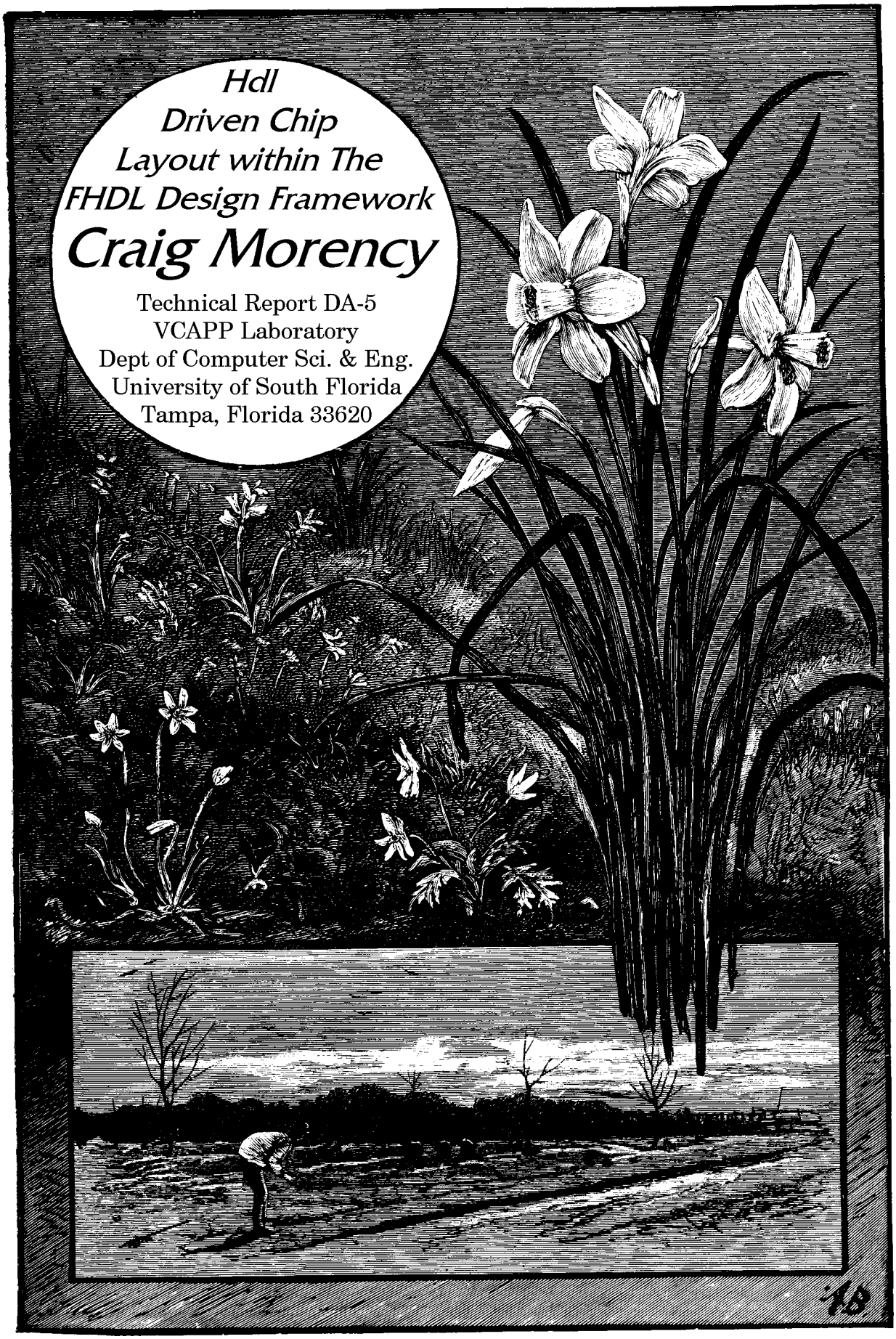

Hdl Driven Chip Layout within The FHDL Design Framework

Craig Morency

Technical Report DA-5
VCAPP Laboratory
Dept of Computer Sci. & Eng.
University of South Florida
Tampa, Florida 33620

