These applied labels appear as "x_0", "x_1", and "x_2" in Figure 7.

In this description all of the cells tile neatly. The notation is, however, sufficiently flexible to accommodate offset cells as well as overlapping ones. For this purpose, the alignment operators "--" and "|" are used in combination with labels present in the leaf cells. With these operators, cells are aligned so that the labels with the same name are superimposed.

A feature of the notation not shown here is the ability to reference arrays of parameters. This feature allows the construction of coded circuits like PLAs and ROMs.

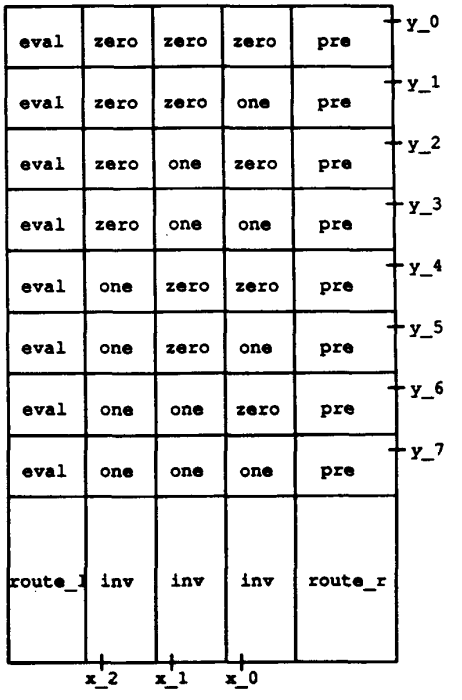

Figure 7: Floor plan of the layout of a three input decoder produced by the layout generator.

3.3 Schematic Description

The description in Figure 6 directs the construction of a schematic diagram of the decoder, as well as the previously discussed layout. Schematic output is always a picture and is intended for documentation purposes only. The schematic leaf cells are pictorial elements created with the drawing package DP [Giuse 83]. These leaf cells are combined into a single file with appropriate "placement" information in a manner similar to the layout construction.

As in the layout, labels are applied to the diagram through the use of formal parameters passed to the leaf cells.

Figure 8 shows the schematic diagram of the decoder produced by the schematic generator from the

layout/schematic description in Figure 6.

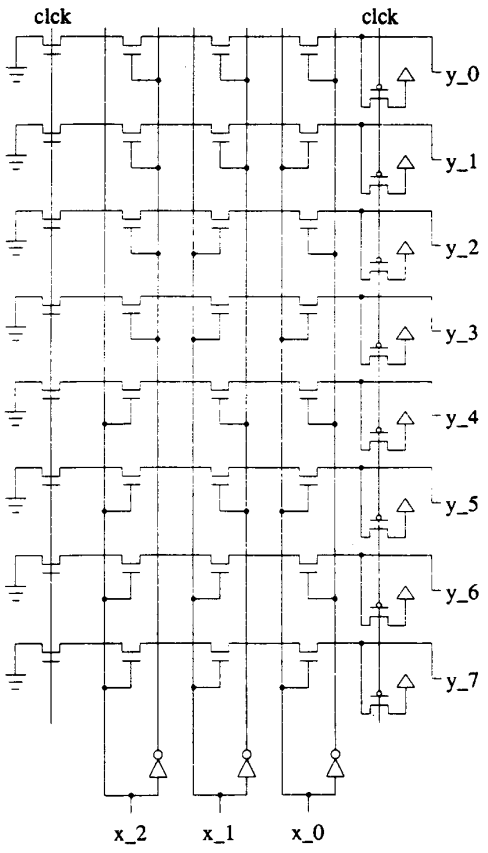

Figure 8: Schematic diagram of a 3 input decoder produced by the schematic generator from the description of Fig. 6

4 Conclusion and Future Work

The high and low level network examples illustrate the application of the notation to top-down design of circuit families. Noteworthy is the fact that both the layout and schematic diagrams are generated from the same description (Figure 6). Also noteworthy is that the low-level network description and the layout/schematic descriptions share a common hierarchy, with differences showing up at the leaf cell level: the shared objects provide a natural link between the three circuit views.

We have applied the notation to a variety of decoders, a family of NxM Baugh-Wooley multipliers, and a multiplier employing redundant binary representations of terms. A graduate VLSI design class has employed the notation in the design of modules comprising a message routing chip, in particular a variable width, variable depth FIFO. The notation has demonstrated utility both as a means of capturing information about circuits at a variety of levels, and as an intermediate database in a design generator environment.

A major extension to the notation would be the addition of a more flexible interconnect strategy for assembling the layout. In many cases the requirement of interlocking leaf cells is a severe constraint. By adding more analysis of cell borders in the layout generator, cell extensions as well as routing could be employed to place and interconnect cells.

5 Acknowledgements

Mary Bailey and Bill Beckett were an immense help in writing the code required to assemble layouts. We acknowledge the support of DARPA for this work under contract #MDA903-85-K-0072.

References

[Bamji 85] C. Bamji, C. Hauck, and J. Allen, "A Design by Example Regular Structure Generator", *Proc. of the 22nd Design Automation Conference*, pp 16-22, (June 1985).

[Becker 87] B. Becker, G. Hotz, R. Kolla, P. Molitor, and H. Osthof, "Hierarchical Design Based on a Calculus of Nets," Proc. of the 24th DEsign Automation Conference, pp. 649-653, (June 1987).

[Beckett 86] W. Beckett, "MOS Circuit Models in Network C", *Proc. of the 23rd Design Automation Conference*, pp 171-8, (June 1986).

[Beckett 85] W. Beckett, "Coordinate Free LAP", Technical Report #86-07-01, Department of Computer Science, University of Washington, (July 1986).

[Giuse 83] D. Giuse, Carnegie Mellon University, (1983).

[Lieberherr 83] K. Lieberherr and S. Knudsen, "*Zeus*: A Hardware Description Language for VLSI", *Proc. of the 20th Design Automation Conference*, pp 17-23 (June 1983).

[Liem 86] M. Liem, "Declarative Descriptions for VLSI Generators", Masters Thesis, University of Washington, Dept. of Computer Science, Technical Report 86-09-03, (June 1986).

[Ousterhout 85] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, "Magic: A VLSI Layout System", Proceedings of the 21st Design Automation Conference, pp. 152-159, (June 1984).

[Sheeran 83] M. Sheeran, "μFP - An Algebraic VLSI Design Language", Ph.D. Dissertation, Oxford University (November 1983).