HDL DRIVEN CHIP LAYOUT WITHIN THE FHDL DESIGN FRAMEWORK

Craig D. Morency
Peter M. Maurer
Zhicheng Wang
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620

ABSTRACT

Work is currently underway at the University of South Florida to integrate automated placement and routing with the language FHDL, for the purpose of general integrated circuit synthesis. Presented is the division of the system into two programs, which separates the tasks of layout synthesis to take advantage of CPU time and memory space. One of the over-riding principles of this work, has been to prevent any modifications of the FHDL format which would restrict the use of the language for multi-level compiled simulation. Some of the modifications which were required are presented. Most important of which is a scheme for the generation of zero/unit delay latch and flip-flop simulation models to match the I/O of actual cells.

HDL DRIVEN CHIP LAYOUT WITHIN THE FHDL DESIGN FRAMEWORK*

Craig D. Morency
Peter M. Maurer
Zhicheng Wang
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620

1. Introduction.

There has been much interest in the nature and flexibility of various circuit specification techniques for synthesis. Developers are having success in implementing a handful of these techniques, and desirable features of each of them are merging into what might ultimately be a standardized interface. The two main types of specification which have developed for synthesis are the graphical type and the modular programming language type. A good example of the first type is the ADAM synthesis system being developed at USC, which requires a dataflow graph as half of the specification for dataflow designs [1]. The programming language type of specification however, is the most popular approach for a more general class of circuits. The IEEE standard VHDL for example was specifically designed for flexibility (see [2] for a complete overview), and is now the foundation of continued research and development in synthesis, for several levels of design specification [3,4,5]

Current research in design automation at the University of South Florida uses USF's hardware design language FHDL as the specification language for an integrated circuit synthesis tool [6]. One desirable feature of any design specification technique is that it be easy to use, especially syntactically. Another is that its format must be capable of supporting the requirements of design at many levels of abstraction. FHDL meets these requirements, and is currently evolving both in the direction of syntheses and in the direction of specification.

2. Generating Layouts.

The FHDL language allows circuits to be specified in several different ways. In particular, a circuit may be described as a ROM, as a PLA, or as a collection of individual gates. In addition, several circuits may be combined in hierarchical fashion to create a larger circuit. We have currently implemented layout procedures for these three types of circuits. These procedures make used of several tools in the Berkeley MAGIC[7] and OCTTOOLS[8] packages. At the present time there are several steps that must be completed manually, but as the tools evolve, less and less manual intervention will be required.

The first step in creating layouts is to break the circuit into a collection of ROMs, PLAs, and standard-cell blocks. Each of these blocks is then routed to the tool capable of handling that type of block. For ROMs and PLAs this procedure is completely automatic, but for standard-cell blocks, some manual intervention is required.

---

* This work was supported by the Defense Research Projects Agency under grant2114-033-LO and by the University of South Florida Center for Microelectronics.

For PLAs, the "Mpla" tool provided with the Berkeley MAGIC package to create layouts for PLAs. The Mpla is a tiling program that creates PLAs by abutting several standard cells to create the layout of the PLA. However, the Mpla program has its own input language that must be used to specify the contents of a PLA. One approach would have been to automatically reformat FHDL specifications into Mpla specifications, but we chose instead to create our own version of Mpla that was capable of accepting FHDL as input. This was possible because the code was well structured, as illustrated in Figure 1. Because the input-parser was contained in a separate model, it was relatively straightforward to replace the Mpla parser with the FHDL parser.

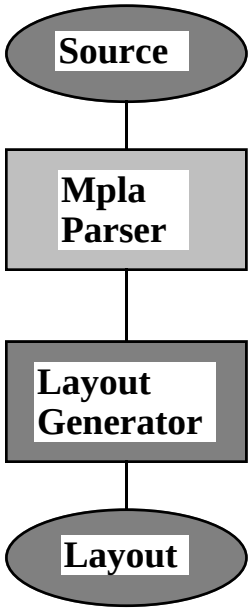

Figure 1. The Structure of Mpla.

Generating layout for ROMs is somewhat more complicated than for PLAs, because it is necessary to generate a complete address decoder in addition to the content array. One approach would have been to encode the address decoder as the AND plane of a PLA, and simply use the existing PLA layout tool, but we chose instead to generate the layout directly. This was relatively simple to do, because both the decoder and the content array can be implemented as collections of CMOS NOR gates such as that pictured in Figure 2.

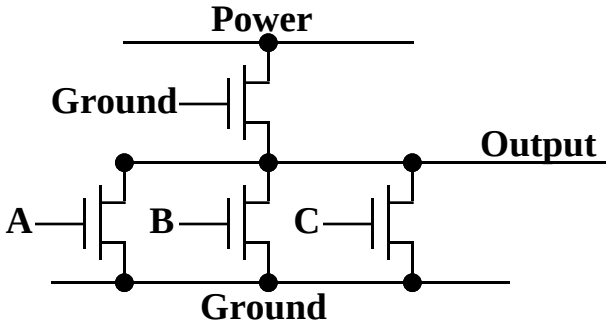

Figure 2. A NOR Gate for ROM Implementation.

The layout produced by the tool is pictured in Figure 3. (This is a simplified view of the actual layout produced by the tool.)

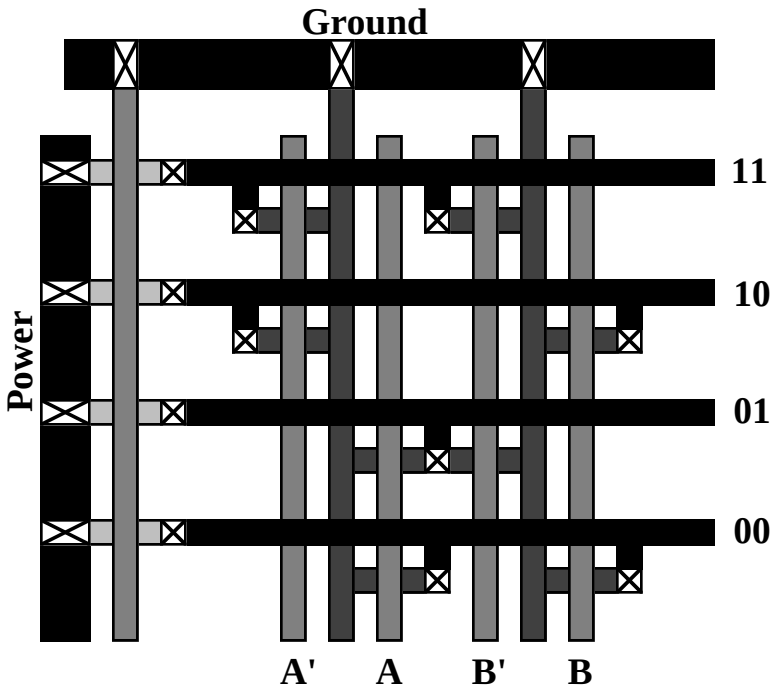

Figure 3. The Layout of a ROM Decoder.

In Figure 3, grey represents polysilicon, black represents metal, white boxes represent connections between layers, while dark and light stripes represent N and P-type diffusion. Each horizontal metal line represents one NOR gate. These lines will be attached to a similar structure that is rotated 90 degrees. Each metal line will be attached to a polysilicon line, and the NOR gates representing the content array will run vertically, with outputs at the bottom. In addition to the structures illustrated in Figure 15, there must be NOT gates to generate A' and B', and NOT gates to invert the outputs of the content array.

The process for laying out standard-cell blocks is more complex than that for generating the layouts of ROMs and PLAs, primarily because standard-cell layout often requires manual intervention to fix problems that arise in the place and route process. To facilitate manual intervention in the process, the standard cell layout tools run under the MAGIC layout editor interface. The primary change in this interface for standard-cell layout is the "place" command. Prior to beginning the layout process, each standard-cell block is placed in a separate file. It is assumed that the user is logged onto the MAGIC system, and is in the process of creating the floorplan of a chip. When the user wishes to place a standard-cell block that has not yet been laid out, he (or she) enters the following command.

place <file-name>

The layout tool then reads the specified file, parses it, and creates input for the TimberWolf automatic placement tool. The TimberWolf package is invoked to place the standard cells, and once the placement of the cells is complete, the output is converted to MAGIC internal structures. At this point the layout of the block, without routing, appears on the user's terminal. The user may then route the block by invoking the MAGIC "route"

command. Once routing is complete, the layout can be cleaned up and compressed using the MAGIC "plow" command.

3. Adapting the Simulation Framework.

Our approach to implementing a layout synthesis capability using FHDL, has been to use the framework constructed for FHDL simulation without change. This allows the same input to be use both for simulation and for automatic layout. In keeping with this goal, we originally attempted to create a single program that was capable of both simulation and automatic layout. Although this approach was successful, the resultant program was too large to be practical. The final result was a collection of programs based around a common parser, but with different code-generation phases. The two programs that are of interest here are the layout program and the simulation program.

The output of the layout program is a cell list and a net list which can be used by the place and route program. The output of the simulation program is an executable program that can be used to simulate the circuit. Because the layout process for ROMs and PLAs is quite different from the standard-cell layout process, the layout generators for ROMs and PLAs were included in the simulation program. They are invoked using a special compiler option that also bypasses the production of executable code.

The cell list produced by the layout program is unique in that it gives cell size and pin location information. For any cell which does not have alternative aspect ratios, such data represents exact final dimensions of the cell. Otherwise, the data represents a tentative choice of layout for a particular component. During the construction of a layout, this information is used to create "phony cells" which contain only the information needed to support placement and routing. This helps reduce CPU time and memory requirements, because individual cells only need to be read from storage once to extract their dimensions and pin locations. Since it is not uncommon to have to run the layout synthesis program more than once to adjust aspect ratios, the added information in the cell list can provide a significant savings in time.

4. Primitive Simulation Modeling

FHDL was originally developed as a language for the functional simulation of designs, and has since proven its flexibility for circuit specification at several levels [6]. Because of this success, additions to the language for layout synthesis have been avoided unless they have a purpose in both IC synthesis and simulation. Any change in the description of a circuit for layout purposes must must include a functional description of a circuit for simulation purposes. This rule enables a designer will be able to begin the first stages of synthesis without reconfiguring the simulation model.

A typical chip development project requires that a circuit be partitioned into components, and that these components be mapped to actual cells which are available for layout. Additionally, these components must be mapped to simulation models to be able to simulate the circuit. Consequently, there must exist a one-to-one mapping between the cells that are used for simulation, and the cells that are used for layout. For simple gates such as NANDs and NORs, this presents no problem. However, for flip-flops, this turns out to be something of a problem. The original collection of flip-flops supported by the FHDL simulation program was chosen primarily for pedagogical purposes, and had little or no relationship to the cells that were actually available for layout. Unfortunately, it is not possible to simply include the available flip-flops in the simulation model, because this set grows to meet the needs of the IC designer. Furthermore, it is impractical to include a simulation model of every type of flip-flop that could ever be used by a designer, because this set is unmanageably large. Because of these problems, it was necessary to modify the method used by the FHDL language to specify flip-flops.

Rather than referencing the built-in models for the flip-flop model which contains the exact pin configuration needed, FHDL was changed to allow a designer to simply list the

pins in the circuit specification. For such primitive component specifications, a simulation model would then constructed to match the given configuration. This is in contrast to most hardware description languages, which require that a separate models be created for each cell in the simulation model library. On the other hand, the new approach forces the user to involve himself more deeply in component simulation techniques than might otherwise be necessary. A method similar to this is presented in the paper by Dholakia et.al., but is based on interactive pop-up windows which require an extra level of designer attention [9]. This change allows the designer to run a simulation using flip-flops with certain input and output pin configurations, even when there is no layout cell available to match it. This problem can be minimized by providing the designer with a set of simulation macro-cells that describe the flip-flops actually contained in the layout library. (FHDL provides automatic mechanisms for handling such libraries.)

To illustrate the new method for specifying flip-flops, consider the example of a single or multi-bit D-type latch. The set of allowable input/output specifications for this type of latch is illustrated in Figure 4.

S	R	D	C	Q	nQ
nS	nR		nC		

Figure 4. I/O Specifications for a D-Latch.

In Figure 4, only one of the values in each column can be chosen for a given latch. S and R stand for SET and RESET respectively, while nS and nR stand for the negation of S and R. The D input is standard and defines the type of latch, while C stands for CLOCK. The Q and nQ specifications determine the type of outputs, and are independent of one another, although at least one must be present. The code for generating simulation output for these specifications is given in Figure 5. This code is written in C and generates output in C. It is assumed that S/nS dominates R/nR.

```
        count = 0;

        printf("OUT = ");

        if (found_S || found_nS)
        {
        if (found_S)
            printf("V[count] | (");
        else if (found_nS)
            printf("~V[count] | (");
        count++;
}

if (found_R || found_nR)
{
        if (found_R)
            printf("V[count] & ");
        else if (found_nR)
            printf("~V[count] & ");
        count++;
}

if (found_C)
        printf("(V[count] & V[count+1] | V[count+1] & ");
else
        printf("(V[count] & ~V[count+1] | ~V[count+1] & ");


if (found_Q)
        printf("V[count+2]);");
else
        printf("~V[count+2]);");

if (found_R || found_nR)                printf(")");

if (found_Q && found_nQ)
        printf("V[count+2] = OUT;");
        printf("V[count+3] = ~OUT;");
else if (found_Q)
        printf("V[count+2] = OUT;");
else
        printf("V[count+2] = ~OUT;");
```

The D-type flip-flop and most other storage components can be considerably more complex than this example. For example, if an edge-triggered flip-flop is specified, the generated code must contain an additional variable to test for rising or falling clock edges.

5. Conclusion

Techniques for automatically generating layout from FHDL specifications have been presented. These techniques allow for the automated layout of ROMs and PLAs and the user-assisted automatic layout of standard-cell blocks. These tools have already been of considerable value to various VLSI projects at the University of South Florida, and as the techniques evolve, will probably be of even greater value. Adaptations of FHDL and its framework to permit layout synthesis were presented. Most important of these is the provision for the generation of simulation models for primitive components, in particular

flip-flops. The implications arising from the development strategy of FHDL, is that design specification can be left open and unrestricted, while the inclusion of automatic tools for layout continues.

REFERENCES

1. Jain, Rajiv, et.al., "Experience with the ADAM Synthesis System", Proceedings 26th Design Automation Conference, 1989, pp.56-61.
2. Waxman, Ronald, L.Saunders, H.Carter, "VHDL links design, test, and maintenance", IEEE Spectrum, May 1989, pp.40-44.
3. Lis, Joseph S., Daniel D. Gajski, "Synthesis From VHDL", IEEE Int. Conf. on Computer Design, 1988, pp.378-381.
4. Newton, Authur R., A.L.Sangiovanni-Vincentelli, "CAD Tools for ASIC Design", Proceedings of the IEEE, June 1987, pp.765-776.
5. de Geus, Aart J., "Logic synthesis speeds ASIC design", IEEE Spectrum, August 1989, pp.27-31.
6. Maurer, Peter M., et.al., "The Florida Hardware Design Language", IEEE Southeastcon Proceedings, 1990.
7. Sechen, C. and A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," *Custom Integrated Circuits Conference*, May 1984.
8. J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, G. S. Taylor, "The Magic VLSI Layout System," *IEEE Design and Test of Computers*, Vol. 2, No. 1, Feb. 1985.
9. Dholakia, Suresh, et.al., "Compilation of Standard Cell Libraries", IEEE Custom Integrated Circuits Conf., 1986, pp.343-346.